The Do's and Don'ts of Quantum Memory in Cryptanalysis



Samuel Jaques
St. Cross College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Trinity 2023

Acknowledgements

A massive thank you to my supervisors, Simon C. Benjamin and Christophe Petit. Science is a method of knowledge, but it is embedded in human institutions, and they gave invaluable help in both aspects.

My academic "siblings" provided huge support. I am glad I had a group that would take all my questions seriously, from the painfully obvious to the unanswerably vague. Thank you to Andrea Basso, Gregory Boyd, Zhenyu Cai, Hans Chan, Adrian Chapman, Mingjie Chen, Daniel Dafcik, Tako Boris Fouotsa, Valerie Gilchrist, Matt Goh, Cica Gustiani, Hamza Jnane, Tyson Jones, Bálint Koczor, Sreehari Kollath, Péter Kutas, Yi-Fu Lai, Abel Laval, Laurane Marco, Sam McArdle, Richard Meister, Simon-Philipp Merz, Romy Minko, Tomoki Moriya, Carlos Outeiral, Arthur Rattew, Cai Shiping, Adam Siegel, Miha Stopar, Armands Strikis, and Charlotte Weitkämper.

During my DPhil I had the opportunity for several internships, which helped broaden my work and fund the DPhil itself. My internship mentors (and co-authors), Michael Naehrig, Thomas Häner, Hart Montgomery, and Arnab Roy, deserve big thanks.

All of my co-authors were great to work with and of course the research in this thesis would not exist without them. Thank you Xavier Bonnetain, Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Craig Gidney, Mike Lodder, Martin Roetteler, Francisco Rodríguez-Henríquez, Razvan Rosie, André Schrottenloher, Fernando Virdia, and others mentioned above.

The support from my wife Tiffany Cassidy was indispensable. She helped me keep a grounded perspective on this journey and patiently listened to my incomprehensible rants about quantum algorithms.

Thanks to my dad and my sister, Kevin Jaques and Natasha Jaques, for sparking an interest in science and computers, and also to my mom, Paula Sostorics, for keeping me interested in the rest of the world.

Every student visa and immigration adviser I talked to before and during my DPhil was tremendously helpful, thank you all.

Finally, I have to thank the essential help from Özlem Türeci and Uğur Şahin, Alexandra Elbakyan, Paul Ginsparg, Joppe W. Bos and Tancrède Lepoint, and numerous StackExchange users.

Abstract

This thesis refines the estimates of the costs to attack three different cryptographic problems. To get there, I first survey some fundamental issues in quantum computing. This narrows the scope of the capabilities of quantum computers and the engineering costs to build and run them. I then focus on "QRAM", a type of quantum memory. The question with QRAM is whether quantum computers could cheaply access memory in superposition. I argue that they could not, and that quantum computers must pay a steep price to access memory.

I then analyse circuits to attack elliptic curve cryptography with Shor's algorithm. I find that even expensive QRAM improves these circuits. Next I provide some optimisations and parallelisation strategies for Kuperberg's algorithm. This algorithm can attack "CSIDH", a type of post-quantum cryptography. Again QRAM is an effective tool: huge classical co-processing costs offset the expense of each quantum memory access. Finally I provide an algorithm for the *golden collision problem*, a generalisation of "meet-in-the-middle" attacks. If stable, error-free memory exists, then this algorithm could acheive an exponential advantage in total cost. However, the run-time of this algorithm is too high, so I show a more effective use of massive arrays of quantum hardware. This second algorithm has a higher cost but parallelises nearly perfectly.

Contents

List of Figures **xi**

1 Introduction **1**

1.1 This Thesis 3

1.2 Outline 5

2 Background **7**

2.1 Notation 7

2.2 Quantum Computing Background 8

2.2.1 The Memory Peripheral Framework 8

2.2.2 Quantum Memory Management 10

2.2.3 Quantum Amplitude Amplification 10

2.3 Cryptography Background 11

2.3.1 NIST Security Levels 12

3 Fundamental Costs **13**

3.1 Gate-Based Computation 14

3.1.1 Active Error Correction 15

3.1.2 Passive Error Correction 19

3.1.3 Other Cost Metrics 21

3.2 Ballistic Computation 22

3.2.1 Hamiltonian Locality 23

3.2.2 Building up or Swapping in 24

3.2.3 Errors 27

3.3 Gates 28

3.3.1 Non-Reversible Gates 28

3.3.2 Reversible Gates 29

3.4 Conclusions 37

4 Quantum Random Access Memory **39**
4.1 Introduction . . . 40
4.2 Definitions and Notation . . . 42
4.2.1 Routing and Readout . . . 44
4.2.2 Alternate Notation . . . 45
4.2.3 Context . . . 46
4.3 Circuit QRAM . . . 47
4.3.1 Lower Bounds . . . 48
4.4 Gate QRAM . . . 49
4.4.1 Active vs. Passive QRAM . . . 50
4.4.2 Path Independence . . . 50
4.4.3 Hamiltonian Construction . . . 51
4.5 Errors . . . 52
4.5.1 Error-corrected components . . . 53
4.5.2 Error Correction . . . 54
4.5.3 Gate Teleportation . . . 55
4.6 Classical RAM . . . 58
4.6.1 Area-Time Metric . . . 59
4.6.2 Active Components . . . 60
4.6.3 Error Propagation and Correction . . . 63
4.6.4 Latency . . . 64
4.7 Conclusion . . . 65

5 Attacking Elliptic Curve Cryptography **67**
5.1 Introduction . . . 68
5.1.1 Discrete Log . . . 69
5.1.2 Elliptic Curves . . . 69
5.1.3 Shor's Algorithm . . . 70
5.2 Arithmetic . . . 71
5.2.1 Multiplication . . . 71
5.3 Windowing Shor's Algorithm . . . 77
5.3.1 Signed Look-ups . . . 77
5.3.2 Memory-efficient Look-ups . . . 79
5.3.3 Cost Analysis . . . 79
5.4 Q# Resource Estimation . . . 80
5.4.1 Depth-Width Inconsistency . . . 81
5.4.2 Measurement Feedback . . . 82
5.4.3 Initial Results . . . 83
5.5 Error Correction . . . 83
5.5.1 Methods . . . 84
5.5.2 Results . . . 87

6 Attacks on CSIDH **91**
6.1 Background . . . 92
6.1.1 Group Action Cryptography . . . 92
6.1.2 CSIDH . . . 93
6.2 The Cost of Kuperberg's Algorithm . . . 95
6.2.1 Phase Vectors . . . 95
6.2.2 Creating Phase Vectors. . . . 96
6.2.3 Collimation . . . 102
6.2.4 Measuring Phase Vectors . . . 108
6.2.5 Summary . . . 111
6.2.6 Cost Analysis . . . 113
6.3 Group Structure . . . 115
6.3.1 Group Representations . . . 116
6.3.2 Incomplete Superpositions . . . 117
6.3.3 Effects of Incomplete Superpositions . . . 120
6.4 Conclusion . . . 126
6.4.1 Security Conclusions . . . 126
6.4.2 SQALE'd CSIDH . . . 127
6.4.3 Discussion . . . 129

7 The Claw-Finding and Golden Collision Problems **131**
7.1 Introduction . . . 134
7.1.1 Golden Collision Finding . . . 134
7.1.2 Quantum Random Walks . . . 136
7.2 Iteration-Based Quantum Random Walk . . . 139
7.2.1 Iterating a function . . . 140
7.2.2 Quantum Data Structure . . . 141
7.2.3 Fraction of marked vertices . . . 147
7.2.4 Summary . . . 150
7.3 Outlook and Comparison . . . 150
7.3.1 Comparison with QAA . . . 151
7.4 Parallel Search . . . 154
7.4.1 Latency . . . 158
7.4.2 Iteration-based . . . 159
7.5 Discussion . . . 160

8 Conclusion **165**

Appendices

A Hamiltonian Limits **169**
A.1 Complexity of Long Hamiltonians 169
A.2 Coherent Evolution Limit 173
A.2.1 Shifts of vectors 173
A.2.2 Energy and Coherence Theorem 186

B Proofs of QRAM Lower Bounds **191**
B.1 Proof of Lemma 4.3.1 191
B.2 Proof of no distillation 192

C QRAM Circuits **197**
C.1 Unary 197
C.2 Select-Swap 198
C.3 Uncomputation 201

D Pebbling **203**
D.1 CSIDH Group Action Pebbling 205
D.2 Claw-Finding Pebbling 206

E Proof for Chapter 6 **209**
E.1 Proof of Theorem 6.3.2 209

F Properties of Iterated Functions **213**

Bibliography **219**

List of Figures

3.1 Non-reversible energy requirements 30
3.2 Energy landscape of a single bit 31
3.3 Laser and qubit interaction 31
3.4 Energy for coherent reversible gates 35
3.5 Time-energy tradeoffs 36

4.1 QRACM distillation and teleportation 56

5.1 Elliptic curve group laws 70
5.2 Montgomery form modular multiplication circuit 73
5.3 Windowed modular multiplication circuit 75
5.4 Signed, windowed, elliptic curve point addition circuit 79
5.5 Q# parallelism 81
5.6 Q# depth-width tradeoff 81
5.7 T gate teleportation 85
5.8 ECDLP qubit types 89
5.9 A slow `CNOT` routing layout for a surface code 90

6.1 Phase vectors 97
6.2 Combining two phase vectors 103
6.3 Collimating a phase vector 106
6.4 Collimation tree 112
6.5 Overlap of search spaces with small CSIDH exponents 119
6.6 Collimation with errors 122
6.7 Simulated noisy collimations 123
6.8 CSIDH attack costs 128

7.1 Claw-finding 132
7.2 Claw-finding to golden collision reduction 135
7.3 Reversible function iteration 141
7.4 A collision of two iterations 142
7.5 Iterating and checking for a collision 143
7.6 Vertex marking state diagram 145

7.7 Parallel quantum claw-finding, part 1 155
7.8 Parallel quantum claw-finding, part 2 156
7.9 Qubit maximum size for 100-year claw-finding attack 162

A.1 Triangular vectors 174
A.2 Proof of Lemma A.2.2 175
A.3 Proof of Lemma A.2.3 176
A.4 Proof of Lemma A.2.4 177
A.5 Proof of Lemma A.2.4 178
A.6 Proof of Lemma A.2.5 179
A.7 Examples of maximum-energy vectors with bounded shifts 181

C.1 Recursive unary QRACM 198
C.2 Select-swap QRACM access 200
C.3 Fanout-and-swap QRAQM 201

1 Introduction

The year is 1975. A burgeoning computer infrastructure demands data security, and you are developing the new Data Encryption Standard (DES). IBM has presented a promising design, but the NSA has mysteriously requested the keys be shortened from 64 bits to 56 bits [Joh98]. What should you do?

The best attack is a brute force key search, so with 56-bit keys, there are only 2^{56} keys to check. The best computer at the time is the Cray 1, capable of 160 million operations per second [Tat17]. If each "operation" can test one key, it will take 171 years to crack a DES key, so you might think you are safe.

Yet, a well-funded adversary could buy 171 Cray 1 computers for USD$7.5 billion (2023 dollars) [Nat09] and crack DES in one year. Or, following Moore's law (already 10 years old in 1975), you might predict the number of transistors – and, presumably, processing power – will double every 18 months or so, and by 1986 the top supercomputer could break DES in a year.

Indeed, DES broke quickly. Not only did computers get cheaper and faster, they also became ubiquitous. The first public break was in 1997, using a distributed array of personal computers [CD97]. Today, modern graphics cards, less than USD$1,000, can break DES in 15 days on average [Gos17].

With hindsight, we can easily see that hardware advances would quickly make 56 bit keys (and even 64 bit keys) insecure.

The year is 2001. You are standardising a replacement for DES, the Advanced Encryption Standard (AES). Now you have decided the minimum key size is 128 bits, with options for 192 and even 256 bits. The best supercomputer at the time can manage 7.3 Terraflops [SDS+01], so if you can compile one AES encryption to a single floating-point operation, it still needs 14 million billion years to break AES-128. If the attack difficulty halves every 18 months, you get about 80 years before this might be possible. That is a lot of breathing room.

With hindsight, that was a great decision. Hardware progressed at the expected speed, no unexpected attacks appeared, and AES became an incredibly useful and ubiquitous standard.

The year is 1977. You are one of Rivest, Shamir, and Adleman, and you have just invented one of the first public-key encryption schemes, named RSA (after yourself) [RSA78]. Fundamentally, its security depends on the hardness of factoring large integers, so you need to choose parameters large enough that factoring is hard.

The best method to factor an n-bit number has complexity $\exp\left(O(\sqrt{n \ln n})\right)$, so you estimate that a 664-bit number will take 3.8 billion years. That is a huge safety margin: with exponential scaling in computing power, it should be 2025 before someone could break this in one year.

However, in 1993, a completely new factoring method is invented [BLP93], with complexity $\exp\left(O(n^{1/3}(\ln n)^{2/3})\right)$. This new "number field sieve" quickly factors huge numbers, and by 2005 an academic team factors the first 663-bit RSA number [BMFK05] – 20 years ahead of the original extrapolation.

In 2009, NIST (a U.S. technology standards organisation) recommended that RSA should use 3072 bits [BBJ+09], based on the complexity of the number field sieve and aiming for an attack complexity of about 2^{128} operations, at least 90 trillion years on the best hardware at the time [SDS+09]. This bought them around 46 doublings before such an attack could be feasible, so maybe 70 years with Moore's law. This parameter set buffers against hardware advances *and* algorithmic

advances. Yet, despite being 30 years old, the number field sieve is still the fastest classical, general-purpose factoring algorithm. Instead of breaking, RSA became so slow that it was almost entirely replaced by *elliptic curve* cryptography (ECC), which has seen no radical improvements to generic attacks since its inception.

With hindsight, it was premature in 1977 to decide on parameters based on known attacks, but by 1993, the best attacks plateaued. How could you have known that algorithmic advances would proceed in this way? In 2009, was 3072 bits overkill?

The year is 2019. You are on the team for "Supersingular Isogeny Key Encapsulation" (SIKE), a scheme invented in 2011 and proposed to be safe against quantum attack [JAC+17], and you are hoping that NIST standardises it. Based on a quantum claw-finding attack, you had previously decided that you need 503-bit primes to match the security of AES-128. Yet, two recent papers [ACVCD+, JS19] argue that you have *over*-estimated the security: you can justifiably lower the key size to 434 bits and claim the same security against known attacks. This is a tempting offer, since your scheme is one of the slowest available, and the smaller keys will make it about 30% faster.

In fact, the conclusions in those papers still hold, at least about the attacks they analyse. But in 2022, an unexpected *algebraic* attack completely broke the scheme [CD23, MMP+23, Rob23]. Now, consumer hardware can break the original 503-bit parameters in under an hour.

With hindsight, SIKE was always doomed. The attack is polynomial time, and even though large enough parameters could still be safe, they would be hopelessly slow to actually use. But how could you have predicted this? Was it reasonable to lower the parameters, based on analyses of known attacks, since we have no knowledge of more efficient attacks until it is too late?

1.1 This Thesis

The year is 2023. Our current cryptographic infrastructure relies on RSA and elliptic curves. *Quantum* computers could break either method in polynomial time,

so we want to switch to cryptography that is safe against quantum computers. Yet how do we know what will be hard for quantum computers? How can we guess how hard any problem needs to be for it to be safe? How soon do we need to switch?

All of these questions mirror the questions that cryptographers faced over the last 50 years. We want to be like AES, not DES; like ECC, not RSA. For quantum computers, the questions might even be harder, because quantum computers of today are nearly useless. The best quantum computer today can only run a few hundred operations before it is overwhelmed with errors [AAB+19].

The risks of being too pessimistic about a quantum attacker are obvious: someone will build a powerful quantum computer and break our computer security. Yet being too conservative is risky also.

Firstly, many current schemes are "battle-hardened", with decades of research on the fundamental problems, implementations, and protocols. As an example, it would be tragic to "upgrade" Signal (an encrypted messaging protocol) to protect against a future quantum attack and discover that the implementation was faulty. Major news outlets use Signal to protect their sources; given how dangerous whistleblowing can be, a faulty Signal implementation could kill people.

Second, engineers want efficiency, and will strip away aspects of security to get it (ex: [RDK+22], SHA-2 is still used in all secure internet connections, despite the standardisation of SHA-3 [WV21]). If we set parameters of some post-quantum scheme to protect against a planet-sized quantum attack, users might simply not use it. They might keep using ECC, and then they will be vulnerable when quantum computers get big enough to break ECC.

Some quantum attacks are practically impossible because they use galactic scale algorithms or rely on a stack of tenuous assumptions about quantum engineering. Other attacks are ambitious but possible, and it is sensible to defend ourselves from these. To separate these kinds of attacks and those in between, in this thesis I look carefully at the fundamental notions of cost in quantum computing, especially around quantum *memory*, and specifically on "QRAM". I then apply these

conclusions to some particular quantum attacks, finding some that are plausible and others that are not.

What does it mean if the best attack, quantum or classical, is outside of any sensible threat model? As RSA and SIKE show us, breakthroughs in cryptanalysis could destroy any scheme. Predicting this kind of progress is even more difficult than predicting hardware progress. One might still choose schemes or parameters where the best attacks are galactic scale, not to protect against those attacks, but to protect against unknown cryptanalysis progress. However, a few extra bits of security might be useless against new cryptanalysis. In cases where the best quantum attacks are outlandish, one might decide that a few more bits of security adds little *marginal* benefit. Practical concerns like performance and ease-of-implementation could outweight this benefit.

There are also other avenues to this kind of security. "Hybrid" schemes, using both post-quantum and traditional cryptography, can be proven safe as long as *either* scheme is secure. Someone who wanted to use SIKE would have been safe if they used a hybrid scheme, but not if they used the 900-bit parameter set.

1.2 Outline

I begin with a background on quantum computing in Chapter 2. Here I will also discuss the notion of a quantum attack, to frame the kinds of questions this thesis aims to address.

In Chapter 3, I consider the fundamental costs of a quantum computation. The goal here is to somewhat constrain the capabilities of a quantum attacker, while remaining sufficiently open to improvements in quantum technology. I prove some bounds on "ballistic" computation and the energy costs of certain computational technologies. This chapter will also explain the various notions of "cost" that I will use in all subsequent chapters to evaluate quantum algorithms.

Chapter 4 consists of several sections of my work in [JR23] and starts to apply the notions from Chapter 3 to the concept of quantum random access memory (QRAM). I will explain what QRAM is and how it might be constructed as a circuit,

and at what cost. I also criticise the notion of QRAM as a single operation, using the previous arguments against ballistic computation.

With this background in place, I will step through progressively more implausible quantum algorithms. This starts in Chapter 5, where I give circuits to attack elliptic curve cryptography, a type of cryptography used to secure 98% of secure internet connections [WV21]. This follows my work published as [HJN+20]. This chapter focuses on the effective use of QRAM, where an attacker can use expensive QRAM circuits and still see an advantage in computation cost.

Moving on, Chapter 6 addresses the quantum attacks on "CSIDH", a key encapsulation method proposed to be quantum-secure. This follows my work in [CSCDJRH21], and explores quantum circuit optimisations and time-memory trade-offs. Here again, QRAM is an effective tool, and I conclude that the dominant cost in the attack is classically processing a table of memory, not the quantum access. I further discuss some justification for a riskier choice of key space for CSIDH.

In Chapter 7, I consider the golden collision problem, for which the best algorithms are purely exponential. This is based on my work [JS21], which shows that for this problem there is an exponential advantage to using QRAM, even accounting for the expensive access costs, *if* we make a dubious assumption that the quantum memory remains coherent without active error correction. We also provided the best parallel algorithm for the problem.

2 Background

2.1 Notation

Be the change you want to see in the world.

Unknown; often attributed to Mahatma Gandhi

I use the circle constant $\tau \approx 6.28\ldots$, defined as the ratio of a circle's circumference to its radius, instead of the more typical π.

Instead of the typical scientific notation of $x \cdot 10^y$, I will use a more computer-science-friendly powers-of-two notation, i.e., $1.x \cdot 2^y$. For those more familiar with powers of 10, just divide the exponent by 3 for a good approximation.

Throughout this thesis I will use the asymptotic notation $O(\cdot)$, $\Omega(\cdot)$, $\Theta(\cdot)$, $o(\cdot)$, and $\omega(\cdot)$. Each of these is a set of functions, though I will occasionally abuse notation and write (e.g.) $f(n) = O(n^4)$, or $f(n) = 3n^2 + O(n)$.

Without giving formal definitions, I briefly explain:

- $f(n) \in O(g(n))$ means $g(n)$ is an asymptotic upper bound for $f(n)$;
- $f(n) \in \Omega(g(n))$ means $g(n)$ is an asymptotic lower bound for $f(n)$;
- $f(n) \in \Theta(g(n))$ means $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$, i.e., they have asymptotically the same order of growth;

- $f(n) \in o(g(n))$ means $g(n)$ is asymptotically a "strict" upper bound for $f(n)$, i.e., $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
- $f(n) \in \omega(g(n))$ means $g(n)$ is asymptotically a "strict" lower bound for $f(n)$.

Following the standard cast of characters in cryptography, I imagine a scenario where Alice and Bob are trying to communicate securely, and Eve is trying to eavesdrop, sabotage, or otherwise attack their communication. Thus, in this thesis Eve will always be the one who owns and runs a quantum computer.

2.2 Quantum Computing Background

I assume a basic familiarity with quantum computing, and this section describes how this thesis will treat quantum computers. For a thorough introduction to quantum computing, [NC00] is standard choice, and for a faster crash course I recommend the introduction of my master's thesis [Jaq19].

I use "computational basis" to refer to the basis $\{|0\rangle, |1\rangle\}$ and any tensor product of these vectors. Throughout, I ignore global phases and normalisation of quantum states. When describing algorithms, I will ignore steps that add or remove qubits in the state $|0\rangle$, and use $|0\rangle$ sometimes to refer to a multi-qubit zero state, $|0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle$.

2.2.1 The Memory Peripheral Framework

This thesis will work in the Memory Peripheral framework of [JS19]. Section 2 of that paper describes the mathematical formalism, so here I will give an intuitive definition. At its core, the goal is to model quantum computers as a peripheral of a classical computer.

The framework considers *memory controllers* and *memory peripherals*. The memory peripheral will be a physical device like a qubit, a classical DRAM chip, a magnetic tape, etc. The memory controller performs *memory operations* on the memory peripheral, which modify the physical state of the memory peripheral.

Quantum gates and measurements are memory operations on a qubit; read/write instructions are operations on DRAM and tape.

Between operations, the state in the memory peripheral may evolve naturally because of the physical properties of the memory peripheral. A qubit could degrade because of noise (as could DRAM or tape). However, a more sophisticated memory peripheral might do something useful between operations. If the Hamiltonian of a quantum system is precisely managed, the evolution of that Hamiltonian can do useful computation.

To capture this, a memory peripheral is represented as $\mathsf{A} = (\mathcal{H}^{\mathsf{A}}, H^{\mathsf{A}})$, where $\mathcal{H}^{\mathsf{A}}$ is a Hilbert space representing possible computational states (e.g., $\mathbb{C}^2$ for a qubit) and H^{A} is the Hamiltonian for that space (e.g., 0 for a static, error-free qubit; X for a qubit undergoing rotation). Memory operations can either directly modify the state in the Hilbert space $\mathcal{H}^{\mathsf{A}}$, or add terms to the Hamiltonian H^{A}. I will refer to a memory peripheral with $H^{\mathsf{A}} = 0$ as "static".

Costs in the Memory Peripheral Framework

Given a memory peripheral model, there is also a *memory controller*. This is a co-processor for the machine, which is responsible for scheduling and managing the computation. Depending on the computational model of the memory controller, we will add some operations that allow it to apply a certain subset of memory operations to the memory peripheral. For example, a parallel random access machine (PRAM) could have a set of instructions that allow it to apply any Clifford+T+measurement operation to specific qubits of qubit memory. Or a more complicated model (representing a *less* powerful machine) could have restricted instructions, where it can only apply T gates to certain kinds of qubits.

Each of these operations will be given some cost, in the same units as any other computations the memory controller might perform. In the case of a PRAM, classical operations are often given unit cost, so we can give unit cost as well to the memory operations. Or in a more complex model that accounts for latency, we might give a cost in units of time based on the physical location of the memory operation.

The goal here is to account for the cost of a computation in terms of the interventions that a controller must perform. This allows a direct comparison between classical and quantum costs, as it measures quantum costs in units of classical computation. In many cases this will ignore significant costs of computation; for example, counting PRAM operations ignores the energy costs to cool certain types of quantum computers. However, the flexibility of this framework means that if I decide I *do* care about energy, I can instead cost the PRAM computation in energy, and assign appropriate energy costs to each operation on the memory peripheral.

2.2.2 Quantum Memory Management

Throughout, I will use the concept of *in-place* versus *out-of-place* operations. Given a function $f : \{0,1\}^n \to \{0,1\}^m$, an in-place circuit to compute $f(x)$ from a state x will have the action

$$U_f |x\rangle = |f(x)\rangle \tag{2.1}$$

while an out-of-place circuit has the action

$$U_f |x\rangle |0\rangle^m = |x\rangle |f(x)\rangle . \tag{2.2}$$

Because of reversibility, we cannot have an in-place circuit for a non-invertible function. However, some invertible functions still do not have efficient in-place circuits.

2.2.3 Quantum Amplitude Amplification

Quantum amplitude amplification (hereinafter "QAA") [BHMT02] is an essential tool in quantum cryptanalysis. As a generalisation of Grover's search algorithm [Gro96], it allows Eve to solve any search problem with a square-root speed-up. That is, suppose she has a unitary $U_{\text{diffusion}}$ such that

$$U_{\text{diffusion}} |0\rangle^n = |\psi\rangle := \alpha |\psi_{\text{good}}\rangle + \beta |\psi_{\text{bad}}\rangle , \tag{2.3}$$

and she has another unitary U_{check} such that

$$U_{\text{check}} |\psi_{\text{good}}\rangle = - |\psi_{\text{good}}\rangle$$

$$U_{\text{check}} |\psi_{\text{bad}}\rangle = |\psi_{\text{bad}}\rangle .$$

Then QAA allows Eve to prepare the state $|\psi_{\text{good}}\rangle$ with $\Theta(\frac{1}{|\alpha|})$ calls to $U_{\text{diffusion}}$ and U_{check}. Since she has a $\frac{1}{|\alpha|^2}$ chance of measuring $|\psi_{\text{good}}\rangle$ if she measures the initially prepared state, QAA is considered a square-root speed-up.

Typically, we use QAA for general search problems: Eve is searching for a bit string that satisfies some property (e.g., it is the secret key to some cryptosystem). She can easily make a uniform superposition of all bit strings with H gates as $U_{\text{diffusion}}$, and implement a check for the property as U_{check}. In this case, if the proportion of "good" bitstrings is p, then she needs $\Theta(1/\sqrt{p})$ iterations, compared with $\Theta(1/p)$ classically.

QAA parallelises badly [Zal99]. The optimal strategy is equivalent to dividing the search space among different processors. With P processors, each one must do $\Theta(\frac{1}{\alpha\sqrt{P}})$ iterations, but this means the total number of calls to each unitary (added up over all the processors) *increases* with the number of processors, to $\Theta(\frac{\sqrt{P}}{\alpha})$.

2.3 Cryptography Background

Very little of this thesis depends on specific notions in cryptography. All of the attacks are on *public-key* cryptography, a type of cryptography where Alice computes a private key and a public key, and publishes the public key. Bob uses the public key to encrypt a message, and Alice can use her private key to decrypt it.

I consider only key recovery attacks on such a system: Eve takes Alice's public key (and any parameters of the scheme) and computes Alice's private key from the public key. This is a simple passive attack that completely breaks the public-key scheme, since Eve can decrypt any message from Bob once Eve has Alice's private key. Hence, it should be hard, and my motivating question is deciding how hard this attack is for Eve.

Each chapter will explain specific cryptographic schemes. Throughout, I will reference AES, the "Advanced Encryption Standard" [U.S01]. This is a symmetric key scheme, meaning Alice and Bob must already share the same secret to use it securely.

2.3.1 NIST Security Levels

The National Institute of Standards and Technology (NIST) is standardising post-quantum cryptography. As part of this process they defined security targets for different schemes. These targets are set relative to the security of symmetric cryptography, specifically AES and the Standard Hash Algorithm (SHA). Specifically, they are set as in Table 2.1.

Security Level	Threshold
1	AES-128 key search
2	SHA-256 collision search
3	AES-192 key search
4	SHA-384 collision search
5	AES-256 key search

Table 2.1: NIST Security Levels.

3
Fundamental Costs

Contributions. The first section is largely a survey of existing works, and a background on error correcting codes. The second and third are original to this thesis and otherwise unpublished.

I start by considering fundamental costs of a quantum computation. My goal is to attempt some lower bounds on how hard a quantum computation can possibly be.

This chapter has three sections. The first explores a gate-based model of computation, where we are limited to a small set of simple interventions (e.g., Clifford+T gates). This will be the cost model I use in later chapters for actual cryptanalysis, and as such this section will be limited to more grounded statements about quantum computers.

The second section will explore a more permissive model of computation, where the computer performs complicated operations without intervention from the memory controller. This will be a bit more speculative.

Finally, the third section tries to set some even more fundamental limits, such as the minimum amount of energy for one gate. This will be the least rigorous, but gives intuitive context for later results.

When attempting to model a quantum computer at this level, we have to make some decisions about realism. For example, a surface code *works*, provably, so I will call it a "realistic" guess for a quantum computing architecture in the near-to-mid future. In Chapter 5 I will assume a surface code architecture, and claim specific amounts of specific resources to attack certain kinds of cryptography. If someone invents a better architecture, then these claims will be wrong. The farther into the future I imagine the attack, then the more likely that a "realistic" assumption is proven wrong by a theoretical advance. Thus, I will refer to assumptions as more "conservative" when they are optimistic about future progress in the field – such as efficient asymptotically good LDPC codes (see Section 3.1.1). The most "conservative" statements will be bound only by provable limits, such as the speed of light and the no-signalling theorem.

3.1 Gate-Based Computation

My first notion of a quantum computer is as a memory peripheral where we construct all memory operations out of a small set of basic operations. The motivating example is a qubit memory with Clifford+T+measurement operations, but we could also

imagine a different gate set or qudits (i.e., the basic memory peripheral object is $\mathsf{Q} = (\mathbb{C}^d, 0)$ in the notation of Section 2.2.1).

More precisely, I restrict the memory controller to have access to a finite set of memory operations of bounded arity. Some real-world systems seem to contain an infinite family of gates, such as all single-qubit rotations, but in practice there is a finite level of precision in these rotations, so I can assume we have only a finite set of rotations without losing any power in the model.

Ultimately, we want to assume that each memory peripheral is static (the Hamiltonian is 0), but this is not a realistic assumption. On a spectrum of engineering capability, memory peripherals can be worse than neutral; that is, the natural evolution may subject the system to noise. We can model this by having each memory peripheral include a thermal bath, e.g., $\mathsf{Q} = (\mathbb{C}^2 \otimes \mathcal{H}^B, H_{SB} + H_B)$ where $H_{SB}(T)$ is an interaction term. Basic state preparation operations will create the state $|0\rangle\langle 0| \otimes \rho_\beta$, where ρ_β is the Gibbs state of temperature β.

The specifics of the interaction between the system and the bath will govern exactly how noisy this system is. An exact model would be useful for NISQ-type systems, where we optimise our algorithms for the physical system and its noise. For cryptanalysis, we instead want to abstract away the noise.

For that purpose, I assume there is an error-correcting code. I follow [JS19] and split into two broad types: *active* and *passive*.

3.1.1 Active Error Correction

Active error correction means that, at finite temperature, the system will decohere without intervention. The memory controller must continuously intervene to preserve the state.

An example is surface code error correction, where measurement cycles are necessary to preserve the quantum states.

The model I use for this system is based on *logical* qubits, where the space of memory peripherals is generated by $\mathsf{Q} = (\mathbb{C}^2, 0)$. The memory operations do not modify the states' Hamiltonian; therefore, we are limited to quantum channels of

bounded arity. This is, up to polynomial factors, equivalent to the set of Clifford+T gates, as well as measurement in the $\{|0\rangle, |1\rangle\}$ basis and state preparation of $|0\rangle$. I will refer to this as the set of Clifford+TM operations.

Because one must intervene to correct errors, all memory operations, including the identity, carry some cost to the memory controller. [JS19] took a conservative assumption of unit cost per memory operation. This is a lower bound for this model, but I will expand on more realistic assumptions in Chapter 5 and Chapter 6.

This is the model most unfavourable to a quantum attacker, with huge impacts on cryptanalysis.

Surface Codes

Surface codes are a promising family of quantum error-correcting codes. A two-dimensional lattice of qubits with nearest-neighbour connectivity can implement a surface code.

I will explain only the relevant features of these codes, not the details. Good references (from which most claims in this section derive) are [FMMC12] and [HFDM12]. A surface code has a *distance* d, such that:

- each logical qubit requires $2(d+1)^2$ physical qubits,
- many logical operations take time proportional to d *cycles*, each of which consists of a state preparation, 4 sequential CNOT or C-Z gates, and a measurement,
- the resulting logical error rate is approximately equal to $0.1 \cdot (100\epsilon)^{\frac{d+1}{2}}$, where ϵ is the error rate of the physical qubit operations.

The final formula is an approximation from [GE21], and depends on the specific error rates of the different components. Notice that if $\epsilon \geq 1/100$, the error rate does not decrease with distance. This forms a "threshold", below which the surface code is helpful, and above which it is not. Though analytic lower bounds exist (e.g. [Fow12]), the threshold is often determined with simulation experiments and depends on the specific error model.

To correct errors, Eve must measure half of the qubits and solve a minimum-weight perfect matching problem for each logical qubit. This must be done roughly every d surface code cycles. This is a lot of computation for some classical co-processor to solve these problems. Detecting an error projects it into either logical X or Z errors, which the co-processor will correct "logically" by commuting them through all subsequent operations.

Costs. With W logical qubits and a code distance d, Eve must process at least $d^2\mathsf{W}$ bits of information in every code cycle, and solve W matching problems every d cycles. Either the classical co-processor is at least W times faster than the cycle time, or – much more likely – Eve uses an array of parallel co-processors, on the order of one per logical qubit. Thus, a surface code with many logical qubits implies a significant array of classical co-processors.

Either way, if she runs the computation for some time t, then the total computational power she *could* have employed is proportional to t times the number of logical qubits. We could also consider the total classical computation performed (which will be proportional to energy consumption), which is more complicated because Eve might be able to adaptively "switch off" portions of the machine and stop error-correction on them, if she does not need those qubits.

Overall, a good first approximation for the cost of a surface code computation is the number of physical qubits times the number of cycles. Chapter 6 and Chapter 7 take this approach, while Chapter 5 goes into more detail.

Other Active Codes

While surface codes are versatile and effective, their asymptotic *rate* is 0: the ratio of logical qubits to physical qubits actually decreases as the code size increases. They also have difficulties with T gates.

We must thus ask what other codes the future could hold. First, I dismiss codes where the *syndrome measurements* have more than a constant number of interactions. A syndrome measurement is a measurement of some joint property of a set of qubits (e.g., their parity), which detects whether errors have occurred

and projects the states onto particular errors when they do. Hamiltonian locality justifies the restriction to constant interactions: I do not allow a single operation to interact with that many qubits. While we might be able to build up such a syndrome measurement from sequential smaller ones (which is precisely what the surface code does), this increases the amount of time between subsequent syndrome measurements, which increases the effective physical error rate (i.e., the rate of errors between measurements), and it requires building a large stabiliser measurement out of gates of bounded fanin.

Similarly, I disregard any code where each qubit is involved in more than a constant number of syndrome measurements, since contention for that qubit will also increase the time or number of operations of each measurement.

Codes satisfying these restrictions, having fixed-weight syndrome measurements and a constant number of syndromes for each qubit, are called *low-density parity check* (LDPC) codes. Recent results constructed asymptotically good LDPC codes for quantum computers [PK21, PK22]; however, they require long-range interactions for the syndrome measurements. In fact, a result of [BK22b] shows that long-range interactions are *necessary*: if the code has distance $d \propto n^\alpha$, it will require $\tilde{\Omega}(n)$ interactions of length $\tilde{\Omega}(n^{\alpha/2})$.

Are long-range interactions feasible? Chapter 7 will discuss signal propagation limits, but at the size of a single logical qubit, we may not care. Certainly such codes are infeasible for today's superconducting qubits, which have only nearest-neighbour connectivity. Trapped ion quantum computers are brought up often as a candidate architecture, since the ions in one trap have all-to-all connectivity. However, it may not be feasible to load a single trap with the number of ions needed for these LDPC codes[1]. Other issues include the design of efficient operations on the logical qubits.

For these reasons I am skeptical about the future of LDPC codes. However, we can also consider how an actively-corrected cost model would change with such codes. Some facts still hold:

[1]While optical links can connect ion traps[SNN+20], [BK22a] show that the number of optical links from each trap must be proportional to the number of qubits in each trap for asymptotically good rates.

1. Eve will still pay some cost per logical qubit per time step, since she must actively suppress errors.

2. The physical:logical qubit overhead is still greater than 1, so the number of logical qubits still gives a lower bound on the number of physical qubits.

3. There is no universal, transversal gate set (by the Gottesman-Knill theorem [Got98]), so *some* gate(s) must still be difficult to apply.

Given this, the cost of an actively corrected model still has a term proportional to DW, regardless of advances in error-correcting codes.

3.1.2 Passive Error Correction

It could be possible that we can engineer a Hamiltonian for each logical qubit such that, at finite temperature, the interaction between the system state and the bath is suppressed. This is known as *passive* correction.

Currently, it is provably impossible to construct spatially local passive error correction in 2 dimensions for broad families of codes [BLP+16]. Future research could discover passive error correction in the remaining codes, or could prove impossibility for them as well.

Three-dimensional passive error correction is similarly still open. However, truly three-dimensional memory might not be possible. The basic objection is the square-cube law: the amount of heat the system must dissipate (precisely to passively correct the errors) will be proportional to the total amount of memory, which is proportional to the volume of the computer. In contrast, the capacity of heat dissipation is proportional to the surface area of the computer. As the computer gets larger, the volume gets much larger than the surface area, implying the rate of heat dissipation per unit area must increase.

These are all asymptotic results, and there is some reason to be hopeful about actively corrected memory with limited active correction. For example, a trapped ion today can be kept coherent for several hours [WLQ+21]. If such qubits could form an actively-corrected memory, there would still be a cost proportional to

qubits×depth, but the constant(s) of proportionality could be low enough that it would effectively act like passively-corrected memory.

Some have presented the idea of embedding a higher-dimensional code into three-dimensional space. Fundamentally, this is impossible: one can always describe the embedding as a three-dimensional code, and so if passively-corrected memory is impossible in three-dimensions, the embedded code cannot be passively-corrected.

The criteria of passive correction [BLP$^+$16] is that, as we increase the system size, the error rates converge to zero. With an embedded higher-dimensional code, this fails because at some scale the system can no longer effectively implement the higher-dimensional code. Eventually, the connectivity between the qubits becomes completely non-local. However, until that point, error rates will scale as the higher-dimensional code predicts. This could give us a quantum memory with good error scaling, but with a fixed system size bound.

Implications of Passively Corrected Memory

The main difference between passive and active correction is that the identity gate has no cost with passive correction. This means the total cost in the memory peripheral framework is simply the gate cost.

There could be a more complicated landscape of gate costs. For example, certain gates might be much more expensive for a passively corrected memory than others, just as T gates are more expensive in the surface code. However, without knowing the structure of passively corrected memory (again, it might not even be possible), one cannot give such detail. Instead, I will simply count the total number of gates from the Clifford+T+measurement set, and this acts as an upper bound (up to constants) of the true cost.

One can readily see that for an algorithm of depth D and with W qubits, the gate cost G satisfies $\mathsf{G} \leq \mathsf{DW}$, since we can apply at most one gate to every qubit at every time step. Thus, the so-called [JS19] "G-cost" is never greater than the "DW-cost" of actively-corrected memory. For most algorithms, they are equivalent up to polynomial factors. Generally, it would be inefficient to leave qubits idle for

a long time. Further arguments in this vein, justifying a "time-area model", are in Chapter 4. In Chapter 7, I show that there is a problem where the best known G-cost is exponentially smaller than the best known DW-cost.

3.1.3 Other Cost Metrics

In the memory peripheral framework, the cost of an algorithm is the total number of interventions from the memory controller. Is that always the right cost?

We could apply a different cost model to the memory controller, such as total runtime, total memory consumption, or more exotic metrics (e.g. the "competitive ratio" [AB99]). The key benefit of the memory peripheral framework is that we are perfectly free to do this, as long as we compare algorithms consistently.

I focus on the total number of interventions because typically I am comparing to a highly parallelisable classical algorithm. For example, if we only care about total runtime, then there is an easy polynomial time algorithm for any problem in NP: list all possible inputs, check them all in parallel, and output the result. This relies on vast parallelism, but if our memory controller cannot execute instructions in parallel then it must make each memory operation sequentially as well. In that case, the runtime itself becomes proportional to the total number of interventions.

With vast parallelism (which is well justified for the surface code, for example), we could happen on a situation where the fastest algorithm requires more total operations than a different algorithm, but someone (like an adversary Eve) cares only about solving the problem as fast as possible, no matter the cost. In this situation, the total number of interventions is an inappropriate cost; *however*, this situation almost never occurs.

Firstly, a real adversary Eve almost certainly has hard limits on both her total hardware and her total runtime. Thus, the absolute minimum time is often unavailable, and instead she cares about minimising runtime within her hardware budget, or vice versa.

Second, essentially all classical algorithms that I consider in this thesis can be parallelised nearly perfectly. In this case, the algorithm acheiving the minimum

runtime under a hardware budget is precisely the same algorithm that minimises the total number of operations.

When these assumptions fail, I will make note of it. Signal latency (the time for information to travel between parts of the computer) is often the culprit, since it can vastly increase the runtime of specific gates. Section 7.4 gives an example.

3.2 Ballistic Computation

Here I imagine a more permissive model of computation, where Eve can set a memory peripheral to some complicated Hamiltonian and let it experience a lengthier evolution. How much computation can this do?

Feynman's original quantum computing design was like this [Fey85], but was driven forward by measurements. Kitaev's version evolved on its own via a clock [KSV02], so this shows that, in theory, an entire computation could be ballistic. Is this realistic?

Here I will give several limitations on this model, and what they might imply for future quantum computers.

In the memory peripheral framework, we construct a Hamiltonian by memory operations. I assume the set of available memory operations is finite, and we build complicated Hamiltonians by adding more terms. At a basic level, a memory controller needs to address which memory operation to perform, and it has a finite amount of memory to do this.

A second basic limitation is that we *must* have passively corrected memory. Our goal is to allow the system to evolve without intervention; if we need to intervene regularly to correct errors, there is little reason to prefer a ballistic computation instead of a sequence of active interventions.

Indeed, general Hamiltonian simulation results tell us that we can simulate a Hamiltonian H for time t and error ϵ with gate cost $O(\frac{t}{\sqrt{\epsilon}})$ [CMN$^+$18]. This can get arbitrarily bad in ϵ; however, for most cases where we would use H to perform some specific computation (e.g.: QRAM), we can find bespoke sequences of gates that perform much better than generic Hamiltonian simulation.

3.2.1 Hamiltonian Locality

My first assumption about how one can construct Hamiltonians is that each irreducible memory operation is only capable of making a bounded-arity change to the Hamiltonian. That is, given a Hamiltonian H on a memory peripheral $\mathsf{A}_1 \otimes \cdots \otimes \mathsf{A}_n$, each memory operation can only add a term H' with support on a bounded number of A_i.

This means we will be restricted to k-local Hamiltonians. This is a surprisingly restrictive class of objects.

Lemma 3.2.1. *Let $\mathcal{H}_1, \ldots, \mathcal{H}_n$ be Hilbert spaces, and let $H_i = \{H_{ij}\}_{j=1}^{\ell_i}$ be a collection of self-adjoint operators whose real span covers $\mathcal{B}(\mathcal{H}_i)$. Then the real span of tensor products of at most k operators, each from some H_i, is strictly contained in the span of at most $k+1$ operators.*

Proof. A simple basis-counting argument suffices. Without loss of generality we can assume H_i is a (real) basis that contains the identity. Then all tensor products of distinct elements from each H_i are linearly independent. However, we can see that the set of such tensor products such that at most k elements are non-identity is strictly contained in the set of at most $k+1$ non-identity elements. Since they are linearly independent, the result holds. □

This forbids us from constructing a Hamiltonian such as $e^{iX\otimes X\otimes X\otimes\cdots\otimes X}$. In fact, in the literature I have found no description of even a 3-local term in an experimentally realised Hamiltonian.

Already this limits the power of ballistic evolution. Perez-Delgado and Vinjanampathy [PDV21] take a model of computation where we are limited to Hamiltonians of a particular energy, defined as the difference between the maximum and minimum eigenvalue of the Hamiltonian. You can see that $X^{\otimes n}$ has the same energy as X in this model, so they argue that using such high-weight Hamiltonians allows "coherent parallelisation", leading to extraordinary results like sorting in $O(\frac{1}{n}\log n)$ time.

This paper has received substantial criticism for these claims, though I believe the conclusions are valid given the model they use. If one believes that the conclusions

cannot be true, then the assumptions must fail somehow. This section gives a particular reason why the model is unrealistic: it relies on high-weight Hamiltonian terms, which are not physically realisable if we can only add k-local terms for fixed k.

Higher-weight terms can be approximated in certain cases. Lloyd and Terhal's construction in Section A of [LT16] provides an example, but needs a large energy gap to do so.

3.2.2 Building up or Swapping in

Suppose Eve wants to enact a Hamiltonion $H = \sum_{i=1}^{n} H_i$, where H_i can be added with irreducible memory operations, so that she can evolve some state by $e^{itH/\hbar}$ for some time t. Constructing a memory peripheral which has this Hamiltonian requires n operations. Our first question is whether Eve can re-use this Hamiltonian.

That is, consider the difference between applying microwave pulses to a system of superconducting qubits, compared to inputting a photon into an optical layout. In the first case, we actively intervene to make each change to the Hamiltonian of the qubits (viz. adding a term for each microwave pulse), and then we actively intervene to restore the Hamiltonian to its original state (by ending the pulses). In the second case, we establish the Hamiltonian of the system by the arrangement of mirrors and waveguides, and this Hamiltonian remains fixed. We simply swap in the quantum state (a photon) by one active intervention, and we can do this multiple times.

Slightly more formally, we can ask whether it is always possible, given two memory peripherals A and B where B has a large Hamiltonian, to apply an irreducible memory operation acting on $\mathsf{A}\otimes\mathsf{B}$. If so, I will call that a "swapping in" Hamiltonian, and otherwise a "building up" Hamiltonian.

Building Up

It should be clear that building up Hamiltonians will be more costly in most cases, as if Eve wants to apply $e^{itH/\hbar}$ to different states at different times, she needs to create H again, at cost proportional to n. QRAM is a motivating example: if we need to access the same memory multiple times, we do not want to pay the same cost each time.

This might still be worthwhile if Eve wants to leave the system under H for a long time, i.e., if simulating $e^{itH/\hbar}$ with basic gates would be less efficient. I can make this more precise:

Theorem 3.2.1. *Let $\mathcal{G}$ be a set of mutually orthogonal circuits (i.e., for any $U_1, U_2 \in \mathcal{G}$, there is some input state such that $\langle\psi| U_2^\dagger U_1 |\psi\rangle = 0$). Let $\mathcal{H}$ be a set of Hamiltonians constructed by choosing n k-local terms from a set of N Hamiltonian terms of energy at most E on a system of W qudits, such that there exists some $t \in \mathbb{R}$ such that for all $U \in \mathcal{G}$, there is $H \in \mathcal{H}$ such that $U = e^{it'H/\hbar}$ for $t < t'$. Then*

$$ntE + n \log \mathsf{W} \in \Omega(\log |\mathcal{G}|). \tag{3.1}$$

To clarify this theorem, if there is a large family of circuits that Eve wants to enact with ballistic Hamiltonian evolution, she needs the product of the number of interventions, times the time that she allows the Hamiltonian to evolve, to grow logarithmically in the size of circuits. However, notice that if Eve has a budget of G gates, she can likely create a family of at least 2^G mutually orthogonal circuits, based on counting her choices of where to put each gate. Chapter 4 shows an example of such a circuit family.

To think about this result intuitively, it is somewhat obvious: if Eve wants to do better than a gate-based computation, she ought to be evolving the Hamiltonian for a non-trivial amount of time.

There are two limitations to this result I want to emphasise. The first, which is favourable to Eve, is that the result shows a bound for Hamiltonians trying to simulate circuits. I have not shown that circuits can simulate any Hamiltonian; instead we only have the generic Hamiltonian simulation result of $O(\frac{t}{\sqrt{\epsilon}})$ gates. Thus, there is still room for advantage.

Second, unfavourable to Eve, is that I have not proven that advantage *does* exist for $ntE \geq \Omega(\log |\mathcal{G}|)$. It might be that that Eve actually needs $n \geq \Omega(\log |\mathcal{G}|)$ to capture all circuits in $\mathcal{G}$, *regardless* of the total runtime or energy.

It is unclear which is more probable. A small Hamiltonian (say, $H = X$) can only produce a small number of distant unitaries, since there is only a small amount

of time before there is a coherent evolution of its eigenvalues. In contrast, a large Hamiltonian could potentially trace a long and winding path through the space of unitaries as one increases t in $e^{itH/\hbar}$.

My overall point is that if Eve wants *fast* Hamiltonian evolution, the "building-up" strategy is likely useless. When the time is limited, the complexity of evolution that the Hamiltonian can drive is related to the number of interventions she used to create it, but if she must create the Hamiltonian each time, she might as well just run the computation with gates.

Swapping In

To "swap in" to a Hamiltonian, Eve would spend n interventions to create a memory peripheral C with a complex Hamiltonian H, but she would prepare it with a "neutral" state, such as a low-energy eigenvector of H.

This means that C will be "idle" once Eve prepares it. To use it, she would take some other peripheral A, and swap the state ρ from A to C. Then she waits some time t for ρ to evolve under $e^{itH/\hbar}$, then swaps the state back into A.

In more concrete terms, C could be an optical bench with a series of mirrors and A could be a photonic mode containing a photon pulse. Eve could send the pulse into the optical setup C, allow it to propagate through the mirrors, lenses, etc., and then once it exits the set up, she could use a fiber optic cable to guide it back to A (or perhaps to some other system).

Somewhere in between lie most of the proposals for passive QRAM gates (see Chapter 4), which consist of some complicated device that is built and initialised *once*, and into which different access signals are sent.

This somewhat avoids the bounds of Theorem 3.2.1, since the cost to Eve to make Q queries to a given ballistic evolution of time t would be n interventions and time Qt.

However, there are distinct engineering challenges. To name a few:

- swapping in the state and allowing it to evolve must not perturb the Hamiltonian;
- the swapping must not change the state;

- the Hamiltonian itself must remain completely fixed between uses.

Despite these issues, I cannot prove that such a mechanism is impossible.

3.2.3 Errors

A system to build up a Hamiltonian will necessarily have some errors. I will model this as follows: every term added to the Hamiltonian will actually be a random variable, given by $H_i + \epsilon H_i^{err}$, where H_i is the intended term, and H_i^{err} is some normalised random error term.

With n terms, we will end up with a Hamiltonian $\tilde{H} = H + \epsilon H^{err}$, where $H^{err} = \sum_{i=1}^{n} H_i^{err}$. The question we need to address is how different $\tilde{U}(t) := e^{it\tilde{H}/\hbar}$ will be from $U(t) = e^{itH/\hbar}$.

A large body of literature exists to analyse errors at the gate level, but Hamiltonian errors are under-explored. Quantum control theory considers such errors (e.g., the survey in [DP10]) but focuses on continuous and active control, not a long ballistic computation.

With gates, errors grow additively. That is, if we want to apply $U_n U_{n-1} \dots U_1$, but each unitary has some error of size ϵ, the total error will be approximately $n\epsilon$. For a rigorous definition and derivation, see [NC00].

From Equation (A.11), if we have that $\|t\epsilon H^{err}\| \leq \hbar\delta e^{-t/\hbar}$, for some small δ, then

$$\begin{aligned} \|e^{it\tilde{H}/\hbar} - e^{itH/\hbar}\| \leq & e^{t/\hbar}\left(e^{t\|\epsilon H^{err}\|\hbar} - 1\right) && (3.2)\\ \leq & e^{t/\hbar}\left(e^{\delta e^{-t/\hbar}} - 1\right) && (3.3)\\ = & \delta + O\left(\delta^2 e^{-t/\hbar}\right) && (3.4) \end{aligned}$$

Generally, if $\|t\epsilon H^{err}\|$ is small, then the overall error is bounded. This means if Eve only wants a short time of Hamiltonian evolution, she is safe. For longer time evolutions, she has no guarantee of success. And unfortunately for us, we have no guarantees that she *cannot* succeed, i.e., we have no lower bounds on the error.

To explain why, recall that we can explicitly write out the evolution with error as:

$$e^{it\tilde{H}/\hbar} = e^{it(H+\epsilon H^{err})/\hbar} = \sum_{k=0}^{\infty} \frac{(it)^k}{\hbar^k k!}(H + \epsilon H^{err})^k \quad (3.5)$$

$$= \underbrace{\sum_{k=0}^{\infty} \frac{(it)^k}{\hbar^k k!} H^k}_{=e^{itH/\hbar}} + \underbrace{\sum_{k=1}^{\infty} \frac{(it)^k}{\hbar^k k!} \sum_{j=1}^{k} \epsilon^j \binom{k : H}{j : H^{err}}}_{\text{error}}. \tag{3.6}$$

We have no guarantee that the error term will be non-zero. Granted, any quantum device engineer would be extremely sceptical that the error term will have precisely the form necessary to cancel out this error evolution, but it is *theoretically* possible. In fact, [HLGJ21] shows that certain types of QRAM can have error of size $O(\epsilon \cdot \text{polylog}(n))$.

3.3 Gates

This section considers the cost of a single change to the computational state: a gate. The lower bounds I will give are almost certainly extremely loose, but I present them to illustrate how we need more permissive physical assumptions for galactic scale algorithms to be possible at all.

3.3.1 Non-Reversible Gates

Landauer argued that a non-reversible gate must produce $k_B T \ln(2)$ J of heat [Lan89]. The argument is that all physical processes are reversible, so if a gate maps any input state (say, $|0\rangle$ or $|1\rangle$) to a single output (say, $|0\rangle$), then it must change some environmental state to do so. If we are uncertain about the state of the computational system, we then become uncertain about the environmental system. If there is any energy difference between the two states, then there is an unknown energy flow to the environment, i.e., heat.

Often we think of classical computing as non-reversible and quantum computing as reversible, but classical computing can be reversible, and quantum computing can be irreversible. Not only are there measurement-based quantum computing techniques, but an actively-corrected quantum memory uses measurements to ultimately dissipate errors as heat. A stabiliser code using N physical qubits to encode one logical qubit will need $N - 1$ stabiliser measurements, so the number

of measurement cycles times the number of physical qubits gives a lower bound on the number of non-reversible gates.

To put these numbers in perspective, consider Figure 3.1. Landauer's bound is meaningless for Shor's algorithm, but shows that some attacks are infeasible with non-reversible gates.

3.3.2 Reversible Gates

We see that at sufficiently large scales, even Landauer's limit makes computation infeasible. We might thus consider purely reversible computation, noting that this requires passively-corrected memory, with all the caveats of such a technology. Still, I divide this into two sub-cases, corresponding to whether different computational states have different energy levels or the same, as in Figure 3.2.

To give examples of these cases, superconducting and trapped ion qubits have different energy levels for $|0\rangle$ and $|1\rangle$ states. In contrast, dual-rail photonic qubits have the same energy level, since the photon's energy is independent of its location.

Distinct Energy Levels

Here I give a novel lower bound on the energy to coherently drive a quantum system between states of different energy levels. A full proof is in the appendix; here I give an intuitive argument based on lasers.

Suppose we have a qubit where $|0\rangle$ has lower energy than $|1\rangle$, and the qubit's state is currently $\alpha|0\rangle + \beta|1\rangle$. If the difference in energy is equal to the energy of one photon, we might use a laser pulse to drive the system through an X gate, such as in Figure 3.3. Specifically, if the laser is in a state

$$\sum_{n=0}^{\infty} e^{\lambda/2} \sqrt{\frac{\lambda^n}{n!}} |n\rangle \tag{3.7}$$

in the Fock basis, then after driving an X transition, the joint state of qubit and laser is

$$\alpha \sum_{n=1}^{\infty} e^{\lambda/2} \sqrt{\frac{\lambda^n}{n!}} |n-1\rangle |1\rangle + \beta \sum_{n=1}^{\infty} e^{\lambda/2} \sqrt{\frac{\lambda^n}{n!}} |n+1\rangle |0\rangle \tag{3.8}$$

By gaining or losing energy, the qubit became entangled with the laser!

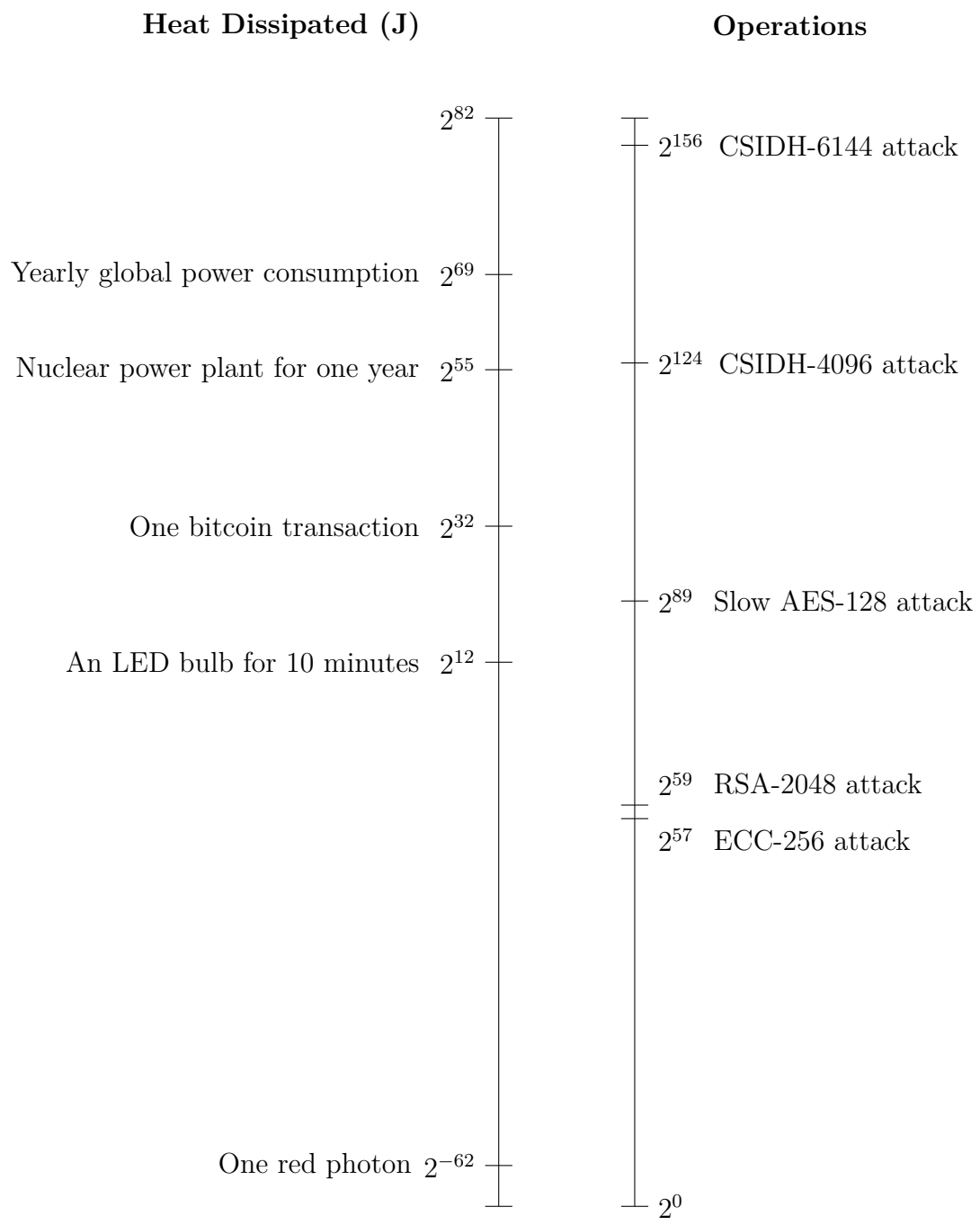
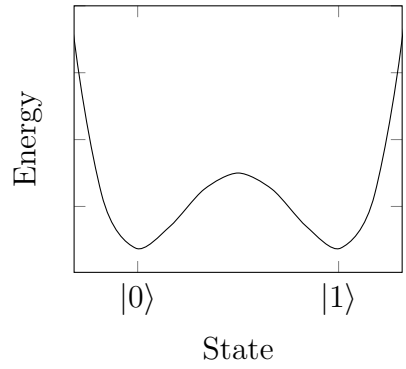
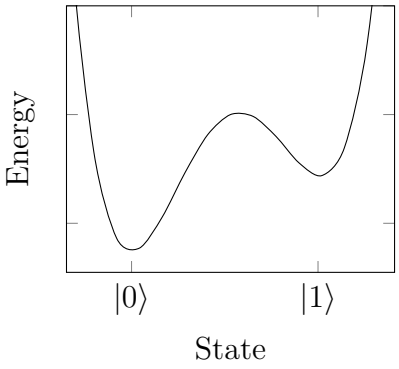


Figure 3.1: Energy consumption assuming one non-reversible gate per physical qubit per code cycle at 273 K. ECC is from Chapter 5, CSIDH is from Chapter 6 and [CSCDJRH21], and RSA is from [GE21]. AES attacks are from [JNRV20], with "slow" meaning there is no depth limit.

This could be a critical problem if we accidentally measured the precise energy level of the laser before and after this interaction (say, by measuring the change in energy of the lasing medium). Can we avoid this? There was some debate



(a) Sketch of a system where two states have the same energy levels.



(b) Sketch of a system where two states have the different energy levels.

Figure 3.2: Different ways to store information in some system, according to the energy of that system. Notice the energy barrier between the states.

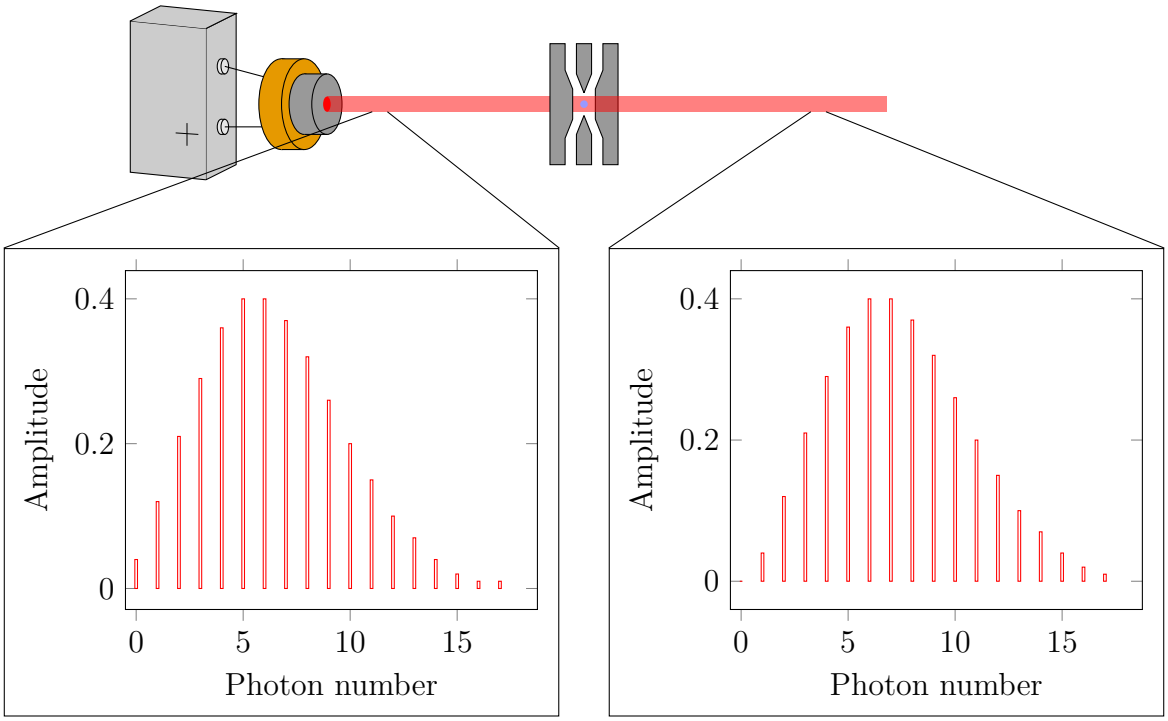


Figure 3.3: Sketch of a laser pulse interacting with a qubit. Can you tell whether the qubit was in the $|0\rangle$ or $|1\rangle$ state before the interaction? Should each photon number state be entangled with the state of the battery, which supplied the energy of the photons? When the laser pulse is absorbed by some material, will that measure the photon number?

([Mø97] and rebuttals) about whether coherent laser states even exist ([BRS06] summarises nicely, but dodges some of the question by assuming separate spheres of description for classical and quantum systems). It seems like the interaction of the laser with a bulk material, after it interacts with the qubit, will effectively measure

its electric field which is very nearly a basis of coherent states like Equation (3.7). For my purposes, it does not matter because

- we know this *does* work, since superconducting and trapped ion qubits use interactions just like this;
- the lower bound I will derive holds even for coherent states.

The idea of the lower bound is that for the interaction to be coherent, there must be as little difference as possible between the state entangled with $|0\rangle$ and $|1\rangle$, so that they are hardly entangled at all. Inspecting Equation (3.8), you can see that there is less entanglement as the laser state gets "wider", but this makes the average photon number – and thus average energy – much higher.

Generalising this, I suppose that an initial quantum state could be either $|\phi_0\rangle$ or $|\psi_0\rangle$ or any superposition thereof, and the action of some gate U maps them to $|\phi_1\rangle$ and $|\psi_1\rangle$ respectively, with energy gaps E_ϕ and E_ψ. For this to occur, it means that there is an environment state $|\chi_{env}\rangle$ and a gate U such that

$$U |\phi_0\rangle |\chi_{env}\rangle \mapsto |\phi_1\rangle |\phi_{env}\rangle \tag{3.9}$$

and

$$U |\psi_0\rangle |\chi_{env}\rangle \mapsto |\psi_1\rangle |\psi_{env}\rangle \tag{3.10}$$

Critically, conservation of energy implies that $|\phi_{env}\rangle$ must have lost as much energy as $|\phi_1\rangle$ gained, and similarly for $|\psi_{env}\rangle$.

Our goal with U is to emulate the map $\tilde{U}$, with action

$$\tilde{U} |\phi_0\rangle = |\phi_1\rangle \text{ and } \tilde{U} |\psi_0\rangle = |\psi_1\rangle \tag{3.11}$$

which is clearly impossible because of conservation of energy. Instead, we hope that the channel $\mathrm{Tr}_{env} \circ U$ is close to the channel given by $\tilde{U}$.

Theorem 3.3.1. *Suppose $|\phi_0\rangle$ and $|\phi_1\rangle$ have an energy gap of E_ϕ, and $|\psi_0\rangle$ and $|\psi_1\rangle$ have an energy gap of E_ψ. If the expected energy of the environmental state is at most E (over the ground state energy), then letting $\Delta = E_\phi - E_\psi$, there exist*

input states such that fidelity of the output of U compared to the output of $\tilde{U}$ is at most

$$1 - \frac{2^5}{3^3}\frac{\Delta^3}{E^3} \tag{3.12}$$

The proof is in Appendix A.2. This may not be a tight bound; I also show in Proposition A.2.1 that there is a state of energy E with fidelity at least $1 - O(\frac{\Delta^2}{E^2})$. More or less, the state in the energy basis is

$$|\chi_{env}\rangle = \sum_{n=0}^{2\frac{E+\Delta}{\Delta}} \sqrt{\frac{\Delta}{E+\Delta}} \sin\left(\frac{n\tau\Delta}{4(E+\Delta)}\right) |n\Delta\rangle\,. \tag{3.13}$$

Theorem 3.3.1 shows a few things. From an engineering point of view, it means Eve wants her environment state to contain states which are only integer multiples of the energy gap between the two states she might want to move between. Further, she should optimise for a distribution of energy levels which has a large overlap with a shift. A Poisson distribution (as in a laser state) is a good choice, or the state in Equation (3.13).

An interesting follow-up question is whether Landauer's limit will apply in this case as well. Even though the transition itself is reversible, we can never learn whether the transition occurred or not. Heat is defined as energy with classical uncertainty, but here we have quantum "uncertainty" about the energy levels.

The key question for energy bounds is whether this high-energy environment can be "re-used" or whether we must dissipate the full energy after each gate. That is, could Eve redirect the laser pulse back to other qubits after one interaction?

If so, then the environmental state will pick up some information about the quantum state with each interaction, so I can treat the case of N sequential interactions as one interaction with an energy gap of $N\Delta$. Then Theorem 3.3.1 implies that at energy E the overall fidelity is at most $1 - O(\frac{N^3\Delta^3}{E^3})$, so the energy must scale as $\Omega(N\Delta)$ to maintain a constant fidelity.

If Eve uses a distinct environmental state each time, then the infidelity of each of the N interactions must be $\frac{1}{N}$ times the overall target fidelity; if $\frac{\Delta^3}{E^3} \in O(\frac{1}{N})$ then this means $E \in \Omega(N^{1/3}\Delta)$. However, since the environmental state after each

interaction cannot be reused, the total energy use is $NE \in \Omega(N^{4/3}\Delta)$ – worse than the bound from reusing the environmental state.

Figure 3.4 contextualises this bound, assuming Eve re-uses the environment state. I excluded Shor's algorithm, since I doubt we will have passively-corrected memory for that case. Like Landauer's limit, this gives roughly a constant energy bound per operation. Here, the exact bound will depend on the system's energy gap. A microwave photon's worth of energy gives a bound which is a factor of 2^{11} smaller than Landauer's; a red light photon's worth of energy is a factor of 2 larger than Landauer's. Again, most galactic-scale algorithms are infeasible thanks to this bound, so even reversible computation is not enough: Eve cannot allow computational states to have distinct energy levels.

Equal Energy Levels

If two computational states have equal energy levels, then if there is some interaction with the environment, Arrhenius' law states that qubit's coherence time is proportional to $e^{-E/k_B T}$ where T is the temperature and E is the *energy barrier* between the states. Putting this into context is more difficult, since the constants of proportionality will depend on the technology. Moreoever, it is the weakest bound yet, since the required energy will scale only logarithmically with the depth of the algorithm; even at cryptographic scales, that will be meaningless.

Instead, let us focus on the Margolus-Levitin bound [ML98], which states that the minimum time for a Hamiltonian of "energy" E (the difference between the average energy and the ground state energy) to evolve one state to an orthogonal state is $h/4E$. This means if Eve has a fixed time limit for an algorithm, her required energy per operation scales with the runtime of the algorithm.

Eve could escape this bound by moving from a gate-based computation to a ballistic computation. Jordan [Jor17] modifies the Feynman-Kitaev Hamiltonian to emulate an arbitrary number of gates with an arbitrarily low energy. I rely on the arguments from Section 3.2 to focus on gate-based computation; Jordan's construction requires $\Omega(n)$ interventions to emulate a circuit of n gates, as Theorem 3.2.1 predicts.

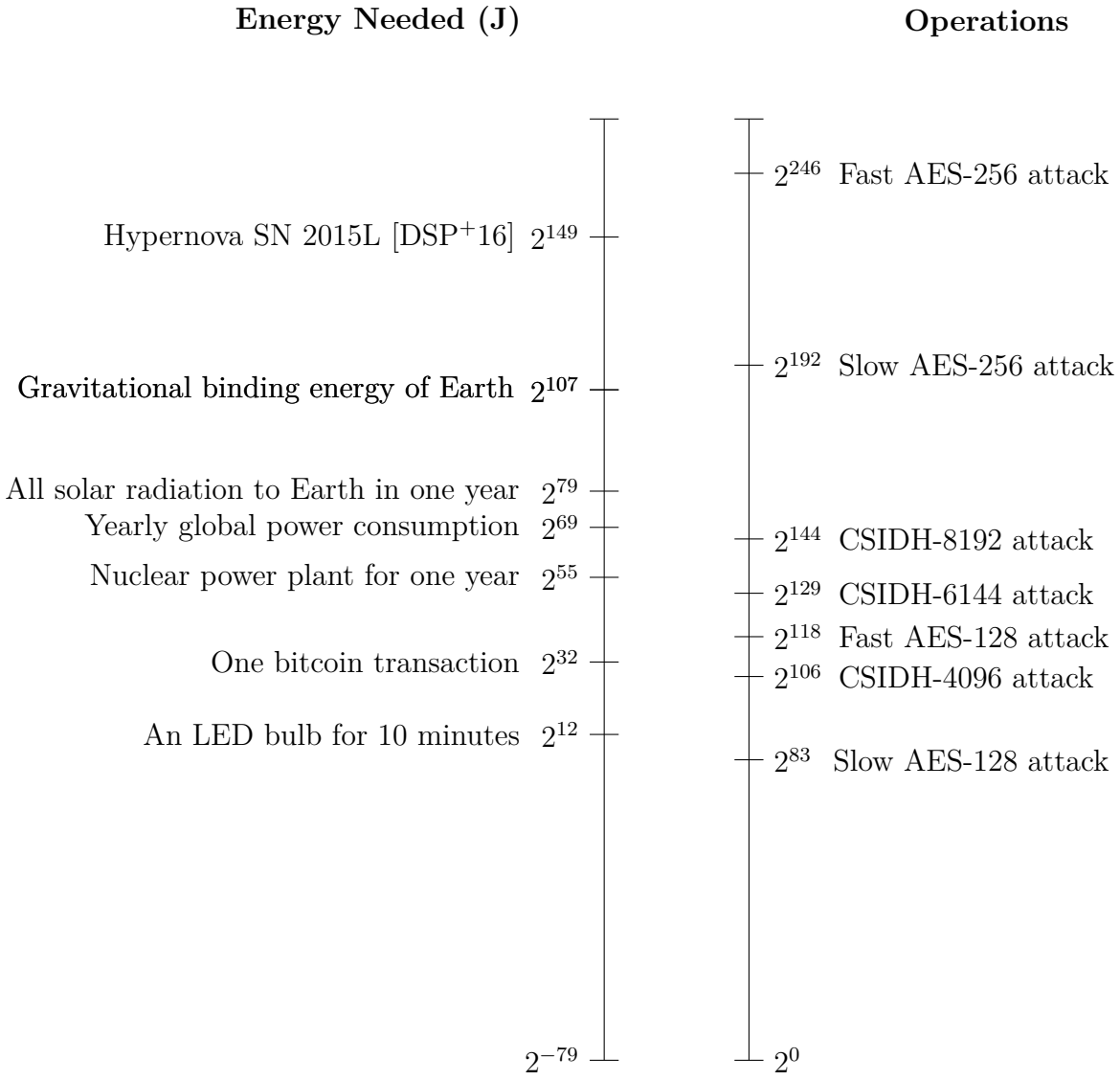


Figure 3.4: Minimum energy to coherently drive the computations for these attacks, from Theorem 3.3.1, assuming an energy gap Δ of $1.96 \cdot 2^{-80}$ J (one microwave photon) and a target fidelity of 0.9. CSIDH costs are the "G-cost" from the script in [CSCDJRH21]; AES costs are from [JNRV20], where "slow" is a depth limit of 2^{96} and "fast" is a depth limit of 2^{40}.

Figure 3.5 shows the energy bounds this gives for "real-world" attacks, showing almost no limitation. Again there is a question of whether the energy of the Hamiltonian is consumed in some sense by each operation; I assumed it is re-used. Further, I used the gate count for the number of operations, assuming that applying m gates in parallel would take m times more energy, so Eve is indifferent between using more energy to make the gates faster or more energy to apply them in parallel.

The difference between Figure 3.5 and Figure 3.4 is whether distinct computational states are at distinct energy levels. To actually reach the Margolus-Levitin bound, Eve would need (among other things) a method to "borrow" energy to drive the computational state over the energy barrier to perform gates without losing *any* energy to the environment and without revealing any information about the original state during this process. This is not *provably* impossible, but shows the extreme physical assumptions that we need to avoid the previous lower bounds.

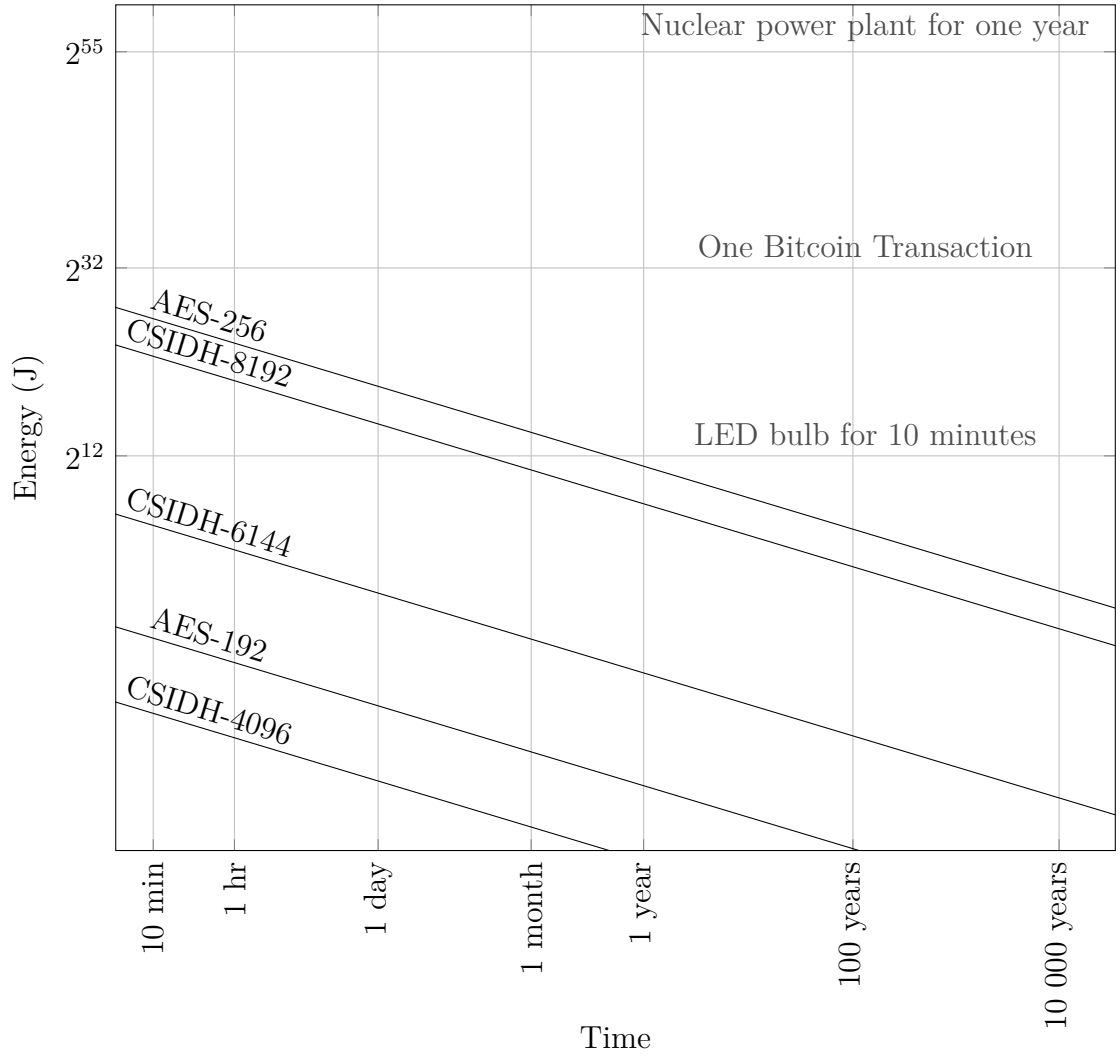


Figure 3.5: Energy required based on the Margolus-Levitin bound to complete attacks in the time specified. The costs are based on number of gates with no specified depth limit, since I assume that the runtime is decreased by increasing the energy of the computational Hamiltonian. Compared to Figure 3.4, here I assume no energy gap between computational states.

3.4 Conclusions

Right now, surface codes and similar topological codes are the most promising path to large-scale quantum computing simply because they have the fewest unsolved problems necessary to build them. Hence, I will use surface codes as the architecture to estimate costs in Chapter 5 on Shor's algorithm, the most near-term cryptographic attack from quantum computers.

For Chapter 6, I will also make estimates based on surface codes, but as Figure 3.1 shows, running the attack with a surface code is infeasible. That's good! We want to choose parameters where the attack is infeasible. However, it raises a question of whether the architecture will be improved. That is, will Eve find it easier to construct a planet-sized machine to attack CSIDH-6144, or to solve the problem of passively-corrected quantum memory?

As we move to "galactic scale" algorithms, Eve will need to solve these theoretical problems to escape more restrictive lower bounds like Landauer's limit. We can see that bounds like Margolus-Levitin are almost meaningless for real-world cryptographic parameters, so there is no *fundamental* reason that an adversary could not someday break AES-256. Instead there is merely a common sense notion that 2^{246} operations is simply unreasonable.

Section 3.2 highlights some issues with ballistic computation, which are not strong enough to rule out ballistic computation, but do exemplify some overlooked issues. My particular critique is that ballistic QRAM is unlikely, and I expand on that argument in Chapter 4.

4

Quantum Random Access Memory

Contributions. This chapter is a condensed version of my work in [JR23], with the included sections taken verbatim. My co-author Arthur Ratthew wrote the analysis of quantum linear algebra in that paper, but that does not appear in this chapter.

4.1 Introduction

To frame this chapter, I first summarise existing arguments for quantum random access memory (QRAM) and try to explain, conceptually, how to think about the nature of a QRAM device.

Recall that the memory peripheral framework highlights a view of a quantum computer as a peripheral of a classical device. The crux of QRAM is that if a query state is a superposition over all addresses, then for the device or circuit to respond appropriately, it must perform a memory access over *all* addresses simultaneously. While "it performs all possible computations at once" is a classic misunderstanding of quantum computing, it applies more directly to QRAM: imagining the memory laid out in space, a QRAM access must transfer some information to each of the bits in memory if it hopes to correctly perform a superposition of memory accesses. This sets up an immediate contrast to classical memory, which can instead adaptively direct a signal through different parts of a memory circuit.

In the memory peripheral framework, we need to apply gates for every possible pattern of memory access, and since each gate requires some action from the controller – be it time, computation, or energy – the relevant cost is proportional to the total size of the memory. The problems with high time or energy costs are obvious, but the framework highlights that a computational cost is really an *opportunity* cost. If we have a dedicated co-processor for every $O(1)$ qubits, could they do something besides run the quantum computer?

This captures arguments against particular uses of QRAM. If we are allowed to reprogram the control circuitry, then:

- two-to-one collision finding is just as fast for the quantum computer or its controller [Ber09]
- claw finding is *faster* for the controller than the quantum computer [JS19]
- the advantage of quantum matrix inversion drops from exponential to (at best) polynomial [CHI+18]

In short, a massive quantum computer implies a massive classical co-processor to control it, and that co-processor can quickly solve many problems on its own.

Proponents of QRAM can then argue that this is the wrong way to conceptualise QRAM. In classical computers, RAM is a separate device, manufactured differently than the CPU, designed specifically to perform memory accesses. From the CPU's perspective, it sends a signal into the RAM and waits for the RAM to process that signal on its own. An analogous quantum device would not incur the costs of external control, but would need incredible engineering to function correctly without that control.

The main technical contributions of this chapter are a careful analysis of the requirements and assumptions of this other perspective. In short, I argue that:

- If QRAM requires no intervention, it is "ballistic", and must be described by a single Hamiltonian;
- No intervention means no active error correction, and hence each physical component must intrinsically have near-zero noise levels;
- Unless the device is carefully designed to limit error propagation, physical component error rates must be impossibly low;
- To be compatible with the rest of a fault tolerant computation, we need some way to apply QRAM to qubits in an error corrected code;
- In particular, a QRAM gate cannot be teleported, so we must resort to other techniques which risk decohering the state.

In many ways the classical analogy fails, and the engineering requirements of a *cheap* QRAM device are beyond daunting, and access to N bits of QRAM will likely always cost $\Omega(N)$.

I stress that QRAM can still be a useful tool. Chapters 5, 6, and 7 give examples of cases where QRAM provides an advantage to a quantum attacker, even accounting for the arguments in this chapter.

4.2 Definitions and Notation

I will follow the notation of Kuperberg [Kup13], who proposes a taxonomy of four types of memory, based on the product of two criteria:

- Does the device store classical or quantum data? (CM for classical memory or QM for quantum memory)
- Can the device access this data in superposition or not? (CRA for classical access or QRA for quantum access)

This results in four types of memory:

CRACM (Classical access classical memory): This is classical memory that can only be accessed classically, i.e., the memory we are all familiar with today. Kuperberg does not actually distinguish the *speed* of memory access. Comparing quantum and classical, we do not really care. In this sense DRAM, flash memory, hard disk drives, and even magnetic tapes can all be considered CRACM. I discuss more properties of CRACM in Section 4.6.

CRAQM (Classical access quantum memory): Here we have quantum memory, but which can only be accessed classically. The notion of "access" is somewhat fuzzy: obviously a classical controller cannot read quantum memory without destructively measuring it. My notion of access will be more akin to "addressable", meaning the classical computer can find the required bits of quantum memory. This means essentially all qubits are CRAQM, since any quantum circuit will require a classical controller to find the necessary qubits to apply the gates for that circuit.

Classically addressing qubits is a challenging and extremely important problem (e.g., [JPC+19]), but it is fundamentally different and mostly unrelated to what this chapter addresses.

QRACM (Quantum access classical memory): I formally define "QRACM" as follows:

Definition 4.2.1. *Quantum random-access classical memory (QRACM) is a collection of unitarities $U_{QRACM}(T)$, where $T \in \{0,1\}^N$ is a table of data, such that for all states $|i\rangle$ in the computational basis, $0 \leq i \leq N-1$,*

$$U_{QRACM}(T)\,|i\rangle\,|0\rangle \mapsto |i\rangle\,|T_i\rangle\,. \tag{4.1}$$

Here T is the classical memory and $|i\rangle$ is an address input. I call the register with $|i\rangle$ the *address register* and the bit which transforms from $|0\rangle$ to $|T_i\rangle$ the *output register*. Since $U_{QRACM}(T)$ is unitary, superposition access follows by linearity. This definition specifies a unitary action, but I do not require the device to be unitary: a quantum channel whose action is identical to (or approximates) conjugation by $U_{QRACM}(T)$ will still be considered QRACM (for example, the method in [BGM+19] uses measurement-based uncomputation).

In this definition, the table T parameterises the unitaries, since QRACM must take some classical data as input. This means that there is no single QRACM "gate": there is a *family* of gates, parameterised by T. In fact, since T has N entries of $\{0,1\}$, there are 2^N possible memories.

I take N as the number of elements in the table that the QRACM must access, which I will sometimes call *words* (for cases where they are larger than single bits). This means the address register has $n := \lceil \lg N \rceil$ qubits. Other papers choose a notation focusing on n, the number of qubits in the address, so that the number of words in the table is 2^n (and the number of possible tables is 2^{2^n}). Using n and 2^n emphasises that, for certain algorithms, the size of the table will be exponential compared to other aspects of the input and running time. I will mainly use N and $\lceil \lg N \rceil$ since the classical table must be input to the algorithm, and therefore anything $O(N)$ is linear in the input size, but I use n throughout as shorthand for $\lceil \lg N \rceil$.

Of course most applications require the elements of T to be more than single bits, but concatenating single-qubit QRACM gives multi-bit QRACM.

QRAQM (Quantum access quantum memory): I formally define this as:

Definition 4.2.2. *Quantum random-access quantum memory (QRAQM) is a collection of unitaries $U_{QRAQM}(N)$, where $N \in \mathbb{N}$, such that for all states $|i\rangle$ with $0 \leq i \leq N-1$ and all N-qubit states $|T_0, T_1, \ldots, T_{N-1}\rangle$ (referred to as the* data register*, where $T_0, \ldots, T_{N-1}$ is a bitstring),*

$$U_{QRAQM} |i\rangle |0\rangle |T_0, \ldots, T_{N-1}\rangle = |i\rangle |T_i\rangle |T_0, \ldots, T_{N-1}\rangle \tag{4.2}$$

The difference now is that the memory itself is a quantum state, meaning we could have a superposition of different tables. Here the gate is implicitly parameterised by the table size, and needs to suffice for all possible states of that size.

This definition only shows a unitary that can *read* the quantum memory, but one can show that unitarity forces

$$U_{QRAQM} |i\rangle |1\rangle |T_0, \ldots, T_{N-1}\rangle = |i\rangle |1 \oplus T_i\rangle |T_0, \ldots, T_{N-1}\rangle \tag{4.3}$$

and from this, one can conjugate $U_{QRAQM}(N)$ with Hadamard gates on all qubits in the output and data registers to construct a write operation.

The literature still has some ambiguity about this definition. For example, we could define a SWAP-QRAQM such that

$$\begin{aligned} &U_{S-QRAQM} |i\rangle |\psi\rangle |\phi_0\rangle \ldots |\phi_{N-1}\rangle \\ &\qquad = |i\rangle |\phi_i\rangle |\phi_0\rangle \ldots |\phi_{i-1}\rangle |\psi\rangle |\phi_{i+1}\rangle |\phi_{N-1}\rangle . \end{aligned} \tag{4.4}$$

SWAP-QRAQM is equivalent to the previous read-and-write QRAQM, if some auxiliary qubits and local gates are used.

4.2.1 Routing and Readout

Imprecisely, QRACM and QRAQM have two tasks: the first is routing and the second is readout.

By analogy, consider retrieving data classically from a network database. A request for the data will contain some address, and this will need to be appropriately routed to a physical location, such as a specific sector on a specific drive in a specific

building. Once the signal reaches that location, the data must be copied out before it can be returned. Following the example, this would involve reading the magnetisation of the hard drive.

Difficulties can arise from either routing or readout, but I stress that both are necessary. [Kup13] states "Our own suggestion for a QRACM architecture is to express classical data with a 2-dimensional grid of pixels that rotate the polarization of light. (A liquid crystal display has a layer that does exactly that.) When a photon passes through such a grid, its polarization qubit reads the pixel grid in superposition." This seems to satisfy the requirements of the readout, but does not address routing: how do we generate the photon in such a way that its direction is precisely entangled with the address register, so that for state $|i\rangle$, the photon travels to the ith pixel in this grid?

Generally, the routing problem seems more difficult, and in some cases solving the routing problem would easily give us good QRAM with a mechanism such as the above. Thus, I mainly focus on routing in this chapter.

For that reason, I also will use the term "QRAM" to refer to a routing technology that would work for either QRACM or QRAQM with only modest changes to the readout mechanism.

4.2.2 Alternate Notation

"QRAM" is used inconsistently throughout the literature. It is sometimes used for QRAQM or QRACM. It is also sometimes meant to specifically refer to an operation that implements QRACM, but via a single gate.

"QROM" has been used to refer to QRACM implemented as a circuit (e.g., from Clifford+T gates) (e.g., [BGB+18]). QRACM must be read-only, so using QRAM to refer to QRAQM and QROM to refer to QRACM would be a consistent alternative notation; however, since the literature uses QRAM to refer to both kinds, I opt against this.

In classical computing, devices such as hard drives or tapes *can* implement the functionality of random-access memory, though the access times are longer and

more variable than devices such as DRAM or flash memory. Analogously, different proposals achieve the same required function of QRACM – to access memory in superposition – but with radically different costs and access times[1].

To be more precise than this, I will use *circuit*-QRAM to refer to proposals to construct superposition access using a standard gate set, especially Clifford+T.

I will use *gate*-QRAM to refer to proposals for QRAM consisting of a specialised device. I further categorise into *active* gate-QRAM, which requires $\Omega(N)$ active interventions from the classical controller for each access, and *passive* gate QRAM, which requires $o(N)$ active interventions.

If we imagine memory access by routing an address through a binary tree, the tree must have $N-1$ nodes (if N is a power of 2), and each node must do some computation on the incoming signal. The difference between active and passive gate-QRAM is whether that computation requires some external process to make it happen, or whether it proceeds on its own with no external intervention.

4.2.3 Context

There may be a disconnect in the discourse around the feasibility of QRAM. Proponents and hardware designers may only be considering small-scale regimes of up to 2^{30} bits of QRAM, in which case the arguments presented here and elsewhere *may* be surmountable: classical computational components in 2004 had error rates of 2^{-30} to 2^{-40} [Tez04], so perhaps quantum components could reach the same level.

However, at the scale of 2^{30} bits, my conclusions are also less relevant; we know (Section 4.3) how to perform N-bit QRAM with a circuit of $O(N)$ gates, so any algorithm using 2^n bits of QRAM will face only a 2^n overhead if physical QRAM assumptions are wrong. For something like 2^{20} bits, this does not make a large difference, especially considering the extreme uncertainties in future quantum computing overheads and architectures.

In contrast, at the scale of 2^{50} bits, cheap passive QRAM causes a drastic change in algorithm costs, which has incentivised the research into algorithms that use

[1] Arguably "quantum superposition access memory" is a more accurate term, but the acronym feels too egotistical.

QRAM. Yet, even minute physical costs become relevent if they scale with memory size. For perspective, if we lose a single visible photon's worth of energy for every bit in memory for each access, accessing 2^{50} bits costs about 1 kJ.

As we move into cryptographic scales, QRAM becomes clearly absurd. A single photon's worth of energy per bit means 1 *giga*-Joule per access if we have 2^{80} bits. Even more extreme: in a Fabry-Perot interforometer, the momentum change from a single microwave photon on a 10 g mirror would shift the mirror's position by a portion of 2^{-98} of the cavity's length. If the memory needs 2^{-128} precision (as one proposed QRAM technology [GLM08a]), then this shift is a billion times larger than the precision needed.

Most of my conclusions in this chapter state that an N-bit QRAM access will need energy/computation/some other cost to scale as $\Omega(N)$, and/or each physical component will need error rates of $O(1/N)$. These are asymptotic claims, and it is beyond what we can say today to claim that at at small-scale memory sizes, these terms will dominate other more relevant costs. However, at cryptographic scales and beyond, we must consider these factors.

4.3 Circuit QRAM

One can build QRACM circuits out of simpler Clifford+T+measurement gates. The cost is linear in the number of bits of memory. Specifically, Appendix C summarises work in this area proving the following:

Theorem 4.3.1. *For any depth D and width W such that $D \in \Omega(\log N)$, $W \in \Omega(\log N)$, and $DW = \Theta(N \log N)$, there exists a family of QRACM circuits with depth D and width W.*

This may feel unreasonably high; however, in a bounded fan-in circuit model, this is essentially optimal.

As this thesis will show, there are many applications where QRAM is a useful tool, even with this high cost.

4.3.1 Lower Bounds

The key point of these lower bounds is that with N bits of memory in a QRACM, there are 2^N possible QRACM gates, depending on the values in those bits. However, the number of different quantum circuits is roughly exponential in the product of the number of qubits and the circuit depth, which one can see because we have a finite set of choices of gate for each qubit at each time step. Thus, we need $N \approx$depth$\times$width. This is the same logic as a classic result of Shannon [Sha49], since QRAM can implement any n-bit function.

More precisely, I prove the following Lemma in the appendix:

Lemma 4.3.1. *The set of all distinct quantum circuits on W qubits of depth D using G gates from a fixed set of g possible gates of fanin at most k, has size at most $2^{kG\left(\lg(\frac{DWg}{G\sqrt{k}})\right)}$.*

From this the lower bound follows:

Theorem 4.3.2. *For any N, there exists a table of size N such that for any quantum circuit defined as above that implements $U_{QRACM}(T)$, the number of gates G will be in $\Omega(N)$.*

Proof. There are 2^N different tables of size N, and each will require a distinct quantum operation. This means we require

$$N \leq kG\left(\lg(\tfrac{DWg}{G\sqrt{k}})\right) \tag{4.5}$$

and the result follows readily. □

Theorem 4.3.2 states that there *exists* a table with these high resource requirements. There are many "tables" with much lower requirements. For example, if precisely one entry of the table is 1 and the rest are 0, then a multi-controlled `NOT` will implement the required QRACM gate. However, inspecting the proof, we see that only an exponentially small fraction of tables can be implemented with fewer gates than this.

Surface codes break with the model of "bounded fanin" somewhat, since a logical CNOT gate can have arbitrarily many targets. One can simulate this with traditional CNOT gates using at most twice as many qubits and constant depth, so the asymptotic lower bound still holds.

Notably, the lower bound applies to *circuits*, meaning measurements are forbidden. My simple counting arguments fall apart quickly: one could simply measure the entire input, look it up in the table classically, and output the result. It is easy to see that there are 2^N distinct "operations" of this type, but they fail to implement QRACM. To be useful, the measurements need to be mostly independent of the input state, but I leave it to future work to incorporate this into the counting arguments. If the feedback from measurements is simple enough to replace it with quantum control without drastically increasing the resource requirements, the bounds will hold. [YZ22] provide a similar lower bound through different techniques which are also restricted to measurement-free circuits.

4.4 Gate QRAM

The gate lower bounds for circuit QRAM are unpalatable for certain applications, but there is no reason that all parts of a future quantum computer are built as circuits in a small gate set. Like GPUs or RAM in classical computers today, it is easy to imagine specialised devices for specific tasks. In this section, I imagine a device that performs a QRACM or QRAQM access, and our goal is something much more efficient than implementing this access as a circuit.

This section asks whether such a specialised device can be built and run efficiently. Not all tasks can have specialised hardware; it is unreasonable to expect a submodule which solves 3-SAT in quantum polynomial time. My question is whether QRAM access is a simpler problem.

To give a quick sense that QRACM is generally difficult, consider that if the table is $T = [0, 0, 0, 1]$, then QRACM access to this table is equivalent to a Toffoli gate. Hence, Clifford+QRAM is a universal gate set.

In this section I lay out several issues that any proposed QRAM device must address to be competitive with circuit-based QRAM.

4.4.1 Active vs. Passive QRAM

Recall that in the memory peripheral model, we cost quantum operations by the number of interventions by the classical controller. Here I extend that slightly to include the energy cost of the external controller. For example, we would count the cost to run a laser.

An *active* QRAM gate is one with $\Omega(N)$ cost in this sense for each QRAM access. Active gate QRAM will still fail asymptotically in the same way as circuit QRAM. Recalling the scale of memory in Section 4.2.3, even if the constant in this asymptotic term is tiny, it quickly add up when the memory sizes are 2^{50} bits or larger.

There may be a constant-factor advantage to active gate QRAM, but Appendix C shows that the constants for circuit QRAM are already low. Moreover, when comparing a quantum algorithm using QRAM to a classical algorithm, the quantum algorithm must overcome a great deal of overhead to have advantage (e.g., Google's quantum team estimates that a quadratic advantage is insufficient in the near term [BMN+21]; error correction nearly eliminates the advantage of quantum lattice sieving [AGPS20]).

For these reasons, I discount any active gate QRAM, though I cannot rule out a constant-factor advantage from such a device.

Notice that passive QRAM (with $o(N)$ cost for each access) needs both the readout and routing stages to be passive. Even if our readout is quite low-energy – say, if we can read directly from a non-volatile classical memory – if the routing stage has cost $\Omega(N)$, the entire QRAM access would have cost $\Omega(N)$.

4.4.2 Path Independence

Accessing memory in superposition means that we must not destroy any of the states in superposition, which means that no external system, including the controller

enacting the memory, can learn *anything* about the memory access itself. A major obstacle here is the path in which the memory access travels.

In contrast, consider one aspect of classical memory access that QRAM cannot emulate. If there is a tree of routing nodes as in the bucket-brigade QRAM, a classical computer can keep all of the nodes in a low- or zero-power "idle" mode, where a node activates only when it receives a signal from its parent node. Such a device would consume energy only in proportion to $O(\log N)$, not $O(N)$. QRAM cannot do this because if different paths consume different energy or different interventions, this effectively measures which path the address took. Thus, if a QRAM gate requires any active intervention (such as a controlled swap like the circuit QRAM), then the controller *must* apply that intervention to *all* nodes, immediately shooting the cost to $\Omega(N)$.

In their description of bucket-brigade QRAM, [GLM08b] describe an "active" path through the tree for each address. If a node requires energy to become active, or an active node needs some intervention to perform its role in the device, then *all* nodes must receive that energy or intervention.

4.4.3 Hamiltonian Construction

A fully passive gate implies that the Hamiltonian describing the entire QRAM device must be constant throughout the operation of the gate, since any change to the Hamiltonian requires external intervention. This means there must be some Hamiltonian $H_{QRACM}(T)$ and time t such that

$$U_{QRACM}(T) = e^{iH_{QRACM}(T)t/\hbar}. \tag{4.6}$$

I also allow a passive gate with $o(N)$ interventions, so I could have

$$U_{QRACM}(T) = \prod_{j=1}^{k} e^{iH_{QRACM}(T,j)t_j/\hbar} \tag{4.7}$$

for $k \in o(N)$, but the arguments I present here will be much the same.

The arguments from Chapter 3 apply here directly. For example, if changing the system's Hamiltonian needs $\Omega(N)$ interventions (e.g., if one needs photon pulses on

all nodes of a routing tree), then the QRAM is active. We could instead try to design a system where we can evolve to the desired Hamiltonian with fewer interventions, but recalling Theorem 3.2.1, the set of QRACM gates is mutually orthogonal, giving:

Theorem 4.4.1. *Let $\mathcal{U}_{ballistic}$ be the set of all unitaries on W qubits formed by Hamiltonians with up to n terms, each of which is at most k-local and with energy between $-E$ and E, and then evolving the full Hamiltonian for time up to t. If all QRACM gates for tables of size N are in $\mathcal{U}_{ballistic}$, then*

$$ntE + n \log W \geq \Omega(N). \tag{4.8}$$

That is, there could be a trade-off of the number of interventions against time and energy, but if any two are fixed, the other must scale with the QRACM size.

Alternatively, one could construct a Hamiltonian such that there is an "idle" eigenstate in the QRAM, and we only need to perturb the system to start it. For example, one of the proposals for bucket-brigade QRAM imagines excitable atoms at each node, excited by input photons. Adding a photon would start the system on the desired evolution.

I addressed such ballistic computation in Chapter 3. Given the arguments on error rates, I expect that without careful thought, errors must be inversely proportional to the number of interventions in the Hamiltonian.

Overall, I highlight the notion of a single Hamiltonian for the QRAM to exemplify the extreme engineering challenges of such a device, which are far above those of even a large-scale fault-tolerant quantum computer.

4.5 Errors

Defining the exact error rate of QRAM is tricky, since there are several definitions of the error rate of a channel (e.g., average fidelity vs. minimum fidelity). For applications like those in Chapter 5, average fidelity is likely more relevant because the algorithm is robust against small systematic errors, whereas for QAA using QRAM, we need something like minimum fidelity.

Generally, I will use these metrics to measure the quality of the QRAM gates and their components, though somewhat imprecisely. "Error rate" will refer to the infidelity, $1 - F(\cdot, \cdot)$. For the overall QRAM, I want to compare the output of our real QRAM device (modeled as a quantum channel) to the ideal behaviour: a unitary channel defined by $U_{QRACM}(T)$ or $U_{QRAQM}(T)$.

Error rates. It is well-known that a quantum algorithm with G gates can succeed if each gate has an error rate of $O(1/G)$, and this is often a necessary condition as well (with even more stringent conditions if idle qubits are also error-prone). The error rates in curent quantum devices are too high for circuits with more than a few gates, and experts seem to generally expect the future to centre around fault-tolerant error-correcting codes, not spectacular jumps in physical error rates [MP21].

For this reason I dismiss any QRAM technology requiring error rates per *component* to scale inversely with the number of bits of memory, regarding it as just as unlikely as quantum computing without error correction.

Where this might not matter is applications with only a small number of QRAM accesses. As an example, if an algorithm needs 2^{20} Hadamard gates but only 2^8 QRAM accesses, then physical error rates of 2^{-10} are too high for Hadamard gates but just fine for the QRAM. In this case we would not need error correction, and in fact we could decode a logical state to a physical state, apply the QRAM, then re-encode to the logical state. This is a risky process for a quantum state, but if we make few total accesses to the QRAM then we also do few total decodings. This might be helfpul, but I stress again that this limits the total size of the QRAM, giving it at best a constant-factor advantage.

For large-scale QRAM, which must be error-corrected, I consider two approaches: error correction within the QRAM, or QRAM within the error correction.

4.5.1 Error-corrected components

Suppose that we attempt to suppress errors in each component of a QRAM device using some form of error correction, such as [CDEH$^+$21] propose for their method:

"scaling up the qRAM necessitates further exploration in converting each tree node to a logical qubit and adapting quantum error correction."

If this error correction requires active intervention, like a surface code, we have essentially recreated circuit QRAM. Even if the QRAM device uses specialised hardware, active error correction implies intervention from the controller and energy dissipation for each component, pushing us back into the regime of active gate QRAM with a $\Omega(N)$ cost of access.

Passive error correction, where the system naturally removes errors by evolving to lower energy states, is dubious (recall Section 3.1.2). Methods for passive error correction would be a massive breakthrough, but it would also not be enough for passive QRAM. Such a device would still need a single Hamiltonian which directs the entire QRAM access to achieve the imagined costs of passive QRAM.

Because of these difficulties, I assume error-correction within the QRAM that does not lead to $\Omega(N)$ access costs is infeasible. Thus, to avoid strict error requirements, we would want a QRAM gate such that the total error of each access relative to the error rate of each component is only $O(\text{polylog}(N))$. For example, if $N = 2^{50}$ and the error scales as $\lg^2 N$, we "only" need error rates of 2^{-21} in each component to achieve a total error rate of 2^{-10}.

4.5.2 Error Correction

Instead of correcting the errors in the components of a QRAM device, the device could be noisy and we apply error correction to it. Because the set of possible QRACM gates includes the CNOT and Toffoli gates, the Gottesman-Knill theorem means we cannot expect to apply QRACM transversally to any error-correcting code for which Clifford gates are also transversal (such as a surface code). Here I imagine a few different approaches, which are certainly non-exhaustive:

1. apply QRACM gates independently of the code and attempt to inject the results into another fault-tolerant quantum system;

2. construct an error-correcting code for which QRACM is transversal, and inject *other* gates into this code as needed;

3. code switch from a more practical code to a QRACM-transversal code;

4. find a non-transversal method to fault-tolerantly apply QRACM gates.

I am pessimistic about QRAM error correction, for reasons I discuss below, but this area needs more research. Finding constructions for satisfactory codes, or proving more impossibility results, would shed more light on the feasibility of QRAM.

Transversal codes. There are a few issues with a QRACM-transversal code.

First, QRACM with n address bits is in the $n+1$ level of the Clifford hierarchy, since it contains the n-multi-controlled NOT (consider access to a table T where only the $N-1$ entry is 1, the rest are 0). Using [PY15, Theorem 2], this means QRACM cannot be transversal for any family of subsystem codes that can aymptotically correct more than a $1/n$ probability of qubit loss (though, even in the extreme regimes I considered, this only forces qubit loss rates below 1/100, a common assumption anyway).

If such a code exists, it will also be unwieldy for computations, as it will require another procedure like state injection to perform H gates (since Toffoli+H is universal). Switching from a more practical code to a QRACM-specific code would need a series of intermediate codes to preserve the logical state, lest it decohere during the transition.

4.5.3 Gate Teleportation

Universal quantum computation in a surface code is possible via magic state distillation and gate teleportation. Several auxiliary qubits are prepared in a fixed state, the controller applies a physical T gate to each one, then distills them together with measurement to create a much higher fidelity state. Using this one state, the T gate can be teleported onto any qubit in the surface code.

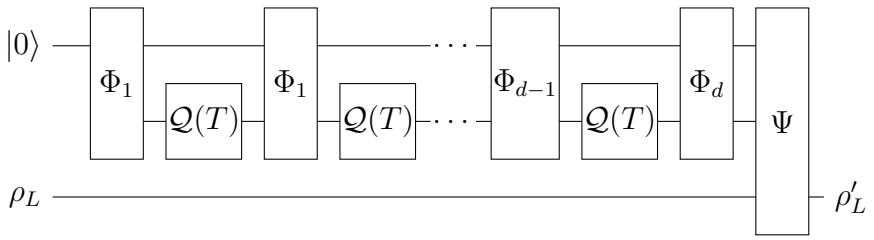


Figure 4.1: Schematic of a QRACM disillation and teleportation. The channels $\Phi_1, \ldots, \Phi_d$ attempt to distill the output of accesses to the physical QRACM table T, denoted $\mathcal{Q}(T)$. A final teleportation channel Ψ attempts to use the resulting distilled state to "teleport" the QRACM gate onto a logical state input ρ_L.

For an analogous process with QRACM, I define the following gate teleportation channel, shown in Figure 4.1. Let $\mathcal{H}_Q$ be the address and output space of the QRACM, and let $\mathcal{Q}(T) : B(\mathcal{H}_Q) \to B(\mathcal{H}_Q)$ be a channel representing the physical QRACM gate. I assume $\mathcal{Q}(T) = \mathcal{N}_1 \circ \mathcal{U}_{QRACM}(T) \circ \mathcal{N}_0$ for all T, meaning that our QRACM process is modelled as a perfect QRACM access sandwiched between two noise channels. I model in this way so that the noise channels are not dependent on the table itself, which aids in the proofs.

We have d distillation channels $\Phi_1, \ldots, \Phi_d$, with $\Phi_i : B(\mathcal{H}_{i-1} \otimes \mathcal{H}_Q) \to B(\mathcal{H}_i \otimes \mathcal{H}_Q)$, and the entire state distillation process produces a state $\rho_{distill}$ as follows:

$$\begin{aligned}\rho_{distill}(T) :=& \Phi_d \circ (I_{d-1} \otimes \mathcal{Q}(T)) \circ \\ & \Phi_{d-1} \circ (I_{d-2} \otimes \mathcal{Q}(T)) \circ \\ & \cdots \\ & \Phi_2 \circ (I_1 \otimes \mathcal{Q}(T)) \circ \Phi_1(|0\rangle\langle 0|)\end{aligned} \tag{4.9}$$

Letting $\mathcal{H}_{QL}$ represent the space of a logically encoded input and output space, we have a final teleportation channel $\Psi : B(\mathcal{H}_d \otimes \mathcal{H}_Q \otimes \mathcal{H}_{QL}) \to B(\mathcal{H}_{QL})$. We create a logical QRACM channel $\mathcal{Q}_L(T)$ by

$$\mathcal{Q}_L(T)(\rho) = \Psi(\rho_{distill}(T) \otimes \rho). \tag{4.10}$$

The goal is for $\mathcal{Q}_L(T)(\rho)$ to approximate a QRACM access on the logical state.

In the appendix I prove Theorem B.2.1, summarised approximately as:

Theorem 4.5.1. *Suppose a QRACM distillation and teleportation process uses d calls to the physical QRACM gate. Then the minimum fidelity of this distillation process is bounded by $\frac{3}{4} + 2\frac{\sqrt{d}}{N}$.*

That is, if we want a high fidelity QRACM operation to fit into a fault-tolerant computation, we must use resources that are asymptotically similar to those required to do fault-tolerant circuit QRACM.

The idea of the proof is fairly simple: considering the process in Figure 4.1, the input states to the QRACM are fixed. This means there is always some address j whose amplitude in the input state is on the order of $O(1/N)$. If we make a new table T' such that $T'[j] \neq T[j]$, then this is barely distinguishable to the state distillation process, since so little of the input to the QRACM will detect this difference. Yet, once we distill this state, it must suffice for all logical inputs. In particular, when our logical input is precisely the modified address j, the output between tables T and T' should be orthogonal. This is a contradiction: the distilled states are too close to create this orthogonal output.

A key component of this proof is that the state distillation is the same process for all tables, which is somewhat unrealistic. The specific pre- and post-processing channels could be adapted to the specific table at hand. However, counting arguments similar to Lemma B.1.1 might still apply; using only $o(N)$ resources in the distillation could mean that a single process handles an exponential number of different tables, and the proof of Theorem B.2.1 will still suffice. Unlike Lemma B.1.1, measurements are necessarily involved in distillation and teleportation, so formalising this line of reasoning would need a more sophisticated argument.

Theorem B.2.1 is a harsh blow to certain approaches to QRAM, but not all. It states that minimum fidelity is bounded, so applications with "broad" inputs to the QRAM, like look-ups to classical pre-computation, are not yet forbidden. However, a QAA search over a database in QRAM will suffer greatly. As the QAA hones in on the correct output, its input to the QRACM will get closer to this forbidden logical input. As a concrete example, if T and T' differ in precisely the address that QAA needs to find, then a distilled-and-teleported QRACM will obviously fail.

As a final note, since a QRAQM process implies a QRACM process by simply initialising the quantum memory to a classical input, Theorem B.2.1 forbids similar QRAQM distillation and teleportation processes.

Heralded QRAM. By "heralded" I mean any QRAM technology involving some measurement which indicates whether the QRAM access succeeded or not, typically also implying that, if it succeeded, the fidelity of the resulting access is very high.

Heralded processes are perfect candidates for gate teleportation because we can try as many times as we want to produce a single good magic state, then teleport it into the code. The impossibility of QRAM state teleportation is a grave blow to heralded QRAM. We cannot produce a resource state independently of the "live" data to which we want to apply the QRAM, so we need to instead input the real data into the QRAM, somehow ensuring that when the QRAM fails, this does not decohere the input.

Logical Inputs. Another potential route to save a process like this is to use the logical input as part of the QRACM access. This is more analogous to a transveral QRACM access, albeit with a very broad definition of "transversal" meaning only that we use a physical gate as part of the process to enact a logical gate.

This is a risky process, since the QRACM might noisily decohere the logical input. We want the full logical QRACM gate to have a much higher fidelity than the physical QRACM gate, so there must be little-to-no correlation between the logical state and the input to the QRACM. Intuitively, this suggests an extension of Theorem B.2.1 might exist, since if there is no correlation, then a similar table-swapping trick might work. I leave the details of a proof or disproof this intuition to future work.

4.6 Classical RAM

The extensive list of restrictions in Section 4.4 raises an important question of whether these same issues apply to classical memory. Here I apply the same arguments as Section 4.4 to classical memory. Either classical memory is fundamentally

easier and escapes the restrictions of quantum memory, or the restrictions are already accounted in real-world analysis and use cases of large-scale classical memory.

4.6.1 Area-Time Metric

The central thesis of the memory peripheral model, and most critiques of QRAM, is that the qubits in the QRAM (and possibly the memory controllers themselves) could perform other computations more efficiently than just the QRAM access. Succienctly, N bits of QRAM is more like $\Theta(N)$ parallel processors ([Ste16] states this explicitly).

This same argument applies to classical memory, and can mainly be justified by arguing that the cost to construct one bit of memory is proportional to the cost to construct one processor ([Ber01, Ber09] cost algorithms in currency for this reason). While the ratio of costs might be quite high, it is constant relative to the input size of any problem, suggesting that the ratio of memory to processors should stay roughly constant. Some real-world examples of this phenomenon:

- Google's "IMAGEN" [SCS+22], a huge text-to-image diffusion model, used 10 TB of public data plus an internal dataset of a similar size. The training used 512 TPU-v4 chips. Adding up all the independent processing units (i.e., arithmetic logic units (ALUs)), each chip has 2^{21} parallel "processors", for a total of 2^{30} processors [Goo22]. This means there is only about 18 kB of memory per ALU.

- GPT-3 used 570 GB of training data and used 175 billion parameters [BMR+20]. They use a supercomputer with more than 285,000 CPUs and 10,000 NVIDA V100 GPUs. Each GPU has 2^{12} CUDA cores, totalling at least 2^{26} parallel processing units, giving 11 kB per processor.

- In the current largest supercomputer in the world, "Frontier" [Cho22], each node has 5 TB of memory and 4 AMD Instinct MI250X GPUs. Each GPU has 14,080 stream processors [AMD22]. This memory/processor ratio is much higher, but still not huge, at about 89 MB/processor.

While the individual processors in a GPU or TPU are not what we usually think of as a "processor", they are capable of sequential computation. Machine learning benefits because it is an incredibly parallel task, but many analyses of algorithms with QRAM also present an alternative parallel classical algorithm.

All of this favours the *area-time* cost model for classical computation, where the cost is the product of the total amount of hardware by the runtime of an algorithm.

A further motivation is parallelism, as in [BP95]. Many classical algorithms parallelise perfectly, like discrete log attacks, classical lattice attacks, and unstructured search. This gives little reason to favour a computer with a large amount of memory with few processors, since for only a constant factor increase, many processors can share that memory and achieve the same result faster. As [BL13] say about a chip with most of its components idle: "any such chip is obviously highly suboptimal: for essentially the same investment in chip area one can build a much more active chip that stores the same data and that at the same time performs many other useful computations". More directly, someone paying for server time literally pays for the area-time product of their computation.

To summarise, the notion that the qubits and controllers in QRAM can be repurposed to more fruitful tasks is an argument that carries over directly to classical computing, and it is actually applied to large, high-performance computing for vast improvements in runtime with little increase in total cost.

4.6.2 Active Components

By far the biggest difference between quantum and classical memory is that classical memory only needs $O(\log N)$ active components to access memory. I sketch here how such a memory would work: there is a binary tree of routers, all of which are powered off. Each router is capable of receiving a signal which it amplifies and uses to decide to start drawing power. To route a memory request, the router in layer i of the tree, once activated, uses the ith address bit to decide whether to forward the address to its left or right child node. The process then repeats.

In this way, only $O(\log N)$ routers are actually on. It is able to do this because there is no restriction about leaking access patterns to the environment. Contrast this to a quantum memory: if it were given a superposition of all memory addresses, each router would need to be activated. Requesting a smaller superposition cannot reduce the number of active routers, since that would requiring knowing that the state had zero support over some addresses.

Flash memory. Real-world classical memory devices do not precisely follow this model. Following the description of [CGH+17], flash memory is divided into chips, which are divided into dies, which are divided into planes. Within each plane are thousands of *blocks*, each of which contains hundreds of *pages*, each of which contains thousands of bits. The device does not address individual bits, but full pages.

To access a specific block within a plane, two multiplexers (similar in function to a QRAM routing tree) decode the address of the block into a column and a row. A signal is sent along a wire at that column and at that row such that it activates the block at a memory location at the intersection of the selected column and row. From there, another multiplexer selects the appropriate page, and the device acts on that page (to read, write, etc.).

Abstracting this slightly, each block performs the equivalent of an AND gate: it returns the memory address only if there is a signal on the column bit line *and* on the row word line. Thus, as with QRAM, the number of "gates" is proportional to the number of pages of memory; however, in contrast to QRAM, only the "gates" on the active lines consume energy. In other words, the energy consumption to address one block should be proportional to the size of the length of each column and the length of each row (approximately the square root of the total number of blocks), and experiment confirms this [Moh10].

This experimental result implies also that the analogous process *within* the block, to address the page, consumes energy only in the blocks on the active row and column lines, at most. This is substantially fewer than the total number of blocks, each of which has its own page decoder, meaning that the page decoding

process matches the abstract approach of only activating the addressed sub-unit of memory. Here the analogy stops: addressing a single page within a block sends different signals to addressed and non-addressed pages, so the energy per active block is proportional to the size of the block.

Above the block level, to achieve minimum energy consumption we would expect each die to only activate the necessary planes, and each chip to only activate the necessary dies. Instead, the exact opposite happens: flash SSDs will group blocks and pages on different chips and planes into *superblocks* and *superpages*, which are accessed simultaneously. The purpose is so "the SSD can fully exploit the internal parallelism offered by multiple planes/chips, which in turn maximizes write throughput" [CGH+17]. Here again we see the "repurpose memory into processors" maxim: SSD designers expect that any application needing access to the SSD will have enough processors to reap more benefit from fast parallel access than from a more energy-efficient design.

Active memory. Any volatile classical memory, such as DRAM, is analogous to active gate QRAM, in that the total energy cost for N bits is $\Omega(N)$. In fact, it is even worse: volatile memory requires $\Omega(N)$ energy *per clock cycle*.

Despite this, DRAM sees widespread use. For consumer-level computers, RAM is often not the highest energy component, consuming less than any other component in an idle smartphone [CH10] and similarly for laptops. Though the asymptotic cost might increase with memory size, at practical sizes this is irrelevant. This cost might matter for large-scale applications; however, as in Section 4.6.1, in practice other components are also scaled up for these applications, so the energy draw of the RAM is not the main cost. Again, the area-time model captures this: total power draw increases with both the amount of volatile memory and the number of processors, so efficient machines will scale both proportionally.

4.6.3 Error Propagation and Correction

Section 4.4 dismissed QRAM technologies that require error rates to scale as $O(1/N)$ per component. One could make the same argument that if a single gate at one of the row and column lines of DRAM or flash memory spontaneously emits current, then it will create an error in the RAM. Thus, we should expect classical RAM to need component error rates to also scale as $O(1/N)$. Why is this acceptable classically?

The first reason is that extremely high fidelity classical components are fundamentally much easier to produce. Classical components have essentially one dimension of error: bit flips. In contrast, quantum components suffer from analogous bit-flip errors, but also phase-flip errors. Two-dimensonal passively corrected memory, which is provably impossible for quantum memory except in esoteric settings [BLP+16], is easy and ubiquitous classically (viz. hard disk drives).

While this analogy holds for non-volatile memory, volatile classical memory has a further advantage from the continuous energy flow to help correct errors. The technique of a "cold boot attack" highlights that volatile DRAM is somewhat non-volatile, but it just has an unacceptably high error rate unless power is continuously applied. Recall that we cannot do this to a passive gate QRAM, or else it becomes an active gate QRAM. Again, a QRAM device could work perfectly well if it *does* pay this cost (e.g. circuit QRAM), but certain algorithms lose their advantage.

Second, classical memory can, and does, perform error correction. Circuit QRAM is allowed to require $O(1/N)$ error because if it is part of an error corrected quantum computer, that level of noise can be acheived with polylogarithmic overhead. Beyond the consumer level, classical memory *also* uses error correcting codes because the overall failure rate is too high. The error rate for RAM, in failures per Mbit per million hours of operation, is somewhere around 25-75 failures [SPW11] or 0.2 to 0.8 [ZDM+19], though with high variance not only between devices but also over time for a given device.

This error rate shows two points: first, it is low enough to be negligible for most consumer applications, and second, it is still high enough to be relevant at large

scales. This exactly matches my claims about QRAM. The classical difference is that the achievable error rate for physical components is much, much smaller.

Finally, errors present less of a problem for classical computing because of the no-cloning theorem. Modern large-scale, high-performance computing often follows the "checkpoint/restart model" [Gio17], where the computer periodically saves the entire state of memory so that it can resume in the case of a fault. A quantum computer cannot do this because of the no-cloning theorem. Instead, the error rate of each operation must be driven so low that the *entire* computation can proceed without fault.

4.6.4 Latency

All memory devices, classical and quantum, must contend with signal latency and connectivity. If connections between qubits are limited to fixed length, then because we live in Euclidean space (a) each qubit can have only a constant number of connections, and (b) sending data between qubits on opposite ends of the memory requires at least $W^{1/d+o(1)}$ connections in d-dimensional space, if there are W total qubits. Long-range connections still suffer from signal latency, where – asymptotically – the time for signals to propagate across a memory device is still $W^{1/d+o(1)}$.

I emphasise that even light-speed signals have noticeable latency: for a 3 GHz processor, light travels only 10 cm in each clock cycle. We often do not see this, because (again) huge classical data structures typically co-exist with enormous numbers of parallel processors (as [BP95] predict based on this reasoning). Classical memory latency has led to tremendous effort in efficient caching and routing. In contrast, quantum memory access "takes the mother of all cache misses" [Sch21], since any timing information effectively measures which path the access took, and hence which address.

Thus, the case where latency would be an issue – where a single processor attempts to process an enormous amount of data with completely unpredictable queries – is extremely rare for classical computers. Yet QRAM applications often

assume a single quantum processor, and even when assuming parallel access like [BBG+13], the quantum memory cannot efficiently cache. Overall, quantum memory latency seems more difficult to manage than classical.

4.7 Conclusion

Large scale, high quality QRAM which is cheap to use is unlikely. It would require passively-corrected quantum memory and highly complex, high-fidelity ballistic computation. I argued in Section 4.4 and Section 4.5 that these are much stronger assumptions about quantum computing hardware than general large-scale, fault-tolerant quantum computing. Even if those assumptions come true, QRAM may still be difficult: building a ballistic computation is complicated (Theorem 4.4.1) and it may not interact well with the rest of a fault-tolerant quantum computation (Theorem 4.5.1). In the full version of this work [JR23] I survey proposed gate-QRAM technologies and all of them were either implicitly active or required unreasonably low error rates.

All of that said, I cannot *prove* that QRAM will not be possible. However, many powerful computing devices (say, a polynomial time SAT-solver) are not provably impossible, but we do not assume they will be built. My claim is that QRAM should be considered infeasible until proven otherwise, instead of the reverse. While classical RAM appears to set a precedent for such devices, Section 4.6 argues that this is an inappropriate comparison: QRAM is more difficult relative to other quantum computations, than classical RAM is difficult relative to classical computation.

Despite these critiques, QRAM will likely be a crucial tool for quantum algorithms. In a circuit model, QRAM has a linear cost, but optimized circuits keep the constants low. The rest of this thesis will consider some uses of circuit QRAM.

5

Attacking Elliptic Curve Cryptography

Contributions. This chapter closely follows [HJN+20]. This was a highly collaborative work. The subroutines using windowing with quantum memory were mostly my contribution. Beyond this publication, this chapter presents the results of working around some bugs in the quantum resource estimation that affected the original estimates and includes some new estimates of error correction overhead.

The most immediate threat to cryptography from quantum computers is Shor's algorithm [Sho94]. This will break the RSA cryptosystem and all cryptography built around the hardness of the discrete log problem. This includes all elliptic curve cryptography (ECC)[1].

This attack is easier because it is only polynomial time. More than that, it is relatively fast: the majority of the cost of running Shor's algorithm is running a quantum analogue of the encryption function itself. For this reason, my goal in this chapter is to make that computation more efficient.

Building on [RNSL17], my main contribution in [HJN+20] that I present in this chapter is to use QRACM to accelerate two aspects of the computation: modular multiplication and the overall point multiplication. These are based on *windowing*, a traditional classical method of accelerating point multiplication [BGMW92] first adapted to quantum computing in [Gid19c]. My contribution was to extend windowing to multiplying two quantum registers and to give optimisations specific to elliptic curve arithmetic.

These techniques save a constant and logarithmic factor, respectively. The conclusion is that ECC is easier to attack than RSA, which was the consensus before the new RSA estimates in [GE21]. Our work in [HJN+20] allowed the ECC attack to catch up to these new estimates.

Further, I also give an analysis of the resources accounting for error correction. I conclude that optimising logical qubit count does translate into an optimised physical qubit count. Whereas RSA-2048 could be factored with 20 million qubits in 8 hours [GE21], I show that ECC-256 could be broken with 6 million qubits in 7 hours.

5.1 Introduction

For a more thorough introduction to the discrete log problem and elliptic curve cryptography, see [KL20]. For Shor's algorithm, see [NC00].

[1] I consider isogeny-based cryptography a separate category.

5.1.1 Discrete Log

This chapter addresses the following problem:

Problem 5.1.1 (Discrete Logarithm (or Discrete Log or DLOG))**.** *Let G be an abelian group. Given two points $P, Q \in G$, find $n \in \mathbb{N}$ such that $Q = nP$.*

Solving this is sufficient (though possibly not necessary) to break a spectacular array of modern cryptography. As one example, the certificate my computer just received from youtube.com, verifying that I am connecting to Google, is signed with "ECDSA". That means my computer received a string of data, shown below,

04:BD:DF:F6:8E:04:62:C8:E2:11:12:C4:BF:46:2F:7A:96:7B:ED:04:C4
83:25:7C:44:8D:13:BF:06:68:18:10:04:7D:AC:D1:A3:73:27:B1:BE:6C
98:B3:8E:D2:7E:4C:8A:B9:11:00:FD:B0:BF:91:F4:C0:5F:37:8E:40:CA:F4:0F

representing Q in a group called P-256. If Eve can solve the discrete log problem for this point with respect to the base point

$x =$0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296
$y =$0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5

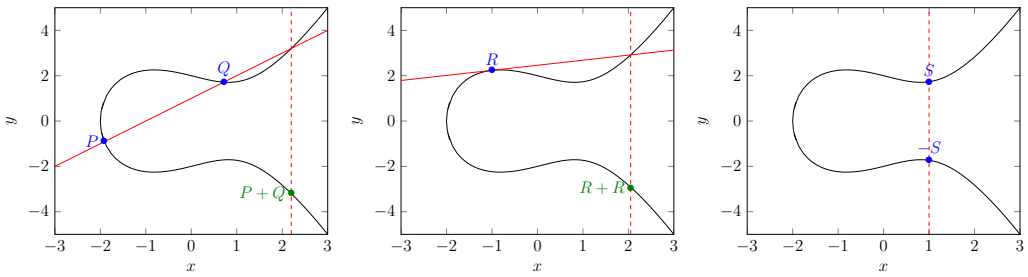
then Eve can sign anything on behalf of Google!

My connection to youtube was encrypted with keys established from an elliptic curve key exchange, meaning that if anyone eavesdropped on my connection (which would be easy to do, since I am on a wireless network), they would see another point Q such that, if they solved the DLOG problem for this point, they could decrypt the entire communication.

5.1.2 Elliptic Curves

An elliptic curve $E[\mathbb{F}_q]$ (for our purposes) is a group whose elements are the collection of points $(x, y) \in \mathbb{F}_q^2$ satisfying an equation

$$y^2 = x^3 + Ax + B \mod q \tag{5.1}$$



(a) Adding distinct points. **(b)** Adding a point to itself. **(c)** The inverse of a point.

Figure 5.1: Geometric description of the elliptic curve group laws.

for $A, B \in \mathbb{F}_q$, together with a "point at infinity" $\mathcal{O}$. Technically this is only the "affine Weierstrass" form. The addition law is defined geometrically, as shown in Figure 5.1.

Given A and B from Equation (5.1), Schoof's algorithm can find the number of points in polynomial time [Sch85]. Several standard curves exist for cryptography, where the number of points is known and is a large prime.

5.1.3 Shor's Algorithm

Given a DLOG problem with points P and Q, with P having prime order equal to q, Shor's algorithm solves DLOG in three steps:

1. Create a superposition of points $\sum_{a,b\in(\mathbb{Z}/q\mathbb{Z})} |a\rangle\,|b\rangle$.

2. Use the first two registers to control addition of multiples of P and Q, to create $\sum_{a,b\in(\mathbb{Z}/q\mathbb{Z})} |a\rangle\,|b\rangle\,|aP + bQ\rangle$, then measure the third register to obtain a point R.

3. Apply a quantum Fourier transform (QFT) to each of the first two registers, then measure the result.

Throughout, I will let $n = \lceil \lg q \rceil$ and let s be the value such that $Q = sP$.

[NC00] explain the method in more detail, but with high probability the measurement produces two integers $j, k \in (\mathbb{Z}/2^n\mathbb{Z})$ such that $\frac{j - sk \mod 2^n}{2^n}$ is on the order of $\frac{1}{p}$. From this Eve can solve for s.

Step 1 (the superposition) can be easily done with $2n$ Hadamard gates and a conditional measurement (to ensure that the points are less than q), and Step 3 can also be done easily. In fact most papers assume it is done "semi-classically" (e.g. [PZ03, RNSL17]), meaning we alternatively use only a few bits of a and b to control addition of multiples of P and Q, then QFT those bits and measure immediately [GN96]. The middle step, performing the scalar multiplication in superposition, is the most difficult, and what this chapter will focus on.

5.2 Arithmetic

To add distinct points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, one computes

$$\lambda = \frac{y_P - y_Q}{x_P - x_Q}, \quad x_{P+Q} = \lambda^2 - x_Q - x_P, \quad y_{P+Q} = (x_P - x_{P+Q})\lambda - y_P, \tag{5.2}$$

where all arithmetic is performed in the field defining the curve. I will focus on prime-order moduli, so all of the above is done modulo a large prime p ([BBvHL20, PWLK22] are concurrent and subsequent work in binary fields).

You can see that we need addition, subtraction, multiplication, squaring, and division modulo p. Circuits for all of these are in [RNSL17], which we improved on in [HJN+20]. I will focus on my contributions using QRACM.

5.2.1 Multiplication

Suppose Eve wants to multiply two integers $|x\rangle$ and $|y\rangle$. Assume they are represented as qubits, i.e. $|x\rangle = |x_0\rangle\,|x_1\rangle \ldots |x_{n-1}\rangle$ such that $x = x_0 + x_1 \cdot 2^1 + \cdots + x_{n-1} \cdot 2^{n-1}$. Then so-called "schoolbook multiplication" produces $|xy\rangle$ by the following:

1: **for** $i = 0$ to $n - 1$ **do**
2: Use x_i to control addition of y to the output register.
3: Double y
4: **end for**

This requires n controlled additions, and since addition requires $c_A n + o(n)$ gates for some constant c_A, this requires $c_A n^2 + o(n^2)$ gates (the depth may be $d_A n + o(n)$

or $d_A \lg n + o(\log n)$, depending on the addition circuit). More sophisticated methods exist [Gid19a], but they are less effective at the range of n I consider.

However, Eve wants $|xy \mod p\rangle$, not just $|xy\rangle$. Naively she could do all the operations above modulo p, but that is rather expensive. [RC18] do an integer multiplication first, then divide by p.

An alternative approach from [RNSL17] was to use Montgomery form integers and add multiples of p as they progress.

Definition 5.2.1 (Montgomery Representation, [Mon85])**.** *For an odd prime p, the Montgomery Representation of an integer $x \in (\mathbb{Z}/p\mathbb{Z})$ is $x2^n \mod p$.*

Given two integers x and y in Montgomery representration, their product would be $xy2^{2n} \mod p$, which has an extra factor of 2^n. Thus, Eve needs to divide by 2^n at the same time as she divides out by p. She uses two facts to make this happen easily:

- If the least-significant-bit of xy is 0, then she can divide by 2 simply by shifting all bits (i.e., ignoring the last bit)
- If the least-significant bit of xy is non-zero, she can make it zero by adding p, which is odd.

This leads to the following algorithm for Montgomery multiplication:

1: **for** $i = 0$ to $n - 1$ **do**
2: Use x_i to control addition of y to the output register.
3: If the least-significant bit of the output register is 1, add p to the output register.
4: Shift the output register 1 qubit to the right.
5: **end for**

Translated to a circuit, it looks like Figure 5.2:

A small hiccup is that this creates entangled "junk" qubits, because Eve needs to translate the "if" block to a controlled addition of p, but that means she must copy out the least significant bit to be the control (adding p in-place is not reversible, as it sets the least significant bit to 0). Applying this circuit in reverse uncomputes both the junk and the output, so this can be done later once the output is no longer needed.

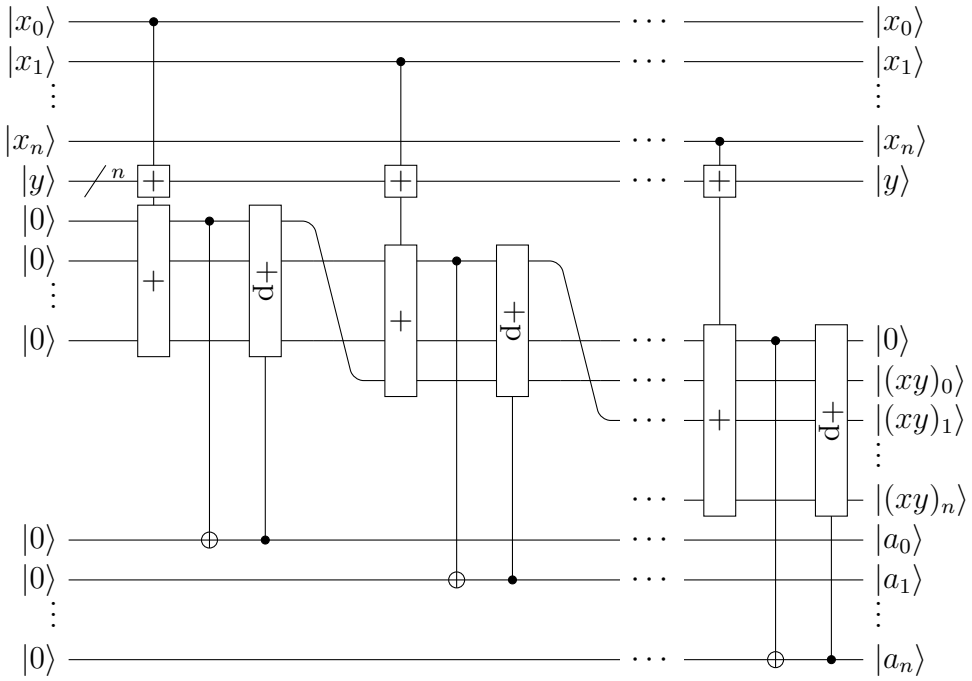


Figure 5.2: Multiplying integers in Montgomery form, as in [RNSL17].

This was the method of [RNSL17], and we improved the uncomputation strategy in [HJN+20].

Windowed Multiplication

The contribution I added was to shrink the cost by offloading computation to the classical co-processor with *windowing*. Windowing is an old classical method, adapted to quantum computing in [Gid19c], but the existing quantum techniques only add classical integers onto quantum integers.

To explain the technique, I will use larger multiples of p to clear out the least significant k bits, rather than just one least significant bit.

First, split $x = x_0 + x_1 \cdot 2^k + \cdots + x_j \cdot 2^{jk}$, such that each "word" x_i has k bits. Starting with $|x_0\rangle |x_1\rangle \ldots |x_j\rangle |y\rangle |z\rangle$ (where $|z\rangle$ is the output register), Eve can use schoolbook multiplication to compute $|x_i y + z\rangle$ in the output register. She then wants to clear the bottom k bits of this output.

Let $m_i = x_i y + z \mod 2^k$; she copies this out to another register. She wants to find some a_i such that $a_i p + m_i \equiv 0 \mod 2^k$. Since p is odd, it is invertible modulo

2^k, so there exists a value $a_i \equiv -m_i p^{-1} \mod 2^k$. Since such a_i exists, Eve wants to add $a_i p$ to the register $|x_i y + z\rangle$. How does she compute $a_i p$ from $|m_i\rangle$?

She does this computation classically. That is, she constructs a table of size 2^k, indexed by possible values of m_i, where each value corresponds to $a_i p$ such that $a_i p \equiv -m_i \mod 2^k$. With such a table in place, she uses QRACM, indexed by m_i, to look up the $n+k$-bit value $a_i p$ corresponding to m_i.

Once this value is looked up, she adds it to the register $|x_i y + z\rangle$. This will set the least-significant k qubits to 0, and then she can shift the result down by k bits to get something equivalent modulo p to $2^{-k}(x_i y + z)$.

If she repeats this for i from 0 to j, starting with $z = 0$, the end result is

$$2^{-n} x_0 y + 2^{-n+k} x_1 y + \cdots + 2^{-k} x_j y \equiv 2^{-n} xy \mod p \tag{5.3}$$

Recalling that $x = 2^n x'$ and $y = 2^n y'$ for integers x' and y' encoded in a Montgomery representation, this gives Eve precisely the result $2^n x'y'$ that she wanted.

There is a minor issue: the result is *equivalent* to $2^n x'y'$ modulo p, but Eve needs a representation between 0 and $p-1$. Does that hold?

Lemma 5.2.1. *After each step of the above algorithm, the result in the output register is at most $p-1$.*

Proof. By induction. Since $z = 0$ is the starting value, the base case holds. After each step, the current value is at most $p-1$, Eve adds $a_i p$ which is at most $(2^k - 1)p$, and then she divides by 2^k. I can thus bound the result by

$$\frac{p - 1 + (2^k - 1)p}{2^k} = p - \frac{1}{2^k} < p \tag{5.4}$$

Since the output is an integer, the result follows. □

As a circuit, this looks like Figure 5.3.

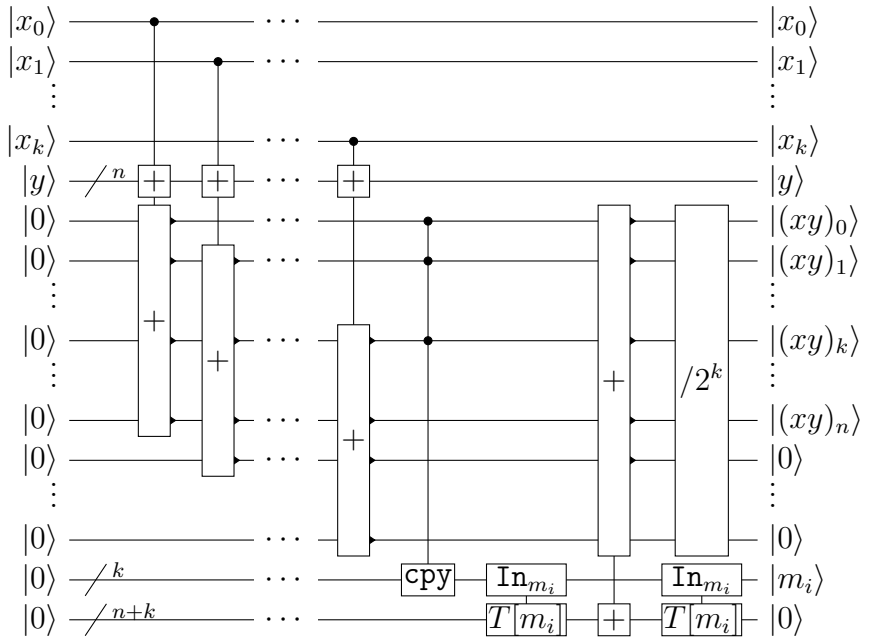


Figure 5.3: One window of a windowed modular multiplication, from [HJN+20]. Here `cpy` simply copies out bits with multiple `CNOT` gates, `In` is the input to a QRACM access to a table T. T is the table such that $T[m_i]$ is a multiple of p equivalent to m_i modulo 2^k.

Cost

Gate cost. A QRACM access to 2^k values costs $4 \cdot 2^k$ T-gates, by [BGB+18]. Let C_A and C_{c-A} be constants such that an n-bit addition costs $C_A n + o(n)$ T-gates and an n-bit controlled addition costs $C_{c-A} n + o(n)$ gates. The cost of an n-bit windowed modular multiplication is:

- $C_{c-A} n^2 + o(n^2)$ gates for the schoolbook multiplications;
- $4 \cdot 2^k$ T-gates for each QRACM look-up, of which Eve needs $2\left\lceil \frac{n}{k} \right\rceil$ repetitions (to both compute and uncompute).
- $C_A(n+k) + o(n+k)$ T-gates for the addition of $a_i p$, of which Eve needs $\left\lceil \frac{n}{k} \right\rceil$ repetitions.

This gives a total cost of

$$C_{c-A} n^2 + o(n^2) + \left\lceil \frac{n}{k} \right\rceil \left(8 \cdot 2^k + C_A(n+k) + o(n+k)\right). \tag{5.5}$$

If I ignore the little-o terms, remove the $\lceil\rceil$, and treat k as a real number instead of an integer, some calculus optimises this when

$$k = \lg n + \lg C_A - 3 - \lg(k \log 2 - 1) \tag{5.6}$$

As a quick approximation, I can set $k = \lg(C_A n/3)$, and the cost becomes

$$C_{c-A} n^2 + o(n^2) + \frac{2C_A n^2}{\lg(C_A n/3)} = C_{c-A} n^2 + o(n^2) \tag{5.7}$$

Asymptotically, this reduces the cost to just the schoolbook multiplication cost! At the scales we considered in [HJN+20], the T-gate cost indeed scaled in this way.

We found the actual optimal k numerically by estimating the full circuit cost for different values of k.

Depth cost. I can do the same analysis for depth, assuming Eve uses a log-depth adder [DKRS04]. If she uses a linear depth adder the analysis is the same as the gate cost. The T-depth of the QRACM look-up is 2^k, addition is $D_A \lg n + o(\lg n)$ T-depth and controlled addition is $D_{c-A} \lg n + o(\lg n)$. Then the total depth is:

- $D_{c-A} n \lg n + o(n \lg n)$ T-depth for the schoolbook multiplications;
- 2^k T-depth for each QRACM look-up, of which Eve needs $2\left\lceil \frac{n}{k} \right\rceil$ repetitions (to both compute and uncompute).
- $D_A \lg(n+k) + o(\lg(n+k))$ T-depth for the addition of $a_i p$, of which Eve needs $\left\lceil \frac{n}{k} \right\rceil$ repetitions.

The total T-depth is then

$$D_A n \lg n + o(n \lg n) + \left\lceil \frac{n}{k} \right\rceil \left(2 \cdot 2^k + D_A \lg(n+k) + o(\lg(n+k))\right). \tag{5.8}$$

The same naive optimisation is less tractable, but I can take a simple value of $k = \lg(D_A \lg(n)/2)$ to obtain

$$D_A n \lg n + o(n \lg n) + \frac{D_A n \lg n}{\lg(D_A \lg(n)/2)} + o(n \lg n / \log\log(n)) = D_A n \lg n + o(n \lg n). \tag{5.9}$$

Again, this is asymptotically the same as schoolbook multiplication, though the lower-order terms converge slower than the case for gate cost. We found a depth more like $2D_A n \lg n$, though this was still around a 22% reduction from unwindowed multiplication.

Width. You might notice that Eve needs an extra $n+k$ qubits for the output of the QRACM-lookup, which seems like a steep price for a constant-factor improvement. However, in the context of elliptic curve point addition, multiplication always takes place when there are enough auxiliary qubits from division that it does not require extra qubits.

5.3 Windowing Shor's Algorithm

Eve can also window the overall algorithm, and I will show some optimisations as well. Recall that her goal is to take a state $|a\rangle |R\rangle$ and compute $|a\rangle |R + aP\rangle$ (and similarly for b and bQ). Traditionally, this is done by representing $|a\rangle = |a_0\rangle |a_1\rangle \dots |a_{n-1}\rangle$ such that $a = a_0 + a_1 \cdot 2^1 + \dots + a_{n-1} \cdot 2^{n-1}$, and each a_i is used to control addition by $2^i \cdot P$. Since P is known classically, Eve can easily precompute all these powers of P.

To window this process, she can simply group the qubits into groups of k qubits, so $a = a_0 + a_1 \cdot 2^k + \dots + a_j \cdot 2^{kj}$. For each i, she pre-computes a table of $2^{ik}P, 2 \cdot 2^{ik}P, 3 \cdot 2^{ik}P, \dots, (2^k - 1)2^{ik}P$. She then does a QRACM look-up into this table, indexed by the word a_i. Then she does a quantum-quantum elliptic curve point addition onto the output register, then uncomputes the point she looked up.

This is the basic method, which we optimised for elliptic curves.

5.3.1 Signed Look-ups

Notice that if Eve looks up a multiple of P such as $a_i P$, she can immediately obtain $-a_i P$ by inverting the y-coordinate (recall that if $P = (x, y)$, $-P = (x, -y)$). On the scale of a full elliptic curve point addition, a single modular inversion is essentially free. Thus, she would rather build tables where she can use one bit of a to control a modular inversion, effectively doubling the table size produced from each lookup.

To detail the arithmetic, take a k-bit word of the address, a_i, and let $a_i = a_{i,top}2^k + a_{i,bot}$ such that $a_{i,bot}$ is $k-1$-bits long and $a_{i,top} \in \{0,1\}$. Eve will use $a_{i,bot}$ to look up $a_{i,bot}2^{ki}P$. She starts by using the negation of $|a_{i,top}\rangle$ to control a bit flip of *all* qubits in $|a_{i,bot}\rangle$, i.e., she flips $|a_{i,bot}\rangle$ if and only if $|a_{i,top}\rangle = |0\rangle$. Flipping all the bits is the same as negating modulo 2^k, so this negates $a_{i,bot}$ modulo 2^k if $a_{i,top} = 0$. She then uses QRACM to look up an elliptic curve point based on the $a_{i,bot}$; this gives a state

$$\begin{cases} |a_{i,bot}\rangle \left|a_{i,bot}2^{ik}P\right\rangle & , \text{ if } a_{i,top} = 1 \\ \left|2^k - a_{i,bot}\right\rangle \left|(2^k - a_{i,bot})2^{ik}P\right\rangle & , \text{ if } a_{i,top} = 0 \end{cases} \tag{5.10}$$

If $a_{i,top} = 0$, she then negates the point she looked up[2]. This makes the point into $\left|(a_{i,bot} - 2^k)2^{ik}P\right\rangle$, so I can now write the state as

$$\left|(2^k a_{i,top} + a_{i,bot} - 2^k)2^{ik}P\right\rangle = \left|a_i 2^{ik}P - 2^{(i+1)k}P\right\rangle \tag{5.11}$$

This looks almost like the desired output, but it has that annoying extra term of $2^{(i-1)k}P$.

Actually that term is fine because it does not depend on a. In more detail, consider that after each look-up, Eve adds the result to the output register. This means once she finishes all additions, she has enacted the map

$$|a\rangle |R\rangle \mapsto |a\rangle \left|R + aP - 2^k P - 2^{2k}P - \cdots - 2^{(m+1)k}P\right\rangle \tag{5.12}$$

but the required interference for Shor's algorithm depends only on the relation between aP and bQ, so the extra terms do not matter. Or more directly, the starting point R was arbitrary; she could have started with $R + 2^k P + 2^{2k}P + \cdots + 2^{(m+1)k}P$ and cancelled out the extra terms.

A remaining subtlety of signed addition is that Eve will need to look up $0P$, the identity point. The optimised addition circuit from [RNSL17], which we started from in [HJN+20], fails when one of the points is the identity. Thus, we modified the data structure slightly: Eve will store a z coordinate with one qubit as either

[2]She could also un-negate $a_{i,bot}$ with the same controlled bit flips, but it is actually more efficient to leave it negated; see Figure 5.4.

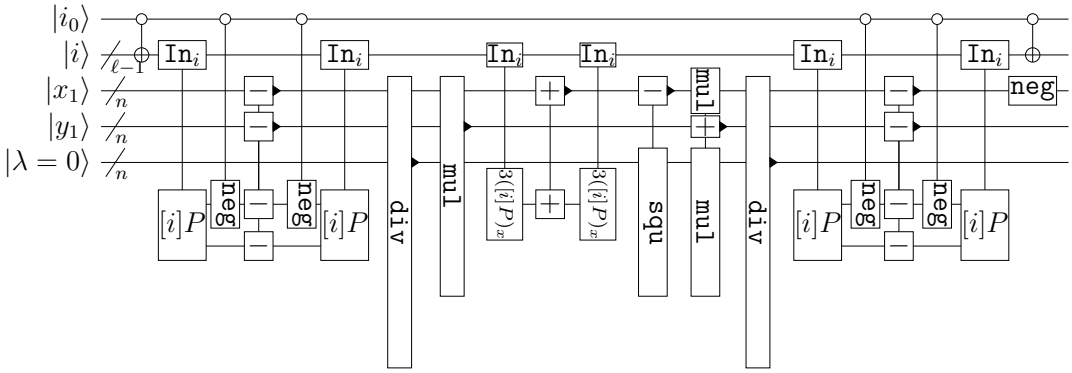


Figure 5.4: One signed, windowed point addition circuit, from [HJN+20]. The width of each circuit block indicates the number of auxiliary qubits it needs.

$|0\rangle$ (for the identity) or $|1\rangle$ (otherwise), and she will use that to control all other parts of the circuit (in fact, it only needs to control two divisions and one negation, so this adds relatively little overhead).

5.3.2 Memory-efficient Look-ups

Naive windowing would write the entire point to a new register, but that takes $2n + 1$ qubits (for x, y, and one for the z coordinate). That is quite high, but Eve can save space by repeating the look-up three times.

More precisely, see Figure 5.4. There are only three points where the second point is mixed with the first, and these points only *add* the coordinates. Luckily, at these points, almost all of the auxiliary qubits are free, so Eve can fit the look-up entirely within the idle auxiliary qubits.

5.3.3 Cost Analysis

A QRACM look-up with the fewest qubits (which is what Eve wants for this case, where she wants only a constant multiple of n for the qubit count) in a table of size 2^k needs $4 \cdot 2^k$ T-gates and T-depth. Since her windows have size $k + 1$ thanks to the signed addition trick, the total cost is

$$\left\lceil \frac{n}{k+1} \right\rceil (\mathsf{A} + 24 \cdot 2^k) \tag{5.13}$$

where A is the cost of point addition, in T-gates or T-depth. We see that $k \approx \lg(\mathsf{A}/24)$ should be optimal. For [HJN+20], we simply checked all window sizes with a short script and chose the optimal size.

This means the total cost is $O(n\frac{\mathsf{A}}{\log \mathsf{A}})$, unlike $O(n\mathsf{A})$ for non-windowed addition. This is a significant saving, since A has gate cost $\Omega(n^2)$ and depth either $\Omega(n \lg n)$ or $\Omega(n^2)$.

5.4 Q# Resource Estimation

Microsoft develops an open source quantum development kit based around a functional language called "Q#" [SGT+18]. We wrote all of the above circuits in Q#, allowing automated tests and resource estimations.

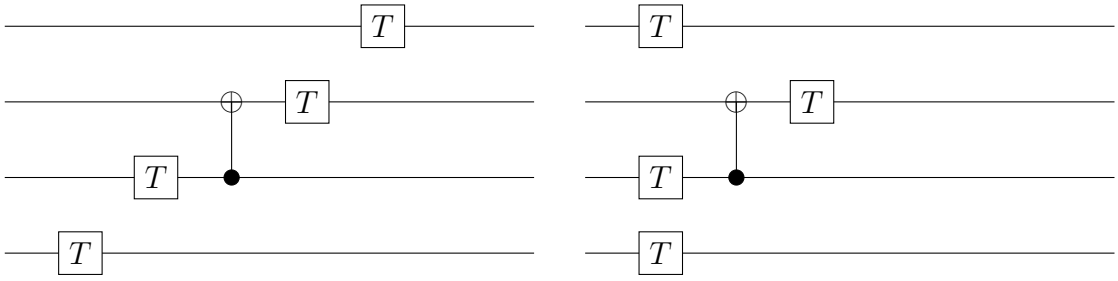
The tests are done by classical computers, and so they cannot test the full algorithm. However, we can test the elliptic curve addition circuits, since these operate only on computational basis states, which are easy to manipulate classically.

Resource estimation is the more interesting feature for my thesis. As the Q# program runs, it sends each gate to a simulator-type object, which can be a simulator or a resource estimator. The resource estimator tracks the total number of gates, their depth, and the number of qubits. This estimator has three features that make it extremely helpful for complicated circuits (e.g., the modular arithmetic library alone has over 2300 lines of code):

- The Q# language is intended to be agnostic to the underlying simulator.
- Rather than manually managing qubits, Q# provides a "`use`" keyword which allocates new qubits and releases them as needed.
- The resource estimator will automatically parallelise any gates that it can, by essentially playing Tetris with the circuit (Figure 5.5).

Unfortunately, these are rather ambitious features, and they have led to two major bugs[3].

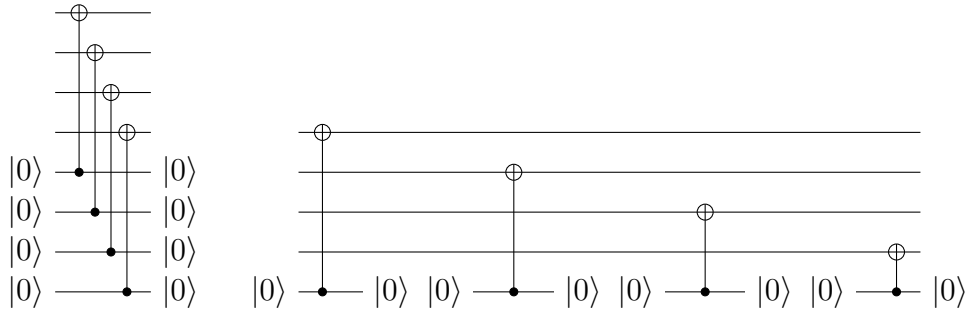
[3]Reported here: `https://github.com/microsoft/qsharp-runtime/issues/1116` and `https://github.com/microsoft/qsharp-runtime/issues/1037`



(a) A sequence of gates as they might be sent to Q#.

(b) The same gates, arranged by dependencies for minimum depth.

Figure 5.5: How Q# parallises gates with Tetris logic.



(a) Low-depth

(b) Low-width

Figure 5.6: Two approaches to construct the same circuit.

5.4.1 Depth-Width Inconsistency

The example we wrote in [JNRV23] highlights the depth-width issue. The following code loops through an input array of 4 qubits. For each qubit, it allocates a new auxiliary qubit, applies a CNOT, then frees the new qubit for other operations.

```
operation DepthWidthTradeoff(inputQubits: Qubit[]) : Unit {
    for i in 0..3 {
        use aux = Qubit() {
            CNOT(aux, inputQubits[i]);
        }
    }
}
```

Assuming CNOT has depth 1, what is the depth and width of this circuit? There are two extreme choices the compiler can make, shown in Figure 5.6.

When we published [HJN+20], the resource estimator would output the depth from the low-depth tradeoff and the width from the low-width tradeoff. This is often impossible. If we chose one strategy at all times, this creates unreasonably

large depths or widths. Moreover, there are hard-to-detect cases where we can have the best of both worlds (e.g., if some other aspect of the looped circuit prevents it from parallelising). The latest version of Q# attempts to detect these cases, but when it chooses low-depth trade-offs, the bug reappears. The low-width version still seems correct, so I used only that for this thesis.

The primary source of the extra "`use`" blocks is in each `AND` or each `CCNOT`. To acheive T-depth 1, these need auxiliary qubits. To fix this, I simply changed the code to use `AND` and `CCNOT` that required no auxiliary qubits, which have T-depth 2 and 3, respectively. This increases the T-depth of the log-depth adder from 39 to 107 for 512-bit inputs, for example.

5.4.2 Measurement Feedback

Taking the second example we wrote in [JNRV23], what is the T-depth of the following circuit?

```
operation MeasurementFeedback(qubits: Qubit[]) : Unit {
    T(qubit[0]);
    if M(qubit[0]) == One {
        T(qubit[1]);
    } else {
        T(qubit[2]);
    }
}
```

It must be 2: we cannot apply the second T gate until we have measured qubit 0, which we cannot do until the first T-gate is finished. However, Q# reports a T-depth of 1, because the simulator backend cannot see the dependency in the classical logic. By design, the classical logic in Q# is invisible to the simulator, but here it must be visible.

Since we used measurement-based uncomputation for `AND`s from [Gid18], this problem created a serious issue. Luckily, there is an easy "hack" to avoid this: Q# also offers a multi-qubit measurement that it reports to have the same cost as a single-qubit measurement. In any situation like the above, I apply a multi-qubit measurement to effectively block all the qubits that depend on the result. This would not be enough for more complicated dependencies, but luckily I have none.

This also makes the circuit incorrect. I thus use a global "`IsTestable`" variable, so that Q# can decide to use the single-qubit measurement for tests and the multi-qubit measurement for resource estimation.

5.4.3 Initial Results

Each elliptic curve point addition is extraordinarily complicated, so running the resource estimation is a lengthy process. Rather than run it for the entirety of Shor's algorithm, we estimated only one point addition with Q#, and then we can multiply the cost by the number of point additions ($2n$, generally).

However, the cost of a point addition depends on the window size. To estimate this, we seperately ran estimates for QRACM look-ups at each window size, and then computed the final cost as:

$$\left\lceil \frac{2n}{k} \right\rceil \cdot \begin{pmatrix} \text{cost of one addition with window size } 8 \\ -3 \cdot \text{cost of QRACM look-up of size } 2^8 \\ +3 \cdot \text{cost of QRACM look-up of size } 2^k \end{pmatrix}. \tag{5.14}$$

This is because one addition uses 3 QRACM look-ups (see Section 5.3.2), so we estimated with a fixed window size and then corrected it. Then we multiplied by the number of additions we needed based on the window size. In fact, we separately accounted for the last addition of P and Q, each of which will have size $n - \left\lfloor \frac{n}{k} \right\rfloor$.

To decide on the best window size, we tried all window sizes up to $\frac{n}{2}$ and chose the lowest-cost (in T-gates, T-gates, and T-depth, for low-width, low-T, and low-depth strategies, respectively).

Doing all of this gave final results, shown in Table 5.1, *which are incorrect*. Section 5.4.1 and Section 5.4.2 explain the bugs that make these estimates over-optimistic.

5.5 Error Correction

Right now, the most plausible approach to run these circuits is use a surface code, so a "full-stack" resource estimatation, such as [GE21] would be best. [GE21] provides a full two-dimensional layout for each operation, which Q# does not provide. Instead, I will estimate with less precision by modifying how Q# counts logical resources.

Circuit	Window Size	Gates				Depth		Width
		Cliffords	Measure	T	Total	T	All gates	Qubits
256-bit modulus								
RNSL	–	–	–	$1.60 \cdot 2^{39}$	–	$1.69 \cdot 2^{36}$	–	2338
Low W	19	$1.32 \cdot 2^{34}$	$1.76 \cdot 2^{26}$	$1.72 \cdot 2^{32}$	$1.45 \cdot 2^{35}$	$1.98 \cdot 2^{30}$	$1.89 \cdot 2^{32}$	2124
Low T	19	$1.75 \cdot 2^{33}$	$1.95 \cdot 2^{27}$	$1.08 \cdot 2^{31}$	$1.80 \cdot 2^{34}$	$1.44 \cdot 2^{29}$	$1.85 \cdot 2^{31}$	2619
Low D	15	$1.04 \cdot 2^{34}$	$1.61 \cdot 2^{28}$	$1.34 \cdot 2^{32}$	$1.40 \cdot 2^{34}$	$1.12 \cdot 2^{24}$	$1.40 \cdot 2^{27}$	2871
384-bit modulus								
RNSL	–	–	–	$1.44 \cdot 2^{41}$	–	$1.51 \cdot 2^{38}$	–	3492
Low W	21	$1.46 \cdot 2^{36}$	$1.23 \cdot 2^{29}$	$1.51 \cdot 2^{34}$	$1.57 \cdot 2^{37}$	$1.68 \cdot 2^{32}$	$1.77 \cdot 2^{34}$	3151
Low T	19	$1.05 \cdot 2^{35}$	$1.28 \cdot 2^{29}$	$1.74 \cdot 2^{32}$	$1.10 \cdot 2^{36}$	$1.21 \cdot 2^{31}$	$1.31 \cdot 2^{33}$	3901
Low D	15	$1.73 \cdot 2^{35}$	$1.34 \cdot 2^{30}$	$1.13 \cdot 2^{34}$	$1.17 \cdot 2^{36}$	$1.23 \cdot 2^{25}$	$1.48 \cdot 2^{28}$	4278
521-bit modulus								
RNSL	–	–	–	$1.81 \cdot 2^{42}$	–	$1.91 \cdot 2^{39}$	–	4727
Low W	22	$1.85 \cdot 2^{37}$	$1.59 \cdot 2^{30}$	$1.82 \cdot 2^{35}$	$1.98 \cdot 2^{38}$	$1.99 \cdot 2^{33}$	$1.09 \cdot 2^{36}$	4258
Low T	20	$1.45 \cdot 2^{35}$	$1.49 \cdot 2^{30}$	$1.00 \cdot 2^{34}$	$1.57 \cdot 2^{36}$	$1.40 \cdot 2^{32}$	$1.54 \cdot 2^{34}$	5273
Low D	15	$1.10 \cdot 2^{37}$	$1.70 \cdot 2^{31}$	$1.43 \cdot 2^{35}$	$1.48 \cdot 2^{37}$	$1.13 \cdot 2^{26}$	$1.27 \cdot 2^{29}$	5789

Table 5.1: Uncorrected resource estimates for Shor's full algorithm to compute the ECDLP from [HJN+20]. RNSL results are taken from [RNSL17]. Rows labeled Low W/Low T/ Low D show estimates for circuits minimising width, T gate count and total depth, respectively. All of these figures are *logical* operations/qubits.

5.5.1 Methods

Distance As in [GE21], I assume that the error rate of the logical qubits, with distance d, scales as $0.1 \cdot (100\epsilon)^{\frac{d+1}{2}}$. An upper bound on the total error is the logical DW-cost times the error on each logical qubit, so I set the distance d to ensure the total error is 0.01.

With a distance of d, each logical qubit requires d^2 physical qubits [HFDM12].

CNOT Gates. With lattice surgery, a `CNOT` gate requires at least $2d^2 + 4d$ extra physical qubits [HFDM12]. Potentially Eve needs far more to route the `CNOT` gates, an issue I will return to. I compute an average `CNOT` rate by taking the total number of `CNOT` gates and dividing by the depth. This is the average number of `CNOT` gates in every surface code cycle; I multiply this by 4 (an arbitrary small number) as an attempt to estimate the maximum number needed at any one time.

The `CNOT` takes $3d$ surface code cycles, so I set its depth to equal 3 in Q#.

Single-Qubit Clifford Gates. An H gate requires no extra qubit, but has depth $2d$ [HFDM12]. S gates can be done in depth $1.5d$ using [Gid23], though

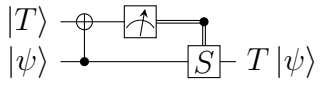


Figure 5.7: T gate teleportation.

this uses an extra qubit. Unfortunately, Q# does not distinguish between single-qubit Clifford gates. Thus, I created a wrapper function for H, S, X, and Z gates. When estimating:

1. Z and X gates do nothing;
2. S gates are applied 3 times.
3. H gates are applied 4 times.

In this way, I can set the depth of each single-qubit Clifford to be 0.5 and the final depth will be proportional to the number of cycles times d.

This loses some information: the extra space for the S gate. However, the only time an S gate is used is in the `AND` or in T teleportation, and both already have extra qubits for the `CNOT`s, so I assume these S gates need no extras.

T gates. In a surface code, applying T gates is far more challenging. The assumed method is to prepare noisy physical states $|T\rangle := |0\rangle + e^{i\tau/8}|1\rangle$ (known as *magic states*), encode these into the code, then distill many of these states into a single high-fidelity logical state of the same form. Once this is done, a circuit like Figure 5.7, known as *teleportation*, will apply this to the target qubit.

A more surface-code-friendly implementation in a Q# algorithm would do this directly, and disallow T gates altogether. In fact, one could be even more sophisticated, since there are specialised distillations for Toffoli gates in particular [GF19b]. Instead, I will take the catalysed T factory from [GF19a], which produces 2 states of the form $|T\rangle$ with $72d^2$ physical qubits and in depth $6.5d$.

Given a state $|T\rangle$, the circuit to inject this into the rest of the algorithm requires one `CNOT` and a conditional S gate, so I assume this has depth $3.75d$ (on average) and requires an extra $2d^2 + 4d$ physical qubits.

The actual resource consumption gets considerably more complicated. Each of the $|T\rangle$-state "factories" can produce $|T\rangle$ states at a rate of 2 every $6.5d$ cycles, and after that they can be stored until use (since each state needs d^2 physical qubits, many fewer than the factory itself). This leaves us a with a "token bucket" kind of problem: how large of a bucket of $|T\rangle$ states does Eve need to ensure that she always has enough when she needs them, and that she has enough $|T\rangle$-state factories to refill it in time?

Precisely answering this question goes beyond what Q# can do currently. Instead, I do the following. I assign depth $4.5d$ to each T, since that is the requirement to teleport the T gate onto the circuit and perform the S gate (half the time the S gate is unnecessary, but if many T gates are done in parallel, with high probability at least one requires the S gate). From the estimated depth, I divide the total depth by the number of T gates and by 6.5, to estimate the number of $|T\rangle$-state factories needed. Then I multiply that by 4 (the same arbitrary factor as for the `CNOT` gates) to guess at the size of the bucket of $|T\rangle$ states.

For a low-depth parameterisation, I know that the addition circuit performs at least n parallel Toffoli gates, each of which uses up to 3 T gates in parallel, so I take a minimum of $3n$ as the $|T\rangle$ state bucket size for this case.

Since Eve repeats the same computations many times in Shor's algorithm, there is no single moment that needs an exceptional number of T gates; they are fairly evenly distributed. Thus, this estimate of the number of factories is reasonable: if Eve had an infinite-sized bucket, there might be a brief warm-up period to fill the bucket but after that the quantum computer should not need to wait for more $|T\rangle$ states. In contrast, the bucket size is the real limiting factor, and might need to be much larger than I estimate. Still, this should be a reasonable first pass.

Given these estimates, I multiply the bucket size by $d^2 + (2d^2 + 4d)$ (to both store and apply the states), and the factory count by $72d^2$, and add these results to the estimate of the extra number of physical qubits.

5.5.2 Results

Table 5.2 shows the corrected results, including the changes to account for surface code costs. The depths are almost unchanged from Table 5.1, despite accounting for the Q# depth-width bug, but this is because the two tables measure different things. In [HJN+20] we counted every gate as depth 1, whereas Table 5.2 used the rules from the last section (e.g., all Pauli-X and -Z gates have depth 0).

Circuit	Window Size	`CNOT` gates		T gates			Depth (cycles)	Logical Qubits
		Total	Rate	Total	Factories	Buckets		
256-bit modulus								
Low W	15	$1.76 \cdot 2^{33}$	0.7	$1.02 \cdot 2^{33}$	1	4	$1.78 \cdot 2^{35}$	2106
Low T	19	$1.85 \cdot 2^{35}$	2.3	$1.08 \cdot 2^{31}$	1	4	$1.16 \cdot 2^{36}$	2633
Low D	12	$1.29 \cdot 2^{33}$	35.7	$1.67 \cdot 2^{32}$	51	768	$1.73 \cdot 2^{28}$	2868
384-bit modulus								
Low W	15	$1.61 \cdot 2^{35}$	0.8	$1.21 \cdot 2^{34}$	1	4	$1.65 \cdot 2^{37}$	3132
Low T	19	$1.13 \cdot 2^{37}$	3.1	$1.74 \cdot 2^{32}$	1	4	$1.10 \cdot 2^{37}$	3913
Low D	12	$1.10 \cdot 2^{35}$	53.7	$1.41 \cdot 2^{34}$	75	1152	$1.96 \cdot 2^{29}$	4275
521-bit modulus								
Low W	16	$1.04 \cdot 2^{37}$	0.7	$1.81 \cdot 2^{36}$	1	4	$1.01 \cdot 2^{39}$	4230
Low T	20	$1.88 \cdot 2^{38}$	4.0	$1.00 \cdot 2^{34}$	1	4	$1.42 \cdot 2^{38}$	5289
Low D	13	$1.33 \cdot 2^{36}$	68.2	$1.65 \cdot 2^{35}$	92	1563	$1.87 \cdot 2^{30}$	5789

Table 5.2: Costs of Shor's algorithm to attack ECC in a surface code architecture. The "rate" of `CNOT` gates is the number of gates divided by the (code distance $\times$ total cycles); see Section 5.5.1 for the computation of factories and buckets. The depth is the total number of cycles needed, divided by the code distance. "Logical qubits" includes only the qubits for the algorithm, and not the extra qubits for `CNOT` and T gates

Using these results, I can make the same physical assumptions as [GE21] and estimate the total physical resources, in Table 5.3, and directly compare the results to [GE21].

Assuming 256-bit curves match 2048-bit RSA, and similarly 384-bit curves match 8192-bit RSA and 521-bit curves match 16384-bit RSA, we can see that elliptic curves are noticeably easier to attack than RSA, especially at higher security levels. Ultimately, RSA uses much bigger integers so basic arithmetic operations are that much more expensive. In fact, this matching between RSA and ECC is based on classical attacks, and because the classical attacks against RSA scale better than those against ECC (subexponential versus fully exponential), the size of

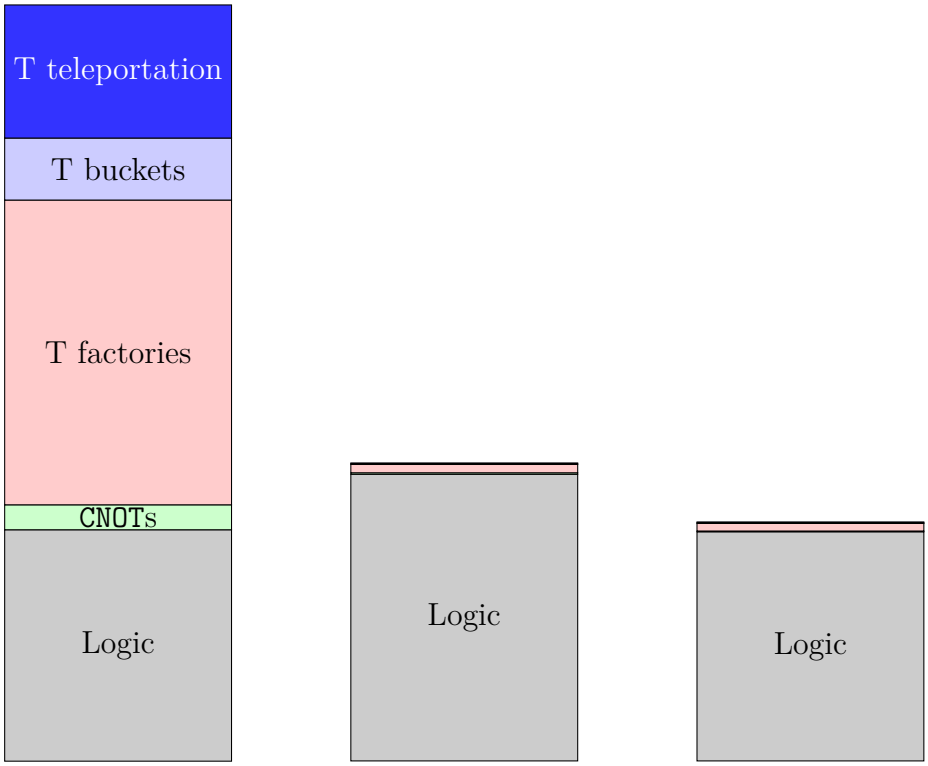
Circuit	Code Distance	Total Cycles	Physical Qubits	Runtime
256-bit modulus				
Low W	30	$1.67 \cdot 2^{40}$	$1.89 \cdot 2^{20}$	21 d
Low T	30	$1.09 \cdot 2^{41}$	$1.17 \cdot 2^{21}$	28 d
Low D	26	$1.41 \cdot 2^{34}$	$1.49 \cdot 2^{22}$	6.7 h
384-bit modulus				
Low W	31	$1.60 \cdot 2^{42}$	$1.48 \cdot 2^{21}$	81 d
Low T	31	$1.07 \cdot 2^{42}$	$1.84 \cdot 2^{21}$	54 d
Low D	27	$1.65 \cdot 2^{35}$	$1.20 \cdot 2^{23}$	15.8 h
521-bit modulus				
Low W	32	$1.01 \cdot 2^{44}$	$1.89 \cdot 2^{22}$	205 d
Low T	32	$1.42 \cdot 2^{43}$	$1.32 \cdot 2^{22}$	145 d
Low D	28	$1.63 \cdot 2^{36}$	$1.67 \cdot 2^{23}$	31.2 h
RSA (from [GE21])				
2048	27	$1.55 \cdot 2^{34}$	$1.19 \cdot 2^{24}$	7.4 h
3072	29	$1.38 \cdot 2^{35}$	$1.13 \cdot 2^{25}$	13.2 h
4096	31	$1.21 \cdot 2^{36}$	$1.64 \cdot 2^{25}$	23.2 h
8192	33	$1.19 \cdot 2^{38}$	$1.04 \cdot 2^{27}$	90.5 h
16384	33	$1.51 \cdot 2^{40}$	$1.01 \cdot 2^{28}$	460.5 h

Table 5.3: Surface code resources to break ECC and RSA. I took the assumptions from [GE21], of a physical qubit error rate of 0.001 and a cycle time of 1 microsecond.

the RSA modulus grows faster to match the same *classical* security level, meaning the quantum security grows faster for RSA.

Take these comparisons with some skepticism, however. [GE21] do a more thorough accounting of surface code resources, including physical layout and the speed of the classical computer (which must handle measurement feedback for, e.g., `AND` uncomputation). Then again, my estimates here are testable thanks to Q#, and in another project [JH22] I improved the Q# simulator and tested that this implementation works correctly, at least for curves up to 10 bits. The same project revealed an inconsequential bug in the data structures in [GE21].

By these estimates, the strategy optimising for logical qubit count *also* minimises physical qubit count. We can see that one T-factory is sufficient for both the low-width and low-T implementation, so there is actually little to gain by optimising T-count at this level. The low-depth implementation has much lower code distance, so the physical qubit requirements for just the logical qubits are almost the same



(a) Low Depth **(b)** Low T **(c)** Low Width

Figure 5.8: Number of physical qubits for different purposes in the attack against ECC-256.

as the low-width (about 2% higher), but to achieve the low depth it needs a huge array of T-factories and space to store the distilled T-states, shown in Figure 5.8. I chose window sizes for the low-width strategy based on the minimum number of physical qubits. Even though larger windows use (slightly) more logical qubits, a larger window means fewer operations and thus a smaller code.

Without a full layout, these numbers are not perfect. Specifically, the overhead for `CNOT` routing seems too small. My calculations assume the low-width stategy has one spare region that all `CNOT` gates will share. I do not know of a method to actually route this around a two-dimensional layout. The low-width strategy only needs 1 `CNOT` per $3d$ cycles, so instead a snaking route of spare qubits (adding roughly 50% overhead, see Figure 5.9) should be enough for all-to-all exclusive `CNOT` connectivity. The low-T strategy needs a few more `CNOT` gates, so $4\times$(# logical qubits) (as in [BKS22]) might be enough. The low-depth strategy would need more sophisticated planning. In the end, `CNOT` routing actually exaggerates the low-width advantage.

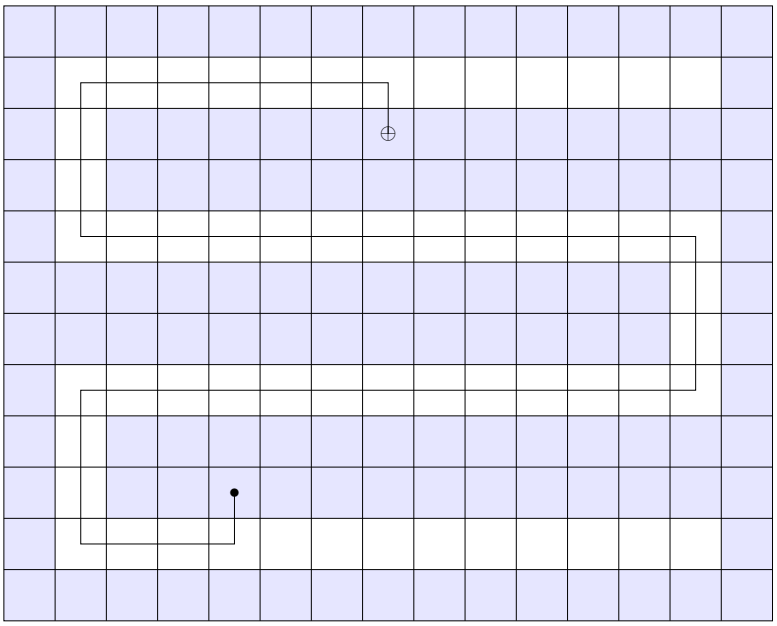


Figure 5.9: A layout allowing all-to-all `CNOT` connectivity with lattice surgery. Logical qubits are in light blue, and "spare" qubits for routing `CNOT`s are in white. An example `CNOT` gate is shown. This layout creates extreme contention for the `CNOT` routing qubits, and can likely only provide one `CNOT` at a time.

A final point is the actual runtime. The low-width implementation, at an optimistic 1 micro-second cycle, still takes 3 weeks to break ECC-256, compared to 7 hours for the low-depth. Eve might decide that a 70× speed-up is worth a 4× hardware increase. As an example, to attack Bitcoin, Eve would need to solve the discrete log between when a transaction is posted and when that transaction is added to the chain, about 10 minutes [HC21], so she would want an even lower-depth implementation or faster hardware.

A common sentiment in quantum circuit design is to minimise the total qubit count (e.g., [HRS17, WWL22]), since today's devices are qubit-limited. I have been skeptical of this philosophy. It is definitely incorrect for QAA (see our arguments in [JNRV20]), and in general since today's devices are also fidelity-limited *and* quantum circuit design is typically done at the logical level, the translation into physical qubits might favour larger circuits that are faster or use fewer T gates. However, here I have proved myself wrong.

6
Attacks on CSIDH

Contributions. This chapter is based off work in [CSCDJRH21]. In this work I provided most of the quantum security analysis, which my co-authors used to develop a secure implementation. Their idea to save costs was to to scale the exponents down; I provided arguments why this should still be quantum-secure. This chapter gives these quantum security analyses.

Commutative Super-singular Isogeny Diffie-Helman (CSIDH, pronounced like "sea-side") is a proposed post-quantum key encapsulation [CLM+18]. It uses isogenies between supersingular elliptic curves to perform a key exchange, though precise definitions of any of those terms will not matter for this chapter.

The mathematics of CSIDH allow it to form the backbone of many cryptographic constructions, but also makes it vulnerable. Uniquely among post-quantum key encapsulations, there is a known quantum attack with subexponential cost. This makes the security controversial: on the one hand, it is easier for quantum computers to attack than any other scheme; on the other hand, RSA suffers from *classical* subexponential attacks, but is widely deployed and generally secure for large enough parameters.

CSIDH is parameterised by a large prime p. Two previous works [BS20, Pei20] claimed CSIDH with a 512-bit prime was insecure against quantum attack. Generally, these papers made conservative assumptions (i.e., pessimistic from the attacker's point of view). Our goal in [CSCDJRH21], from which this chapter is based, was to create a quantum-secure variant of CSIDH, hence why we made our assumptions more optimistic for the attacker. In this chapter I will give the more detailed cost analysis from [CSCDJRH21], which considers parallelism and resource limitations.

6.1 Background

6.1.1 Group Action Cryptography

CSIDH moves just barely away from the structure of quantum-insecure cryptography, which works directly over groups, and works instead with group *actions*. Here we have an abelian group G and some set S (hopefully with little helpful structure) such that for any $g \in G$ and $x \in S$, we can easily compute $g \star x$, in such a way that $h \star (g \star x) = (hg) \star x$ for any $h \in G$. We can construct a problem analogous to DLOG for this case:

Problem 6.1.1 (Vectorisation, [Smi18])**.** *Given a group G acting freely and transitively on a set S, and given $x, y \in S$, find $g \in G$ such that $g \star x = y$.*

"Free" means that if $g \star x = x$ for any x, then g is the identity; "transitive" means that for any $x, y \in S$, there exists some g such that $g \star x = y$.

You can see that DLOG is a vectorisation problem, where the set S is itself a group of some order N, and G is the group $(\mathbb{Z}/N\mathbb{Z})$. A key difference is that in DLOG, you can take two elements $x, y \in S$ and add them together such that $g \star (x + y) = g \star x + g \star y$. This addition is key to computing arbitrary multiples of the challenge points in the DLOG problem, which Shor's algorithm needs. However, arbitrary group actions might not have this property, so Shor's algorithm cannot generally solve vectorisation.

A free, transitive, and abelian group action allows some nice Diffie-Hellman-like cryptography ([CLM+18] has 416 citations as of writing; [BKP20, BKV19, FP21, MTT20] are a small sample of constructions from CSIDH) but since I focus on attacks, I will ignore the constructions.

6.1.2 CSIDH

CSIDH starts with a prime p such that $p + 1$ is divisible by many small primes $\ell_1, \ldots, \ell_n$. Recalling the definition of an elliptic curve from Chapter 5, we consider the set $\mathcal{E}$ of all elliptic curves defined over p up to isomorphism (e.g., by taking different coefficients for the equation, Equation (5.1), defining the curve). We then restrict to *supersingular* curves: those that have exactly $p + 1$ points over $\mathbb{F}_p$. I will denote this set by $\mathcal{E}_{ss}$.

For each elliptic curve $E \in \mathcal{E}_{ss}$, and each prime ℓ_i, there is a special group homomorphism called an *isogeny*[1], mapping points in E to some other elliptic curve E'. We can consider each isogeny as a function $\sigma_{\ell_i} : \mathcal{E}_{ss} \to \mathcal{E}_{ss}$, indexed by these primes ℓ_i. To compute $\sigma_{\ell_i}(E)$, we find the isogeny corresponding to ℓ_i and evaluate what curve this isogeny maps to. This computation is relatively fast and straightforward.

A surprising fact is that, for supersingular curves over $\mathbb{F}_p$, these maps commute: given any ℓ_i and ℓ_j, $\sigma_{\ell_i} \circ \sigma_{\ell_j} = \sigma_{\ell_j} \circ \sigma_{\ell_i}$. A second fact is that there are two isogenies

[1] Generally there is an isogeny for each subgroup of E; for CSIDH in particular there are two subgroups defined over $\mathbb{F}_p$ for each ℓ_i.

defined by each ℓ_i, which are inverses of each other. Thus, we can form a free group out of these isogenies as a subgroup of all bijections on $\mathcal{E}_{ss}$. We can even represent any element in this group as a vector in $\mathbb{Z}^n$, i.e.,

$$\vec{a} = (a_1, a_2, \ldots, a_n) \tag{6.1}$$

which is interpreted as the function

$$\sigma_{\ell_1}^{a_1} \circ \sigma_{\ell_2}^{a_2} \circ \cdots \circ \sigma_{\ell_n}^{a_n} \tag{6.2}$$

where

$$\sigma_{\ell_i}^{a_i} = \underbrace{\sigma_{\ell_i} \circ \sigma_{\ell_i} \circ \cdots \circ \sigma_{\ell_i}}_{a_i \text{ times}} \tag{6.3}$$

such that the function given by $\vec{a}_1 + \vec{a}_2$ is the same as the function for $\vec{a}_1$ composed with the function for $\vec{a}_2$.

The set $\mathcal{E}_{ss}$ is finite (the coefficients in each elliptic curve come from a finite set), in fact of size $O(\sqrt{p})$. Thus, there must be infinitely many vectors $v\mathbb{Z}^n$ such that the isogeny defined by v is the identity. These will form a subgroup of $\mathbb{Z}^n$ (i.e, a lattice) and so the group we are actually interested in is the quotient $\mathbb{Z}^n/\langle v_1, \ldots, v_n \rangle$. We call this the *class group*.

This introduction will baffle any number theorist, since the class group is a far more fundamental object that only has this lattice quotient structure in the special case of CSIDH; however, this quotient group is more natural from the perspective of the quantum attack.

This gives us the required group action: the group is the class group, with supersingular elliptic curves as the set it acts on. Thus, the hard problem to solve is that Eve is given two elliptic curves E_0 (a starting curve, hence "0") and E_A (Alice's public key, hence "A"), and Eve needs to find the secret isogeny σ such that $\sigma \star E_0 = E_A$.

6.2 The Cost of Kuperberg's Algorithm

Kuperberg's algorithm [Kup13] solves the "abelian hidden shift problem", a slight generalisation of vectorisation. I will present it specifically for vectorisation, including some optimisations from [Pei20] and our work in [CSCDJRH21].

Recall that the vectorisation challenge is two points x and y in the set S, and we seek a secret $s \in G$ such that $s \star x = y$. For now I assume G is a cyclic group of order N, so I regard s as an integer in $\{0, \ldots, N-1\}$.

6.2.1 Phase Vectors

The data structure Eve will use is a *phase vector*, defined as follows:

Definition 6.2.1 (Phase Vector)**.** *A* phase vector *is a tuple* $(|\psi\rangle, J, L, A, H)$*, such that:*

1. *The state* $|\psi\rangle$ *is*
$$|\psi\rangle = \sum_{j=0}^{L-1} |j\rangle\,|J(j)\rangle\,\zeta_s^{J(j)} \tag{6.4}$$
where j *and* $J(j)$ *are represented by binary strings, and* $\zeta_s := e^{is/N}$ *(*s *is the secret).*

2. L *is an integer called the* length

3. $J : \{0, \ldots, L-1\} \to \{0, \ldots, N-1\}$ *is the* phase function*, a function stored as a table in classical memory, such that* $J(i) = 0$ *for at least one* i*.*

4. A *is the* altitude*, such that* A *is the greatest common divisor of all outputs of* J*.*

5. H *is an integer called the* height*, such that* $H \geq H_0$ *where*
$$H_0 := \min_{k \in \mathbb{Z}} \left\{ \max_{i,j \leq L-1} \{|J(i) - J(j) + kN|\} \right\}. \tag{6.5}$$

The classical controller knows J*,* L*,* A*, and* H*.*

To visualise these, see Figure 6.1. A phase vector contains some information about the secret, but unless the phases are condensed (low height) or well-aligned (large altitude), Eve cannot recover any information. Kuperberg gave a method called *collimation* to trade length for improvements in height or altitude. This gives the high-level approach of Kuperberg's algorithm:

1. Eve uses the group action to create initial phase vectors, which are short and unhelpful;

2. Eve combines two phase vectors to make them longer and then collimates them to improve height and/or altitude. She repeats this (which needs many new input phase vectors) until she has a perfect phase vector: one such that $L = H/A$ and $J(j) = jA$ for all j.

3. A quantum Fourier transform on a perfect phase vector reveals about $\lg L$ bits of the secret (with some noise).

4. Eve repeats this entire process until she recovers all the bits.

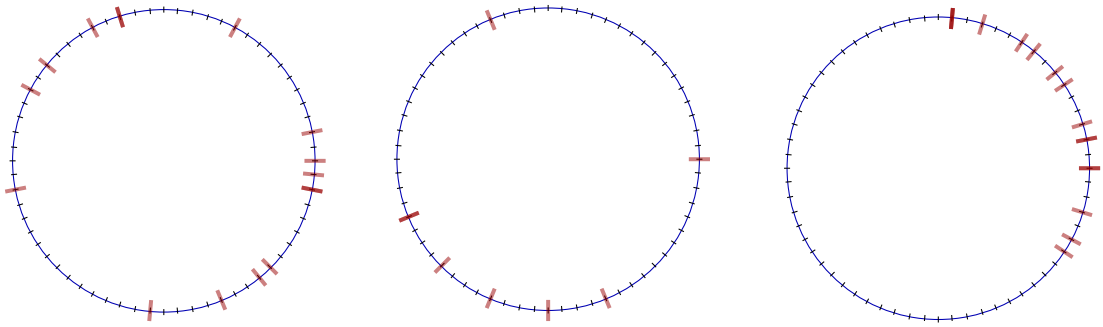
I will detail each of these steps and some optimisations.

Since the order of ζ_s divides N and Eve can ignore global phase, J is invariant modulo N and invariant under adding a constant to all components. This would make the altitude ill-defined except that I require $J(i) = 0$ for some i and $J(j) \leq N - 1$ for all j. This captures the notion of evenly-spaced phases.

6.2.2 Creating Phase Vectors.

To create phase vectors, Eve will need to compute the action of the class group in superposition, which means she needs to know the structure of the class group.

The class group structure can be pre-computed, shown in Algorithm 1. Step 1 is polynomial time with a quantum computer [Kit96], though Step 3 is a lattice problem which is hard in general. However, the dimension of the lattice is n, which is relatively small. For CSIDH-512, the lattice has size 74, and today's lattice



(a) A bad phase vector (phases are disordered). It has length 16, height 51 and altitude 1. I generated the phases randomly, but this still left a gap of size 13 between two phases (in the lower left of the circle), hence the height is only 51. Of course, as the length increases, the expected size of such gaps will shrink.

(b) A good phase vector (phases are evenly spaced). It has length 8, height 44 and altitude 4. Similar to Figure 6.1a, I generated up to height 64 but a random "gap" of size 20 appeared. Because the altitude is greater than 1, the phases are spaced out evenly.

(c) A good phase vector (phases are in a narrow range). It has length 16, height 24 and altitude 1. Because the height is less than 64, the phases are restricted one part of the circle. I selected phases randomly, and there is some overlap: states with the same phase

Figure 6.1: Representation of phase vectors in a group of size $N = 64$. The larger red ticks are the phase multipliers. Since the phase multipliers are considered modulo N, I represented them on a circle, where orientation of the circle does not matter because the state is invariant up to global phase.

Algorithm 1 CSIDH Group action precomputation

Input: A CSIDH prime $p = \ell_1 \dots \ell_n - 1$
Output: A short basis of the lattice L such that $\mathbb{Z}^n/L$ is the class group for p

1: Use a variant of Shor's algorithm to find a basis B of the lattice L
2: Find a generator g of the class group, given L
3: Use a lattice solver to find a short basis B' from B
4: **return** B'

solving records (set by academic labs, not well-funded nation-state adversaries) are around dimension 150-180 [SG10, DSvW21].

Eve will need a uniform superposition of group elements to create phase vectors. In CSIDH-512, they use exponents from -5 to 5 to create a nearly uniform distribution. A *nearly* uniform superposition may not be good enough for Kuperberg's algorithm (Section 6.3.2 shows many problems with it). Eve needs a way to sample uniformly from the group.

For cyclic groups, she can pick a generator $g \in G$ and create a superposition $\sum_{x=0}^{N-1} |x\rangle$ and compute $|xg\rangle$. In CSIDH, a generator g will be some vector in $\mathbb{Z}^n$, so

$N \cdot g$ will have exponentially large components. Since the size of the components (either the maximum or the sum) governs the cost of computing isogenies, this vector must be reduced in the lattice.

This all leads to Algorithm 2, an algorithm to create new phase vectors.

Subgroups

Suppose the class group G can be written as $G = C_{n_1} \times C_{n_2} \times \cdots \times C_{n_t}$, with each C_{n_i} as a prime-order cyclic group. Ideally, Eve could take a Pohlig-Hellman approach and attack each subgroup separately. Unfortunately (or fortunately, from the perspective of the user), this may not be possible for Kuperberg's algorithm. If I write Alice's secret s in this format, as $s = (s_1, \ldots, s_t)$, then if Eve tries to create a superposition over just one subgroup, she gets

$$\sum_{x \in C_{p_1}} |(x, 0, \ldots, 0)\rangle \left(|0\rangle \left|(x, 0, \ldots, 0) \star E_0\right\rangle + |1\rangle \left|(x, 0, \ldots, 0) \star E_A\right\rangle\right) \tag{6.6}$$

When she measures the final state, she gets some E_k such that $(x, 0, \ldots, 0) \star E_0 = E_k$ or $(x, 0, \ldots, 0) \star E_A = (x + s_1, s_1, \ldots, s_t) \star E_0 = E_k$. These *cannot* be equal unless $s_2, \ldots, s_t$ are all 0.

More generally, she cannot create the proper superposition unless she has some overlap between applying the exponent vector to E_0 and E_A, and this will not happen unless she either has a superposition over all of G or she matches part of the secret. If she can match part of the secret, then she has already solved or guessed the secret!

In the usual Pohlig-Hellman attack against discrete log [PH78], there is a group homomorphism (exponentiation by a cofactor) to a smaller subgroup. Here, Eve would need an efficient function on the set of elliptic curves that that would map them to the image of a subgroup of the class group. The set of supersingular elliptic curves does not seem to have enough structure for such a function.

Thus, she must create a superposition over all of G, with each element represented perhaps as $(x_1, \ldots, x_t)$. Then once she measures and does a QFT, she must apply the QFT to the entire state $|x_1\rangle |x_2\rangle \ldots |x_t\rangle$. This gives her something like

Algorithm 2 Creating phase vectors

Input: A CSIDH prime $p = \ell_1 \dots \ell_n - 1$, a short basis B of a lattice L such that the class group is isomorphic to $\mathbb{Z}^n/L$, a generator $g \in \mathbb{Z}^n$ of the class group.

Output: A phase vector

1: Create a uniform superposition from 0 to $N-1$ (where N is the order of the class group)

Current state: $\sum_{x=0}^{N-1} |x\rangle$

2: Multiply the first register by the generator $g = (g_1, \dots, g_n)$

Current state: $\sum_{x=0}^{N-1} |x\rangle |xg_1\rangle \dots |xg_n\rangle$

3: Use Babai's algorithm [Bab86] to find a short vector $v = (v_1, \dots, v_n)$ such that $g - v$ is in the lattice L

Current state: $\sum_{x=0}^{N-1} |x\rangle |xg_1\rangle \dots |xg_n\rangle |v_1\rangle \dots |v_n\rangle$

4: Initialise a qubit in $|0\rangle + |1\rangle$

Current state: $\sum_{x=0}^{N-1} |x\rangle |xg_1\rangle \dots |xg_n\rangle |v_1\rangle \dots |v_n\rangle (|0\rangle + |1\rangle)$

5: Use an isogeny function (e.g., [BLMP19]) to compute $\sigma_1^{v_1} \dots \sigma_n^{v_n}(E_0)$, controlled by the first qubit equalling 0

Current state: $\sum_{x=0}^{N-1} |x\rangle |xg_1\rangle \dots |xg_n\rangle |v_1\rangle \dots |v_n\rangle (|0\rangle |v \star E_0\rangle + |1\rangle |0\rangle)$

6: Use an isogeny function to compute $v \star E_0 = \sigma_1^{v_1} \dots \sigma_n^{v_n}(E_A)$, controlled by the first qubit equalling 1

Current state: $\sum_{x=0}^{N-1} |x\rangle |xg_1\rangle \dots |xg_n\rangle |v_1\rangle \dots |v_n\rangle (|0\rangle |v \star E_0\rangle + |1\rangle |v \star E_A\rangle)$

7: Uncompute the intermediate computations

Current state: $\sum_{x=0}^{N-1} |x\rangle (|0\rangle |v \star E_0\rangle + |1\rangle |v \star E_A\rangle)$

8: Measure the last register and discard the result

Current state: $|x\rangle |0\rangle + |x+s\rangle |1\rangle$

9: Apply a QFT to the first register

Current state: $\sum_{k=0}^{N-1} e^{i\tau \frac{kx}{N}} |k\rangle |0\rangle + e^{i\tau \frac{k(x+s)}{N}} |k\rangle |1\rangle$

10: Measure the first register; let the result be k

Current state: $e^{i\tau \frac{kx}{N}} |0\rangle + e^{i\tau \frac{k(x+s)}{N}} |1\rangle$

11: (Ignore global phase)

Current state: $|0\rangle + e^{i\tau \frac{ks}{N}} |1\rangle = |0\rangle + \zeta_s^k |1\rangle$

12: Set J such that $J(0) = 0$ and $J(1) = k$

13: Write $J(1)$ to another register, controlled by the qubit

Current state: $|\psi\rangle = |0\rangle |0\rangle + \zeta_s^k |1\rangle |k\rangle = |0\rangle |J(0)\rangle + \zeta_s^k |1\rangle |J(1)\rangle$

14: **return** $(|\psi\rangle, J, L = 2, A = k, H = k)$

a "multi-phase" vector, e.g.,

$$|0\rangle + \zeta_{s_1}^{k_1} \dots \zeta_{s_t}^{k_t} |1\rangle \tag{6.7}$$

for classically-known phases k_i. In later sections I will show how this affects other aspects of the algorithm.

Cost Analysis

First, I ignore the costs of the precomputations, since they need only be done once, and the results are so useful for cryptographic protocols that Eve might have honest parties do these computations for her. For example, today we know the class group and a short basis for CSIDH-512, thanks to a work creating more efficient signatures [BKV19].

Next, I ignore the costs of the multiplication of g by x (Step 2), and the costs of Babai's algorithm (Step 3), since both will be insignificant compared to the isogeny computations. Bonnetain and Schrottenloher [BS20] reach this conclusion by analysing this step in detail.

The main cost is the isogeny computation, Steps 5 and 6. [BLMP19] provide a constant-time classical implementation for CSIDH-512 using 2^{40} non-linear classical gates (e.g., `AND`, `NAND`, etc.), arguing that a quantum implementation should be at least this expensive and likely much more. There are a few reasons this may not be true:

- Eve can window the computation (as in Section 5.3). Because the group action commutes, she can select the first k exponents and precompute all possible isogenies. This has a precomputation cost of $(2C+1)^k$ isogenies, where C is the maximum value the exponent might have. She can then use the first k registers for a QRACM lookup of the pre-computed elliptic curve. The cost savings should be logarithmic: the isogeny computation cost is approximately linear in the number of exponents, so when $k = O(\log n)$ the cost savings will match the cost of the QRACM access.

Even in a far-future regime where we assign nearly equal costs to classical and quantum operations, precomputing all the isogenies is a fixed cost, and Eve can re-use access to that table for all future isogeny computations.

- Quantum arithmetic could be made cheaper than classical arithmetic. Of all things that will likely be optimised by the time this algorithm might be implemented, quantum arithmetic is a front-runner, since it is so ubiquitous.

Thus, in [CSCDJRH21] we took the 2^{40} number from [BLMP19] as the total number of `AND` gates to compute an isogeny for CSIDH-512, assumed those were the only gates, and estimated that the oracle cost will grow quadratically with the bit-length of the prime.

However, quantum computing has a memory management issue because of uncomputing. Eve does not want to allocate an extra 2^{40} qubits to do each `AND` out-of-place. We translated this into a *pebbling* game, specifically the quantum "spooky" variant due to Gidney [Gid19b] and expanded in [KSS21]. We followed straightforward methods from these references, so the full details are only in the Appendix D.1.

The overhead ends up quite small: for example, the overhead for CSIDH-512 is only a factor of about 30.

However, the CSIDH group action is more complicated than a straight-line program. Specifically, each "pebble" is, in the most general form, the full state of computation. Thus, we must estimate how large this is. During the classical arithmetic, the memory cost is $3b+3b\lg b$, where $b \approx \sqrt{\ell_{max}} \approx \sqrt{\frac{\log p}{\log\log p}}$ is the degree of the largest isogeny computed. We thus assumed that the "state" of computation for a quantum implementation would have the same size.

This gave our estimate of the group action cost; however, we chose to ignore even this cost. It seems like the most imprecise estimate, and at the scales we considered this cost was negligible anyway.

6.2.3 Collimation

Collimation is the term Kuperberg coined to describe the process of improving a phase vector. It decreases height or increases altitude of a phase vector (good), but reduces the length (bad). Thus, Eve first needs a method to lengthen phase vectors.

Lengthening Phase Vectors

The output of Algorithm 2 always gives Eve a phase vector of length 2, where $J(0) = 0$ and $J(1) = k$. The altitude and height are also k (since $\gcd(0, k) = k$). To create phase vectors of a greater length, she simply concatenates, as in Algorithm 3 and Figure 6.2.

Algorithm 3 Enlarging phase vectors

Input: Two phase vectors $(|\psi_1\rangle, J_1, 2^{\ell_1}, A_1, H_1)$ and $(|\psi_2\rangle, J_2, 2^{\ell_2}, A_2, H_2)$
Output: A single, larger phase vector
1: Set $H = \max\{H_1 + H_2, N\}$
2: Set $A = \gcd(A_1, A_2)$
3: Set $L = 2^{\ell_1} \cdot 2^{\ell_2}$
4: Set J to an empty list of size L
5: **for** i in 0 to $2^{\ell_1} - 1$ **do**
6: **for** j in 0 to $2^{\ell_2} - 1$ **do**
7: Set $J[i \cdot 2^{\ell_2} + j] = J_1(i) + J_2(j)$
8: **end for**
9: **end for**
10: **return** $(|\psi_1\rangle \otimes |\psi_2\rangle, L, J, A, H)$

To see why this works, notice that when $L_1 = 2^{\ell_1}$ and $L_2 = 2^{\ell_2}$,

$$|\psi_1\rangle \otimes |\psi_2\rangle = \sum_{j_1=0}^{L_1-1} \sum_{j_2=0}^{L_2-1} |j_1\rangle |j_2\rangle \zeta_s^{J_1(j_1)} \zeta_s^{J_2(j_2)} \tag{6.8}$$

$$= \sum_{j_1=0}^{L_1-1} \sum_{j_2=0}^{L_2-1} \left|j_1 2^{\ell_2} + j_2\right\rangle \zeta_s^{J_1(j_1)+J_2(j_2)}, \tag{6.9}$$

exactly matching the J, H, A, and L that Eve calculated for the new phase vector. Thus, the length increases multiplicatively, but the height also increases and the altitude also decreases.

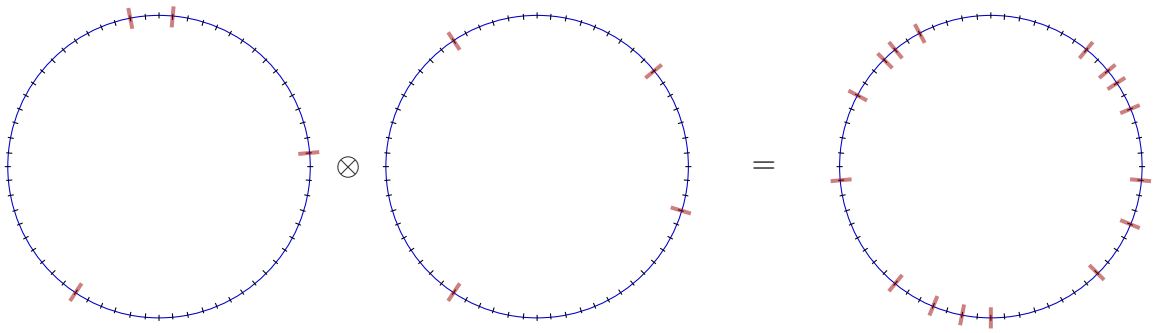


Figure 6.2: Combining two phase vectors of length 4 into one of length 16. Again, both input phase vectors were generated randomly, so they have heights 40 and 44, respectively, but the height of the product increased to 52. Both started with altitude 1, so the output also has altitude 1.

Collimation

The algorithm for collimation is in Algorithm 5 and illustrated in Figure 6.3. Compared to Kuperberg's method and Peikert's refinements, in [CSCDJRH21] we kept a register with the phase multiplier $J(j)$ itself, so that we would not need to use a QRACM look-up to obtain it or remove it. This means each collimation needs only one forward QRACM computation and one QRACM *uncomputation*. Appendix C.3 details the uncomputation process, from [BGM+19], which is cheaper than a forward computation. The previous methods used 3 forward QRACM look-ups.

In Algorithm 5, Eve iterated through all phases to build the phase function of the collimated phase vector. This takes $O(L)$ time. In Algorithm 5, with r inputs, this naive approach would take longer, so Step 9 references a new problem.

To briefly explain the method of collimation, measuring the quotient Q of the phase multiplier will remove many phases from superposition, but those phases that are left will all be in the range from QS to $(Q+1)S$. Since global phase is irrelevant, this effectively reduces the total height to S. In this way, Eve can trade length for height.

Since only some indices j remain out of the original L, Eve needs to permute them to "condense" them. This is the purpose of Algorithm 4 and all the steps from 9 onward, and the only reason for the QRACM look-up. Without condensing

Algorithm 4 Collimation Post-Processing

Input: r phase vectors $(|\psi\rangle_i, J_i, L_i, A_i, H_i)$, a parameter S, a list $\mathcal{J}$ of r-tuples
Output: Parameters L', A', H' and functions
$J' : \{0, \ldots, L'-1\} \to \{0, \ldots, S\}$,
$\pi : \mathcal{J} \to \{0, \ldots, L'-1\}$
$\pi^{-1} : \{0, \ldots, L'-1\} \to \mathcal{J}$

1: Set J', π, and π^{-1} to empty dictionaries
2: Set $L' = |\mathcal{J}|$, $H_{max} = 0$, $H_{min} = S-1$, $A' = 0$
3: **for** $i = 0$ to $L'-1$ **do**
4: Set $\vec{j} = \mathcal{J}[i]$
5: Set $J'(i) = J(\vec{j}) \bmod S$
6: Set $\pi(\vec{j}) = i$ and $\pi^{-1}(i) = \vec{j}$
7: Set $H_{max} = \max\{H_{max}, J'(i)\}$
8: Set $H_{min} = \min\{H_{min}, J'(i)\}$
9: Set $A' = \gcd(A', J'(i) - J'(0))$
10: **end for**
11: Set $J'(i) = J'(i) - H_{min}$ for all i from 0 to $L'-1$
12: **return** L', A', $H_{max} - H_{min}$, J', π, and π^{-1}

the indices, the number of qubits in each phase vector would increase exponentially with the number of collimations.

Eve could have measured the remainder instead of the quotient. This would instead increase the altitude, since the difference between any two phases would be a multiple of S, so (again ignoring global phase) the gcd of all phases is at least S.

By repeatedly enlarging and collimating phase vectors, she can reduce the height to as low as she wants, or the altitude as high as she wants.

Collimation Permutation Problem

In Step 9 of Algorithm 5, Eve needs to rebuild the classical table for the phase function J that the collimated vector will have. In [CSCDJRH21] we called this the "collimation permutation problem":

Problem 6.2.1 (Collimation Permutation)**.** *Let L, H, S and N be integers such that $N \geq H \gg S \gg L$. Given an input of r sorted lists $J_1, \ldots, J_r$, each containing L uniformly random numbers from 0 to $H-1$, and an integer Q, list all r-tuples from $J_1 \times \cdots \times J_r$ such that their sum (modulo N) is in $\{QS, QS+1, \ldots, (Q+1)S-1\}$.*

Algorithm 5 Collimation (for height)

Input: r phase vectors $(|\psi\rangle_i, J_i, L_i, A_i, H_i)$, a parameter S
Output: A phase vector with smaller L and height of S

1: (The joint state of $|\psi_1\rangle \otimes \cdots \otimes |\psi_r\rangle$)
Current state: $\sum_{j_1=0}^{L_1-1} \cdots \sum_{j_r=0}^{L_r-1} \zeta_s^{J(j_1)+\cdots+J(j_r)} |j_1\rangle \ldots |j_r\rangle |J_1(j_1)\rangle |J_2(j_2)\rangle \ldots |J_r(j_r)\rangle$
2: (let $\vec{j} = (j_1, \ldots, j_r)$ and $J(\vec{j}) = J_1(j_1) + \cdots + J_r(j_r)$)
Current state: $\sum_{j_1=0}^{L_1-1} \cdots \sum_{j_r=0}^{L_r-1} \zeta_s^{J(\vec{j})} \left|\vec{j}\right\rangle |J_1(j_1)\rangle |J_2(j_2)\rangle \ldots |J_r(j_r)\rangle$
3: **for** $i = 2$ to r **do**
4: Add the ith phase register onto the first phase register
Current state: $\sum_{j_1=0}^{L_1-1} \cdots \sum_{j_r=0}^{L_r-1} \zeta_s^{J(\vec{j})} \left|\vec{j}\right\rangle |J_1(j_1) + \cdots + J_i(j_i)\rangle |J_2(j_2)\rangle \ldots |J_r(j_r)\rangle$
5: **end for**
6: (let $J(\vec{j}) = J_1(j_1) + \cdots + J_r(j_r)$)
Current state: $\sum_{j_1=0}^{L_1-1} \cdots \sum_{j_r=0}^{L_r-1} \zeta_s^{J(\vec{j})} \left|\vec{j}\right\rangle \left|J(\vec{j})\right\rangle |J_2(j_2)\rangle \ldots |J_r(j_r)\rangle$
7: Divide the first phase register in-place by S
Current state: $\sum_{j_1=0}^{L_1-1} \cdots \sum_{j_r=0}^{L_r-1} \zeta_s^{J(\vec{j})} \left|\vec{j}\right\rangle \left|J(\vec{j}) \bmod S\right\rangle \left|\left\lfloor \frac{J(\vec{j})}{S} \right\rfloor\right\rangle |J_2(j_2)\rangle \ldots |J_r(j_r)\rangle$
8: Measure the register with the quotient; let the result be Q
Current state: $\sum_{\vec{j}:\left\lfloor \frac{J(\vec{j})}{S} \right\rfloor = Q} \zeta_s^{J(\vec{j})} \left|\vec{j}\right\rangle \left|J(\vec{j}) \bmod S\right\rangle |J_2(j_2)\rangle \ldots |J_r(j_r)\rangle$
9: Solve the **collimation permutation problem** (Problem 6.2.1) to find $\mathcal{J} = \left\{\vec{j} \middle| \left\lfloor \frac{J(\vec{j})}{S} \right\rfloor = Q\right\}$.
10: Run Algorithm 4 with all r phase vectors, S, and $\mathcal{J}$ as inputs. Set L', A', H' and J', π, π^{-1} to the output.
11: Use a QRACM look-up into π, with $\vec{j}$ as the address
Current state: $\sum_{\vec{j}\in\mathcal{J}} \zeta_s^{J(\vec{j})} \left|\vec{j}\right\rangle \left|J(\vec{j}) \bmod S\right\rangle |J_2(j_2)\rangle \ldots |J_r(j_r)\rangle \left|\pi(\vec{j})\right\rangle$
12: Use a **phase QRACM uncomputation** (Appendix C.3) on all the red registers in the previous line
Current state: $\sum_{\vec{j}\in\mathcal{J}} \zeta_s^{J(\vec{j})} \left|J(\vec{j}) \bmod S\right\rangle \left|\pi(\vec{j})\right\rangle$
13: (relabel: $j' = \pi(\vec{j})$, $J'(j') = J(\vec{j}) + QS$)
Current state: $\sum_{j'=0}^{L'-1} \zeta_s^{J'(j')+QS} |J'(j')\rangle |j'\rangle$
14: (ignore global phase)
Current state: $|\psi\rangle = \sum_{j'=0}^{L'-1} \zeta_s^{J'(j')} |J'(j')\rangle |j'\rangle$
15: **return** $(|\psi\rangle, J', L', A', H')$

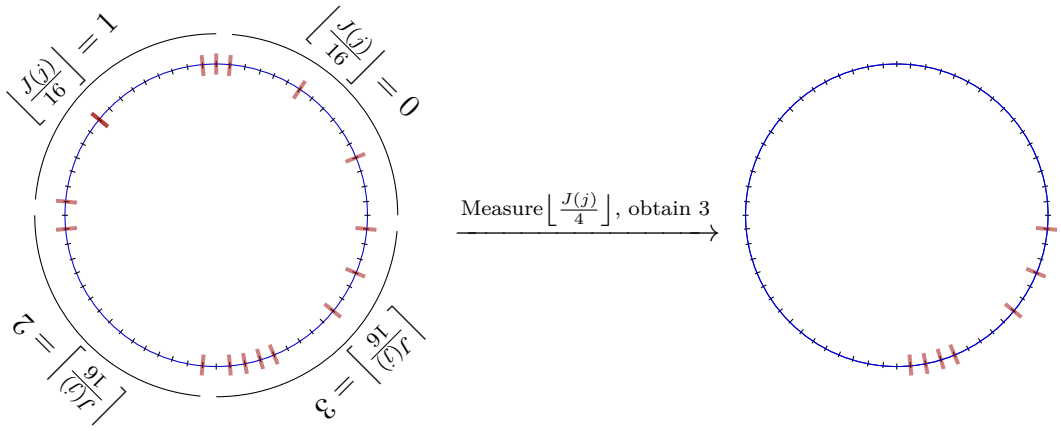


Figure 6.3: Collimating a phase vector of length 16, height 50, and altitude 1. Eve measures partial information about the phase (the quotient when dividing by 16 in this case). This truncates the phases to only those matching the measurement. This reduces the height to 15, but also reduces the length to 7. There is randomness in the resulting length: If Eve had measured 2 instead of 0, the resulting phase vector would have length only 2.

In Kuperberg's algorithm, Eve will always have $S \geq L^r$ except for the last tuple. This means that the set of all possible sums will be "sparse". The assumption of uniformity is justified for the initial phase vectors, which are uniform up to N, and since each collimation produces sums modulo some S, they will continue to be uniform.

I discuss three approaches below. We will find that solving this problem is the majority cost of Kuperberg's algorithm, so improvement in this area will have a big impact on both CSIDH and any other commutative group action protocol.

Sort-and-search. A simple classical approach is to divide r in half, and merge the first $\frac{r}{2}$ lists into one sorted list (assume r is even for ease of exposition), sorted by the value of the sum. This costs $O(L^{r/2} \log L)$ and needs $L^{r/2}$ memory for the final list. Then for each tuple $(j_{r/2+1}, \ldots, j_r)$ of the remaining lists, Eve computes its partial sum J, and looks for anything between $QS - J$ and $(Q+1)S - J - 1$ in her sorted list of partial sums. This costs $O(L^{r/2} \log L)$ as well, if she parallelises in such a way that she can amortise the classical memory access costs. This method can parallelise as fast as a sort can parallelise. I assume it parallelises perfectly, in fact.

In our numerical estimates, this ended up as the most cost-effective method. While it has a high memory requirement, depth limits force Eve to parallelise so much that the memory consumption of the larger sorted list is not the most costly component.

Because we expect a uniform distribution of phases, Eve could use faster heuristic sorting methods (e.g. [Neu98]) in time $O(L^{r/2})$ instead of $O(L^{r/2} \log L)$. We did not use these in the estimates in [CSCDJRH21]; re-running the estimates shows this would save about 1 bit.

QAA. Eve can apply QAA (Section 2.2.3) to the set of all R-tuples of elements. This would require $O(L^{\frac{r-1}{2}} \lg L)$ iterations, but each iteration needs $O(r + \lg L)$ QRACM accesses to read the elements of the lists. Thus, the total cost is $O(L^{\frac{r+1}{2}} \lg L)$, strictly more than the classical sort-and-search. Moreoever, it parallelises quite badly.

Subset-sum. There are more efficient algorithms for subset-sum, such as [BBSS20]. However, their speed-up is mostly due to partial guessing: Eve would guess the sum of the first $r/2$ components (for example), solve subset-sum for just these components, then solve the corresponding sum for the remaining components. However, since the space of possible sums is much larger than L^r, almost all guesses for the partial sum will fail. To put this another way, even guessing all possible values from QS to $(Q+1)S-1$ has cost S, which is much greater than the L^r, the cost of a brute force search. Thus, I did not see a method to adapt these techniques to our problem.

Subgroups

To briefly justify how this works when using a generating set of low-order elements of the class group group, recall Equation (6.7), where the phases on each address $|j\rangle$ would be a product like $\zeta_{s_1}^{J_1(j)} \dots \zeta_{s_n}^{J_n(j)}$, where J_i is the phase function for the ith component of the secret. That is, the the state in a phase vector will be

$$\sum_{j=0}^{L-1} \zeta_{s_1}^{J_1(j)} \dots \zeta_{s_n}^{J_n(j)} |j\rangle \,. \tag{6.10}$$

Eve can simply collimate one component at a time, and the other components of the secret will follow along. Once she collimates one component (say, the first) to have a height of 0, the resulting phase vector will have the form

$$\sum_{j=0}^{L-1} \zeta_{s_2}^{J_2(j)} \dots \zeta_{s_n}^{J_n(j)} |j\rangle . \tag{6.11}$$

Continuing in this way, Eve can eventually reduce it to the single-component case. The complexity of collimating all of the other components away will be given by the product of the sizes of all the components, which is just N, the order of the group. Hence, this collimation has almost exactly the same cost as in the cyclic case.

6.2.4 Measuring Phase Vectors

The perfect phase vector is one such that $L = H/A$ and $J(j) = jA$ for all j. Collimation reduces H and increases A, to produce one with $L \geq H/A$, after which Eve will execute Algorithm 6 to obtain some bits of the secret.

Briefly, Eve's goal is to have a phase vector where the phase function J is a multiple of the identity. The steps up to Step 7 serve only to rearrange the indices to make that true. Once that is done, the QFT reveals the secret almost immediately: you can see that because we sum over j'' in Step 14, the sum will be close to 0 except for values k such that

$$\frac{As}{N} - \frac{k}{K} \approx 0. \tag{6.12}$$

More precisely, following Peikert [Pei20], let $\theta_k = \frac{kA}{H} + \frac{sA}{N}$. We can see in Step 14 that the amplitude of each k is

$$\frac{A}{H} \frac{1 - e^{i\tau K\theta_k}}{1 - e^{i\tau\theta_k}} \equiv \frac{A}{H} \frac{\sin(\tau L\theta_k/2)}{\sin(\tau\theta_k/2)} \tag{6.13}$$

(ignoring global phase). For those values of k such that $|\theta_k| \leq 1/2K$, then we have $\tau K\theta_k/2 \leq \tau/4$, so $\sin(\tau K\theta_k/2) \geq 2K\theta_k$. Similarly, we know that $\sin(\tau\theta_k/2) \leq \tau\theta_k/2$, so we get a lower bound of

$$\frac{A}{H} \frac{2K\theta_k}{\tau\theta_k/2} = \frac{4}{\tau} \tag{6.14}$$

Algorithm 6 Measuring a phase vector

Input: A phase vector $(|\psi\rangle, J, L, A, H)$ such that $L \geq H/A$
Output: A guess for the secret value s

1: Set $K = \left\lfloor \frac{H}{A} \right\rfloor$
2: Set J', π, π^{-1} to empty dictionaries
3: Set π such that $J(\pi(0)), J(\pi(1)), \ldots, J(\pi(L-1))$ has the form of repeated blocks of $(0, A, 2A, \ldots, (A-1)\cdot K)$ of the largest size possible (say, m blocks)
4: Set π^{-1} to the inverse of π, and J' such that $J'(j') = J(\pi^{-1}(j'))$
5: (let $j' = \pi(j)$)
Current state: $\sum_{j'=0}^{L-1} \zeta_s^{J'(j')} \left|\pi^{-1}(j')\right\rangle |j'\rangle$
6: Divide the second register by K in-place
Current state: $\sum_{j'=0}^{L-1} \zeta_s^{J'(j')} \left|\pi^{-1}(j')\right\rangle |j' \bmod K\rangle \left|\left\lfloor \frac{j'}{K} \right\rfloor\right\rangle$
7: Measure the second register; let X be the result.
8: **if** $X = m$ **then**
9: **return** failure
10: **end if**
Current state: $\sum_{j'=XK}^{(X+1)K-1} \zeta_s^{J'(j')} \left|\pi^{-1}(j')\right\rangle |j' \bmod K\rangle$
11: Use a QRACM uncomputation to remove the first register (noting that $j' \bmod K$ and K precisely determine $\pi^{-1}(j')$)
Current state: $\sum_{j'=XK}^{(X+1)K-1} \zeta_s^{J'(j')} |j' \bmod K\rangle$
12: (Notice that $J'(j') = A \cdot (j' \bmod K)$ if $j' < mK$)
Current state: $\sum_{j'=XK}^{(X+1)K-1} \zeta_s^{A\cdot(j' \bmod K)} |j' \bmod K\rangle$
13: (Relabel $j'' = j' - XK = j' \bmod K$)
Current state: $\sum_{j''=0}^{K-1} \zeta_s^{A\cdot j''} |j''\rangle$
14: Apply a QFT to the state
Current state: $\sum_{k=0}^{K-1} |k\rangle \sum_{j''=0}^{K-1} e^{i\tau \frac{Aj''s}{N}} e^{i\tau \frac{kj''}{K}}$
15: Measure the state; let the result be k
16: **return** $\frac{kN}{H}$

since $\frac{H}{A} = K$. This is the amplitude for such k, so the square gives the probaiblity of measuring such k. This means with probability at least $\frac{16}{\tau^2} \approx 0.41$, Eve obtains some k such that $|\theta_k| = |\frac{kA}{H} + \frac{sA}{N}| \leq \frac{1}{2K}$. Expanding this a bit more and assuming $K = \frac{H}{A}$ (i.e., ignoring the floor function), it means

$$-\frac{1}{2H}A \leq \frac{k}{H}A + \frac{s}{N}A \leq \frac{1}{2H}A \tag{6.15}$$

$$-\frac{Nk}{H} - \frac{N}{2H} \leq s \leq -\frac{Nk}{H} + \frac{N}{2H} \tag{6.16}$$

That is, Eve computes $k\frac{N}{H}$ as her best guess for s, and this will be correct except for a window of size $\frac{N}{H}$.

This is not exactly true, since there may be no k satisfying $|\theta_k| \leq \frac{1}{2K}$. We know $k \leq K$, and if A is too small, this might not satisfy Equation (6.16). Peikert's solution is to finish all the collimations first, then concatenate all the final phase vectors and do a $\lg(N)$-bit QFT at the very end. This makes little difference for Eve since storing the final states are small for both classical and quantum storage.

Alternatively, before the QFT, in Step 13, Eve can use phase gates to add an extra phase of $e^{-i\tau\frac{Aj''s'}{N}}$, where s' is her best guess for s so far. This means we can set

$$\theta_k := \frac{k}{H}A + \frac{s - s'}{N}A \tag{6.17}$$

and follow the same logic. Eve will end up with an approximation of $s - s'$, and since she knows s', she can construct a better guess of s as well and continue this iteratively.

Intuitively, the factor of A in the phase means Eve is only measuring the bits "above" A, and the length of k limits her to obtaining $\lg(K) \approx \lg(L)$ bits at one time. We started with $L \approx K$, so I assume $\lg(K) \approx \lg(L)$. However, this process is not exact, and from computer simulations Peikert [Pei20] found that this process reveals about $\lg(L) - 2$ bits.

Overall this process reveals about $\lg(L) - 2$ sequential bits at whatever location Eve chooses, which she decides by the altitude. Since she can decide to collimate to reduce height or increase altitude, she can select precisely which bits of s to recover.

6.2.5 Summary

Putting this all together, recall the sketch of Kuperberg's algorithm, illustrated in Figure 6.4:

- Use the group action oracle to produce initial phase states of length 2.
- Combine phase states to a high enough length, then collimate to decrease height or increase altitude. Repeat this process, creating a tree structure of phase states.
- Once there is a state where height/altitude approximately equals length, apply a Quantum Fourier Transform and measure to reveal lg(length) − 2 bits of the secret.
- Repeat the process to obtain all bits of the secret.

This is Kuperberg's algorithm. For a quick sketch of runtime, given two phase vectors of length L, Eve can combine them and collimate to another vector of length L, and this should reduce the ratio of H/A by a factor of approximately L. Since H/A is roughly N for initial phase states, she needs to recurse this collimation about $\log_L(N)$ times. The number of leaves in this "tree" is roughly $2^{O(\log_L(N))}$, so that is how many times she needs to query the group oracle. However, each collimation requires a QRACM access to $O(L)$ bits of memory, so the gate cost is roughly

$$2^{O(\log_L(N))}O(L). \tag{6.18}$$

You can see that as L increases, the number of collimations and oracle calls decreases, but the cost of memory access increases.

If I take L such that $\log^2(L) \approx \log N$, then this comes out to

$$2^{O(\sqrt{\log N})} \tag{6.19}$$

i.e., sub-exponential.

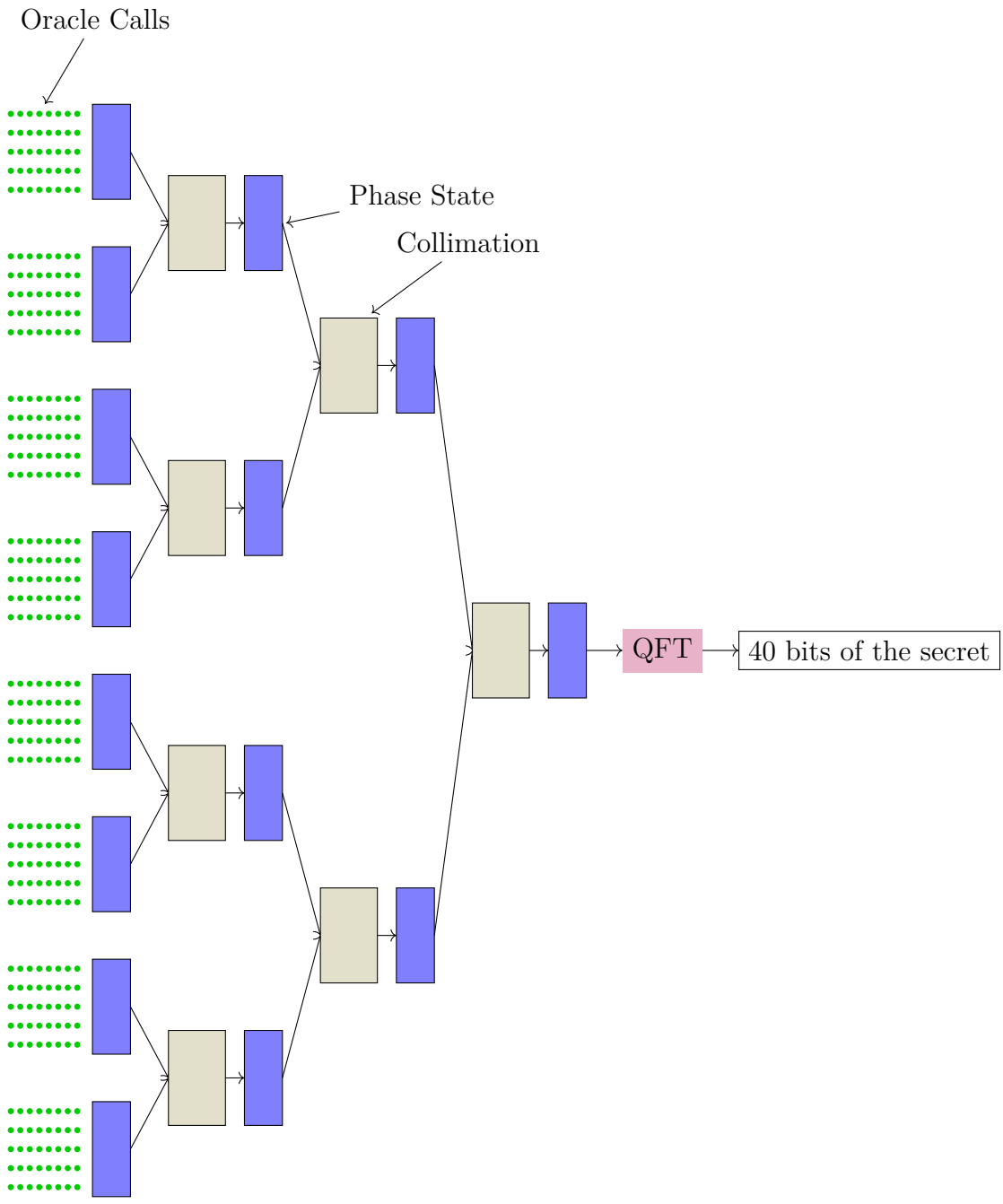


Figure 6.4: Tree of collimations in Kuperberg's algorithm. Phase states to the right will be better, meaning the height is lower or the altitude is higher. Notice that Eve could traverse this tree depth-first sequentially (with minimum memory requirements), or parallelise the initial (left-most) layers of the tree. While this illustration has only 3 layers, an effective, real attack against CSIDH-4096 might use 51 layers.

Subgroups

Recall (Section 6.2.2) that Eve can create better input phase vectors if she can split the class group into a product, and she knows some components of the secret in this representation. However, she must form at least one collimation tree that uses input phase vectors with random phases from the entire group. After that she might be able to use the outputs from previous collimations to start with smaller superpositions and thus better input phase vectors.

In the best case for Eve, the class group can be decomposed into a product of many small-order subgroups. Each collimation tree will reveal some components of the secret (of size approximately $L/4$, since she learned about $\lg L - 2$ bits), so each collimation tree will focus on a subgroup of size approximately $L/4$ times smaller than the last. Thus, the cost is primarily the cost of the *first* and largest collimation tree.

In the worst case for Eve, there is a large prime-order component. Learning some bits of the secret in this component will not make the remaining collimations any easier, so she will need to perform many full-size collimations.

What is the group structure of CSIDH? Conjecturally, class groups should be mostly cyclic [CL84]. As an example, CSIDH-512 has a class group of order 2^{256} but can be split into components such that the largest has size 2^{135} [BKV19]. We assumed Eve did not use an adaptive strategy and repeated the full collimation tree. Our estimates found that the total number of collimation trees is about 2^7, even for large primes, so a smooth group structure could reduce attack costs by at most about 7 bits. Exploiting CSIDH-512's group structure drops about 3 bits.

6.2.6 Cost Analysis

Here I give a more precise cost analysis and explain how we numerically estimated costs.

Number of Collimations

I assume that when Eve collimates r phase vectors of length L, she chooses parameters so that the resulting phase vector will also have length L. Thus,

each collimation reduces the height by a factor of $L^{r-1}c_r$, where $c_r = \sqrt{\frac{6}{r\tau}}$ (see [CSCDJRH21] for details). Intuitively, there are L^r phases in superposition with r phase vectors, and if the phases are distributed uniformly up to their height, then intervals of size L^{r-1} will have about L states in each one.

Since phase vectors start with a height of N and Eve wants a height of L, she needs

$$h = \left\lceil \frac{\lg(N/L)}{\lg(L^{r-1}c_r)} \right\rceil \tag{6.20}$$

layers of collimation. Because of the rounding, the height of the first layer need only be $H_1 := L \cdot (L^{r-1}c_r)^{h-1}$, so she can afford a slightly smaller length L_0 for initial phase vectors, since $N/H_1 < L^{r-1}c_r$.

From Peikert, I assume each layer has probability 0.98 of failing. Taking r^h leaf nodes each of length L_0 and assuming some fail, this tells us how many oracle calls Eve needs. I add a $2^{0.3}$ "fudge factor" from Peikert as well.

Depth Limits

Suppose Eve has a fixed depth limit and must parallelise.

Firstly, she can compute multiple collimation trees in parallel, since each produces independent parts of the secret, so this parallelises perfectly.

The tree of collimations parallelises extremely well: Eve can collimate separate pairs of nodes in the tree independently. The minimum depth is h sequential collimations, but up to that point I can assume she parallelises perfectly.

The oracle calls can also parallelise perfectly. Eve needs a minimum parallelisation of the oracle to ensure there are enough input phase vectors into the trees of collimation. If she parallelises more than this, she must store extra phase vectors from the oracle.

Within the collimation, the QRACM access parallelises almost perfectly using the high-width QRACM circuit from [BGM+19]. The sort-and-search parallelises nearly perfectly as well; I assume it parallelises perfectly.

Final Estimates

Recall that the size of the collimation tree depends on the length of the phase vectors and the arity of each collimation, which are parameters of the attack. Rather than cleverly optimise these, we simply tried all lengths and arities and chose the least costly attack. The concrete attack costs are highly discontinuous anyway.

For each fixed arity and phase vector length, we used the methods outlined above to estimate the costs of each collimation step, deciding based on a depth limit how they must parallelise. We also calculated how many group action computations are necessary to create the input phase vectors, and then used a unit cost for each of these computations. While this is obviously an underestimate, using the estimation methods from Section 6.2.2 made *no* change to the final costs for primes larger than 3072 bits.

Given the cost of each collimation, we multiply by $\lg(N)/(\lg(L)-2)$, the number of collimation trees necessary to recover all bits of the secret.

The results are in Section 6.4. Since these describe a generic attack on the vectorisation problem in a CSIDH class group, I will now turn to whether there are more effective attacks for restricted problems.

6.3 Group Structure

The best classical attack so far against CSIDH is a meet-in-the-middle attack (see Chapter 7). Taking exponents between $-m$ and m, this has cost on the order of $O(\sqrt{(2m+1)^n})$ (with an equal amount of memory). For the huge primes needed to escape Kuperberg's algorithm, n becomes so large that we could take m quite small and the classical attack would still be cryptographically infeasible. Since the size of m governs the cost of performing the key exchange, we asked the question: is a small m still secure against quantum attack?

We argued that small exponents will still be secure, mainly because the group structure of class group is quite distinct from these exponents. Specifically, we focused on three reasons:

1. The representation of elements of the class group must be homomorphic to integers represented as bitstrings;

2. An incomplete superposition of states will not produce phase vectors correctly;

3. An incorrect input phase vector seems impossible to detect or correct, and rapidly corrupts later levels of the collimation sieve.

I expand these points in detail.

6.3.1 Group Representations

If Eve represents points in the group as vectors $\vec{v} \in L$, then her input states before the QFT will have the form

$$|0\rangle |\vec{v}\rangle + |1\rangle |\vec{v} + \vec{s}\rangle \tag{6.21}$$

and she will need to apply a QFT. The QFT must respect the group structure in the sense that the phase from $\vec{v} + \vec{s}$ must be the phase from $\vec{v}$ multiplied by the phase from $\vec{s}$. More or less, she will need to apply a separate p-ary QFT to each state in this superposition, taking the state to

$$\sum_{\vec{k} \in L^*} |0\rangle \left|\vec{k}\right\rangle e^{i\tau \frac{\vec{k}\cdot\vec{v}}{p}} + |1\rangle \left|\vec{k}\right\rangle e^{i\tau \frac{\vec{k}\cdot(\vec{v}+\vec{s})}{p}} \tag{6.22}$$

From here, Eve can collimate as before, but she will need to collimate each of the n components, each of which has $\lceil \lg(p) \rceil$ bits. This is far more than the expected $\frac{1}{2}\lg(p)$ bits she will collimate if she treats the class group as a cyclic group.

She could attempt to do a QFT over each component using a smaller modulus than p; call it m. However, this will cause issues if $v_i + s_i \mod p \neq v_i + s_i \mod m$, as the QFT will not be properly homomorphic. A large m would suffice if, with overwhelming probability, $v_i + s_i < m$. Later in Theorem 6.3.1 I show how large such m must be.

More generally, the problem with this technique is that Eve is really doing a QFT over $\mathbb{Z}_p^n$ and ignoring the lattice structure[2]. Instead, suppose Eve wants a

[2] A QFT over $\mathbb{Z}_p^n$ is defined as $|v_1, \ldots, v_n\rangle \mapsto \sum_{\vec{k} \in \mathbb{Z}_p^n} \exp(i\tau \frac{v_1 k_1 + \cdots + v_n k_n}{p}) |k_1, \ldots, k_n\rangle$, extended linearly.

representation of group elements that uses this structure. She will need a QFT on this representation which perfectly captures the group structure. In Section 6.2.2 I presented the choice of representing a cyclic group G as elements of $(\mathbb{Z}/N\mathbb{Z})$, as multiples of some generator of G. In fact, any representation Eve chooses will be efficiently isomorphic to such a representation, as follows: Suppose Eve has some QFT that suffices for her representation. This will have the action

$$|x\rangle \mapsto \sum_{k=0}^{N-1} e^{i\tau \frac{f(k,x)}{N}} |k\rangle \tag{6.23}$$

whatever state "$|x\rangle$" is, and with f being bilinear. If she then composes this with the inverse of a traditional integer QFT, that will compute an isomorphism from her original representation to $(\mathbb{Z}/N\mathbb{Z})$.

Thus, we can assume Eve chooses to use multiples of a generator. In this case, we need to ask whether small exponents can be efficiently "compressed" in this representation. This should not happen, since the space of all low-weight exponent vectors spans all of $\mathbb{Z}^n$, so they must generate the entire class group. Moreoever, there is no reason to expect low-weight exponent vectors to match the group structure of $(\mathbb{Z}/N\mathbb{Z})$ in any way, so I expect them to correspond to random multiples of the generator. Hence, there should be no way for Eve to find a smaller representation.

6.3.2 Incomplete Superpositions

Taking one step back from the QFT, Eve must produce a superposition of the form

$$|0\rangle \sum_{g\in G} |g\rangle |g \star E_0\rangle + |1\rangle \sum_{g\in G} |g\rangle |g \star E_A\rangle \tag{6.24}$$

But since she knows that s was sampled from a smaller subset, she could try to sample from two different subsets H_0 and H_1, as follows:

$$|0\rangle \sum_{g\in H_0} |g\rangle |g \star E_0\rangle + |1\rangle \sum_{g\in H_1} |g\rangle |g \star E_A\rangle \,. \tag{6.25}$$

Here she needs $|H_0| = |H_1|$ for the normalisation to match.

At this point Eve measures the final register. Let $Z = \{g \star E_0 | g \in H_0\} \cup \{g \star E_A | g \in H_1\}$ be the set of all possible values Eve could measure. I define three subsets to partition Z:

$$Z_0 = \{g \star E_0 | g \in H_0\} \setminus \{g \star E_A | g \in H_1\} \tag{6.26}$$

$$Z_1 = \{g \star E_A | g \in H_1\} \setminus \{g \star E_0 | g \in H_0\} \tag{6.27}$$

$$Z_+ = \{g \star E_0 | g \in H_0\} \cap \{g \star E_A | g \in H_1\} \tag{6.28}$$

Then there are three cases, depending on the value v that Eve measures:

- If $z \in Z_+$, the resulting state will be $|0\rangle |g\rangle + |1\rangle |g + s\rangle$, as she wants.
- If $z \in Z_0$, the resulting state will be just $|0\rangle |g\rangle$.
- If $z \in Z_1$, the resulting state will be $|1\rangle |g\rangle$.

Only the first case is useful to Eve. Section 6.3.3 will address the problem of distinguishing these cases. For now, let us consider the probability of the good case, $z \in Z_+$. The probability of this is $|Z_+|/|H_0|$, but notice that $|Z_+| = |H_0 \cap (s + H_1)|$. How can Eve select H_0 and H_1 to maximise this without knowing s?

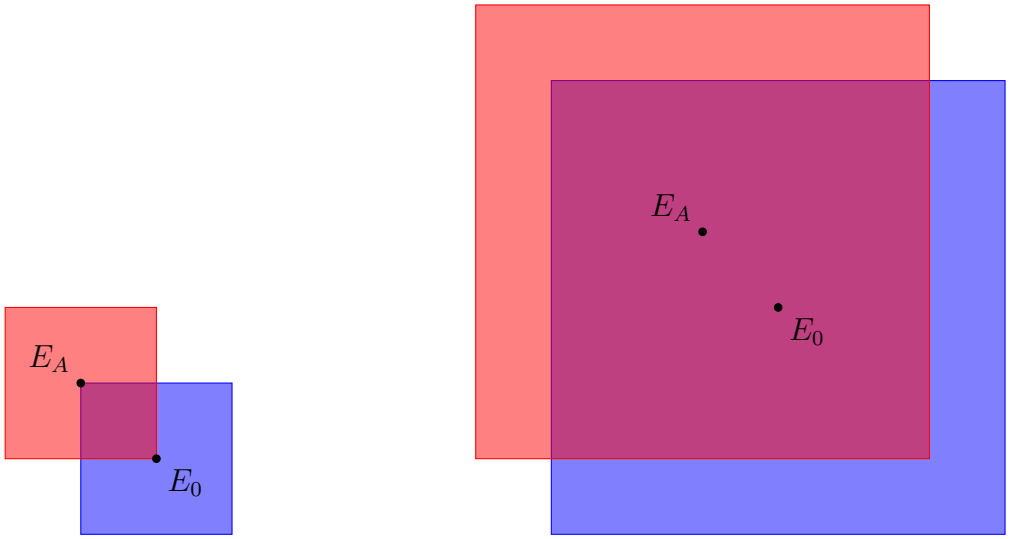
Theorem 6.3.1 (Theorem 4.1,[CSCDJRH21]). *If Eve samples exponent vectors with elements in $\{-m, \ldots, +m\}$, then for a key in $\{-1, 1\}^n$, the probability of a successful phase vector is*

$$\left(\frac{2m}{2m+1}\right)^n. \tag{6.29}$$

Figure 6.5 illustrates this.

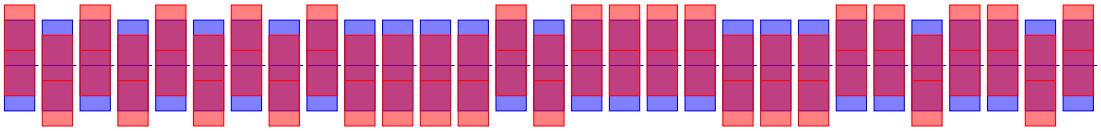
Proof. When Eve measures the final register, there are $2(2m+1)^n$ states in superposition: $(2m+1)^n$ exponent vectors for each of $|0\rangle$ and $|1\rangle$ in the first qubit. All these states have equal probability.

Here, H_0 and H_1 both equal $\{-m, \ldots, +m\}^n$. For small m, the hypercube of width m is likely contained entirely within the fundamental parallelepiped of L, so the set of curves $\{g \star E_0 | g \in H_0\}$ is in bijection with a hypercube of exponent vectors centered at 0 with width $(2m+1)$. The set of curves $\{g \star E_A | g \in H_1\}$ is



(a) Overlap with $m = 1$ in dimension $n = 2$, where $s = (-1, 1)$.

(b) Overlap with $m = 3$ in dimension $n = 2$, where $s = (-1, 1)$.



(c) Overlap in $n = 29$ dimensions, with each dimension represented as a separate column. This means the total proportion of overlap is the *product* of the overlap of each column, not the sum.

Figure 6.5: Visualising Theorem 6.3.1 by showing the overlap between the set of points reached from E_0 (blue) with the set of points reachable from E_A (red), where E_A is offset by some random key in $\{-1, 1\}^n$.

the same hypercube, offset by s. The intersection of these hypercubes has volume $(2m)^n$. □

As an example, suppose Eve wants to apply a QFT to the vector representation as in the last section. In [CSCDJRH21] we chose $n = 417$ for CSIDH-4192, so since the class group likely has size around 2^{2048}, Eve can only choose $m = 900 = 2^{2048/417}$ before collimating the vector representation would be more expensive than the generator-multiple representation. However, such a small m gives a probability of only 0.79 of producing a good input state.

I conclude that there is no clear method for Eve to reliably produce good input states from an incomplete superposition. Can she recover from a bad state, however?

6.3.3 Effects of Incomplete Superpositions

I will define a defective phase vector with fidelity q as a phase vector $(|\psi\rangle, J, L, A, H)$ with an extra set $B \subseteq L$ with $|B| = qL$, such that

$$|\psi\rangle = \sum_{j \in B} e^{i\tau \frac{J(j)s}{N}} |j\rangle \tag{6.30}$$

That is, the actual state in superposition only contains some of the phases that Eve thinks it does (her classical information J expects many more states). The quantum fidelity between the actual state and Eve's imagined state is q. The incomplete states from Section 6.3.2 will have fidelity $q = \frac{1}{2}$.

The problem here is that Eve cannot determine the fidelity of an input state. Worse, fidelity is multiplicative: Given two phase vectors V_1 and V_2 with fidelity q_1 and q_2, if we attempt to combine them as $|\psi_1\rangle \otimes |\psi_2\rangle$, the fidelity is $q_1 q_2$. Since she must combine at least two phase vectors before each collimation, the total multiplicative fidelity loss grows rapidly.

If Eve's final phase vector before measurement has fidelity q with respect to the state she wants, then she would only expect $\lg(L) + \lg(q)$ bits from measuring (note $\lg(q) < 0$). As a rough argument why: if two states have fidelity q, the probability of measuring the same result between them is at most q [NC00]. Thus, in the most favourable case for Eve, the bottom $\lg(q)$ bits of her measured value must be uniformly random to make the probability of a match sufficiently low.

Unitary operations cannot increase fidelity, but quantum operations with measurement can. I must thus argue that the collimation process is unlikely to help.

Recall that when Eve collimates two phase vectors of length L, their joint state before measurement has L^2 phases in superposition, and she measures a value parameterised so that she only keeps L of these phases, i.e., each phase has a probability of $\frac{1}{L}$ of being measured. This means if the *real* number of states in superposition in each of the two input states is qL, then she ends up with $\frac{(qL)^2}{L} = q^2 L$ phases in the result. She will *think* she has L phases, because her classical lists of phases for each phase vector each have L phases, so when she

solves the collimation permutation problem she will find about L states. Thus, the resulting fidelity should be q^2.

There is some variance in this process, and indeed it is slightly more probable to measure a value that matches many states in superposition, as in Figure 6.6. That is, collimation increases fidelity; however, when q is large this effect is relatively small, since the possible measurements will be spread out. When q is small this effect does not matter since the number of states is too small to be useful.

Theorem 6.3.2 (Theorem 4.2,[CSCDJRH21]). *Suppose an initial phase vector has length L, fidelity $q < \frac{1}{2}$, and height H. If Eve collimates this to a new height H', the resulting defective phase vector has expected fidelity at most*

$$q + 4\sqrt{\frac{\ln(L')}{L'}} \tag{6.31}$$

for $L' := \frac{H}{H'}Lq \geq 40$.

The proof is in Appendix E.1.

What Theorem 6.3.2 shows us is that for small q, Eve can only linearly increase the fidelity with each collimation. For collimations after the first level, in Appendix E I prove a new result for this thesis:

Lemma 6.3.1. *Suppose Eve collimates two phase vectors of length L and fidelity $q \leq \frac{\sqrt{2}-1}{2\sqrt{2}} \approx 0.146$. Modelling the phases of the joint distribution as i.i.d. triangular variables, then the probability that the collimated state has more than $\frac{q}{2}L$ states in superposition is at most $q^2 + O(\frac{1}{qL})$.*

Though the fidelity might increase slightly by randomly finding fewer fake phases, Lemma 6.3.1 shows a high probability of an exponential decrease in the number of states in superposition. Since there many sequential collimations (e.g., 51 for CSIDH-4096) this will rapidly decrease the number of states to 1. Lemma 6.3.1 is slightly incorrect, as the phases of the joint state are all sums of phases of two starting vectors so they will not be distributed independently. However, simulating the true behaviour of collimation, in Figure 6.7, shows the expected pattern.

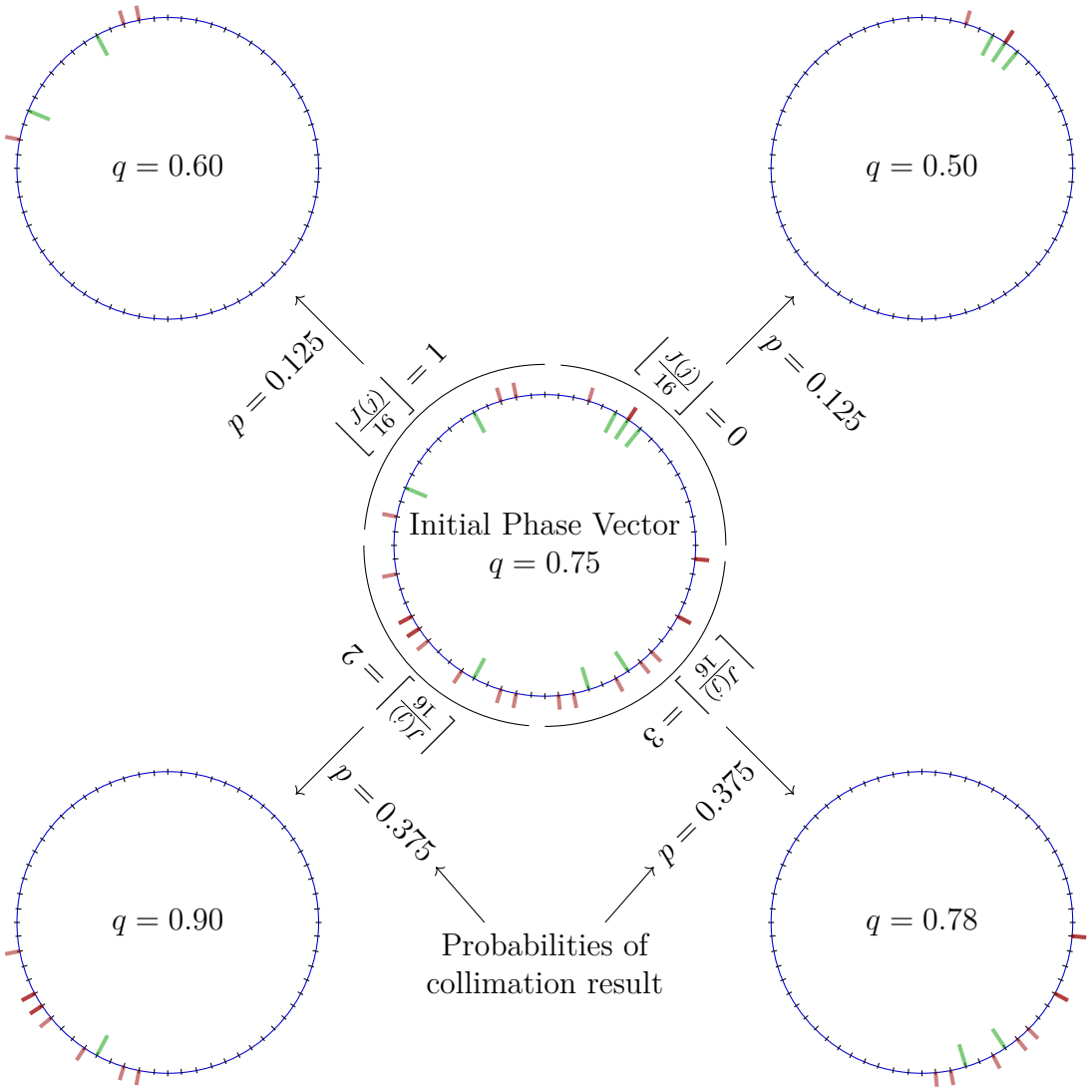


Figure 6.6: Collimation from a low-fidelity phase vector. q denotes fidelity. The initial phase vector is in the middle, and the four possible outputs of collimation are around it. Real phases are in red, "fake" phases (those Eve believes exist, but are not in the quantum state) are in green. Certain collimation results increase fidelity and others decrease it. The interesting phenomenon is that the probability of each collimation result is proportional to the number of real phases in that region, which means collimation results that *increase* fidelity are more likely. Theorem 6.3.2 shows that at larger lengths, the scale of this effect is not enough to really help.

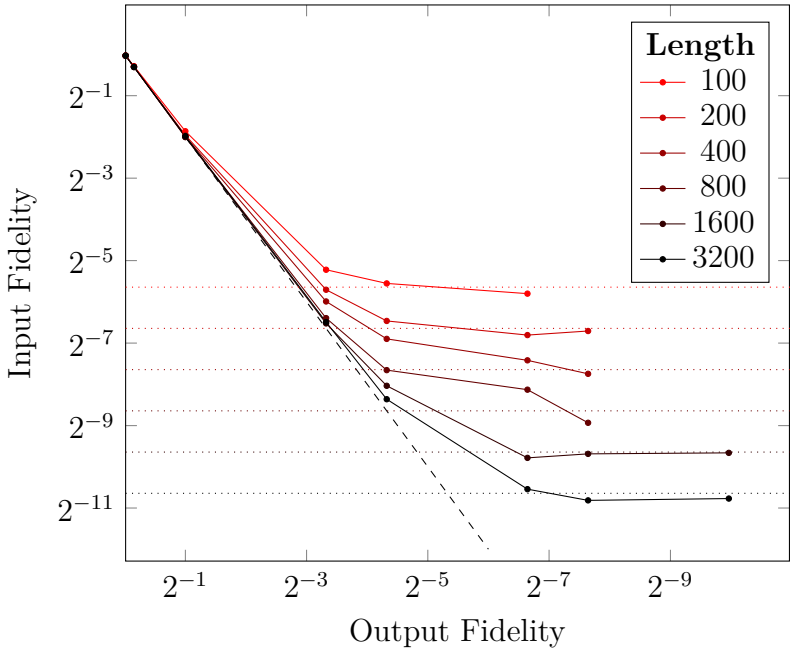


Figure 6.7: Output fidelity from simulating a collimation of two vectors of some input fidelity, averaged over 100 samples. I set the input height and target height to the cube and the square of the length, respectively. The different curves are different lengths of input vector. The dashed line is the square of the input fidelity, which will be the joint fidelity of the two input vectors. Each curve seems to follow the square of the inputs, except that it plateaus at 1/(2×length) (shown by dotted lines). Lines between marks are only for readability.

The core problem is that, as argued previously, combining two phase vectors to start the collimation decreases their fidelity multiplicatively, and the collimation process does not increase it enough to catch up. The multiplicative loss is clearly visible in Figure 6.7 while the collimation increase is essentially invisible. This gives Eve two choices:

- If she parameterises her collimation so that each state has more than a $\frac{1}{L}$ chance of being measured, then (in expectation) there will be more than L phase states in her classical table J that will be consistent with that measurement. Thus, the number of phases Eve must track will grow exponentially as she traverses down her tree of collimations, creating infeasible classical storage requirements.

- If she parameterises her collimation so that each state has less than a $\frac{1}{L}$ chance of being measured, then the number of quantum states will decrease

to less than qL, which is fewer than she started with. Thus, she will face an exponential decrease in the number of states in superposition.

Once the fidelity is $1/L$, Lemma 6.3.1 no longer applies, but at this point the phase vector is useless as there is only one quantum state left. Global phase is irrelevant, so all information about the secret (which was in the phase) is gone. The convergence to $1/L$ instead of 0 occurs because Eve will always measure a value that corresponds to at least one real state.

If Eve collimates two phase vectors of length L and fidelity $q = 1/L$, this means each has only one state in superposition. This means she will measure the unique joint state, so there will still be only one state in superposition, out of an expected L fake phases.

In short, if the input fidelity is greater than $\frac{1}{L}$, then combining phase vectors to start collimation will reduce the fidelity much faster than the collimation might increase it; if the input fidelity is already at $\frac{1}{L}$, the state is completely useless. Thus, using incomplete superpositions as input seems disastrous for Eve.

My final question is whether Eve could take many phase vectors, some which might be defective, and combine them together to make one good phase vector, similar to magic state distillation. More formally:

Problem 6.3.1 (Probabilistic Phase Vector Distillation (PPVD), Problem 4.1 from [CSCDJRH21])**.** *Let s be an unknown secret value. Suppose Eve has an input of n input states $|\phi_k\rangle$ with labels k such that with probability p, $|\phi_k\rangle = |0\rangle + e^{i\tau\frac{ks}{N}}|1\rangle$, with probability $\frac{1-p}{2}$, $|\phi_k\rangle = |0\rangle$ and with probability $\frac{1-p}{2}$, $|\phi_k\rangle = |1\rangle$.*

With some probability ϵ, either output 0 for failure or output 1 and t states $|\phi_{j_1}\rangle, \ldots, |\phi_{j_t}\rangle$ and their associated phase multipliers j_1 such that, for all i,

$$|\phi_{j_1}\rangle = |0\rangle + e^{i\tau\frac{j_i s}{N}}|1\rangle . \tag{6.32}$$

This seems difficult to solve, essentially because the distillation process must somehow produce the correct phase vectors for an *unknown* secret. Thus, if we attempt to measure at any point, our measurement result needs to be correctly

interpreted for any secret. Any method to do this seems too powerful: it would then be able to generate phase vectors from the defective inputs $|0\rangle$ and $|1\rangle$. Since Eve can prepare those states "for free", she could use them to generate extra phase vectors, which should not be possible.

To make this slightly more precise:

Lemma 6.3.2 (Lemma 4.2 from [CSCDJRH21])**.** *If the collimation sieve gives the optimal query complexity for the dihedral hidden subgroup problem, then no process can solve PPVD with $t\epsilon > pn$.*

Verbatim from [CSCDJRH21]. For a contradiction, let $t\epsilon > pn$. Assume Eve has a perfect phase vector oracle, from which she makes n initial queries. She then takes pn of these phase vectors and shuffles them together with $|0\rangle$ and $|1\rangle$ vectors. Then she runs the process that solves the PPVD.

If it succeeds, it produces t new phase vectors, which she adds to a growing list. If it fails, she calls the phase vector oracle another t times. Either way she has $t - np$ new phase vectors, and in the first case she did not need to call the oracle. Thus, each iteration calls the oracle $t(1 - \epsilon)$ times on average. Eve repeats this process to create all the phase vectors that the collimation sieve needs.

If the collimation sieve requires Q states, this process calls the oracle $\frac{Q}{t-np}t(1-\epsilon)$ times. If $t\epsilon > pn$, then

$$\frac{Q}{t - np}t(1 - \epsilon) < \frac{Q}{t - t\epsilon}t(1 - \epsilon) = Q \tag{6.33}$$

and thus she solved the dihedral hidden subgroup problem with fewer than Q states. □

This makes me skeptical that this problem can be solved in general. Still, a method to do this would help, slightly: Eve might be able to use smaller exponent vectors and distill them into good inputs. However, she would need to ensure that this process does not result in her making so many extra oracle queries that it negates the total cost saving.

6.4 Conclusion

6.4.1 Security Conclusions

Our goal was to compare to NIST's post-quantum security thresholds, which are based on key search for the AES cipher. Unfortunately for CSIDH, the best attack against AES is QAA, which costs more (in hardware, gates, and depth-width product) as it parallelises more. In contrast, Kuperberg's attack parallelises almost perfectly. Does this make CSIDH insecure?

Perhaps it should, since parallelism is necessary for real-world attacks. The hardware requirements for low-depth AES attacks are absurd (e.g. 2^{80} qubits for an attack of depth 2^{40} [JNRV20]), so one can reasonably claim AES is unbreakable by known attacks in depth 2^{40}, whereas CSIDH-512 might be. In that sense, CSIDH-512 is not as secure as AES-128.

On the other hand, after a certain point, even a "low-cost" attack has unbelievable hardware requirements. We should not defend ourselves against an attack requiring a planet-sized quantum computer. If we are security-conscious, we could instead add a "buffer" of security against *realistic* known attacks.

Thus, in [CSCDJRH21] we made *slightly* aggressive assumptions about the adversary's capabilities. Recalling the NIST security levels (Section 2.3.1), we set hardware limits of:

- 2^{80} for NIST level 1, from [ACVCD$^+$]
- 2^{100} for NIST level 2, the memory contained in a "New York City-sized memory made of petabyte micro-SD cards" [Sch20]
- 2^{119} for NIST level 3, the memory of a 15 mm shell of such cards around the Earth [Sch20].

On top of this, we gave an overhead of 2^{10} classical operations for each unit of DW-cost to account for error correction. [GE21] estimated an overhead of 2^{17} for each logical gate, so we assumed progress in in error correction. We assumed this

represented a surface-code-like architecture, so the depth overhead of each quantum operation is $2^{\frac{10}{3}}$ and the physical qubit overhead is $2^{10\cdot\frac{2}{3}}$.

Combining all of the estimations from Section 6.2 with these costs, we wrote a Python script[3] to estimate the cost of the attack under different depth and hardware limits. Figure 6.8 gives the results.

You can see that CSIDH suffers from its parallelism, e.g., CSIDH-3072 is more secure than AES-128 except at small depths. Our ultimate conclusion was that CSIDH-4096 meets NIST level 1, CSIDH-6144 meets NIST level 2, and CSIDH-8192 meets NIST level 3.

Kuperberg's algorithm is a tremendously effective use of QRACM. Notice that in a collimation, Eve only performs *one* look-up into any given table; thus, the classical cost to even write the table is always at most a constant factor less than the quantum cost to perform the look-up. Indeed, we found that the majority cost was not any QRACM look-up, but solving the collimation permutation problem to construct the table which the QRACM will access. Thus, these conclusions are nearly indifferent to the arguments in Chapter 4: cheap QRAM would make little difference to the cost of Kupberberg's algorithm.

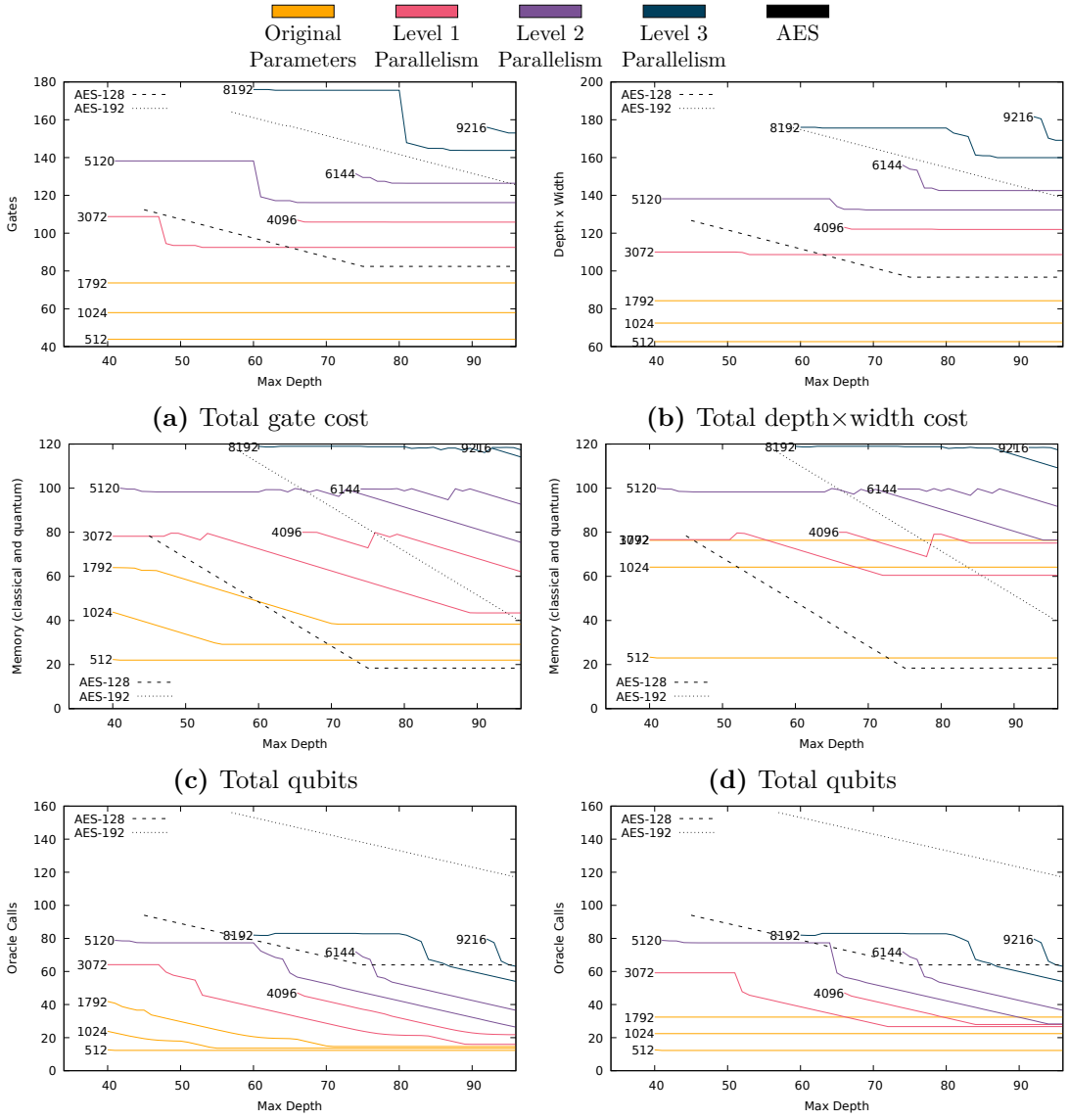
6.4.2 SQALE'd CSIDH

The goal of all the arguments in Section 6.3 was to justify the use of small exponents in a practical implementation, since these will be much faster. This led to our "SQALE"d CSIDH, which stands for **S**ubliner Vélu **Q**uantum-resistant isogeny **A**ction with **L**ow **E**xponents. The "sublinear Vélu" refers to the method of computing the isogenies [BFLS20]; it is quantum-resistant thanks to the large primes, and it uses low exponents.

My colleagues on this project wrote an optimised constant-time implementation, and Table 6.1 shows the performance.

Performance is measured in *gigacycles*.

[3] Available at `https://github.com/JJChiDguez/sqale-csidh-velusqrt/tree/main/quantum-cost-estimation`.



(a) Total gate cost

(b) Total depth×width cost

(c) Total qubits

(d) Total qubits

(e) Total oracle calls

(f) Total oracle calls

Figure 6.8: Costs of the quantum collimation sieve attack under various hardware limits, from [CSCDJRH21]. Coloured solid lines are the costs of the collimation sieve at primes of bit lengths from 512 to 9126; dotted lines are the cost of key search on AES, from [JNRV20], with the same memory limits and overhead as our analysis. All figures are logarithmic in base 2. Plots on the left are parameterised to minimise gate cost, plots on the right to minimise DW-cost. Larger primes achieving lower depth (e.g., 5120 vs. 4096) is due to increased memory limits. Level {1,2,3} parallelism refers to the hardware limits in the text.

NIST Security Level	**Quantum Security** (bits)	**Prime size** (bits)	**Performance** (gigacycles)
Level 1	124	4096	23.3
Level 2	148	6144	74.8
Level 3	>160	8192	199.1

Table 6.1: Summary of results, from [CSCDJRH21].

6.4.3 Discussion

We saw how to optimise Kuperberg's algorithm to match the costs of attacks against AES. Mainly to compensate for bad parallelism in the AES attack, CSIDH-4096 is the smallest parameter set matching AES-128. Unfortunately, the performance of CSIDH-4096 is tremendously impractical. What should we do?

Personally, I am most cautious of the rapid breakthroughs in *classical* cryptanalysis of isogeny-based cryptography [CD23, MMP+23, Rob23]; I put far higher odds that in my lifetime there is a complete classical break, rather than a quantum computer powerful enough to break it.

For now, assume that Kuperberg's algorithm remains the best attack.

On the one hand, this speaks against NIST's security levels, since smaller parameters like CSIDH-512 are still difficult to attack. The cost of group action computations (to produce input phase vectors) were insignificant for the "SQALE"d parameters; however, our estimates of the cost to attack CSIDH-512 jumped from 2^{63} to 2^{87} operations when we accounted for these costs. This is a *huge* computation, and perhaps we undercounted costs in the group action (what about spatial layout? counting linear operations? more rigorous pebbling?). Surely we have decades before a quantum attacker could break CSIDH-512?

The alternative view is that breaking ECC-256, by the same metrics[4], is roughly 2^{52} operations. This makes CSIDH-512 only 2^{35} times harder to break than today's cryptography. Thinking of the lessons from Chapter 1, is this hard enough? Moreover, most of the attack cost against CSIDH-512 is in group action

[4]Our CSIDH estimates assumed only 2^{10} overhead from the surface code and the methods from Chapter 5 would estimate 2^{15} overhead. Thus, I dropped a factor of 2^5 from the ECC-256 estimate in Chapter 5.

computations, and there is a reasonable intuition that a cryptosystem vulnerable to an attack taking only 2^{22} oracle calls should not be called “secure”.

7

The Claw-Finding and Golden Collision Problems

Contributions. This chapter is based on a joint work with André Schrottenloher [JS21]. This was a highly collaborative work. I was the main contributor of the algorithm that uses iterated distinguished point finding, and the random function analysis necessary for its runtime, which this chapter presents. André noticed that Quantum Amplitude Amplification was a simpler approach to find distinguished points with the same runtime; together, we worked out how this technique could also apply to the "Multi-Grover" algorithm from [BBG+13], detailed in Section 7.4.

The Data Encryption Standard (DES) was standardised with 56-bit keys (see Chapter 1), but once cryptographers started to realise that these were too small, they looked for ways to expand the protocol. A naive first approach is to use 2 keys, and encrypt with both, one after the other:

$$\text{DOUBLE-DES}((k_1, k_2), m) = \text{DES}(k_2, \text{DES}(k_1, m)). \tag{7.1}$$

However, this is vulnerable to a *meet-in-the-middle* attack: given a plaintext-ciphertext pair (m, c), Eve creates a table of $\text{DES}(k_1, m)$ for all possible k_1 (which is only 2^{56} messages), then she iterates through all k_2 and checks if $\text{DES}^{-1}(k_2, c)$ is in the table. If it is, she has found both halves of the key. The total complexity is still roughly 2^{56}.

This is a specific instance of a ubiquitous cryptographic problem, called *claw-finding* (Figure 7.1):

Problem 7.0.1 (Claw-Finding)**.** *Let $f : X \to S$ and $g : Y \to S$ be two functions. Find $x \in X$ and $y \in Y$ such that $f(x) = g(y)$.*

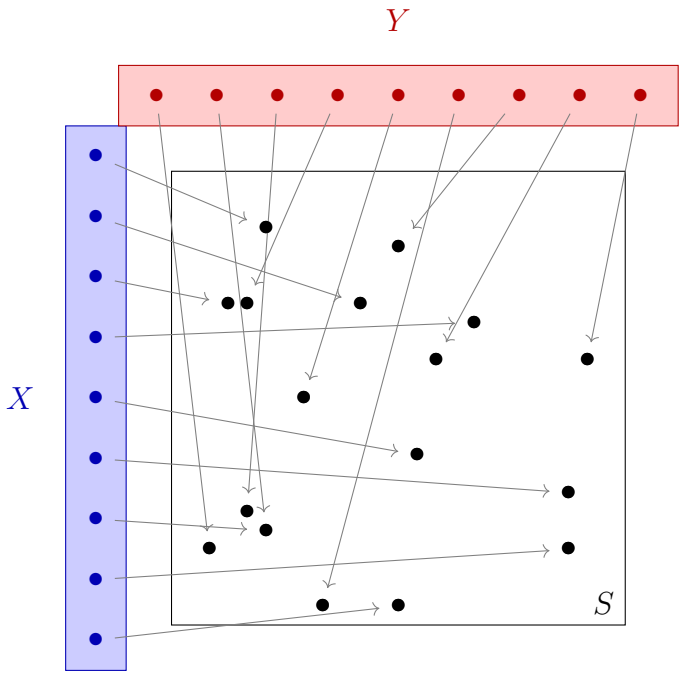


Figure 7.1: The claw-finding problem. The arrows represent f and g. There is one claw, can you find it?

Some cases where this appears in cryptography:

1. Discrete log: Given an instance of a discrete log problem $Q = sP$ in a group G of order q, we can write $s = am + b$ for $m \approx \sqrt{Q}$. Then letting $f : \{0, \ldots, m\} \to G$ given by $f(a) = amP$ and $g : \{0, \ldots, m\} \to G$ given by $g(b) = Q - bP$, solving the claw-finding problem gives a pair (a, b) such that $f(a) = amP = g(b) = Q - bP$, so $s = am + b$. Of course, discrete log is much easier to solve with a quantum computer than claw-finding! Even classical computers can do better for finite field discrete log (e.g. [BGG+20]).

2. Learning-with-errors: The Learning-with-errors (LWE) problem asks to find $s \in \mathbb{F}_q^n$ given $A \in \mathbb{F}_q^{m \times n}$ and $b = As + e$, where both s and $e \in \mathbb{F}_q^m$ are sampled from some small distribution. This is the underlying problem behind all known fully homomorphic encryption constructions (e.g. [GSW13, CGGI19, BGV12]) and the new NIST standards for post-quantum cryptography [BDK+18, DKL+18].

   If we split $n = n_1 + n_2$, and split A column-wise as $A = A_1 \| A_2$, then let $f : \mathbb{F}_q^{n_1} \to \mathbb{F}_q^m$ be given by $f(s_1) = A_1 s_1$ and $g : \mathbb{F}_q^{n_2} \to \mathbb{F}_q^m$ be given by $g(s_2) = b - A_2 s_2$, then this is *almost* a claw-finding problem. For the secret s_1 and s_2, we have that $f(s_1) - g(s_2) = e$, which is small, so we know that $f(s_1)$ and $g(s_2)$ are close. Any technique to compress close vectors (say, rounding) will transform this into a true claw-finding problem.

3. Supersingular Isogeny Key Encapsulation (SIKE): SIKE was an isogeny-based key encapsulation method that predated CSIDH [JAC+17]. Unlike CSIDH, no abelian class group produces the isogenies, and for SIKE's parameters the only known group action was inefficient [KMPW21]. The hard problem was to compute an isogeny of degree ℓ^e between two known elliptic curves E_0 and E_A, where ℓ is a small prime. As a claw-finding problem the function f would compute an isogeny of degree $\ell^{e/2}$ from curve E_0, and g would compute an isogeny of degree $\ell^{e/2}$ from E_A. If they met, that provided the correct secret isogeny.

In fact, this was the original motivation for our work on this problem. However, in summer 2022, SIKE was completely broken *classically* [CD23, MMP+23, Rob23]. In short, the lack of structure made it harder for honest parties to compute their secret isogenies, so the protocol needed to send extra pieces of information called *torsion points*. Those torsion points provided enough information for an attacker to completely break the scheme in a matter of minutes on consumer hardware.

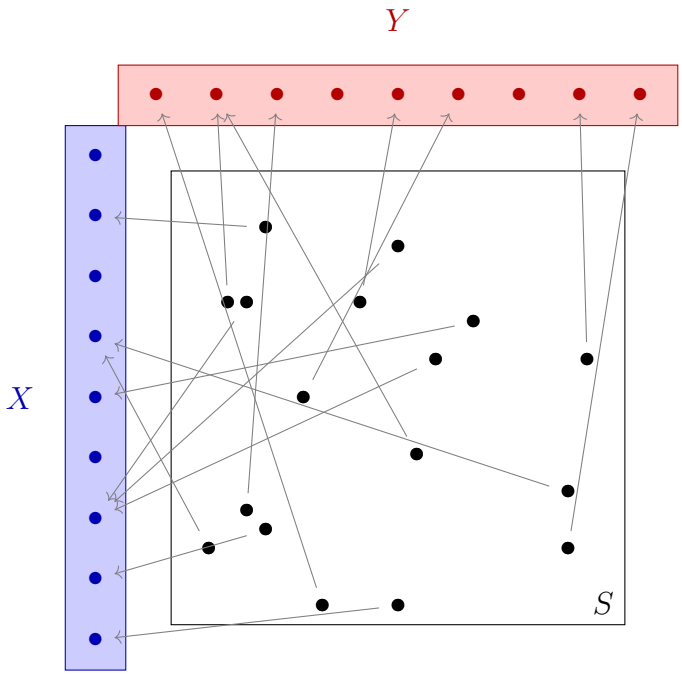
4. In certain badly-designed cryptographic protocols, part of the ciphertext can leak the XOR of two blocks of plaintext. This is such a common mistake that its history stretches from attacks on the Lorenz cipher in World War II [McK06] up to the KRACK attack on the Wi-Fi Protected Access protocol in 2016 [VP17]. Since the plaintext space might be easy to enumerate (say, English language strings), then $f : X \to S$ can map some input space onto these plaintexts, and $g : X \to S$ will XOR the plaintext with the known XOR result. If $f(x) = g(y)$, Eve recovers the two plaintext blocks.

In terms of quantum query complexity, the claw-finding problem is completely solved: there are matching upper and lower bounds [AS04, Tan07]. However, in *gate* complexity (or depth-width complexity), this is not the case [JS19]. This chapter explores quantum algorithms that provide the best known gate complexity, and the best known *cost* under depth restrictions.

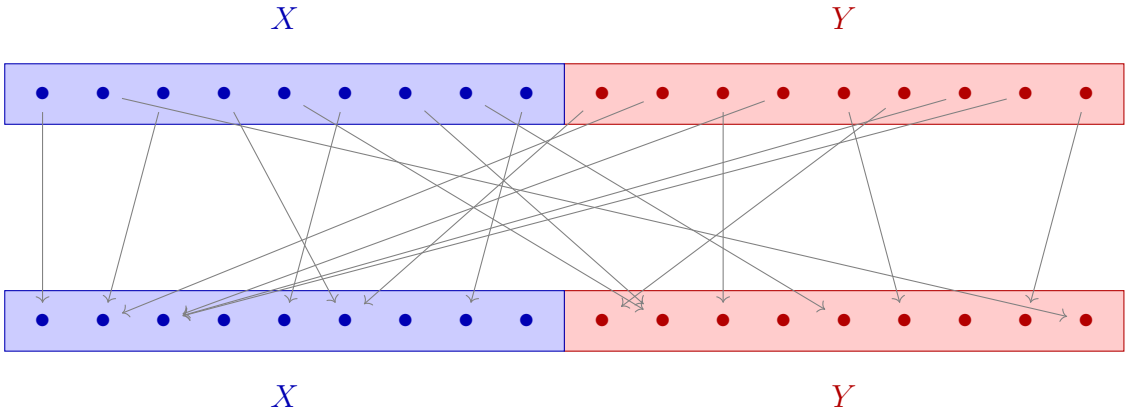
7.1 Introduction

7.1.1 Golden Collision Finding

If we have access to a selection of random functions $F : S \to X \sqcup Y$ (here $\sqcup$ means a disjoint union), then we can define $h : X \sqcup Y \to X \sqcup Y$ by $h(x) = F(f(x))$ and $h(y) = F(g(y))$. The claw is now a collision: two points x and y such that $h(x) = h(y)$. However, we expect this to introduce a large number of spurious collisions, as in Figure 7.2. Thus, we define the "golden collision problem":



(a) A random function from $S :\to X \sqcup Y$



(b) Composing f and g with the random function from $S \to X \sqcup Y$, to make a random function from $X \sqcup Y$ to itself. Notice how many more collisions there are!

Figure 7.2: From the set-up in Figure 7.1, this illustrates the reduction to golden collision finding. There are 9 collisions. The golden collision is the collision on the second element of X.

Problem 7.1.1 (Golden Collision)**.** *Let $h : X \to X$ be a function, and $g : X \times X \to \{0,1\}$ be a "check" function such that there exists a unique pair $x, y \in X$ such that $h(x) = h(y)$ and $g(x,y) = 1$. Find x and y.*

This is the function Eve will solve instead of directly solving claw-finding. For convenience, I will let $N = |X|$.

7.1.2 Quantum Random Walks

The framework of [MNRS11] provides a method to perform quantum random walks, which are almost always done on a Johnson graph. In this context, this means Eve will have large sets of size R as subsets of some set X, and she will need three subroutines:

- Set-up: This initialises a state $\sum_{S\subseteq X,|S|=R} |S\rangle$. I denote the cost as S.
- Update: This takes a state $|S\rangle$, denoting a subset of X, and replaces a random element in S with a random new element, where the "randomness" is in fact a superposition[1]:

$$|S\rangle \mapsto |S\rangle \sum_{x\in S, y\in X} |(S\setminus\{x\})\cup\{y\}\rangle\,. \tag{7.2}$$

  I denote the cost as U.
- Check: This takes a state $|S\rangle$ and checks if the two elements of the golden collision are contained in S, adding a phase if they are.

If I let ϵ be the probability that a set of size R contains the golden collision, [MNRS11] show that the cost of this algorithm is

$$O\left(\mathsf{S} + \frac{1}{\sqrt{\epsilon}}\left(\sqrt{R}\mathsf{U} + \mathsf{C}\right)\right). \tag{7.3}$$

There are a few common tricks for these random walks. The first is that $|S\rangle$ can be any history-independent quantum data structure representing the set S. Following [JS19], Eve will store each element of S as a tuple $(x : h(x))$, and S itself is a sorted array. Basic array operations (insertion, deletion, and search) have gate cost $\tilde{O}(R)$ and depth $O(\log(R))$.

Second, when inserting into an array, it is then easy to check for collisions, since the array is sorted. Thus, Eve will maintain a global flag qubit for each set S, and update the flag when inserting or removing elements. This makes the check cost $O(1)$, so I will ignore it going forward. She can also keep a fixed memory address

[1] The correct definition uses $y \in X\setminus x$, but using $y \in X$ adds negligible error in most cases [JS19].

to store the golden collision when it is in S, or pay an $O(S \log S)$ cost after the random walk finishes to sort the measured list and find the golden collision.

Finally, quantum random walks can be nested with other techniques. For example, if there are multiple sets $X_1, \ldots, X_n$, Eve could use QAA to search *which* X_i contains the golden collision, using a quantum random walk as the check operator for QAA. Or she could use QAA to help compute the update step. This chapter will use both techniques.

Previous claw-finding algorithms. [Tan07] provided the first application of quantum random walks to claw-finding. The method adapts easily to golden collision finding, so Algorithm 7 presents it in those terms for consistency. [Tan07] showed that in the query model, this problem has cost $O(|X|^{2/3})$ (with a matching lower bound from [AS04]).

Algorithm 7 Golden collision finding, as in [Tan07, Amb07]

Input: Functions $h : X \to X$ and $g : X^2 \to \{0, 1\}$ from a golden collision problem
Output: The golden collision (x_g, y_g)

Apply a **Quantum Random Walk**:
  Vertices: Subsets of X of size R
    Data structure: A list of R pairs $(x, h(x))$, ordered by $h(x)$
      A global counter c
  Marked vertices: Sets that contain x_g and y_g
  Setup:
    Set $S = \emptyset$, $c = 0$
    for $i = 1$ to R **do**
      Select random $x \in X$
      Insert(x, S)
    end for
  Update of vertex S:
    Select random y from S
    Adjoint **Insert**(y, S) (see Algorithm 8)
    Select a random $x \in X$
    Insert(x, S)
  Check:
    if $c > 0$ **then**
      Mark vertex
    end if

Algorithm 8 Inserting into a set during a random walk.

Insert (x, S):
1: Compute $h(x)$
2: **for** $z \in S$ such that $h(z) = h(x)$ **do**
3: **if** $g(z, x) = 1$ **then**
4: Increment c by 1
5: **end if**
6: **end for**
7: Insert $(x, h(x))$ into the list in S

However, this algorithm uses a substantial amount of memory, specifically QRAQM. Given the strong arguments in Chapter 4, we might question the gate cost. We did this in [JS19] and showed that the gate cost scales as $\tilde{O}(|X|)$, which is the same (up to poly-logarithmic factors) as QAA.

The surprising result is expanding the asymptotic factors shows a slight saving compared to Grover. Specifically, let H_G be the gate cost to compute the function h defining the problem, and suppose each element of Eve's list needs m qubits (e.g., to store both $x \in X$ and $f(x)$). Then the gate cost of Tani's algorithm can be brought as low as

$$O\left(|X|\sqrt{m\mathsf{H}_G}\right) \tag{7.4}$$

which is a factor of $\sqrt{\mathsf{H}_G/m}$ better than Grover's algorithm. As a brief explanation, the cost of an update, which involves inserting a new element x into a list S, includes two factors: the cost to update the memory (roughly $O(mR)$), and the cost to compute $h(x)$. This means the total cost, the general case of Equation (7.4), is something like

$$O\left(\mathsf{S} + \frac{|X|}{\sqrt{R}}(mR + \mathsf{H}_g)\right) \tag{7.5}$$

(recall that the check cost is $O(1)$, and convince yourself that $\epsilon \approx \frac{R^2}{|X|^2}$). We see that as long as $mR \leq \mathsf{H}_g$, then the cost decreases in R. In words, the random walk needs fewer steps when the sets are larger because each set is more likely to contain the golden collision; however, when the sets get too large, the memory

access cost in each step overwhelms the gains. This trade-off means we can use the memory to offset the costs to compute the function.

Thus, the motivating question of this chapter: can we make the function h *more* expensive, in some way that also reduces ϵ?

7.2 Iteration-Based Quantum Random Walk

To achieve the goal of making the collision function more expensive, we used a trick dating back to at least 1985 [RD82] (and "popularised" by [vOW99]): iterate the function until it reaches a *distinguished point*. A distinguished point will be any point belonging to a suitably small and suitably random subset of X. A simple way to define this is to pick a random k-bit string s and say that x is distinguished if x starts with the string s (here assuming we have an efficient encoding of X into bitstrings).

The set of distinguished points is much smaller than X itself, so the probability of collisions is much higher. If we consider the set $\tilde{h} : X \to X_{dist}$, formed by iterating h until it reaches a distinguished point, we know that the golden collision creates a collision in $\tilde{h}$. Thus, Eve will search for collisions in this smaller image set and use them to find the golden collision.

Classically, finding distinguished points by brute force has the same complexity as iteration. A quantum computer cannot speed up the process of iterating a function, but it *can* speed up a brute force search with QAA. That is the logic of [CNPS17, NPS20], for the problem of finding *any* collision. It also applies to golden collision finding, but surprisingly, iterations work just as well. The unexpected benefit is that iterating a function produces a large set of predecessors of the distinguished points, which makes it easier to find the golden collision.

Eve's general strategy will be to use the quantum random walk algorithm of [Tan07] on $\tilde{h}$, for a certain number of steps. For most random functions h it will fail, so she will choose a random $h' : X \to X$ and compose h with h' and try again. This process – searching for a random function h' that works well – will be *itself* done with QAA, whose oracle is the quantum random walk.

7.2.1 Iterating a function

I first discuss how to iterate the function. Eve's goal here is to map a state $|x\rangle$ (for $x \in X$) to $|x\rangle |h^n(x)\rangle |n\rangle$ such that $h^n(x)$ is distinguished. Here I am using h to refer to whatever random function Eve is currently checking.

Since h is generally not reversible, Eve must pebble the function somehow. This can be done with $O(n)$ steps in a straightforward way with $O(\sqrt{n})$ memory. With spooky pebbling, Eve can do this in $O(n \log n)$ steps with only $O(\log n)$ memory (see Appendix D.2).

Unfortunately, the controller cannot know whether $h^n(x)$ is distinguished; that can only be checked and written to a quantum register. This means Eve must choose some number of iterations n_c such that almost all inputs x will reach a distinguished point for $n \leq n_c$. [vOW99] show experimentally that if the fraction of distinguished points is θ, then the number of iterations to reach a distinguished point is geometrically distributed with a mean of $1/\theta$. We should expect this, since the output of h should approximate a random choice of an element of X, which will have a $1/\theta$ chance of being distinguished.

This means if she chooses $n_c = c/\theta$ for some constant c, the proportion of points that need more than n_c iterations is e^{-c}.

Finally, Eve needs a method to extract the distinguished point. She can use a circuit that detects a distinguished point and copies it out, and apply that after every iteration of h. This will cause a problem if there are multiple distingished points, so she can store the copied-out states in an array of size k, and cyclically shuffle it if a new point is copied in, as in Figure 7.3. After finishing all iterations, she can copy out the latest point (the latest point will have more *predecessors*, which will be more useful).

This will succeed as long as Eve does not find more than k points. Since the expected value of distinguished points is $\theta n_c = c$, by Markov's inequality she will exceed k points with probability at most $\frac{c}{k}$. Even if she does exceed k points, she only cares if this happens with any of the predecessors of the golden collision, so this

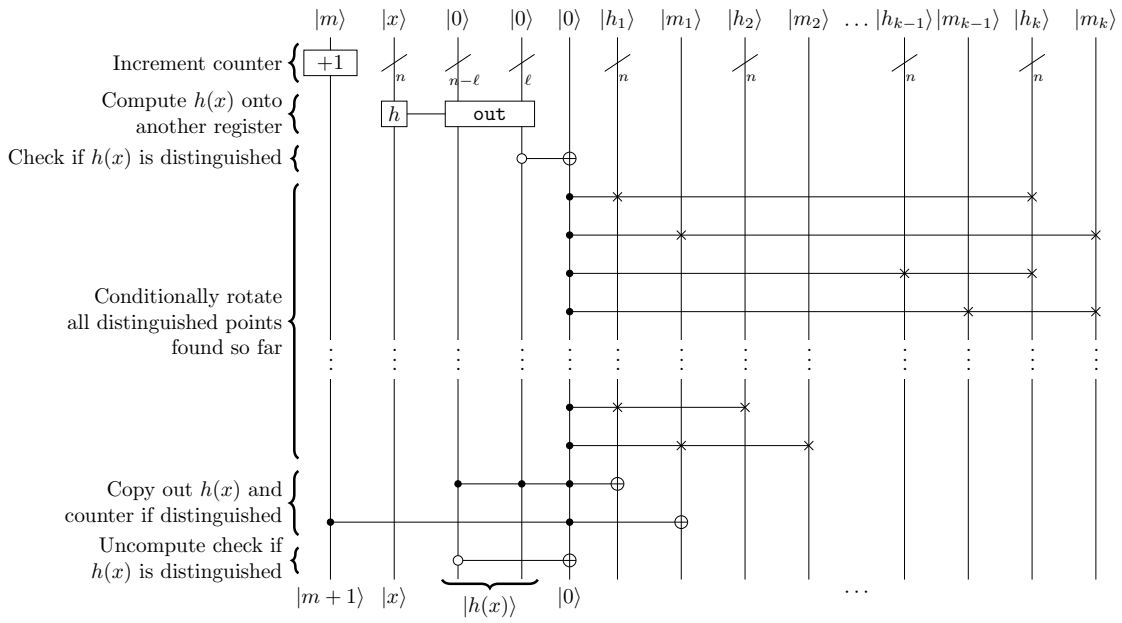


Figure 7.3: Circuit for one iteration of h, checking if the output is distinguished (starting with ℓ 0s) and storing it if it is. h_i, m_i are previously-found distinguished points (initialised to 0), i.e., $h_i = h^{m_i}9x)$ is distinguished.This circuit produces both x and $h(x)$ as "output", but only $h(x)$ is input to the next iteration. Thus some pebbling is necessary.

effectively reduces the value ϵ by a multiplicative factor of $1-\frac{c}{k}$. For a constant $k \geq c$, this increases the runtime of the quantum random walk by a factor of only $O(1)$.

Thus, with $\tilde{O}(\frac{1}{\theta})$ time and $O(\log(1/\theta))$ memory, Eve can compute the iterated function $\tilde{h}$ up to constant error.

7.2.2 Quantum Data Structure

The quantum data structure to store lists in the random walk is an ordered list of states $|x\rangle |h^n(x)\rangle |n\rangle$, together with a counter of the number of points that lead to the golden collision, and a register containing two predecessors of the golden collision, if they exist.

Eve needs a history-independent method to insert a new element in the list and update the collision counter. Inserting into the list is an $O(R)$ cost [JS19]. To update the counter, Eve will run a search to check for the value of $h^n(x)$ in the list before inserting x. The circuit from [JS19] fans out the value, then uses a tree of `CNOT` gates to return the result. In [JS21] we suggest t parallel trees to assemble results

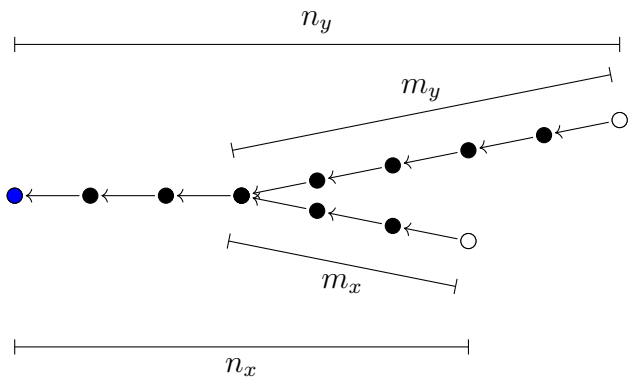


Figure 7.4: A collision of two iterations. The blue point on the left is distinguished, and the white points on the right are the initial values.

to ensure that if there are more than t matches, they are all returned. The trees are connected such that if there are two matches, one match gets copied to the next tree.

If Eve finds a collision, she must determine if it is the golden collision. This involves first finding the point of collision, then checking if it is the golden collision. Given two points $(x, h^{n_x}(x), n_x)$ and $(y, h^{n_y}(y), n_y)$ with $h^{n_x}(x) = h^{n_y}(y)$, she knows there is some minimum m_x, m_y such that $h^{m_x}(x) = h^{m_y}(y)$ (see Figure 7.4). This gives a collision between $x' = h^{m_x-1}(x)$ and $y' = h^{m_y-1}(y)$. If $m_y = 0$ or $m_x = 0$ (what [vOW99] called a "robin hood"), we get no collision, but this is unlikely. This is also the only collision from this tuple: once the trails of x and y collide once, they follow the same path going forward.

To find this point, Eve must iterate the function h for $n_{max} = \max\{n_x, n_y\}$ iterations, applying it to two trails, one each from x and y, as in Figure 7.5. She keeps a global counter r which gets initialised to n_{max} and she decrements in every iteration of h. In each iteration, she compares r to n_x and n_y. She uses the result to control iteration of h, only applying h to the trail for x if $r \leq n_x$. At every iteration, she also applies a circuit which checks if the outputs are the same. The result of that comparison controls whether the last two inputs are copied to an output register. Another qubit, storing whether the output is empty, controls the output check circuit.

The previous algorithm ensures that the collision is copied out. Once it is is copied out, another circuit applies g to check if this is the golden collision, and if so, copies out the result and increments the golden collision counter.

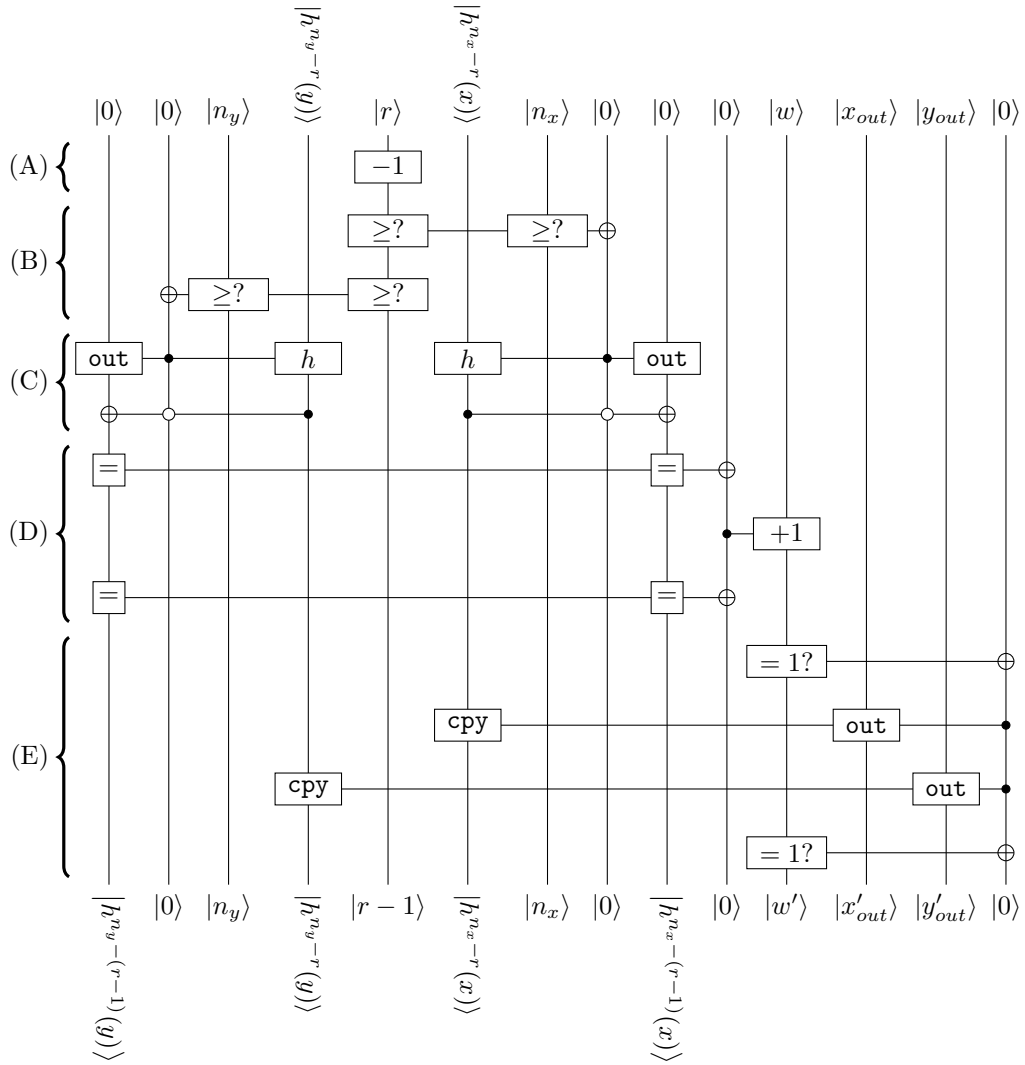


Figure 7.5: Circuit for one iteration of retracing a collision. (A) decrements the countdown r (initialised to $\max\{n_x, n_y\}$). (B) checks if n_x (resp. n_y) is at least as large as r. (C) computes h of the input if the check from (B) was true (writing the result where `out` is), and just copies the input otherwise. (D) checks if there is a collision, incrementing the counter w if so (w is initialised to 0). (E) checks if the counter is exactly 1, meaning it is the first iteration that found the collision, and copies the two inputs to specified output registers if so (denoted by `cpy` for the input and `out` for the output). After cn iterations of this circuit (appropriately pebbled), Eve checks if the output contains the golden collision.

History independence. The circuit above is not yet history independent, since a history dependent data structure must also reliably remove the golden collision if either one of the predecessors is removed. If there are more than 2 predecessors, and they are removed in a different order than they are added, this creates a problem.

This is why Eve needs to handle multiple collisions in a single vertex properly. When there are multiple collisions, she needs a special way to check for the golden collision. The rule from [JS21] is:

- If there is exactly one candidate collision y, check for a golden collision and mark the vertex if so.
- If there are exacty two candidate collisions y_1 and y_2, check for the golden collision *between* y_1 *and* y_2, and mark the vertex if there is such a collision.
- If there are more than two candidate collisions, do not check for the golden collision.

Here, "mark the vertex" means flip a flag qubit. This means if the flag qubit is already $|1\rangle$, it flips back to $|0\rangle$.

The end result is that this is history independent, and a vertex will contain an output (i.e., be "marked") if and only if it has exactly one predecessor of each side of the golden collision. I will show by induction that this holds; Figure 7.6 illustrates this proof.

If there is at most one predecessor of the golden collision, and there has never been more, than this definitely is true.

So suppose Eve has one predecessor y of the golden collision (say it is a predecessor of y_g), and she adds another element x. There is no problem if it is not a predecessor of the golden collision at all, so suppose it is a predecessor of x_g or y_g. In this case there will be exactly one candidate collision, so Eve will check the collision. If x is a predecessor of x_g, Eve will find the golden collision and update the counter. The vertex becomes marked, as it should. If x is a predecessor of y_g, the collision is not golden and the vertex remains unmarked.

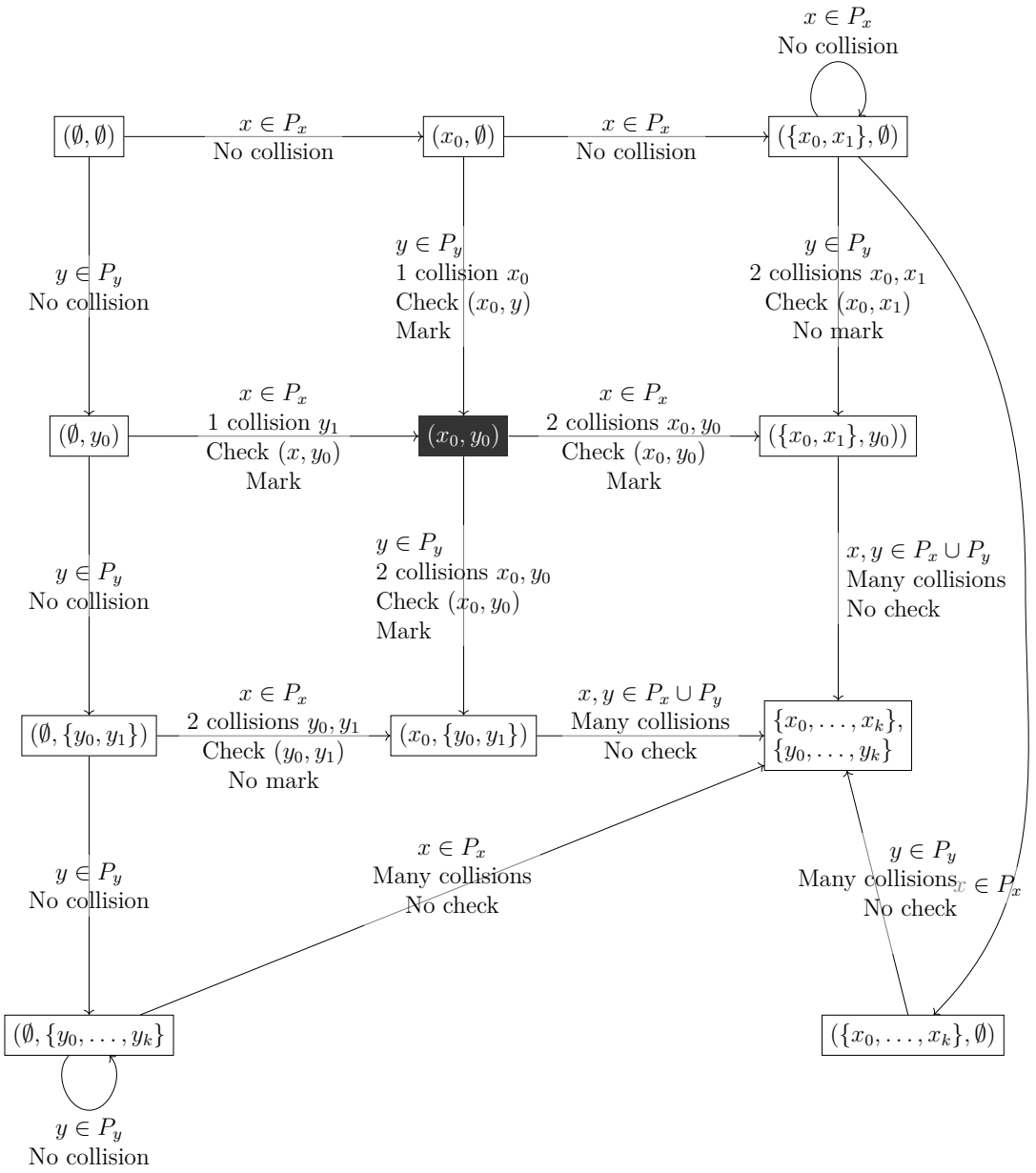


Figure 7.6: State diagram of vertices. Each vertex is labelled by the predecessors it has of x_g and y_g, with edges indicating whether a new predecessor of x_g or y_g is added (letting P_x denote predecessors of x_g and P_y for y_g). The logic of which collisions to check and when to mark the vertices is given in the text. This diagram shows that this will correctly mark only vertices with exactly one predecessor of each half of the golden collision.

Now suppose she has two predecessors of y_g, but none of x_g. As before, adding elements which are not predecessors of x_g creates no problem: there will be no golden collision, so the vertex will remain unmarked. If she adds a predecessor of x_g, she finds two candidate collisions y_1 and y_2. She thus checks for a collision between y_1 and y_2. Finding none, she does not flip the flag qubit, and inserts x. Now the vertex has 3 predecessors of the golden collision and is not marked, as expected.

If she has more than two predecessors of y_g, either there is no golden collision or there are at least 2 matches with a collision, so it will stay unmarked, as required.

Finally, suppose Eve has one predecessor of x_g and one of y_g already in the set, meaning that it is marked. If a newly inserted element x has a collision with these, it is another predecessor of the golden collision, so the set should become unmarked. This will happen because there are two collisions with the new element, so Eve will run the circuit to check if the existing elements form the golden collision. Since they do, she flips the "marked" flag, setting it back to $|0\rangle$.

Unmarking vertices with more predecessors of the golden collision decreases the value ϵ somewhat, but negligibly, since it is highly unlikely to select such points.

Update Cost. Putting this all together, we see that the gate cost to compute $(h^n(x), n)$ from x is $O(\mathsf{H}/\theta)$, using $O(\log(1/\theta))$ quantum memory, where H is the cost to compute h. The check for a collision then costs $O(\mathsf{H}/\theta + \mathsf{G})$ (where G is the cost to compute g, the "goldenness" check); even though everything is controlled, this adds only a constant increase. Finally, Eve pays a cost $\tilde{O}(R)$ for the memory operations to actually add $(x, h^n(x), n)$ to the list.

The final update cost is

$$\mathsf{U} = \tilde{O}\left(\frac{\mathsf{H}}{\theta} + R + \mathsf{G}\right) \tag{7.6}$$

I will assume that G is in $O(\mathsf{H}/\theta)$, to ignore it in the future.

Algorithm 9 summarises the golden collision finding algorithm with iteration.

Algorithm 9 Iteration-based golden collision finding

Input: Functions $h : X \to X$ and $g : X^2 \to \{0, 1\}$ from a golden collision problem
Output: The golden collision (x_g, y_g)

Run **QAA**:
  Diffusion:
    Select a random function $h' : X \to X$
    Set $\tilde{h} = h' \circ h$
  Check:
    Apply a **Quantum Random Walk**:
      Vertices: Subsets of X of size R
        Data structure: A list of R pairs $(x, \tilde{h}^{n_x}(x), n_x)$, ordered by $\tilde{h}^{n_x}(x)$
          A "marked" flag c
      Marked vertices: Sets that contain exactly one predecessor of x_g and one of y_g (the two halves of the golden collision)
      Update of vertex S:
        Select random y from S
        Adjoint **Iterate-and-Insert**(y, S)
        Select a random $x \in X$
        Iterate-and-Insert(x, S) (see Algorithm 10)
      Setup:
        Set $S = \emptyset$, $c = 0$
        for $i = 1$ to R **do**
          Select random $x \in X$
          Iterate-and-Insert(x, S)
        end for
      Check:
        if $c = 1$ **then**
          Mark vertex
        end if
    Flip the phase if $c = 1$ for the set S output by the random walk

7.2.3 Fraction of marked vertices

The number of steps of the quantum random walk is proportional to $\frac{1}{\sqrt{\epsilon}}$, where ϵ is the fraction of marked vertices (i.e., those that find the golden collision). To find this probability, I define *predecessors*:

Definition 7.2.1. *For a function h and an element $x \in X$, the set of* predecessors *of x is*

$$\mathcal{P}_x = \{y \in X | h^n(y) = x, n \geq 0\}. \tag{7.7}$$

I let $P_x = |\mathcal{P}_x|$. Eve needs the golden collision, x_g and y_g, to have both P_{x_g} and

Algorithm 10 Iterating and inserting new elements into a set in a random walk.

Iterate-and-Insert (x, S):
1: **for** $i = 1$ to u **do**
2: Apply the circuit from Figure 7.3, pebbled
3: **end for**
4: Let $\tilde{h}^{n_x}(x)$, n_x be the output of the previous loop
5: Find all $\{z_1, \ldots, z_k\} \in S$ such that $\tilde{h}^{n_z}(z) = \tilde{h}^{n_x}(x)$
6: **if** $k = 1$ or $k = 2$ **then**
7: Set $y_1 = z_1$
8: If $k = 1$ set $y_2 = x$; else $y_2 = z_2$
9: Set $r = \max\{n_{y_1}, n_{y_2}\}$
10: **for** $i = 1$ to u **do**
11: Apply the circuit from Figure 7.5, pebbled
12: **end for**
13: Let $\tilde{x}_G$ and $\tilde{y}_G$ be the outputs of the loop
14: Write $g(\tilde{x}_G, \tilde{y}_G)$ to the flag c
15: Uncompute all previous steps
16: **end if**
17: Insert $(x, \tilde{h}^{n_x}(x), n_x)$ into the list in S

P_{y_g} large. In this section I show that the probability that this happens for a random function. Recall that Eve tries composing h with many other functions h' to create new random functions. The golden collision remains as a collision of $h' \circ h$, but this changes the sets $\mathcal{P}_{x_g}$ and $\mathcal{P}_{y_g}$. We want to know how many functions Eve must try.

The proofs of all of the following claims are in Appendix F.

Lemma 7.2.1 (Lemma B.2, [JS21]). *Fix distinct $x, y \in X$, and let h be a random function under the restriction that $h(x) = h(y)$. Then for $t, s \in o(N)$,*

$$Pr[P_x = t, P_y = s] = \frac{t^{t-1}}{e^t t!} \frac{s^{s-1}}{e^s s!} \left(1 + O(\tfrac{1}{N})\right) \tag{7.8}$$

and the probability that x and y both have at least t predecessors is $\Theta(1/t)$.

Having many predecessors of the function is useless to Eve if they are all in a huge line, since she detects the collision by iterations. I actually need to know how many predecessors are within a fixed distance of the golden collision. It turns out they are fairly close, with high probability:

Lemma 7.2.2 (Lemma B.3, [JS21]). *Let $x, y \in X$ and let n_x be the largest integer such that there there is some point $p \in X$, $p \neq x$, with n_x as the minimum value*

such that $h^{n_x}(p) = x$. *Define* n_y *similarly. Suppose* x *has* t *predecessors and* y *has* s *predecessors. For any* $c > 1$, *the probability that* $n_x > c\sqrt{\tau t}$ *or* $n_y > c\sqrt{\tau s}$ *is at most*

$$\frac{\tau - 6}{3(c-1)^2}\left(1 + O(\tfrac{s}{N}\right). \tag{7.9}$$

Contrasting with [JS21], here I require $p \neq x$, so that if x is in a cycle, the height will be the length of the cycle. This is because Eve is actually fine with cycles as long as they are short enough: she will still detect the golden collision.

Lemma 7.2.2 tells us that Eve does not need to worry much about the predecessors being too far away, as long as there are not too many predecessors. However, what if x has far too many predecessors? Luckily, the distribution from Lemma 7.2.1 shows that this probability decreases quickly.

This gives the theorem I need for the runtime of the random walk:

Theorem 7.2.1 (Theorem B.3, [JS21])**.** *Suppose* h *is a random function such that* x *has at least* t *predecessors and* y *has at least* s *predecessors. Then with probability at least*

$$\frac{\tau - 6}{3(c-1)^2}\left(1 - O(\tfrac{\sqrt{t}+\sqrt{s}}{u})\right), \tag{7.10}$$

the fraction of marked vertices is at least

$$\Omega\left(\frac{R^2 \min\{u^2, t\} \min\{u^2, s\}}{N^2}\right). \tag{7.11}$$

Finally, there is some risk that Eve will not detect a golden collision at all. This could happen if the distinguished point after the golden collision is too far. However, the same geometric distribution argument says that if I split the number of steps u as $u = u_1 + u_2$, then I can use u_1 in Theorem 7.2.1 and claim that the probability of needing more than u_2 steps after the golden collision is at most $e^{-\frac{u_2}{\theta}}$. Overall, Eve still requires only $O(1/\theta)$ steps.

7.2.4 Summary

To summarise the costs and steps that Eve performs, I know that the update cost from Equation (7.6) is

$$\tilde{O}\left(\frac{\mathsf{H}}{\theta} + R\right). \tag{7.12}$$

From Theorem 7.2.1, if x_g and y_g have t and s predecessors (respectively), then with probability $\Omega(1)$ the fraction of marked vertices is

$$\Theta\left(\frac{R^2 \min\{\frac{1}{\theta^2}, t\} \min\{\frac{1}{\theta^2}, s\}}{N^2}\right) \tag{7.13}$$

Thus, for such an h, the cost of the random walk is

$$\tilde{O}\left(\underbrace{R\left(\frac{\mathsf{H}}{\theta} + R\right)}_{\mathsf{S}} + \underbrace{\frac{N\theta^2}{R}}_{1/\sqrt{\epsilon}} \sqrt{R} \underbrace{\left(\frac{\mathsf{H}}{\theta} + R\right)}_{\mathsf{U}}\right) \tag{7.14}$$

Here I have cost the setup as the same as R sequential updates; there are more cost-efficient methods but it will not matter in the end.

However, the overall cost only holds if $t, s \geq 1/\theta^2$. The probability of this occuring is $\Theta(\theta^2)$, by Lemma 7.2.1. Thus, when Eve uses the quantum random walk as the oracle for QAA, searching for the random function h' to compose with h, she will need $O(\frac{1}{\theta})$ iterations. This gives a total cost of

$$\tilde{O}\left(\frac{1}{\theta}\left(\frac{R\mathsf{H}}{\theta} + R^2 + \frac{N\theta\mathsf{H}}{\sqrt{R}} + N\theta^2\sqrt{R}\right)\right). \tag{7.15}$$

This is optimised with $\theta = \frac{\mathsf{H}}{R}$ and $R = N^{2/7}\mathsf{H}^{4/7}$, for a total *gate* cost of

$$\tilde{O}\left(N^{6/7}\mathsf{H}^{5/7}\right). \tag{7.16}$$

7.3 Outlook and Comparison

First, let us consider the merits of this algorithm as a cryptographic attack.

If Eve wants to keep the same gate cost, the runtime of the algorithm is *also* $N^{6/7}$, since

- iteration is impossible to parallelise;
- QAA parallelises badly;
- quantum random walks parallelise badly.

Our paper [JS21] discusses parallelism in more detail, but suffice it to say this algorithm will do a bad job of it.

Moreoever, the total memory requirement is $\Theta(R)$, which is $N^{2/7+o(1)}$. This means that if we consider DW cost, the cost would be $N^{8/7+o(1)}$, which is more than basic QAA!

Thus, this attack is only of interest in a case where Eve has large amounts of time and a fairly large array of passively-corrected memory. However, she has managed to escape the restrictions on QRAQM in Chapter 4: this analysis accounts for a linear access cost to the quantum memory and still manages a useful algorithm.

In fact, the gate cost is lower than QAA, the best known quantum attack without significant quantum memory. Thus, this algorithm provides some interesting points in computational complexity: it shows that there is a problem with an exponential separation between the cost with and without passively-corrected quantum memory, and shows that passively corrected quantum memory is a useful resource, even with linear access costs.

7.3.1 Comparison with QAA

Here I present André Schrottenloher's algorithm as Algorithm 11, which formed a complementary section in our paper [JS21]. It uses a simpler method to find distinguished points: use QAA directly. That is, Eve searches for points x such that $h(x)$ is distinguished. Recall that "distinguished" means that $h(x)$ begins with $\lg(1/\theta)$ bits that match a fixed string s. This creates three changes compared to the iteration-based search:

1. Rather than searching for a random function where the golden collision has many predecessors, Eve now searches for different strings s to define

Algorithm 11 QAA-based golden collision finding

Input: Functions $h : X \to X$ and $g : X^2 \to \{0, 1\}$ from a golden collision problem
Output: The golden collision (x_g, y_g)

Run **QAA**:
  Diffusion:
    Select a random function prefix s of ℓ bits
    Define $X_s = \{x \in X | h(x)$ matches s in ℓ bits$\}$
  Check:
    Apply a **Quantum Random Walk**:
      Vertices: Subsets of X_s of size R
        Data structure: A list of R pairs $(x, \tilde{h}(x))$, ordered by $\tilde{h}(x)$
          A "marked" flag c
      Marked vertices: Sets that contain the golden collision
      Update of vertex S:
        Adjoint **QAA-and-Insert**(s, S) (see Algorithm 12)
        QAA-and-Insert(y, S)
      Setup:
        Set $S = \emptyset$, $c = 0$
        for $i = 1$ to R **do**
          QAA-and-Insert(s, S)
        end for
      Check:
        if $c = 1$ **then**
          Mark vertex
        end if
    Flip this state if $c = 1$ for the set S output by the random walk

Algorithm 12 Inserting new elements into a set in a random walk, using QAA

QAA-and-Insert A set S, a string s:

Run **QAA**:
  Diffusion:
    Select a random point $x \in X$
  Check:
    Compute $h(x)$
    Flip phase if $h(x)$ is distinguished (the first bits of $h(x)$ match s)

"distinguished", such that the output of the golden collision is distinguished. This has probability θ, which is much more likely than the $\frac{1}{\theta^2}$ of the iteration-based search.

2. The cost to find such points with QAA is $O(\frac{1}{\sqrt{\theta}})$ calls to h, compared to $O(\frac{1}{\theta})$ to iterate h (and QAA requires no tricky pebbling or random function analysis).

3. Once Eve guesses the correct prefix, she has effectively reduced the search space from N to $N\theta$, so the fraction of marked vertices is $\frac{R^2}{N^2\theta^2}$, compared to $\frac{R^2}{N^2\theta^4}$ with iterations.

Intuitively, think of it like this: iterating points concentrates them much more on a select number of high-predecessor distinguished points. Lemma 7.2.1 is the crux: the probability of having more than t predecessors only shrinks as $O(1/t)$, but the chance of picking one of those predecessors grows proportional to t^2. Thus, it is easier to pick a random function which amplifies the golden collision substantially, than it is to pick a prefix matching the golden collision. This also means that searching for distinguished points by iteration will produce "higher value" distinguished points than just guessing-and-checking.

Classically, the cost of finding distinguished points is the same by iterating or guessing-and-checking, so it makes more sense to iterate and get higher-value points. The iteration-based approach is the de facto standard for these kinds of problems. In the quantum case, guessing-and-checking gets the advantage of using QAA. This advantage precisely cancels out the advantage from iteration-based distinguished points.

This seems like a complete coincidence to me. To justify this claim, if we wanted "3-collisions", i.e., x, y, z such that $h(x) = h(y) = h(z)$ (and $g(x, y, z) = 1$ for some "goldenness" function), then the fraction of marked vertices decreases to $\Theta(\frac{R^3}{N^3\theta^6})$ for the iteration-based walk[2] and to $\Theta(\frac{R^3}{N^3\theta^3})$ for the QAA-based walk. The probability of finding a function with sufficient predecessors decreases to

[2]I will not derive this, but it should be clear from the previous lemmas and theorems.

$\Theta(\theta^3)$, while the probability of the right distinguished-point-prefix-string is still θ. For both approaches, the cost of each step is the same up to constant factors. Overall, the costs become

$$\underbrace{\tilde{O}\left(\frac{R}{\theta^{3/2}}\left(R+\frac{\mathsf{H}}{\theta}\right)+\frac{N^{3/2}\theta^{3/2}}{R}\left(R+\frac{\mathsf{H}}{\theta}\right)\right)}_{\text{iteration-based}} \tag{7.17}$$

vs.

$$\underbrace{\tilde{O}\left(\frac{R}{\theta^{1/2}}\left(R+\mathsf{H}\theta\right)+\frac{N^{3/2}\theta}{R}\left(R+\frac{\mathsf{H}}{\theta}\right)\right)}_{\text{QAA-based}}, \tag{7.18}$$

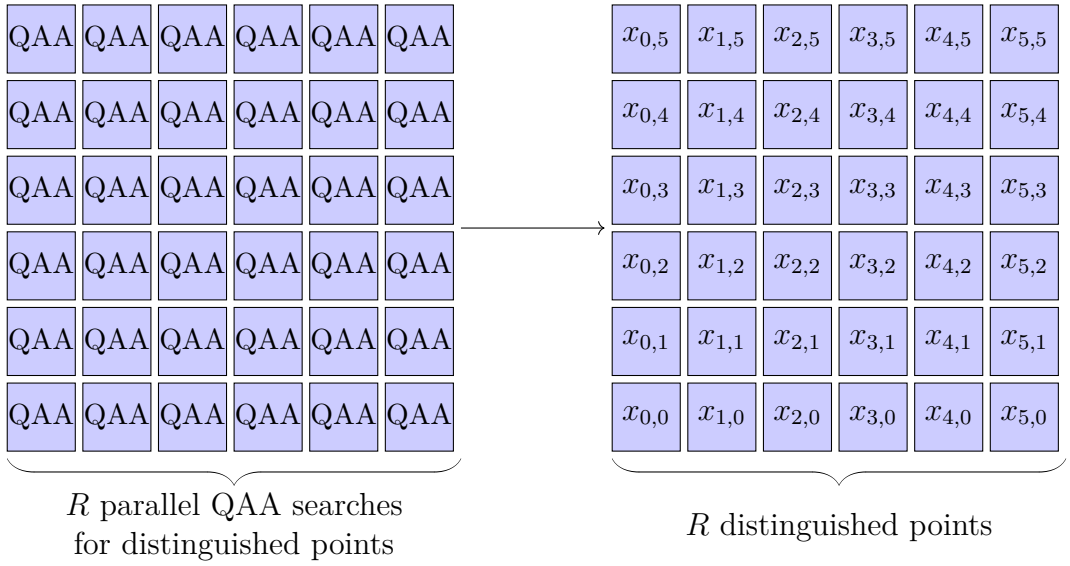
showing an advantage of θ for the iteration-based walk. The iteration-based walk can reach a gate cost of $N^{\frac{21}{20}+o(1)}$ and the QAA-based walk gets to $N^{\frac{15}{14}+o(1)}$. A small difference, but it suggests there is no deeper reason for the equal costs for a golden 2-collision.

7.4 Parallel Search

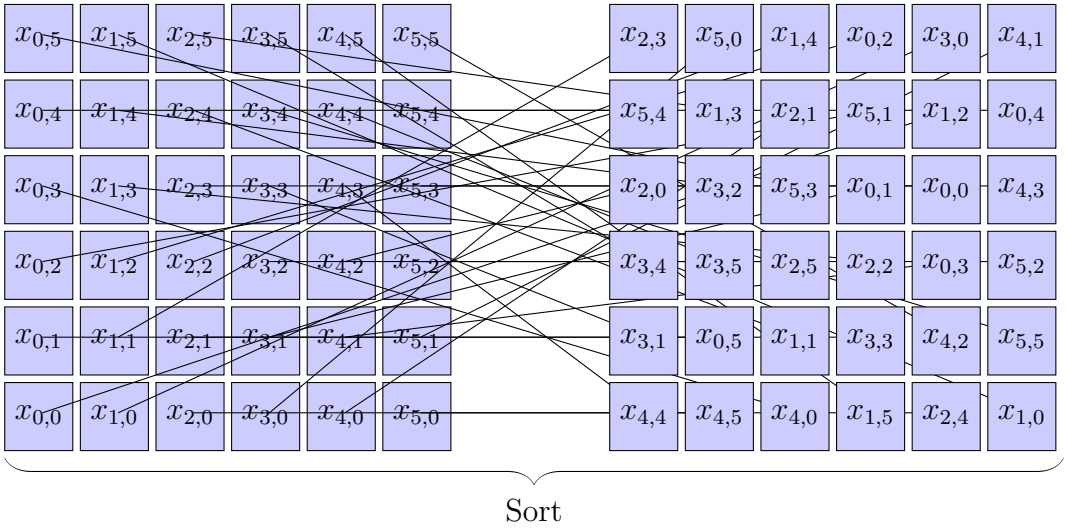
Neither the iteration-based or QAA-based walk parallelise well. Thus, in [JS21] we used the prefix approach with the "multi-Grover" search of [BBG+13].

In brief, if Eve were able to do a QAA search directly over subsets of size R, then this gives almost the same cost as a random walk (Equation (7.3)), *except* the extra factor of $\sqrt{R}$ disappears. This factor is the square root the spectral gap of the underlying graph; you do not need to know anything about it, though. However, a pure QAA complicates the update. If Eve replaces all elements in the set in each iteration, she needs to check for all collisions from scratch. The quantum random walk saves cost by just comparing with one new element at a time.

If Eve has R quantum processors (which, in the memory peripheral models I am using, is equivalent to $\Omega(R)$ qubits of memory), then each processor can select a new element x and compute $h(x)$, and then they can act as a *sorting network* and sort all of this data. Once sorted by $h(x)$, each processor compares to its neighbours and checks for the golden collision. Then they can unsort themselves and assemble the result (including the collision itself).



(a) First step: each machine runs a QAA for pre-images of distinguished points. This is actually the diffusion operation of the larger QAA, and the remaining steps form the check operation.



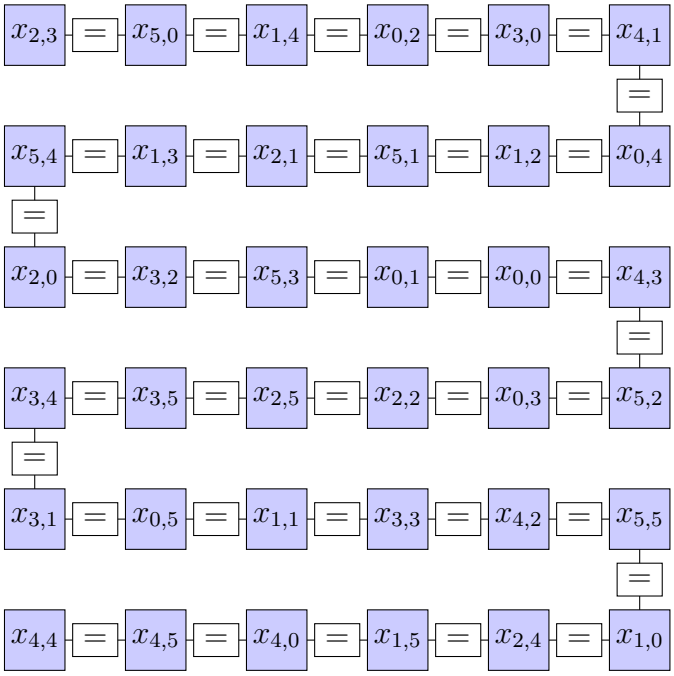
(b) Second step: the processors sort themselves based on the distinguished points

.

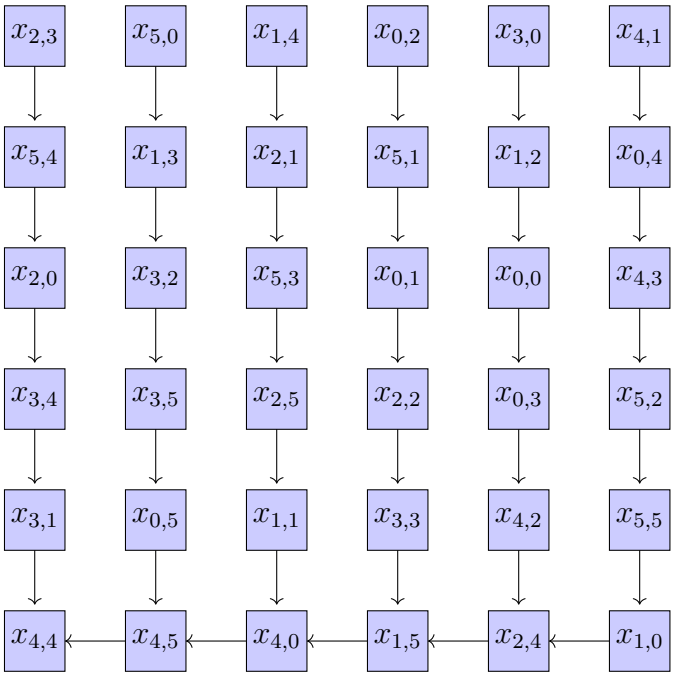
Figure 7.7: The first steps of parallel quantum claw-finding, continued in Figure 7.8.

This is the basic method of [BBG+13], and in [JS21] we combined that with a prefix-based search. Once Eve selects a prefix, she runs a QAA search for the golden collision among preimages of points with that prefix, using the sort-and-compare of [BBG+13]. To find such points, each of the R processors will run its own QAA as a subroutine to find x such that $h(x)$ has the right prefix. Figure 7.7 and Figure 7.8 show this.

This entire process gets wrapped in *another* QAA to find the right prefix. That



(a) Each processor compares its distinguished point to its successor's point. Since they are sorted, this detects any equalities. Each comparison will also check if it is the golden collision.



(b) Each processor sends any golden collision it found to a neighbour, in such a way that the golden collision ends up at some fixed location (here, the bottom left).

Figure 7.8: The later steps of parallel quantum claw-finding, continued from Figure 7.7. Eve does a QAA over subsets of R distinguished points, searching for a subset containing the golden collision. The diffusion operator is the initial QAA in Figure 7.7a, while the remaining steps form the check operator. Once this larger QAA finishes, the state of the processors will be a superposition of sets that contain the golden collision.

is, there are three layers of nested QAA searches. Algorithm 13 details the process.

Algorithm 13 Parallel golden collision finding

Input: Functions $h : X \to X$ and $g : X^2 \to \{0, 1\}$ from a golden collision problem
Output: The golden collision (x_g, y_g)

```
Run QAA:
  Diffusion:
    Select a random prefix s
  Check:
    Run QAA:
      Diffusion:
        for all processors do
          Run QAA:
            Diffusion:
              Select random x ∈ X
            Check:
              Compute h(x)
              Flip phase if h(x) starts with the prefix s
        end for
      Check:
        Each processor has a check flag c_i = 0
        Processors sort their data by h(x_i)
        for i = 1 to P do
          Processor i compares (x_i, h(x_i)) to processor i + 1
          if x_i ≠ x_{i+1} and h(x_i) = h(x_{i+1}) then
            if g(x_i, x_{i+1}) = 1 then
              Set c_i = 1
            end if
          end if
        end for
        Compute a flag c as the OR of all c_i
        Flip phase if c = 1
```

Overall, the gate cost with R processors is

$$\tilde{O}\left(\frac{1}{\sqrt{\theta}}\frac{N\theta}{R}\left(R\frac{\mathsf{H}}{\sqrt{\theta}} + S_G(R)\right)\right) \tag{7.19}$$

where :

- $S_G(R)$ is the cost to sort;
- $R\frac{\mathsf{H}}{\sqrt{\theta}}$ is the cost for all R processors to use QAA to find distinguished points;

- $\frac{N\theta}{R}$ is the cost of the second layer of QAA to find the right set of distinguished points;
- $\frac{1}{\sqrt{\theta}}$ is the number of iterations of the final QAA to find the right prefix.

The depth is

$$\tilde{O}\left(\frac{1}{\sqrt{\theta}}\frac{N\theta}{R}\left(\frac{\mathsf{H}}{\sqrt{\theta}}+S_D(R)\right)\right) \tag{7.20}$$

You can see that the cost will depend substantially on the cost to sort. An optimistic case is where $S_G(R) = O(R\log R)$ and $S_D(R) = O(\log R)$. If Eve has all-to-all connectivity, or even hybercube connectivity, and no latency, then she can achieve these costs.

In this case the gate cost is (up to logarithmic factors) *independent of θ and R.* It is always $N^{1+o(1)}\mathsf{H}$. Similarly, as long as $\frac{\mathsf{H}}{\sqrt{\theta}} \in O(\log R)$, the depth is $N^{1+o(1)}/R$. Thus, this parallelises perfectly and can proceed essentially without distinguished points. [BBG+13] describe essentially this.

7.4.1 Latency

The problem with the previous algorithm is that at the sizes of parallelism where the algorithm might be fast enough to use, we cannot ignore latency.

Instead, let us assume a two-dimensional mesh of processors. In this case $S_G = R^{3/2+o(1)}$ and $S_D = R^{1/2+o(1)}$. Now, the gate cost is minimised when $\frac{\mathsf{H}}{\sqrt{\theta}} = \sqrt{R}$, so we set $\theta = \frac{1}{R}$. The gate cost becomes $N^{1+o(1)}$ in this case, and the depth remains $N^{1+o(1)}/R$. Perfect parallelisation! The sort makes it more expensive to check for collisions, so spending extra time to search for distinguished collisions balances that cost and ensures that collisions are more useful.

There is a slight catch: The number of iterations of the QAA over sets is not actually $O(\frac{N\theta}{R})$, it is $O(\max\{\frac{N\theta}{R}, 1\})$. That is, if Eve expects essentially every set to contain the golden collision, she must still iterate through the set to find it. If she sets $\theta = \frac{1}{R}$, then this means if $R \geq N^{1/2}$ she runs into this problem. In fact, in this case she expects each list to contain every distinguished point!

Thus, this perfect parallelisation only works down to a runtime of $N^{1/2+o(1)}$.

Beyond this, the best Eve can do is to parallelise the QAA by doing things like dividing the search space of prefixes. This increases the gate cost and decreases the depth by the number of parallel machines. With P parallel machines ($0 \leq P \leq \frac{1}{\theta}$), the costs are

$$\tilde{O}\left(\frac{\sqrt{P}}{\sqrt{\theta}}\left(R\frac{\mathsf{H}}{\sqrt{\theta}} + R^{3/2}\right)\right) \text{ gates and } \tilde{O}\left(\frac{1}{\sqrt{P\theta}}\left(\frac{\mathsf{H}}{\sqrt{\theta}} + R^{1/2}\right)\right) \text{ depth} \qquad (7.21)$$

which can be minimised to $\tilde{O}(N\sqrt{P})$ gates and $O(N^{1/2}/\sqrt{P})$ depth (recall that each of these P "machines" has $R = N^{1/2+o(1)}$ qubits). Notice that this still has a minimum depth of $N^{1/4+o(1)}$.

To get even lower depths, each of the QAA searches for distinguished points can parallelise. With P' processors (for each of the R parallel QAA searches) this increases the gate cost from $\tilde{O}(R\frac{\mathsf{H}}{\sqrt{\theta}})$ to $\tilde{O}(R\frac{\mathsf{H}}{\sqrt{\theta}}\sqrt{P'})$ and decreases the depth to $\tilde{O}(\frac{R}{\sqrt{P'}}\frac{\mathsf{H}}{\sqrt{\theta}})$. The overall cost jumps to $N^{3/2+o(1)}\sqrt{P'}$ gates and depth of $N^{1/4+o(1)}/\sqrt{P'}$.

Of course, there is also a hard limit on time proportional to the square root of the memory size, simply for the answer to propagate from one side of the entire machine to the other. The constant factor for this might be negligible compared to other factors I am considering, so that is why I consider these extreme parallelisations separately.

7.4.2 Iteration-based

Originally we ignored the prospects of an iteration-based search within the multi-Grover, mainly because the memory per iteration was $\Theta(1/\sqrt{\theta})$ for a constant-overhead pebbling scheme. However, with spooky pebbling, the memory can decrease to $O(\log(1/\theta))$ – does that save it?

Notice that the *quantum* memory for spooky pebbling scales as $O(\log(1/\theta))$, but the *classical* memory scales as $\Omega(1/\theta)$. Generally, we cannot hope for a spooky pebbling strategy where the sum of classical and quantum memory is less than a reversible classical pebbling strategy: since every ghost pebble has

some classical memory, we would be best served by using a real pebble instead of a ghost pebble at all points.

Thus, with R processors and distinguished points occuring with frequency $1/\theta$, the total hardware is $R/\sqrt{\theta}$ either way. The *cost* of a 2-dimensional sort is still $O(R^{3/2})$, but the time gets larger: the signal latency to send messages between processors is proportional to the square root of the memory size, so the time for such a sort is $\tilde{\Theta}\left(\left(\frac{R}{\sqrt{\theta}}\right)^{1/2}\right)$.

As in the quantum random walk, it takes each processor $O(\frac{\mathsf{H}}{\theta})$ operations to find distinguished points. The probability of a set containing a distinguished point is $\Theta(\frac{R^2}{N^2\theta^4})$, as before, and we must search $O(1/\theta^2)$ different random functions. Putting this all together, the gate cost is

$$\tilde{O}\left(\frac{1}{\theta}\frac{N\theta^2}{R}\left(R\frac{\mathsf{H}}{\theta}+R^{3/2}\right)\right) \tag{7.22}$$

and this is optimised with $\theta = R^{-1/2}$ (instead of $\theta = R^{-1}$), taking the cost to $N^{1+o(1)}$. The depth is

$$\tilde{O}\left(\frac{1}{\theta}\frac{N\theta^2}{R}\left(\frac{\mathsf{H}}{\theta}+\frac{\sqrt{R}}{\theta^{1/4}}\right)\right) \tag{7.23}$$

and when we set $\theta = R^{-1/2}$ this becomes $N^{1+o(1)}/R^{7/8}$. This is worse parallelisation, precisely because of the extra memory for pebbling. If we can ignore signal latency, then this will match the QAA approach.

The minimum depth can be reached when $\frac{N\theta^2}{R} = O(1)$, which (since $\theta^2 = 1/R$) also occurs when $R = \sqrt{N}$.

Thus, using iterations parallelises worse and suffers the same limitations as the prefix-based approach.

7.5 Discussion

The original motivation for this work was to attack the post-quantum protocol SIKE, but since it is completely broken classically, I will consider the general case.

Many cryptographic attacks are known as "galactic algorithms", meaning the parameters are so large that the amount of memory and the runtimes are literally on an astronomical scale.

For example, suppose $N = 2^{256}$ (so that the golden collision problem would have 256 "bits of security"). Then to reach a depth of even 2^{128}, Eve would need at least 2^{128} processors[3]. If each processor weighs 1 milligram, then this would be over a hundred times heavier than our sun.

My interpretation of such results is that golden collision finding with $N = 2^{256}$ is absolutely safe from known attacks. There is no hardware advance that will make this feasible. In fact, even the query lower bound of $\Omega(N^{2/3})$ means an attack would need something on the order of 2^{170} operations, which is hopeless (see Section 3.3). Thus, the algorithmic threat is that the function h can be attacked in a non-black-box way. That is precisely what happened to SIKE.

For smaller N, such as 2^{128}, the conclusions are murkier. If we decide that 2^{128} is our security threshold, then we may not be safe, because the query lower bound for $N = 2^{128}$ is 2^{85}. Yet, the best known gate cost is 2^{107} (the iteration-based quantum random walk of Section 7.2). Should we be concerned? Probably not, since it cannot parallelise well. 2^{107} sequential operations is unbelievably long; if Eve wanted the attack to finish in 100 years, then each operation must take only 2^{-75} seconds. In turn this implies an upper bound on qubit size, since light can only travel 2^{-47} m in that time, or 58 micro-Angstroms (see Figure 7.9).

This implies that we should only consider low depth parallel algorithms, like in Section 7.4. Yet, this brings the gate cost up to N, so $N = 2^{128}$ would be safe from *known* attacks. Is that still too high, though? Recalling NIST's depth limits from Section 2.3.1, as 2^{40}, 2^{64} and 2^{96}, let us consider the limit 2^{40}. To parallelise to that depth with $N = 2^{128}$, Eve would need to break into the more expensive parallelisation, since $N^{1/2} > 2^{40}$. In fact, with $N = 2^{112}$, parallelising to depth 2^{40} brings the gate cost up to 2^{128}.

[3]Here and throughout this section I will treat the constant factors as 1

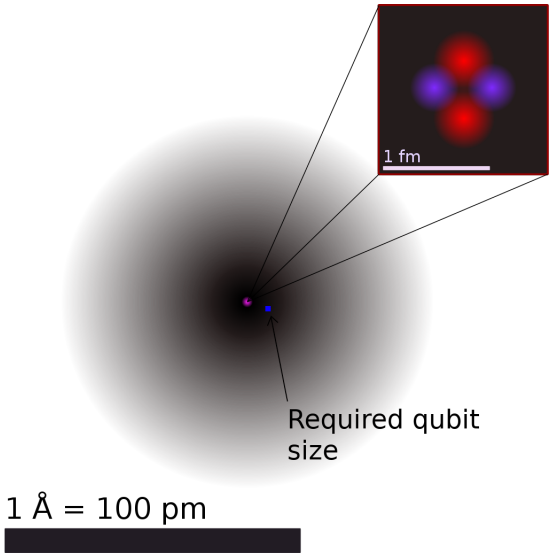


Figure 7.9: A qubit of 2^{-47} m, with a helium atom and nucleus (from [Yzm07]) for scale.

Beyond all of this, one must consider all of the constant or logarithmic factors that this chapter swept under the asymptotic notation, such as the cost to evaluate the function for the golden collision or the costs to do memory operations. This makes these algorithms even more dubious for cryptographic paramters, which is a good thing! For security, we want these problems to be hard, even for quantum computers.

In fact, the best classical algorithm for this problem has gate complexity $\frac{N^{3/2+o(1)}}{\sqrt{M}}$ with memory M such that $M \leq N$ [vOW99]. It also parallelises perfectly. With the $N^{2/7}$ memory of the gate-optimised quantum random walk, the classical algorithm has a cost of $N^{19/14+o(1)}$, still fairly high. With a hardware budget of $N^{1/2}$ (equal to the memory to reach the maximum parallelisation of the parallel QAA), the classical algorithm has gate complexity $N^{5/4+o(1)}$, compared to $N^{1+o(1)}$ for quantum. For something like $N = 2^{128}$, this is a difference of only 2^{32} (playing fast and loose with asymptotics). If each quantum operation has some overhead compared to classical, this shrinks the difference even further.

Overall, in a far future scenario where classical and quantum computers cost nearly the same, and adversaries with trillions of qubits are real enough to cause us concern, these quantum algorithms provide *some* advantage over classical algorithms,

but the underlying problem is still hard enough to form a solid basis for post-quantum cryptography.

8
Conclusion

Cryptography can never be unconditionally proven secure. Our best hope is to narrow down the range of assumptions to a few, simple, and well-tested statements. Yet, they could still turn out to be false.

These assumptions are often algorithmic, but assumptions about hardware and engineering are critical too, and are sometimes implicit in the algorithmic assumptions. This is more obvious with quantum attacks. I want readers to always consider the following question:

Why would a specific engineering breakthough be more likely than a specific cryptanalytic breakthrough?

For example, in Chapter 5 I claimed that a quantum computer would need 2 million noisy qubits and 3 weeks of runtime to break elliptic curve cryptography (ECC). This is about 2^{13} times more qubits and ten times lower qubit error rates than qubits today [IBM23, Goo23], which should help make decisions about how and when to transition to post-quantum crypography.

However, I assumed particular physical properties of the qubits; if the error is 100 times lower, the attack could be done with 2 million qubits in 4 hours or 600,000 qubits with 5 months. Even further, if researchers solve the challenges of good LDPC codes mentioned in Chapter 3, perhaps one might need only a few thousand qubits.

Yet, an unexpected classical algorithm could break ECC tomorrow! Is that more likely than methods to circumvent fundamental challenges in these improved codes?

Predicting scientific advances (or lack thereof) is perilous, but we must, since we need to deploy cryptography right now for today's security. My goal in the first half of this thesis was to point out some particular assumptions about quantum computers that rely on *big* scientific advances.

Changing computational assumptions about an attacker can sometimes radically change the threat. For Shor's algorithm, switching from passive gate QRAM to circuit QRAM actually makes little difference, but the type of active quantum error correction matters greatly. To attack CSIDH-512 (Chapter 6), QRAM assumptions are again mostly irrelevant, but quantum algorithm advances in arithmetic are more impactful. Against CSIDH-4096, assumptions on QRAM, quantum arithmetic, and even the type of active quantum error correction have little impact on the threat of quantum attack. Here an attacker would need low-energy ballistic computation to be a true threat. This would be a tremendous discovery. Personally, I am more concerned about classical attacks using the isogeny structure or better generic algorithms to solve the vectorisation problem.

Finally, Chapter 7 showed that assumptions about gate speed and time budgets are critical to estimate the cost of a claw-finding attack. The stack of physical assumptions needed to realise these algorithms at cryptographic sizes – passive memory, ballistic computation, angstrom-scale qubits – seem far grander than a classical advance on any specific problem.

The message I want to give is that quantum computers will not be omnipotent. It is reasonable to take a restrained model of their capabilities, such as assuming actively-corrected memory and linear-cost QRAM. Of course, a cryptographer could take more generous assumptions. I cannot say that these are wrong, but making such assumptions needs to be an informed decision, and I hope the arguments I made help clarify the strength of such assumptions.

Appendices

A Hamiltonian Limits

A.1 Complexity of Long Hamiltonians

This section comes from [JR23].

Proposition A.1.1. *Suppose H_1 and H_2 are two Hamiltonians of norm at most 1 such that $\|H_1 - H_2\|_2 \leq \epsilon$. If*

$$\|e^{itH_1/\hbar} - e^{itH_2/\hbar}\|_2 \geq \delta \tag{A.1}$$

then

$$\epsilon \geq \frac{\hbar \ln(\delta e^{-t/\hbar} + 1)}{t} \geq \frac{\hbar}{t}\delta e^{-t/\hbar}(1 - \delta e^{-t/\hbar}). \tag{A.2}$$

I will introduce an unusual but helpful notation: for any natural numbers n, j with $j \leq n$, and two matrices A and B,

$$\binom{n : A}{j : B} := \sum_{\vec{x} \in \{0,1\}^n : \|\vec{x}\|_1 = j} B^{x_1} A^{1-x_1} B^{x_2} A^{1-x_2} \cdots B^{x_n} A^{1-x_n} \tag{A.3}$$

In other words, because A and B might not commute, we add up all arrangements of A and B that contain j terms of B and $n - j$ terms of A. This is useful because

$$(A + B)^n = \sum_{j=0}^{n} \binom{n : A}{j : B} \tag{A.4}$$

and if A and B commute, we have $\binom{n:A}{j:B} = \binom{n}{j} A^{n-j} B^j$.

Proof. Suppose that $H_1 - H_2 = \epsilon H_{12}$, where H_{12} has singular values at most 1. I can then argue that

$$\begin{aligned} e^{itH_1/\hbar} =& e^{itH_2/\hbar + it\epsilon H_{12}/\hbar} && \text{(A.5)} \\ =& \sum_{n=0}^{\infty} \frac{i^n t^n}{\hbar^n n!} (H_2 + \epsilon H_{12})^n && \text{(A.6)} \\ =& \underbrace{\sum_{n=0}^{\infty} \frac{i^n t^n}{\hbar^n n!} H_2^n}_{=e^{itH_2/\hbar}} + \sum_{n=0}^{\infty} \frac{i^n t^n}{\hbar^n n!} \sum_{j=1}^{n} \epsilon^j \binom{n : H_2}{j : H_{12}} && \text{(A.7)} \end{aligned}$$

If $\|H_1\| \leq 1$ and $\|H_2\| \leq 1$, I can bound

$$\left\| \binom{n, H_1}{j, H_{12}} \right\| \leq \sum_{\vec{a} \in \{0,1\}^n, \|\vec{a}\|_1 = j} 1 = \binom{n}{j}. \tag{A.8}$$

so we can bound

$$\begin{aligned} \delta \leq \|e^{itH_1/\hbar} - e^{itH_2/\hbar}\| \leq& \sum_{n=0}^{\infty} \frac{t^n}{\hbar^n n!} \sum_{j=1}^{n} \epsilon^j \binom{n}{j} && \text{(A.9)} \\ =& \sum_{n=0}^{\infty} \frac{t^n}{\hbar^n n!} \left((1+\epsilon)^n - 1 \right) && \text{(A.10)} \\ =& e^{t/\hbar}(e^{t\epsilon/\hbar} - 1). && \text{(A.11)} \\ & && \text{(A.12)} \end{aligned}$$

Rearranging gives the result. □

I now define precisely the kind of Hamiltonians I assume Eve might produce.

- The Hamiltonian H will be a linear combination of n terms H_i.
- Each of the terms H_i is chosen from a set of terms of size N, acting on at most W constant-size quantum systems.
- The coefficients in the linear combination are in $[-1, 1]$.
- For all i, $\|H_i\| \leq 1$.

While this restricts her to Hamiltonian terms with energy 1 J, I can scale down the Hamiltonians as long as I scale up the time. Thus, the time t in Theorem A.1.1 should really be thought of as $t'E$ for some real time t'.

Allowing coefficients in $[-1, 1]$ means Eve can choose all coefficients to be small. This is equivalent to choosing larger coefficients and evolving the system for less time. Thus, the theorem covers all choices of time evolution as well, not just a fixed time t.

Theorem A.1.1. *Let $\mathcal{G}$ be a set of mutually orthogonal circuits (i.e., for any $U_1, U_2 \in \mathcal{G}$, there is some input state $|\psi\rangle$ such that $\langle\psi| U_2^\dagger U_1 |\psi\rangle = 0$). Let $\mathcal{H}$ be a set of Hamiltonians constructed in the manner specified above for some n and N such that there exists some $t \in \mathbb{R}$ such that for all $U \in \mathcal{G}$, there is $H \in \mathcal{H}$ such that $U = e^{itH/\hbar}$. Then*

$$\log(|\mathcal{G}|) \in O(nt + n \log \mathsf{W}). \tag{A.13}$$

Proof. First I note that the singular values of $a_1 H_1 + \cdots + a_n H_n$ are bounded by ϵ if $\|a\|_1 < \epsilon$. If Eve chooses the same n Hamiltonians and two coefficient vectors $\vec{a}_1$ and $\vec{a}_2$ such that $\|\vec{a}_1 - \vec{a}_2\|_1 \leq \epsilon$, then the induced Hamiltonians H_1 and H_2 satisfy $\|H_1 - H_2\| \leq \epsilon$.

I then consider how many possible Hamiltonians Eve might be able to produce that are all mutually at distance at least 2ϵ. My goal here is an upper bound. Optimistically, I assume that if Eve chooses two different sets of Hamiltonians, she automatically gets the required distance. Then each choice (there are $\binom{N}{n}$ choices) partitions the space $[-1, 1]^n \subseteq \mathbb{R}^n$ into ℓ_1-balls of radius ϵ. The volume of the space is 2^n and the volume of each ball is $\frac{2^n}{n!}\epsilon^n$, so this gives her at most

$$\binom{N}{n} \frac{n!}{\epsilon^n} \tag{A.14}$$

such matrices.

If she wants to capture all these gates in the set $\mathcal{G}$ with a Hamiltonian, she thus needs

$$\binom{N}{n} \frac{n!}{\epsilon^n} \geq |\mathcal{G}| \tag{A.15}$$

for some ϵ. Rearranging gives

$$\epsilon \leq \left(\binom{N}{n} \frac{n!}{|\mathcal{G}|} \right)^{1/n} \tag{A.16}$$

However, if ϵ is small enough that $e^{t\hbar}(e^{t\hbar\epsilon} - 1) \leq 1$, then by Proposition A.1.1 the unitaries induced by her Hamiltonians are too close to represent these mutually orthogonal gates in $\mathcal{G}$. To put this another way, she needs

$$\epsilon > \frac{\hbar \ln(e^{-t/\hbar} + 1)}{t} \geq \frac{\hbar e^{-t\hbar}}{2t} \tag{A.17}$$

Combining the inequalities in A.16 and A.17 gives

$$\frac{\hbar e^{-t/\hbar}}{2t} \leq \left(\binom{N}{n} \frac{n!}{|\mathcal{G}|} \right)^{1/n} \tag{A.18}$$

I then use Stirling's inequality as follows:

$$\frac{N!}{(N-n)!} \leq \frac{\sqrt{\tau N} \left(\frac{N}{e}\right)^N e^{\frac{1}{12N}}}{\sqrt{\tau(N-n)} \left(\frac{N-n}{e}\right)^{N-n} e^{\frac{1}{12(N-n)+1}}} \tag{A.19}$$

$$\leq N^n e^n \tag{A.20}$$

Combining that with Equation (A.18) gives

$$\frac{\hbar e^{-t/\hbar}}{2t} < Ne|\mathcal{G}|^{-1/n} \tag{A.21}$$

or

$$\ln|\mathcal{G}| < n \ln N + \frac{nt}{\hbar} + n \ln(2t/e) - n \ln(\hbar) \tag{A.22}$$

Here N, the number of different Hamiltonian terms Eve could add, could be large, but I can bound it terms of the width W: if each memory object is a qudit, if Eve tries to add more than d^2 different Hamiltonian terms to a single qudit, then the final Hamiltonian term can be written as a real linear combination of at most d^2 terms. A full formula, if she has W qubits and at most k-local Hamiltonians, will be

$$N \leq d^2\mathsf{W} + d^{2\cdot 2}\binom{\mathsf{W}}{2} + \cdots + d^{2\cdot k}\binom{\mathsf{W}}{k} \in O(\mathsf{W}^k), \tag{A.23}$$

since d is constant.

Thus, $n \ln N = O(nk \log W)$, so the full term is $O(nt + n \log W)$. □

This has a more concrete corollary:

Corollary 1. *Using a family of Hamiltonians with n terms for time t to create a QRACM access to tables of size N must satisfy*

$$nt \in \Omega(N). \tag{A.24}$$

Proof. I use the set of possible QRACM access gates as $\mathcal{G}$ (see Chapter 4 for more detail). There are 2^N possible QRACM access gates, and they are mutually orthogonal, so the result follows immediately. □

A.2 Coherent Evolution Limit

Here I prove Theorem 3.3.1 from Section 3.3.2.

A.2.1 Shifts of vectors

I define the following matrices: a shift S_n with

$$S_n := \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix} \tag{A.25}$$

and an "energy" matrix

$$A_n := \mathrm{diag}(0, 1, 2, \dots, n-1) \tag{A.26}$$

The shift represents the change in energy from interacting with the quantum system, and the energy matrix represents the energy level of the system. For a positive real vector $v \in \mathbb{R}^n$, I aim to minimize $v^T A_n v$ while maximizing $v^T S_n v$.

Lemma A.2.1. *Let $v \in R^n$ be a positive real unit vector such that $v^T S_n v \geq 1 - \epsilon$. Then defining $v_0 = v_{n+1} = 0$, $|v_i + v_{i+1}| \leq \sqrt{2\epsilon}$ for all $i \in \{0, \dots, n\}$.*

Proof. Direct from the definitions and completing squares, I can write:

$$\begin{aligned} 1 - \epsilon \leq & v^T S_n v & \text{(A.27)} \\ = & \sum_{i=1}^{n-1} v_i v_{i+1} & \text{(A.28)} \end{aligned}$$



Figure A.1: $v(t)$ for $t = 10$ and $t = 9$

$$= \frac{1}{2}\left(\sum_{i=1}^{n-1} v_i^2 + v_{i+1}^2 - (v_i - v_{i+1})^2\right) \tag{A.29}$$

$$= \sum_{i=1}^{n} v_i^2 - \tfrac{1}{2}(v_1^2 + v_n^2) - \tfrac{1}{2}\sum_{i=1}^{n-1}(v_i - v_{i+1})^2 \tag{A.30}$$

$$= 1 - \tfrac{1}{2}\sum_{i=0}^{n}(v_i - v_{i+1})^2 \tag{A.31}$$

$$2\epsilon \geq \sum_{i=0}^{n}(v_i - v_{i+1})^2 \tag{A.32}$$

The right-hand-side can be (loosely) lower-bounded by $(v_j - v_{j+1})^2$ for any j, giving the result. □

From Lemma A.2.1, I define a set

$$\mathcal{V}_\epsilon(\delta) := \left\{ v \in \mathbb{R}_+^n \middle| \|v\| = \delta, \text{ for all } i \text{ with } 0 \leq i \leq n, |v_i - v_{i+1}| \leq \sqrt{2\epsilon} \right\}. \tag{A.33}$$

Let $v(t)$ be defined for a parameter $t \in \{1, \ldots, n\}$ as a "triangular vector", shown in Figure A.1:

$$v(t)_i = \begin{cases} \min\{i, t-i\}\sqrt{2\epsilon} & 1 \leq i < t \\ 0 & , i \geq t \end{cases}. \tag{A.34}$$

Intuitely, $v(t)$ should minimise $v^T A_n v$ for all $v \in \mathcal{V}_\epsilon(\|v(t)\|)$: one should be able to minimise this value by "pushing" all the amplitude of the vector to the lowest indices, while ensuring that the start and end are both 0 and that no two indices change by more than $\sqrt{2\epsilon}$. Proving this rigorously is more tedious; if you are convinced that $v(t)$ minimises $v^T A_n v$, feel free to skip to Lemma A.2.7.

First I define a *peak* and a *valley* of a real non-negative vector v. A peak of length ℓ for v will be an index i with $1 \leq i \leq n$ such that:

1. $v_{i-1} < v_i$

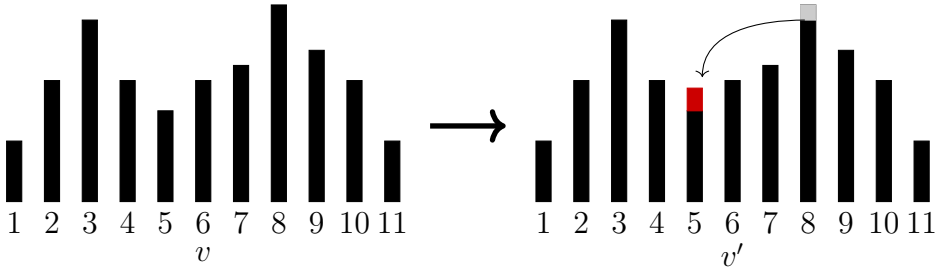


Figure A.2: Illustration of the proof of Lemma A.2.2 by moving mass downward (from entry 7 to entry 5) in a vector with a valley. This reduces the average energy, so the original could not be a minimum. Notice that entry 5 grew by more than entry 7 decreased. This is because we need to conserve the sum of *squares* of the entries.

2. $v_{i+\ell+1} < v_i$ and $v_i = v_j$ for all $j \in \{i, i+1, \ldots, i+\ell\}$.

A valley is the same except the inequalities are flipped. In both definitions I include $v_0 = v_{n+1} = 0$

Lemma A.2.2. *A vector $v \in \mathcal{V}_\epsilon(\delta)$ minimising $v^T A_n v$ has no valleys.*

Proof. Figure A.2 illustrates the proof. Suppose v has a valley at index i of length ℓ_i. Since $v_{i+\ell_i+1} > v_i$, but we know that $v_{n+1} = 0$, then there must be some peak from index j of length ℓ_j, where $j > i + \ell_i$.

Let $\kappa > 0$ be arbitrary; we will set it later.

Let v' be the vector formed by taking $v'_k = v_k$ for all k not in the specified peak and valley, and

$$v'_k = \sqrt{v_k^2 + \frac{\kappa}{\ell_i}} \tag{A.35}$$

for $i \le k \le i + \ell_i$, and

$$v'_k = \sqrt{v_k^2 - \frac{\kappa}{\ell_j}} \tag{A.36}$$

for $j \le k \le j + \ell_j$. Immediately we have that $\|v'\| = \|v\|$. To see that $|v'_k - v'_{k+1}| \le \sqrt{2\epsilon}$ can still hold for some $\kappa > 0$:

1. The difference between any consecutive vector from v'_i to $v'_{i+\ell_i}$ is still 0 (same for v'_j to $v'_{j+\ell_j}$).

2. Since $v_{i-1} > v_i$ and $v_{i+\ell_i+1} > v_{i+\ell}$, and we increase v'_k for $i \le k \le i + \ell_i$, these differences decrease;

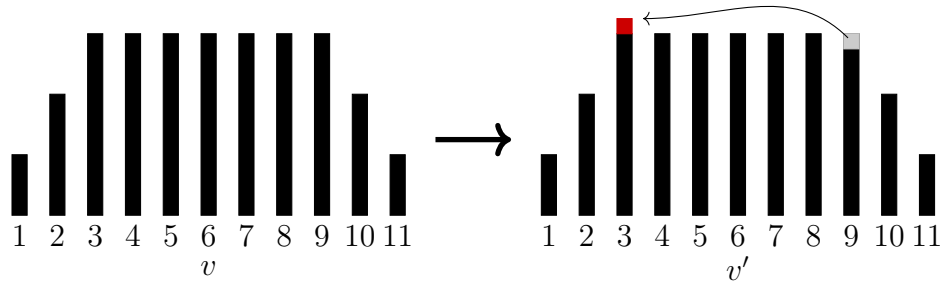


Figure A.3: Illustration of the proof of Lemma A.2.3 by moving mass downward in a vector with a peak of length more than 2.

3. Similarly, since $v_{j-1} < v_j$ and $v_{j+\ell_j+1} < v_{j+\ell}$ and we *decrease* v'_k for $j \leq k \leq j + \ell_j$, these differences also decrease.

In this case we can explicitly write

$$\kappa = \min\left\{\ell_i(v_{i_1-1}^2 - v_{i_1}^2), \ell_i(v_{i_2+1}^2 - v_{i_2}^2), \ell_j(v_{j_1}^2 - v_{j_1-1}^2), m_j(v_{j_2+1}^2 - v_{j_2})\right\} \quad \text{(A.37)}$$

though future proofs will be less explicit.

Finally, notice that

$$v'^T A_n v' = v^T A_n v + \frac{\kappa}{\ell_i}(i + \cdots + i + \ell_i) - \frac{\kappa}{\ell_j}(j + \cdots + j + \ell_j). \quad \text{(A.38)}$$

We know $j + \cdots + (j + \ell_j) \geq \ell_j j$, and $i + \cdots + (i + \ell_i) \leq \ell_i i$, so

$$v'^T A_n v' \leq v^T A_n v + \kappa i - \kappa j < v^T A_n v \quad \text{(A.39)}$$

since $j > i$. Thus, v cannot be a minimum. □

Corollary 2. *Any $v \in \mathcal{V}_\epsilon(\delta)$ minimising $v^T A_n v$ has precisely one peak.*

Proof. By the structure of $\mathcal{V}_\epsilon(\delta)$, any v in this set must have at least one peak, since $v_0 = v_{n+1} = 0$ and at least one component is positive. If there are two distinct peaks, then there must be a valley, so any v minimising $v^T A_n v$ has precisely one peak. □

Lemma A.2.3. *If $v \in \mathcal{V}_\epsilon(\delta)$ minimises $v^T A_n v$, then its peak has length exactly* 1 *or* 2*).*

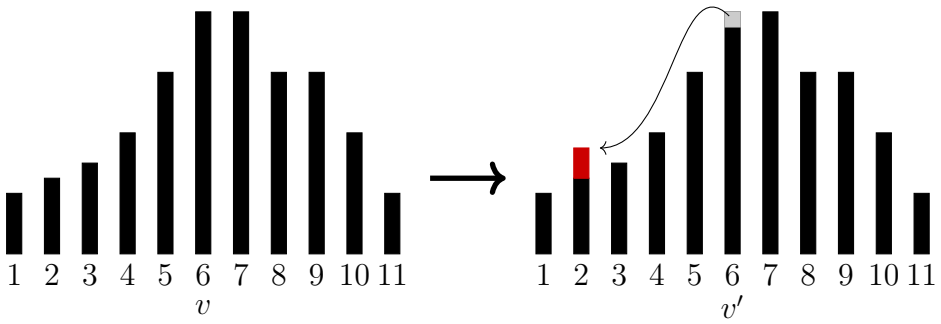


Figure A.4: Illustration of the the first half of the proof of Lemma A.2.4 by moving mass downward in a vector that does not increase as fast as possible.

Proof. Figure A.3 illustrates the proof. Suppose the peak at index i has length $\ell > 2$. Set $\kappa > 0$, and define v' such that $v'_k = v_k$ for all k except $v'_{i+\ell} = \sqrt{v_{i+\ell}^2 - \kappa}$ and $v'_{i+\ell-1} = \sqrt{v_{i+\ell-1}^2 + \kappa}$. Here $\|v'\| = \|v\|$ and the difference inequalities hold because:

1. Since $v_{i+\ell-2} = v_{i+\ell-1}$ (since $\ell > 2$), we can perturb $v_{i+\ell-1}$ up slightly without increasing the difference above $\sqrt{2\epsilon}$.

2. Similarly, $v_{i+\ell-1} = v_{i+\ell}$ so we can perturb them away from each other.

3. Since $v_{i+\ell+1} < v_{i+\ell}$, perturbing $v_{i+\ell}$ downwards slightly will *decrease* the difference between them.

Finally, we can see that $v'^T A_n v' < v^T A_n v$, since a smaller index now has more amplitude. □

Lemma A.2.4. *Let $v \in \mathcal{V}_\epsilon(\delta)$ minimise $v^T A_n v$, and suppose it has a peak at index p of length ℓ. Then if $v_{i+1} > 0$, $|v_i - v_{i+1}| = \sqrt{2\epsilon}$ for $0 \le i \le n$ with the possible exception of $i \in \{p-1, \ldots, p+\ell\}$.*

Proof. Figure A.4 illustrates the first half of the proof. Suppose there is some index i with $0 \le i < p-1$ such that $|v_i - v_{i+1}| < \sqrt{2\epsilon}$. This means $v_{i+1} - v_i < \sqrt{2\epsilon}$.

As in previous proofs, let $v' = v$ except $v'_{i+1} = \sqrt{v_{i+1}^2 + \kappa}$ and $v'_p = \sqrt{v_p^2 - \kappa}$ for some $\kappa > 0$. To show that the difference inequalities hold:

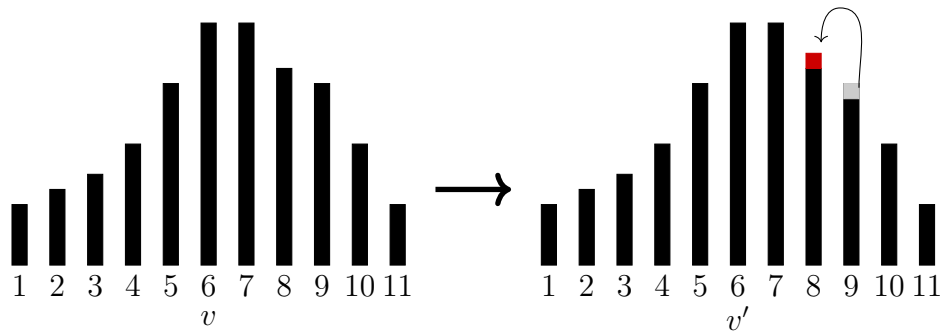


Figure A.5: Illustration of the the second half of the proof of Lemma A.2.4 by moving mass downward in a vector that does not decrease as fast as possible.

- Since $v_{i+1} - v_i < \sqrt{2\epsilon}$, we can perturb v_{i+1} upward without exceeding a difference of $\sqrt{2\epsilon}$.
- Since $v_{i+1} \leq v_{i+2}$ (as $i+1 \leq p$), perturbing v_{i+1} upward cannot increase the difference between them;
- Since v_p is the peak, perturbing it downward will either decrease the subsequent differences (if the peak has length 1) or add a small difference between v_p and v_{p+1} (if the peak has length 2). In either case there is a non-zero perturbation preserving the inequalities.

As in previous lemmas, $\|v'\| = \|v\|$ and $v'^T A_n v' < v^T A_n v$, showing that v cannot be a minimum.

Now suppose there is some index i with $p+\ell < i < n$ such that $|v_i - v_{i+1}| < \sqrt{2\epsilon}$, meaning $v_{i+1} - v_i < \sqrt{2\epsilon}$. Figure A.5 illustrates the proof in this case. By assumption $v_{i+1} > 0$, so we can choose some $\kappa > 0$ and set $v' = v$ except for $v'_i = \sqrt{v_i^2 + \kappa}$ and $v'_{i+1} = \sqrt{v_{i+1}^2 - \kappa}$.

Since $v_{i+2} \leq v_{i+1}$ and $v_{i-1} \geq v_i$, perturbing v_i upward and v_{i+1} downward will cause no problem (and v_{i+1} can be perturbed downward since it is strictly positive). Since the difference between v_i and v_{i+1} was strictly below the bound, this difference can be decreased.

As before, we have pushed amplitude "down", so $v'^T A_n v' < v^T A_n v$, so v was not a minimum. □

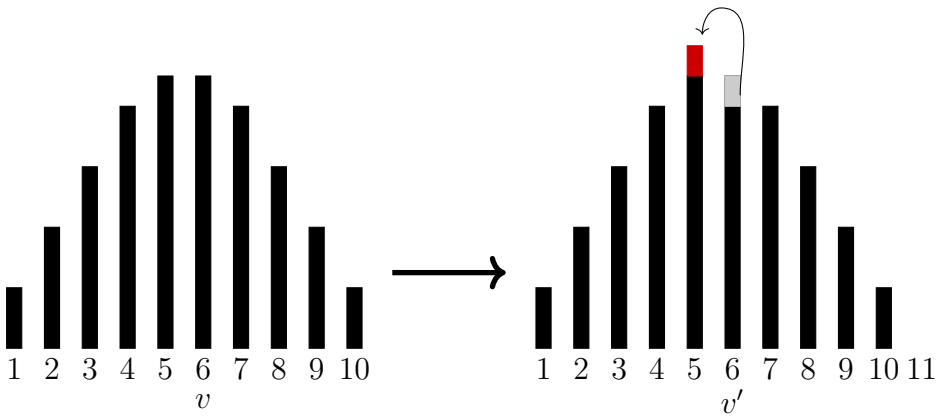


Figure A.6: Illustration of the proof of Lemma A.2.5 by moving mass downward in a vector with both sides of a length-2 peak being less than $\sqrt{2\epsilon}$.

I would like to extend this result to $i = p - 1$, but this requires two sub-cases based on the length of the peak.

Lemma A.2.5. *Let $v \in \mathcal{V}_\epsilon(\delta)$ minimise $v^T A_n v$, and suppose it has a peak at index p of length $\ell = 2$. Then $|v_{p-1} - v_p| = \sqrt{2\epsilon}$.*

Proof. Figure A.6 illustrates the proof. Suppose $v_p - v_{p-1} < \sqrt{2\epsilon}$. We perturb v to v' which is identical, except $v'_p = \sqrt{v_p^2 + \kappa}$ and $v'_{p+\ell} = \sqrt{v_{p+\ell}^2 - \kappa}$. For $\kappa > 0$ this ensures $\|v'\| = \|v\|$ and $v'^T A_n v' < v^T A_n v$.

To ensure the inequalities can hold with $\kappa > 0$:

- Since $v_p - v_{p-1} < \sqrt{2\epsilon}$, v_p can be perturbed upwards a non-zero amount without violating the required inequality;
- Since $v_p = v_{p+1}$ (because $\ell > 1$), it can be perturbed upwards.
- Since $v_{p+\ell} = v_{p+\ell-1}$, it can be perturbed downwards.
- Since $v_{p+\ell} - v_{p+\ell+1} \leq \sqrt{2\epsilon}$, we can perturb $v_{p+\ell}$ downards and the inequality gets stronger.

□

Lemma A.2.6. *Let $v \in \mathcal{V}_\epsilon(\delta)$ minimise $v^T A_n v$ for $\delta \geq \sqrt{2\epsilon}$, and suppose it has a peak at index p of length 1. If $|v_{p-1} - v_p| < \sqrt{2\epsilon}$, then $|v_p - v_{p+1}| = \sqrt{2\epsilon}$*

Proof. Suppose $v_p - v_{p-1} < \sqrt{2\epsilon}$ and $v_p - v_{p+1} < \sqrt{2\epsilon}$. We perturb v to v', which is identical, except $v'_p = \sqrt{v_p^2 + \kappa}$ and $v'_i = cv_i$ for all $i > p$, for a constant $c < 1$ and $\kappa > 0$. For any $\kappa > 0$, by the intermediate value theorem there exists some $c < 1$ to ensure that $\|v'\| = \|v\|$ and $v'^T A_n v' < v^T A_n v$.

A mild exceptional case is if $v_{p+1} = 0$; however, this forces $v_p < \sqrt{2\epsilon}$. In turn this forces $v_{p-1} < \sqrt{2\epsilon}$, which contradicts Lemma A.2.4 unless $p = 1$. But then v_p is the only non-zero component and thus $\|v_p\| < \sqrt{2\epsilon}$. The bound on δ thus avoids this edge case.

Otherwise, to ensure the inequalities can hold with $\kappa > 0$:

- Since $v_p - v_{p-1} < \sqrt{2\epsilon}$, v_p can be perturbed upwards a non-zero amount without violating this inequality.
- Since we only perturbed non-zero v_i for $i > p$, there is some non-zero amount of room to perturb downwards;
- Since $v_p - v_{p+1} < \sqrt{2\epsilon}$, there is room to perturn both v_p upwards and v_{p+1} downwards and preserve this;
- Since we scaled all v'_i by the same amount, we have

$$v'_i - v'_{i+1} = c(v_i - v'_{i+1}) \leq c\sqrt{2\epsilon} < \sqrt{2\epsilon}. \tag{A.40}$$

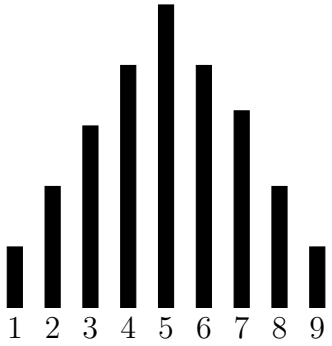
Thus, $v' \in \mathcal{V}_\epsilon(\delta)$ and so v is not a minimum.

□

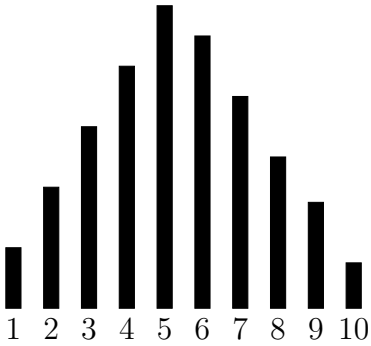
With all of this in place, there are only a few possible shapes for a vector minimising $v^T A_n v$, shown in Figure A.7.

Theorem A.2.1. *Let v be a vector in $\mathcal{V}_\epsilon(\|v(t)\|)$ which minimises $v^T A_n v$. Then $v = v(t)$.*

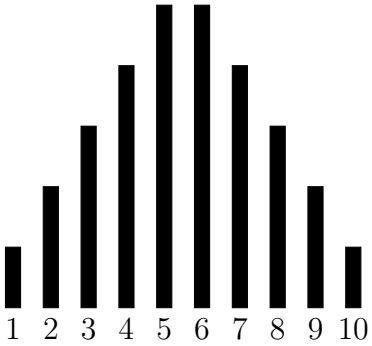
Proof. Let v be such a vector. By Lemma A.2.3, we know that v has a peak of length ℓ with $\ell \in \{1, 2\}$, at some index p. Let $p' = p + \ell - 1$ (the index at the end of the peak).



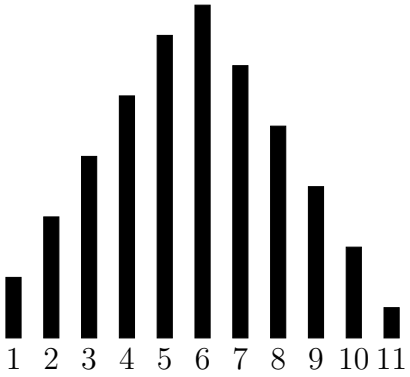
(a) All differences are $\sqrt{2\epsilon}$, peak has length 1 ($v(9)$). Clearly no mass can move and preserve the difference inequalities.



(b) To optimally add ℓ_2-norm to A.7a, none of the first 5 components can increase, so the remaining components must increase.



(c) Increasing the ℓ_2-norm of A.7b eventually creates a peak of length 2. This is $v(10)$.



(d) Increasing the ℓ_2-norm of A.7c creates a peak in component 6, but not the maximum increase at the peak. Increasing the ℓ_2-norm of this creates a vector like A.7a again.

Figure A.7: Possible shapes for a vector maximising $v^T Av$, subject to the constraint that the difference between any two components is at most $\sqrt{2\epsilon}$.

My approach will be to narrow down where the peak of $v(t)$ must be, so that it is at the same place as v. From there, the peak of v might be slightly offset from the peak of $v(t)$, but then the constraint that $\|v\| = \|v(t)\|$ will give the result.

Proof that $v_i \leq v(t)_i$ up to the peak of $v(t)$:

By Lemma A.2.4, we know that $v_i = i\sqrt{2\epsilon} = v(t)_i$ for i up to either p or the peak of $v(t)$, whichever is first. If p (the peak of v) is first, then we know that $v_i \leq v(t)_i$ for all i up to the peak of $v(t)$. This also tells us that if the peak of $v(t)$ comes after or at the peak of v, then the peak of $v(t)$ is at least as large the peak of v.

Special case for $t = 1$:

As a special case, suppose that $v_{p+1} = 0$. This means $v_p \leq \sqrt{2\epsilon}$ since $v_{p+1} = 0$, but we know that $v_{p-1} = (p-1)\sqrt{2\epsilon} < v_p$. Thus, $p = 1$, so $\|v\| = v_p$ and if $\|v\| = \|v(t)\|$ then $t = 1$ and the result holds. Handling this case ensures that $v_{p+1} > 0$ for the rest of the proof.

Proof that $v_i \geq v(t)_i$ for all i after the peak of v:

Recall that p' is the end of the peak of v_p. If $v_{p'+1} < v(t)_{p'+1}$, then $v_i \leq v(t)_i$ for all $i \geq p' + 1$, since both v and $v(t)$ decrease at the same rate (in fact the inequality will be strict until $v(t)_i = 0$). Since we know that $v_{p'+1} \geq v_{p'} - \sqrt{2\epsilon}$ and $v_{p'} > v_{p-1}$, then $v_{p'+1} > v_{p-1} - \sqrt{2\epsilon}$. If the peak $v(t)$ ended at or before $v_{p'}$, then $v(t)_{p'} \leq v_{p'}$. This gives

$$v(t)_{p'+1} \leq v(t)_{p'} - \sqrt{2\epsilon} \leq v_{p'} - \sqrt{2\epsilon} < v_{p'+1}, \tag{A.41}$$

contradicting the earlier assumption. Thus, the peak of $v(t)$ must occur after p', but this means $v_i \leq v(t)_i$ for all i up to and including p'. This means $v_i \leq v(t)_i$ for all i, and since we have at least one strict inequality this would mean $\|v\| < \|v(t)\|$, a contradiction.

Thus, $v_{p'+1} \geq v(t)_{p'+1}$, which in turn means that $v_i \geq v(t)_i$ for all $i \geq p' + 1$, since $v(t)$ is decreasing as fast as possible.

Proof that the peak of $v(t)$ comes at or before p':

If the peak of $v(t)$ was at $p' + 1$ or later, then its peak would need to be greater than the peak of v, contradicting $v_{p'+1} \geq v(t)_{p'+1}$. Thus, the peak of $v(t)$ is at most at p'.

Proof that the peak of $v(t)$ comes at or after p:

If the peak of $v(t)$ came before p, then $v(t)_i < v_i$ for all i after the peak of $v(t)$, since $v(t)$ decreases as fast as possible while v increases at least once after this point. Since $v(t)_i = v_i$ up to the peak of $v(t)$ in this case, this implies that $\|v(t)\| < \|v\|$, a contradiction. Thus, the peak of $v(t) \in \{p, p+1\}$.

Proof that $v = v(t)$:

The constraint on the peak of $v(t)$ means that $v(t)_p \geq v_p$. I split into a few sub-cases, based on the lengths and position of the peaks of v and $v(t)$:

$v(t)$ **has a peak of length 2 or the peak of** $v(t)$ **is at** $p+1$**:** In this case $v(t)_{p+1} \geq v_{p+1}$, in turn meaning that $v(t)_i \geq v_i$ for all $i \geq p+1$. This means $\|v(t)\| \geq \|v\|$, and since we know that $\|v(t)\| = \|v\|$, these inequalities must all be equalities, i.e., $v = v(t)$.

$v(t)$ **has a peak of length** 1 **at** p **and** v **has a peak of length 2:** By Lemma A.2.5, $v_p = v(t)_p$ since both increase by $\sqrt{\epsilon}$ up to that point. Since we saw before that $v_i \geq v(t)_i$ for all $i \geq p' + 1 = p + 1$ (in this case), then $\|v\| \geq \|v(t)\|$. Equality in this ensures $v = v(t)$.

$v(t)$ **and** v **both have peaks of length** 1 **at** p**:** If the peaks of v and $v(t)$ have equal values, then $v'_i \geq v(t)_i$ for all $i > p$, so by the same logic this means $v = v(t)$. Alternatively, if $v_p < v(t)_p$, then by Lemma A.2.6, this means that $v_{p+1} = v_p - \sqrt{2\epsilon} < v(t)_{p+1}$, which contradicts my earlier conclusion that $v_i \geq v(t)_i$ for all $i \geq p' + 1$ (here $p' = p$).

□

Corollary 3. *For $\delta > 0$, let t be such that $\|v(t)\| \leq \delta$. Then for any $v \in \mathcal{V}_\epsilon(\delta)$, $v(t)^T A_n v(t) \leq v^T A_n v$.*

Proof. By Theorem A.2.1, we know that $v(t)$ minimises $v^T A_n v$ for all $v \in \mathcal{V}_\epsilon(\|v(t)\|)$.

Now take any $v \in \mathcal{V}_\epsilon(\delta)$. Scale it by $\frac{\|v(t)\|}{\delta} < 1$ to make v'. Then the inequality constraints still hold (all elements are smaller, so differences are smaller) and $v^T A_n v > v'^T A_n v'$. However, since $v(t)$ is a minimum, then $v'^T A_n v' \geq v(t)^T A_n v(t)$, giving the result. □

Lemma A.2.7. *For any $v \in \mathcal{V}_\epsilon(1)$, $v^T A_n v \geq \frac{4}{3}\epsilon^{-1/3}$.*

Proof. Using Corollary 3, if we pick t such that $\|v(t)\| \leq 1$, then $v(t)^T A_n v(t)$ will give a lower bound.

Computing the norms of $v(t)$ is straightforward, if we split into an even case and an odd case.

Odd: For $t = 2r + 1$, the peak of $v(t)$ has length 2, and we sum up to r on either side; this means

$$\begin{aligned} v(t)^T v(t) =& \sum_{i=1}^{r} i^2 \cdot 2\epsilon + \sum_{j=1}^{r} j^2 \cdot 2\epsilon && \text{(A.42)} \\ =& 4\epsilon \left(\sum_{i=1}^{r} i^2 \right) && \text{(A.43)} \\ =& 4\epsilon \frac{(2r+1)r(r+1)}{6} && \text{(A.44)} \\ 1 \geq& \frac{1}{8}\epsilon t^3 && \text{(A.45)} \\ \frac{2}{\epsilon^{1/3}} \geq& t && \text{(A.46)} \end{aligned}$$

Then we can compute the value of

$$\begin{aligned} v(t)^T A_n v(t) =& \sum_{i=1}^{r} i^2 \cdot 2\epsilon(i-1) + \sum_{j=1}^{r} j^2 \cdot 2\epsilon(2r-j) && \text{(A.47)} \\ =& 2\epsilon \left(\sum_{i=1}^{r} i^3 - i^2 + 2ri^2 - i^3 \right) && \text{(A.48)} \\ =& 2\epsilon(2r-1)\frac{(2r+1)r(r+1)}{6} && \text{(A.49)} \\ \leq& \frac{1}{12}\epsilon t^4 && \text{(A.50)} \\ \leq& \frac{4}{3}\epsilon^{-1/3} && \text{(A.51)} \end{aligned}$$

Even: For $t = 2r$, the peak of $v(t)$ has length one, we sum to $r - 1$ on either side plus a middle term; this means

$$\begin{aligned} v(t)^T A_n v(t) =& \sum_{i=1}^{r-1} i^2 \cdot 2\epsilon + r^2 \cdot 2\epsilon + \sum_{j=1}^{r-1} j^2 \cdot 2\epsilon && \text{(A.52)} \\ =& 2\epsilon \left(r^2 + 2\sum_{i=1}^{r-1} i^2 \right) && \text{(A.53)} \\ =& 2\epsilon \left(r^2 + \frac{(2r-1)r(r-1)}{3} \right) && \text{(A.54)} \\ 1 \geq& 2\epsilon \frac{t^3}{64} && \text{(A.55)} \\ \frac{4}{\epsilon^{1/3}} \geq& t && \text{(A.56)} \end{aligned}$$

And then:

$$v(t)^T A_n v(t) = \sum_{i=1}^{r-1} i^2 \cdot 2\epsilon(i-1) + r^2 \cdot 2\epsilon(r-1) + \sum_{j=1}^{r-1} j^2 \cdot 2\epsilon(2r-j-1) \quad \text{(A.57)}$$

$$=2\epsilon\left(r^3 - r^2 + \sum_{i=1}^{r-1}(2r-2)i^2\right) \tag{A.58}$$

$$=2\epsilon\left(r^3 - r^2 + (r-1)\frac{(2r-1)r(r-1)}{3}\right) \tag{A.59}$$

$$\leq\epsilon\frac{t^4}{8} \tag{A.60}$$

$$\leq 32\epsilon^{-1/3} \tag{A.61}$$

Taking the minimum of these two lower bounds gives the result.

□

Theorem A.2.2. *If $v \in \mathbb{R}^n$ is a positive vector of unit norm such that $v^T S_n v \geq 1-\epsilon$, then $v^T A_n v \geq \frac{4}{3}\epsilon^{-1/3}$.*

Equivalently, if $v^T A_n v \leq E$, then $v^T S_n v \leq 1 - \frac{4^3}{3^3 E^3}$.

Proof. Given such a v, by Lemma A.2.1 we have $v \in \mathcal{V}_\epsilon(1)$. Then Lemma A.2.7 gives the result. □

I will now give an upper bound; a specific vector v such that $v^T S_n v = 1 - \epsilon$ and $v^T A_n v \approx 1 - \frac{1}{\sqrt{\epsilon}}$.

I can define $B_n = S_n^T + S_n$, which is a symmetric matrix, and I see that $v^T B_n v = 2v^T S_n v$.

Lemma A.2.8. *The eigenvalues of B_n are $2cos(\frac{k\tau}{2(n+1)})$ for $1 \leq k \leq n$. The normalized eigenvectors $v^{(k)}$ have components equal to $v_m^{(k)} = \sqrt{\frac{2}{n+1}}\sin(km\tau/2(n+1))$, and if v is the eigenvector of the largest eigenvalue, then $v^T A_n v = \frac{n-1}{2}$.*

Proof. B_n is a Toeplitz matrix, so the eigenvalues are well known. By its structure, the components of an eigenvector v for eigenvalue λ must satisfy $v_{k-1} + v_{k+1} = \lambda v_k$ for $2 \leq k \leq n-2$. Straightforward trigonometry shows that the $v^{(k)}$ described satisfy this, and that this holds for $k = 1$ as well.

Setting $u_m^{(k)} = \sin(km\tau/2(n+1))$, we find its norm (for n odd) as

$$\|u^{(k)}\|^2 = \sum_{m=1}^{n}\sin^2\left(\frac{km\tau}{2(n+1)}\right) \tag{A.62}$$

$$= \sum_{m=1}^{\frac{n-1}{2}} \sin^2\left(\frac{km\tau}{2(n+1)}\right) + \sum_{m=\frac{n+1}{2}}^{n} \sin^2\left(\frac{km\tau}{2(n+1)}\right) \tag{A.63}$$

$$= \sum_{m=1}^{\frac{n-1}{2}} \sin^2\left(\frac{km\tau}{2(n+1)}\right) + \sum_{\ell=0}^{\frac{n-1}{2}} \cos^2\left(\frac{k\ell\tau}{2(n+1)}\right) \tag{A.64}$$

$$= 1 + \frac{n-1}{2} \tag{A.65}$$

and similarly for n even.

Letting Q_n be the permutation matrix which transposes the ith component with the $n-i+1$ component, we see that $Q_n = Q_n^{-1}$ and $Q_n B_n Q_n = B_n$. Because the eigenvectors of B_n are unique, this means any eigenvector of B_n is also an eigenvector of Q_n. Since the eigenvalues of Q_n are ± 1 and all components of v are positive, this means $Q_n v = v$; that is, v is invariant under "flipping" its components.

The space of eigenvectors of Q_n with eigenvalue 1 is spanned by an orthonormal basis given by $y_i := \frac{1}{\sqrt{2}}(e_i + e_{n-i+1})$ for $i = 1$ to $n/2$, plus $y_{mid} = e_{(n-1)/2}$ if n is odd. We can see that $y_i^T A_n y_i = \frac{n-1}{2}$ for all i and for y_{mid}. Thus, for B_n's maximum eigenvector v, which is in the span of these y_i, we have $v^T A_n v = \frac{n-1}{2}$. □

Corollary 4. *For $n \geq 2E-1$, there exists a vector v such that $v^T A_n v = E-1$ and*

$$v^T S_n v = \cos\left(\frac{\tau}{4E}\right) \leq 1 - \frac{\tau^2}{16E^2} + O(\tfrac{1}{E^4}). \tag{A.66}$$

Proof. Take v as the eigenvector for $k = 1$ in Lemma A.2.8 and $n = 2E-1$. This has $v^T B_n v_n = 2\cos(\frac{\tau}{2(n+1)})$, so $v^T S_n v = \cos(\frac{\tau}{2E})$ as required.

Moreover, Lemma A.2.8 says that $v^T A_n v = \frac{n-1}{2} = E-1$, as required. □

A.2.2 Energy and Coherence Theorem

Recall the scenario: There is an environmental state $|\chi_{env}\rangle$ and a unitary U such that

$$U|\phi_0\rangle|\chi_{env}\rangle = |\phi_1\rangle|\phi_{env}\rangle \tag{A.67}$$

$$U|\psi_0\rangle|\chi_{env}\rangle = |\psi_1\rangle|\psi_{env}\rangle \tag{A.68}$$

and Φ_U is the channel formed by appending $|\chi_{env}\rangle\langle\chi_{env}|$, applying U, then tracing out the environmental system. This is intended to mimic the action of $\tilde{U}$, which maps $|\phi_0\rangle$ to $|\phi_1\rangle$ and $|\psi_0\rangle$ to $|\psi_1\rangle$.

Theorem A.2.3. *Suppose $|\phi_0\rangle$ and $|\phi_1\rangle$ have an energy gap of E_ϕ, and $|\psi_0\rangle$ and $|\psi_1\rangle$ have an energy gap of E_ψ. If the expected energy of the environmental state is at most E (over the ground state energy), then letting $\Delta = E_\phi - E_\psi$, there exist input states such that fidelity of the output of Φ_U compared to the output of $\tilde{U}$ is at most*

$$1 - \frac{2^5\Delta^3}{3^3E^3}. \tag{A.69}$$

Proof. I start with an eigendecomposition of $|\chi_{env}\rangle$. I use the eigenvectors of the environmental Hamiltonian, which defines a set of energy levels $\mathcal{E}$. For each energy level E, I project $|\chi_{env}\rangle$ onto the eigenspace for E and normalise it to $|\chi_E\rangle$, so that I can write

$$|\chi_{env}\rangle = \sum_{E\in\mathcal{E}} \alpha_E |\chi_E\rangle \tag{A.70}$$

and since I can define $|\chi_E\rangle$ up to phase, this means I can take $\alpha_E \in \mathbb{R}^+$ without loss of generality. Here the $|\chi_E\rangle$ are mutually orthogonal.

Critically, since U maps $|\phi_0\rangle$ to $|\phi_1\rangle$, which gains energy E_ϕ, it must map every environmental state with energy E to a state with energy $E - E_\phi$. That is, for each energy level E, there is an isometry $P_{E,\phi}$ which maps states of energy E to states of energy $E - E_\phi$:

$$U|\phi_0\rangle|\chi_{env}\rangle = |\phi_1\rangle \sum_{E\in\mathcal{E}} \alpha_E P_{E,\phi}|\chi_E\rangle \tag{A.71}$$

We see a similar behaviour for $|\psi_0\rangle$, but the transition map may be some distict $Q_{E,\psi}$. Thus, if we have a superposition state $\alpha|\phi_0\rangle + \beta|\psi_0\rangle$, the state after applying U is

$$|\phi_1\rangle \sum_{E\in\mathcal{E}} \alpha_E P_{E,\phi}|\chi_E\rangle + |\psi_1\rangle \sum_{E\in\mathcal{E}} \alpha_E Q_{E,\psi}|\chi_E\rangle \tag{A.72}$$

Then we trace out the environmental system. This produces a state (in the $|\phi_1\rangle/|\psi_1\rangle$ basis)

$$\begin{pmatrix} |\alpha|^2 & \beta\overline{\alpha}\gamma \\ \overline{\beta}\alpha\overline{\gamma} & |\beta|^2 \end{pmatrix}, \gamma := \sum_{E,E'\in\mathcal{E}} \alpha_E\overline{\alpha_{E'}} \langle\chi_{E'}| Q^\dagger_{E',\psi} P_{E,\phi}|\chi_E\rangle \tag{A.73}$$

The desired state is identical, except for the desired state will have $\gamma = 1$, so I want to give an upper bound on the absolute value of γ.

We notice immediately that if $E' - E_\psi \neq E - E_\phi$, then the right-hand inner product is 0, since different energy eigenstates are orthogonal. I therefore restrict the range of E' to $E' = E - E_\phi + E_\psi$; for convenience, I will define $\Delta := E_\psi - E_\phi$.

$$\gamma = \sum_{E\in\mathcal{E}} \alpha_E \overline{\alpha_{E-\Delta}} \langle \chi_{E-\Delta} | Q^\dagger_{E-\Delta,\psi} P_{E,\phi} | \chi_E \rangle \tag{A.74}$$

and I can upper-bound the absolute value by the sum of the absolute value of each term, and I can upper bound the inner product as 1. Therefore,

$$|\gamma| \leq \sum_{E\in\mathcal{E}} \alpha_E \overline{\alpha_{E-\Delta}} \tag{A.75}$$

I then partition the set $\mathcal{E}$ into subsets $\mathcal{E}_F$, where if $E, E' \in \mathcal{E}_F$, then $E - E'$ is an integer multiple of Δ. Importantly, this means that $E - \Delta$ will always be in the same partition as E. Then I can partition the sum as

$$|\gamma| \leq \sum_F \sum_{n\in\mathbb{N}} \alpha_{F+n\Delta} \overline{\alpha_{F+(n-1)\Delta}}. \tag{A.76}$$

Define C_F as a normalizing factor $C_F := \sqrt{\sum_{n\in\mathbb{N}} \alpha^2_{F+n\Delta}}$. I define it in this way so that the vector $v_F \in \mathbb{R}^{\mathbb{N}}$, defined to have its nth component equal to $\frac{1}{C_F}\alpha_{F+n\Delta}$, will have unit norm. We then see that

$$|\gamma| \leq \sum_F \lim_{n\to\infty} C_F^2 v_F^T S_n v_F \tag{A.77}$$

where S_n is the shift matrix from Equation (A.25), and v_F is implicitly projected onto the first n components.

We can also see that the expected energy of $|\chi_{env}\rangle$ is

$$\sum_{E\in\mathcal{E}} E\alpha_E^2 = \sum_{F\in\mathcal{F}} \sum_{n\in\mathbb{N}} (F + n\Delta)\alpha^2_{F+n\Delta} = \sum_{F\in\mathcal{F}} C_F^2 \left(F + \lim_{n\to\infty} \Delta v_F^T A_n v_F \right). \tag{A.78}$$

Applying Theorem A.2.2 to each vector v_F, we see that if $v_F^T A_n v_F \leq N_F$ for all n, then $v_F^T S_n v_F \leq 1 - \frac{4^3}{3^3 N_F^3}$ for all n. Notice that this is a concave function of N_F.

Thus,

$$\sum_{F\in\mathcal{F}} C_F^2 v^T S_n v_F \leq C_F^2 \left(1 - \frac{4^3}{3^3 N_F^3}\right) \leq 1 - \frac{4^3}{3^3 \left(\sum_{F\in\mathcal{F}} C_F^2 N_F^3\right)}, \tag{A.79}$$

and since this is independent of n, it holds in the limit as well.

Now notice that if the total energy of the state is no higher than E over the ground state, then Equation (A.78) implies that $\sum_{F\in\mathcal{F}} C_F^2 N_F \leq E/\Delta$. Finally this gives

$$|\gamma| \leq \lim_{n\to\infty} \sum_{F\in\mathcal{F}} C_F^2 v^T S n v_F \leq 1 - \frac{4^3\Delta^3}{3^3 E^3}. \tag{A.80}$$

To relate this to the fidelity, recall that the target state $\alpha|\phi_1\rangle + \beta|\psi_1\rangle$ is pure, so the fidelity between the state in Equation (A.73) and the target state:

$$|\alpha|^4 + |\beta|^4 + 2|\alpha|^2|\beta|^2\gamma \tag{A.81}$$

I can choose α and β "adversarially", i.e., to minimize fidelity. The best I can do is to set $|\alpha|^2 = \frac{1}{2}$, giving $\frac{1}{2} + \frac{\gamma}{2}$. The upper bound on γ gives

$$\text{Fidelity} \leq \frac{1}{2} + \frac{2^5\Delta^3}{3^3 E^3}. \tag{A.82}$$

□

Proposition A.2.1. *Under the same assumptions as Theorem A.2.3, there exists an environmental state χ_{env} of average energy over the minimum energy of E, such that the fidelity is at least*

$$1 - \frac{\tau^2\Delta^2}{16(E+\Delta)^2}. \tag{A.83}$$

Proof. We can start from Equation (A.73), as the derivation before is identical. My first goal is to ensure that $\langle\chi_{E-\Delta}| Q^\dagger_{E-\Delta,\psi} P_{E,\phi} |\chi_E\rangle = 1$ for all E. This is straightforward by induction: Let $R_E = Q^\dagger_{E-\Delta,\psi} P_{E,\phi}$, and select some starting energy level E_0. Then $|\chi_{E_0}\rangle$ can be arbitrary, and I inductively set $|\chi_{E_0+n\Delta}\rangle = R_E^\dagger \left|\chi_{E_0+(n-1)\Delta}\right\rangle$ and $|\chi_{E_0-n\Delta}\rangle = R_E \left|\chi_{E_0-(n-1)\Delta}\right\rangle$ from $n = 1$ to ∞. In turn this means

$$\gamma = \sum_{E\in\mathcal{E}} \alpha_E \alpha_{E-\Delta} \tag{A.84}$$

where I am choosing $\alpha_E \in \mathbb{R}$ and $\alpha_E > 0$ for all $E \in \mathcal{E}$. Then I pick the ground state energy of the environment as E_0 and set $\alpha_E = 0$ for $E < E_0$, $\alpha_E = 0$ for all $E \notin \{E_0 + n\Delta | n \in \mathbb{N}_0\}$. This means I can define $\alpha_n := \alpha_{E_0+n\Delta}$ and write

$$\gamma = \sum_{n=1}^{\infty} \alpha_n \alpha_{n-1} \tag{A.85}$$

Here I appeal to Corollary 4 for an explicit vector $\alpha := (\alpha_0, \ldots, \alpha_N)$ (letting $\alpha_n = 0$ for $n > N$), where $N = 2F - 1$, such that

$$\sum_{n=1}^{\infty} \alpha_n \alpha_{n-1} = \alpha^T S_N \alpha = \cos\left(\frac{\tau}{4F}\right) \geq 1 - \frac{\tau^2}{16F^2}. \tag{A.86}$$

Then I also see that the expected energy of this state is

$$\sum_{n=0}^{\infty} (E_0 + n\Delta)\alpha_n^2 = E_0 + \Delta(F - 1) \tag{A.87}$$

by Corollary 4. If the average energy of the group state is E, then $\Delta(F - 1) = E$; substituting and a basic inequality gives the result. □

B

Proofs of QRAM Lower Bounds

B.1 Proof of Lemma 4.3.1

I start by defining the set of circuits $\mathcal{C}(W, D, G, g, k)$ to be all circuits on W qubits, of circuit depth D, using G gates from a set of size g with fanin at most k. Here I assume all gates have depth 1 and I count circuits based on the arrangement of gates, not by function. Define a "formal circuit" as a function that takes as input a pair of (qubit,time step) and returns as output the particular gate applied to that qubit at that time step.

If two circuits have a different effect on quantum states, they must use different gates, and thus will be distinct formal circuits by this definition. This definition overcounts circuits, since equivalent circuits are counted twice, but we want to upper bound the number of circuits so this is fine.

Proposition B.1.1. *Under this notation,*

$$|\mathcal{C}(W, D, G, g, k)| \leq |\mathcal{C}(W, D, \min\{kG, DW\}, Wg, 1)|. \tag{B.1}$$

Proof. Let U be an ℓ-qubit gate in the gate set of $\mathcal{C}(W, D, g, k)$. I can define ℓ distinct single-qubit gates $U_1, \ldots, U_\ell$ from U, expanding the total number of gates from g to gk, and and then further define W/ℓ multiples of each of these single qubit gates, i.e., $U_1^{(1)}, \ldots, U_\ell^{(1)}, \ldots, U_1^{(m)}, \ldots, U_\ell^{(m)}$ where $m = \lfloor W/\ell \rfloor$. This gives at

most W single-qubit gates from U, so the total number of single-qubit gates defined in this way is at most Wg.

I now define an injective function from $\mathcal{C}(W, D, G, g, k)$ to $\mathcal{C}(W, D, \min\{kG, DW\}, Wg, 1)$. For each formal circuit, at each timestep we take all ℓ-qubit gates $U^{(1)}, \ldots U^{(m)}$ (we know there are fewer tham $m = \lfloor W/\ell \rfloor$ gates of fanin ℓ in one timestep, because there are not enough qubits for more!) and decompose each into $U_1^{(i)}, \ldots, U_\ell^{(i)}$, such that the qubit mapped to the jth input of the ith gate is given gate $U_j^{(i)}$. This increases the total number of gates by at most a factor of k, and cannot increase the number of gates above DW.

For example, a CNOT would split into two gates, a "target" gate and a "control" gate, and if these are reversed, the circuit is different (by our counting).

To show injectivity, suppose two formal circuits C_1 and C_2 in $\mathcal{C}(W, D, G, g, k)$ map to the same formal circuit in $\mathcal{C}(W, D, kG, gW, 1)$. Let U be any gate in C_1; suppose it applies to ℓ qubits $q_1, \ldots, q_\ell$ in time step t. This means there is some i such that the image circuit has $U_1^{(i)}, \ldots, U_\ell^{(i)}$ applied to $q_1, \ldots, q_\ell$ in time step t. By construction of our function, this means there must be some gate in C_2 of the same type as G which is also applied to those same qubits, in the same order, in time step t. Repeating for all gates shows that $C_1 = C_2$. □

Proposition B.1.2. *In the same notation,* $|\mathcal{C}(W, D, G, g, 1)| \leq \binom{DW}{G} g^G$.

Proof. We can straightforwardly count: There are DW possible "slots" for each gate, depending on which qubit and which time step. We choose G slots, and for each one, we select one of the g gates. □

Lemma B.1.1. $|\mathcal{C}(W, D, G, g, k)| \leq \binom{DW}{\min\{kG, DW\}} (Wg)^G$.

Proof. We combine Proposition B.1.1 and Proposition B.1.2 to obtain the result. □

B.2 Proof of no distillation

There are two components of the proof: First, I argue that with only d inputs in the QRACM, we can always find some indices which are "underrepresented", i.e.,

very little of the amplitude of the input state is concentrated on those indices. This means if two tables differ only on those indices, the original states will be negligibly different when given access to one of these two tables. This is more general than necessary: I only need $\ell = 1$ to prove my main theorem.

Lemma B.2.1. *Let $\rho_1, \ldots, \rho_d$ be any collection of states, each in $B(\mathcal{H}_i \otimes \mathcal{H}_{QRACM})$, i.e., part of the state is in the input space for a QRACM gate. Then there exists some set of ℓ indices such that for any two tables T and T' differing only in those ℓ indices, if we define*

$$\delta_i := \|\mathcal{U}_{QRACM}(T)(\rho_i) - \mathcal{U}_{QRACM}(T')(\rho_i)\|_1 \tag{B.2}$$

(with an implicit identity channel on $\mathcal{H}_i$) then

$$\sum_{i=1}^{d} \delta_i \leq \frac{2\ell\sqrt{d}}{N}. \tag{B.3}$$

Proof. Because we care about the application of a perfect QRACM gate, we will purify each of these states, so I consider some purification of ρ_i:

$$|\psi_i\rangle = \sum_{j=0}^{n} \sum_{b\in\{0,1\}} \alpha_{jb} |\psi_{ijb}\rangle |j\rangle |b\rangle \tag{B.4}$$

where I have expressed the input space to the QRACM gate in the computational basis. Here $|\psi_{ijb}\rangle$ is in $\mathcal{H}_i \otimes \mathcal{H}_p$, where $\mathcal{H}_p$ is the space necessary to purify ρ_i. We see that $U_{QRACM}(T) = \sum_{j,b} |j\rangle\langle j| \otimes |b \oplus T[j]\rangle\langle b|$, so that

$$\begin{aligned} &U_{QRACM}(T')^\dagger U_{QRACM}(T) \\ &\qquad = \sum_{j:T'[j]=T[j]} |j\rangle\langle j| \otimes I_2 + \sum_{j:T'[j]\neq T[j]} |j\rangle\langle j| \otimes X \end{aligned} \tag{B.5}$$

where X is the Pauli X. This implies that

$$\langle\psi_i| U_{QRACM}(T')^\dagger U_{QRACM}(T) |\psi_i\rangle \tag{B.6}$$

$$= \sum_{j:T[j]=T'[j], b\in\{0,1\}} |\alpha_{ijb}|^2 + \tag{B.7}$$

$$\sum_{j:T[j]\neq T'[j]} \overline{\alpha_{ij0}}\alpha_{ij1} \langle\psi_{ij0}|\psi_{ij1}\rangle + \overline{\alpha_{ij1}}\alpha_{ij0} \langle\psi_{ij1}|\psi_{ij0}\rangle \tag{B.8}$$

$$\geq 1 - \left(\sum_{j:T[j]\neq T'[j]} |\alpha_{j0}|^2 + |\alpha_{j1}|^2 + \overline{\alpha_{ij0}}\alpha_{ij1} + \overline{\alpha_{ij1}}\alpha_{ij0} \right) \tag{B.9}$$

$$\geq 1 - 2 \sum_{j:T[j]\neq T'[j]} |\alpha_{ij0}|^2 + |\alpha_{ij1}|^2 \tag{B.10}$$

Since $U_{QRACM}(T')$ commutes with tracing out the purifying space, and partial trace can only decrease trace distance, we have that the trace distance between the states after applying QRACM to the two different tables is at most

$$\delta_i := \sqrt{2 \sum_{j:T[j]\neq T'[j]} |\alpha_{ij0}|^2 + |\alpha_{ij1}|^2} \tag{B.11}$$

I then let

$$m_j := \sum_{i=1}^{d} |\alpha_{ij0}|^2 + |\alpha_{ij1}|^2 \tag{B.12}$$

We know that $\sum_{j=0}^{N} m_j = 1$. Thus, for any ℓ, there exist a set $\mathcal{J}$ of d indices such that

$$\sum_{j\in\mathcal{J}} m_j \leq \frac{\ell}{N}. \tag{B.13}$$

If I set T and T' to differ only on $\mathcal{J}$, we see that $\sum_{i=1}^{d} \delta_i^2 \leq \frac{2\ell}{N}$, from which we see that

$$\sum_{i=1}^{d} \delta_i \leq \frac{2\ell\sqrt{d}}{N}. \tag{B.14}$$

□

I now argue that since so little of the input state is concentrated on certain indices, I can replace any table T with a different table that differs on those indices, and this will be indistinguishable to the QRACM distillation-and-teleportation process. Since the process needs to handle all logical inputs, it must fail on that index.

Theorem B.2.1. *Suppose there is a QRACM state distillation-and-teleportation process, accessing tables of size $N = 2^n$ and making at most d calls to the physical QRACM. Then the minimal fidelity for this process is at most*

$$\frac{3}{4} + 2\frac{\sqrt{d}}{N}. \tag{B.15}$$

Proof. Let T be any table, and label the states input into the QRAM device during the distillation process as $\rho_1(T), \ldots, \rho_d(T)$. Let $\mathcal{J}$ be the set of ℓ indices implied by Lemma B.2.1 for these states, and let T' be a table that differs from T in those ℓ indices. I will show by induction that

$$\|\rho_i(T') - \rho_i(T)\|_1 \leq \sum_{j=1}^{i} \delta_j \tag{B.16}$$

with δ_i as in Lemma B.2.1. This holds for $i = 0$ since both take $|0\rangle\langle 0|$ as the starting state.

For induction, recall from the definition of the distillation process that

$$\rho_{i+1}(T) = \Phi_{i+1} \circ (I \otimes \mathcal{Q}(T))\rho_i(T) \tag{B.17}$$

(similarly for T'). Since the channel Φ_{i+1} can only reduce the trace distance between then by the data processing inequality, we obtain:

$$\begin{aligned}
&\|\rho_{i+1}(T') - \rho_{i+1}(T)\|_1 && \text{(B.18)}\\
&\quad \leq \|(I \otimes \mathcal{Q}(T'))\rho_i(T') - (I \otimes \mathcal{Q})(T)\rho_i(T)\|_1 && \text{(B.19)}\\
&\quad \leq \|((I \otimes \mathcal{Q}(T) - (I \otimes \mathcal{Q}(T'))\rho_i(T)\|_1 && \text{(B.20)}\\
&\qquad + \|(I \otimes \mathcal{Q}(T'))(\rho_i(T') - \rho_i(T))\|_1 && \text{(B.21)}\\
&\quad \leq \delta_{i+1} + \sum_{j=1}^{i} \delta_j && \text{(B.22)}
\end{aligned}$$

using Lemma B.2.1.

I finally note that $\rho_d(T) = \rho_{distill}(T)$, the state input into the final teleportation process. This means the distance between $\rho_{distill}(T)$ and $\rho_{distill}(T')$ is at most $\frac{2\ell\sqrt{d}}{N}$ by Lemma B.2.1. Again by the data processing inequality, the final teleportation process cannot increase the distance. Denoting the full process as $\tilde{\mathcal{U}}_{QRACM}(T)$, we see that for any logical input ρ_L, we obtain

$$\|(\tilde{\mathcal{U}}_{QRACM}(T) - \tilde{\mathcal{U}}_{QRACM}(T'))\rho_L\|_1 \leq \frac{2\ell\sqrt{d}}{N} \tag{B.23}$$

However, let $\rho_L = |j\rangle\langle j| \otimes |0\rangle\langle 0|$ for some j with $T[j] \neq T'[j]$. Then we have for the *ideal* QRACM access,

$$\|(\mathcal{U}_{QRACM}(T) - \mathcal{U}_{QRACM}(T'))(\rho_L)\|_1 = 1 \tag{B.24}$$

But by the triangle inequality,

$$\begin{aligned} 1 =& \|(\mathcal{U}_{QRACM}(T) - \mathcal{U}_{QRACM}(T'))(\rho_L)\|_1 && \text{(B.25)} \\ \leq& \|(\mathcal{U}_{QRACM}(T) - \tilde{\mathcal{U}}_{QRACM}(T))\rho_L\|_1 && \text{(B.26)} \\ &+ \|(\mathcal{U}_{QRACM}(T') - \tilde{\mathcal{U}}_{QRACM}(T'))\rho_L\|_1 && \text{(B.27)} \\ &+ \|(\tilde{\mathcal{U}}_{QRACM}(T) - \tilde{\mathcal{U}}_{QRACM}(T'))\rho_L\|_1 && \text{(B.28)} \\ 1 - \frac{2\ell\sqrt{d}}{N} \leq& \|(\mathcal{U}_{QRACM}(T) - \tilde{\mathcal{U}}_{QRACM}(T))\rho_L\|_1 && \text{(B.29)} \\ &+ \|(\mathcal{U}_{QRACM}(T') - \tilde{\mathcal{U}}_{QRACM}(T'))\rho_L\|_1 && \text{(B.30)} \end{aligned}$$

meaning (WLOG) that the distance between the realised QRACM channel and the ideal QRACM channel, for state ρ_L on access to table T, is at least $\frac{1}{2} - \frac{\ell\sqrt{d}}{N}$. Using known relations between trace distance and fidelity and setting $\ell = 1$ gives the result. □

C
QRAM Circuits

Here I show efficient QRAM constructions. None of this is my own work; however, I use these explicitly in Chapter 5 and Chapter 6, and their existence proves Theorem 4.3.1.

C.1 Unary

The simplest method for QRACM is a check-and-copy. That is, given a table T with N elements in $\{0,1\}^m$, and a state with an address register `addr`, do the following:

for $i = 0$ to $N-1$ **do**
  if `addr` $= i$ **then**
    Use `CNOT`s to write $T[i]$ to the output
  end if
end for

Here the "if" really means "control these operations based on this clause".

Checking the address costs $O(\log N)$ gates and can be done in depth $O(\log \log N)$, while copying out the result takes $O(m)$ gates and can be done in depth $O(m)$ with teleportation. Thus, the total gate cost is $O(N(\log N + m))$.

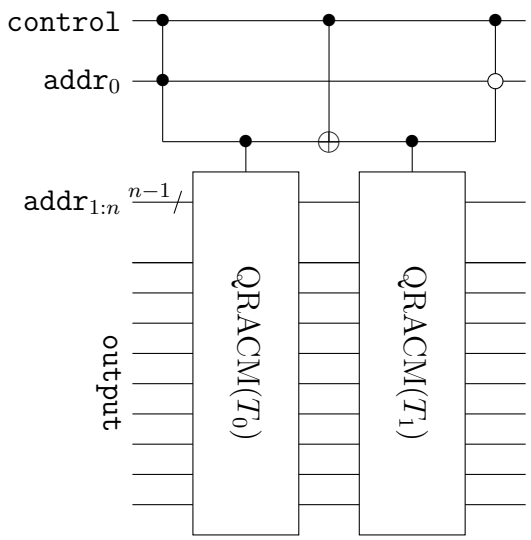


Figure C.1: Circuit for recursive case of unary QRACM from [BGB+18].

In [BGB+18], they show that with a unary encoding, checking each address costs only $O(1)$ gates. In short, Eve iterates from $i = 0$ to $N - 1$ more efficiently than a naive sequential order, instead ensuring that each subsequent number differs in only one bit when written in binary.

This is equivalent to the following recursive algorithm for *controlled* QRACM: The base case, a one-element table, can simply be written out with `CNOT` gates from the control qubit.

Then the recursive algorithm divides T into T_0 (all elements where the address starts with 0) and T_1 (all elements where the address starts with 1), both of which ignore the first address bit. Then we order the addresses in T_0 and T_1 such that the last address of T_0 is the first address of T_1. Then Figure C.1 shows the circuit.

You can see that each layer uses a constant number of gates of overhead, so we end up with a constant times N for the checks. The only part not in $O(n)$ is writing the output, which costs $O(Nm)$.

C.2 Select-Swap

From [JR23].

[LKS18] describe a technique (shown in Figure C.2) that reduces T-count even further, at the expense of more ancilla qubits (which can be in an arbitrary state,

however). The technique appears also in [CDS+22] and [BGM+19]. This method is analogous to "paging" in classical memory. In most classical memory devices, it is just as efficient to retrieve larger chunks of memory (called "pages") as it would be to retrieve single bits. Hence, the memory device retrieves a page and a second device will extract the specific word within that page. Analogously, this QRACM circuit uses a QRACM circuit on a larger page of the classical table, then uses a QRAQM access on that page.

The T-gate cost of unary QRACM (and depth) is independent of the length of each word in the memory table. From any table T with N entries in $\{0,1\}$, Eve can construct a table T' with $N/2^\ell$ entries in $\{0,1\}^\ell$, simply by grouping ℓ consecutive entries – a page – as one word in T'. Thus, to begin access to T, Eve starts with a unary QRACM access to T', writing the output in an ℓ-bit auxiliary register. She does not need the last ℓ bits of the address to do this, since the page she retrieves has every element that the last ℓ bits could address.

Once this is done, she wants to extract the precise bit that the last ℓ bits address. Since the required page is already in the auxiliary register, she uses a QRAQM access to do this. The address in the page for this QRAQM access is the last ℓ bits of the original address, and she writes the output to the original output register.

After this, she uncomputes the QRAQM access and uncomputes the original unary QRACM access.

To see why this works, notice that after the first QRACM access, the 2^ℓ auxiliary bits are now in a superposition, with the contents of these bits entangled with the high bits of the address register. Thus, a *single* QRAQM access extracts the correct output for each of these states in superposition.

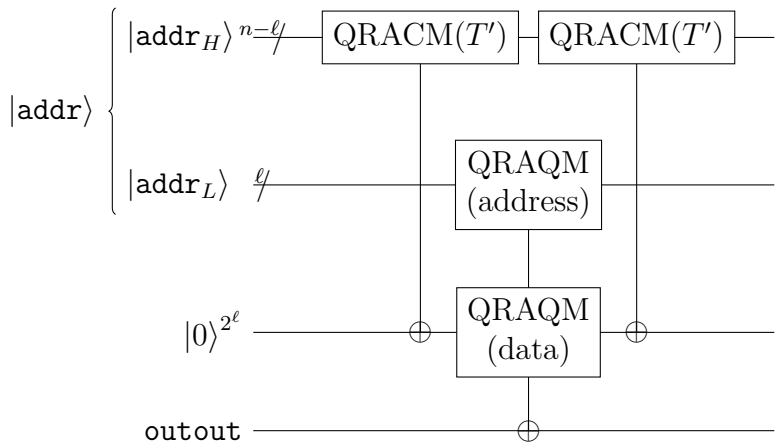


Figure C.2: Select-swap QRACM access, from [JR23]. T' is the table formed from T by combining each page of 2^ℓ consecutive bits of T into one word in T'.

Cost. The cost of the first unary QRACM access is $O(N/2^\ell)$ T-gates, in depth $O(N/2^\ell)$ generally. For the second QRAQM acess, it needs $\tilde{O}(2^\ell)$ gates and has depth $O(\ell)$. To minimize T-gates, a page size of $2^\ell \approx \sqrt{N}$ gives a total T-cost and depth of $O(\sqrt{N})$.

However, notice that the total gate cost is still $\Theta(N)$, since writing each word of T' requires $\Theta(2^\ell)$ gates. Thus, the majority of the gates in this circuit are multi-fanout CNOTs. If these are cheap, then this circuit is cheap overall.

The qubit requirements for the T-gate-optimal parameters are also $\Theta(\sqrt{N})$. Using other parameters, this can reach almost any combination of depth and width whose product is $\Theta(N)$. Even better, this method can use dirty qubits (i.e., qubits which may hold data necessary for other parts of the algorithm) for all but one word in each page.

The circuit in the appendix of [BGM+19] achieves this asymptotic cost, but they use a measurement-based uncomputation in both the QRAQM and QRACM access, and a second look-up to clean up the phases from this uncomputation. This optimizes the non-asymptotic gate counts. They use a QRAQM circuit based on controlled swaps, rather than the bucket-brigade; see Figure C.3 for an analogous technique that keeps depth and T-count small by using high-fanout CNOT gates. The circuits in [BGM+19] achieve the lowest T-count of any QRACM circuit to date.

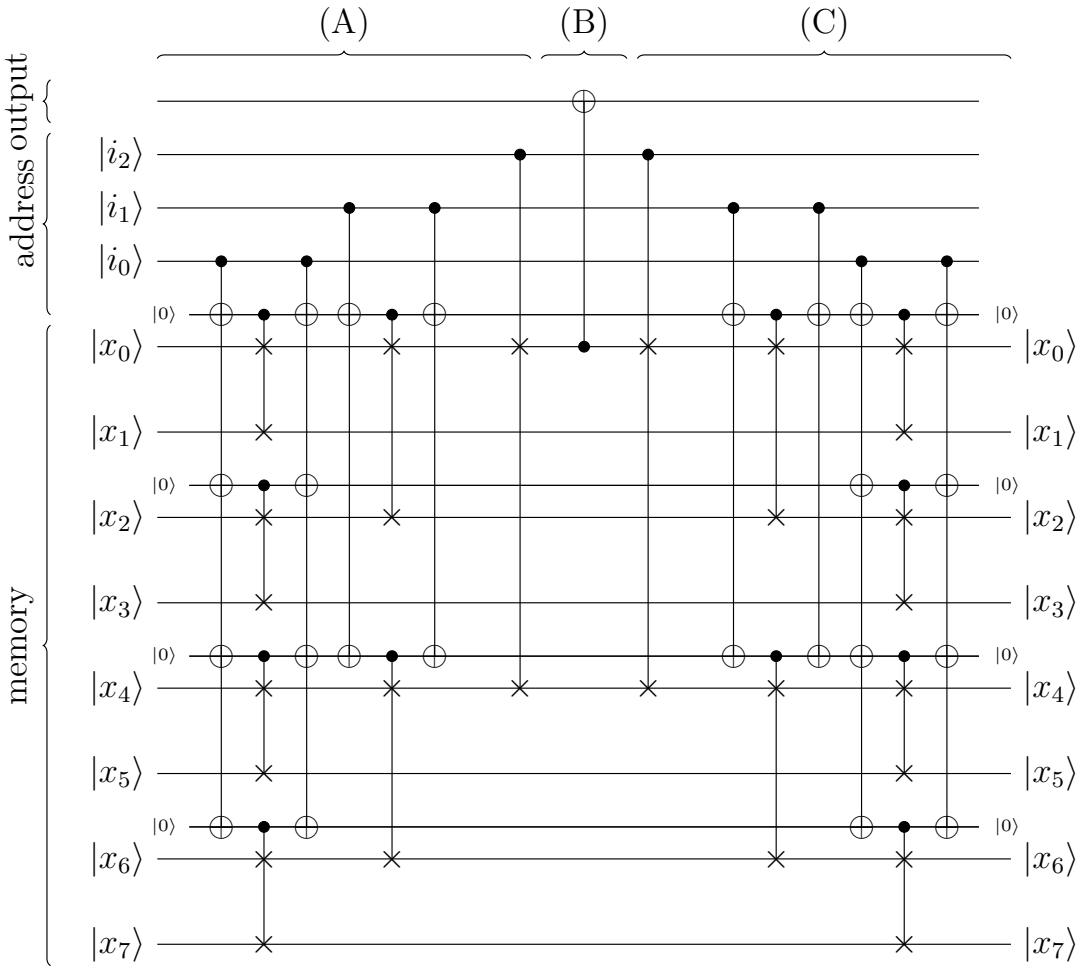


Figure C.3: Fanout-and-swap QRAQM for $N = 8$. (A) uses a binary tree of swaps to move $|x_i\rangle$ into the register for $|x_0\rangle$, where (B) copies it out (replacing this CNOT with a SWAP would produce a SWAP-QRAQM). Then (C) restores the memory register to its original state.

C.3 Uncomputation

The select-swap circuit Figure C.2 needs to uncompute the extra 2^ℓ qubits, and stage (C) of Figure C.3 must also uncompute the data after swapping. Uncomputing QRACM look-ups also has general applciations; see Section 6.2.3. Can Eve use measurement-based uncomputation to save costs?

[BGM+19] provide a method. Put very simply, Eve will measure all the qubits in the $|+\rangle\,/\,|-\rangle$ basis. This creates a binary string $\vec{v}$, and she knows that there is an unwanted phase on every state $|i\rangle$ in the address register such that $\vec{v} \cdot \vec{T[i]} = 1$ (treating $T[i]$ as a binary string, and an inner product over $\mathbb{F}_2$). Thus, she can

compute a new table F such that $F[i] = 1$ if and only if $\vec{v} \cdot \vec{T[i]} = 1$. Then she could do a QRACM look-up in F, writing the result to a qubit initialised in $|-\rangle$, to clean up the phase.

In fact, Eve can use the select-swap trick as well (here it is backwards, so more like swap-select). The idea is that Eve prepares an auxiliary register in the state $|-\rangle |+\rangle^{2^\ell - 1}$, then uses the first ℓ bits of the address to swap the $|-\rangle$ state to the ℓth position in the register. She prepares a table F' of size $N/2^\ell$, such that $F'[i] = (F[i2^\ell], F[i2^\ell + 1], \ldots, F[i2^\ell + 2^\ell - 1])$. You can see that if she "writes" that word onto the auxiliary registers, it adds a phase $F[i2^\ell + j]$ if and only if the $|-\rangle$ qubit was swapped into the jth position, which occurs if and only if the bottom ℓ bits of the address encode j. That is precisely what should happen.

Thus far the savings are minimal because the size of F is still N, the size of T. Eve has only saved the extra `CNOT` gates of writing large outputs. However, if the original table had m-bit outputs which Eve uncomputed, she might as well use those m bits as auxiliary qubits for a swap-select uncomputation, which will then be approximately $1/m$ the depth and T-cost of reversible uncomputation.

As a final step, Eve must uncompute the shuffled auxiliary qubits. In [BGM+19] this uses measurement-based uncomptuation and just Clifford gates, in logarithmic depth.

D
Pebbling

Bennett introduced pebble games in [Ben89]. They abstract the memory management of a straight-line, reversible computer program. I give a quick definition here:

Definition D.0.1 (Pebbling game)**.** *A pebbling game of length n and space m is a series of "moves" $x_0, \ldots x_t$, each represented by a binary vector of length n. The ith component of this vector is 1 if there is a "pebble" at that spot, and 0 otherwise. The rules are as follows:*

- *We want to start from no pebbles, and move a pebble to the end. Formally, x_0 and x_t are all zeroes, but there is some x_i such that the nth component of x_i is 1.*
- *We only have m pebbles. Formally, the hamming weight of all x_i is at most m.*
- *From a move x_i, we are only allowed to add or remove a pebble in front of a previous pebble (or at the start). Formally, $x_{i+1} \oplus x_i$ has Hamming weight 1, and if the non-zero component is in position j, then either $x_{i,j-1} = 1$ or $j = 1$.*

You can think of adding or removing a pebble as computing or uncomputing the next step in a computation. We have n steps of computation, and our goal is to reach the last one and leave no junk memory behind. The final step of computation

is really in space $n+1$, and we need one extra step to place a pebble there. For later recursive analysis, this description will be easier.

Knill [Kni92] proved thorough lower bounds for straight-line pebbling games. Real computing problems, like the CSIDH group action, are unlikely to be straight-line programs, but solving this problem for a more general directed acyclic graph is trickier.

Spooky pebbling is the fun name for a quantum-only method of pebbling, due to Gidney [Gid19b] and expanded in [KSS21]. The insight is that if we have a quantum state like this:

$$\sum_x \alpha_x |x\rangle |f(x)\rangle \tag{D.1}$$

where $f(x)$ is a single bit, then we can measure the last register in the X-basis without damaging the state, as follows:

$$\sum_x \alpha_x |x\rangle |f(x)\rangle \xrightarrow{H \text{ on last qubit}} \sum_x \alpha_x |x\rangle \left(|0\rangle + (-1)^{f(x)} |1\rangle\right) \tag{D.2}$$

$$\xrightarrow{\text{measure in } |0\rangle/|1\rangle \text{ basis}} \begin{cases} \sum_x \alpha_x |x\rangle & \text{, if measured 0} \\ \sum_x \alpha_x (-1)^{f(x)} |x\rangle & \text{, if measured 1} \end{cases} \tag{D.3}$$

Measuring 0 means we uncomputed for free; measuring 1 means we need to remove the phase. One way to remove the phase is to recompute $f(x)$ and apply a Z gate, which seems like it hasn't saved us any computation, except (a) we might be able to remove the phase slightly cheaper (this is precisely what the `AND` uncomptuation circuit does, and it needs no T gates to uncompute!), and (b) the qubit we measured is now free for re-use until we want to remove the phase.

Doing this in a pebble game means that we can use this measurement to make a phase, or a "ghost pebble": we still need to uncompute the pebble, but it does not count towards our pebble budget. Let us define this formally.

Definition D.0.2 (Spooky pebble game)**.** *A spooky pebble game of size n, memory m is a series of moves $x_1, \ldots, x_t$, such that each x_i is a vector with entries in $\{-1, 0, 1\}$ of length n, with the following rules:*

- *x_1 and x_t are all zeroes, but there is some x_i such that the nth component of x_i is* 1.
- *Each x_i can have at most m entries of* 1.
- *x_{i+1} can differ from x_i in at most one entry. Let this entry be j:*
  - *j can change from* 1 *to* -1 *with no restrictions;*
  - *for any other change, the $j-1$ entry of x_i must be* 1.

In this definition, ghosting a pebble is guaranteed to create a ghost pebble. For single bits, we saw that we only have a 1/2 chance of creating a ghost. However, if each pebble is a large register (say, 128 bits or more), then the presence of some unwanted phase is essentially guaranteed.

There is a natural recursive structure to pebbling games, both classical and spooky. If we have a pebble at space j, and a pebble (or ghost pebble) at space $j+\ell$, and we have m' pebbles remaining, then any strategy to solve the pebble game for ℓ steps with m' pebbles will work.

D.1 CSIDH Group Action Pebbling

Here I give a short overview of the pebbling strategy we used to estimate the CSIDH group action oracle. There are two basic strategies:

- High memory: If $m \geq n$, we place one pebble after another sequentially, then remove all but the last one in reverse order. This takes $2n-1$ steps.
- Low memory: If $m = 2$, we place one pebble after the last, then ghost the first, and repeat (call this a "two-at-a-time" strategy). This takes $2n$ steps to fill the last entry, but fills every other space with a ghost. We then recurse to uncompute the ghost in space n. This takes $2(n-1)$ steps. Overall, the number of steps is

$$\sum_{i=0}^{n-1} 2(n-i) = n(n-1). \tag{D.4}$$

This gives our recursive strategy for m pebbles. Set $k \approx n^{\frac{m-1}{m}}$. We then use the forward strategy of 2 pebbles to get to the space at $n-k$. We then leave a pebble there, and recurse with m pebbles on the remaining space. The sub-strategy must reach the end, and leave all remaining spaces clear. Then we ghost the pebble at space $n-k$, and repeat: we use two-at-a-time to reach space $n-2k$, leave one pebble there, and recurse with $m-1$ pebbles.

Letting $C(n,m)$ be the number of steps for a pebble game of size n and memory m, our total cost ends up as

$$\left\lceil \frac{n}{k} \right\rceil C(k, m-1) + \sum_{i=0}^{\lfloor n \rfloor k} ik. \tag{D.5}$$

To find an optimal strategy, we set $k = n^{\frac{m-1}{m}}$ and computed costs recursively, using the low memory strategy for the base case. We tried m from $\frac{1}{2}\lg n$ to $5\lg n$ and chose the result with the lowest value of $m\times$total number of steps.

D.2 Claw-Finding Pebbling

To iterate a function in Chapter 7, Eve can use the following strategy for an n-step pebble game with m pebbles:

1. Use the low memory strategy to place a pebble at location $\frac{n}{2}$.
2. Recursively finish the remaining $\frac{n}{2}$ steps with $m-1$ pebbles.
3. Ghost the pebble at $\frac{n}{2}$, then recursively clear phases in the first $\frac{n}{2}$ steps.

The cost is thus

$$C(n,m) = C(\tfrac{n}{2}, m-1) + C(\tfrac{n}{2}, m) + \tfrac{n}{2}. \tag{D.6}$$

I want to inductively show that if $m \geq \lg(n)$, $C(n,m) \leq C_0 n \lg(n)$ for a constant C_0.

For the base case, find some $n_0 \geq 2$ and $C_0 \geq \frac{1}{2}$ such that for all $n \leq n_0$ and any $m \geq \lg(n)$, $C(n,m) \leq C_0 n \lg(n)$. Then I inductively show that for general n, if $m \geq \lg(n)$ then $C(n,m) \leq C_0 n \lg(n)$. The approach given above maintains

the requirement that $m \geq \lg(n)$, since we drop n to $n/2$ when m drops to $m-1$. Then for the cost, taking $n = n_0 + 1$,

$$\begin{aligned} C(n,m) =& C(\tfrac{n}{2}, m-1) + C(\tfrac{n}{2}, m) + \tfrac{n}{2} && \text{(D.7)} \\ \leq& C_0 \tfrac{n}{2} \lg(\tfrac{n}{2}) + C_0 \tfrac{n}{2} \lg(\tfrac{n}{2}) + \tfrac{n}{2} && \text{(D.8)} \\ =& C_0 n \lg(n) - C_0 n + \tfrac{n}{2}]] && \text{(D.9)} \\ \leq& C_0 n \lg(n) && \text{(D.10)} \end{aligned}$$

Thus, taking $m = \lg n$ gives log-linear time.

E

Proof for Chapter 6

E.1 Proof of Theorem 6.3.2

Recall the statement:

Theorem E.1.1 (Theorem 4.2,[CSCDJRH21]). *Suppose an initial phase vector has length L, fidelity $q < \frac{1}{2}$, and height H. If Eve collimates this to a new height H', the resulting defective phase vector has expected fidelity at most*

$$q + 4\sqrt{\frac{\ln(L')}{L'}} \tag{E.1}$$

for $L' := \frac{H}{H'}Lq \geq 40$.

Proof. This follows [hBO]. Let $K = \{K_1, \ldots, K_s\}$ be all possible collimation measurement results (e.g., the value of the phase divided by H'). Let each of the qL states in superposition be i.i.d. random variables X_i with values in K, so we have $p_i := \Pr[X = K_i]$. Because the states are in uniform superposition, imagine that measurement selects one such state X_j. Then let W_j be the number of other states in the superposition (those with the same value as X_j); this will equal $1 + \sum_{i \neq j} 1_{X_i = X_j}$. If we condition on $X_j = K_m$ for some m, we get that

$$W_j | (X_j = K_m) = 1 + \sum_{i \neq j} 1_{X_i = K_m} \sim 1 + \mathrm{Bin}(qL - 1, p_m) \tag{E.2}$$

and this gives us

$$\Pr[W_j = w] = \sum_{m=1}^{s} \Pr[W_j = w \mid X_j = K_m] \underbrace{\Pr[X_j = K_m]}_{=p_m} \tag{E.3}$$

$$= \binom{qL-1}{w-1} \sum_{m=1}^{s} p_m^w (1-p_m)^{qL-w}. \tag{E.4}$$

For the first collimation, p_m is uniformly random and equal to $H'/H = 1/s$, since the phases are uniformly distributed. This means the above equation is just a binomial distribution, so the length of any state after measurement (let X denote this length) has distribution $1 + \mathrm{Bin}(qL-1, H'/H)$. The phases Eve believes she will have is given by the same probability, except with L random variables X_i instead of qL. Thus, the distribution of incorrect phases (those Eve believes are in the superposition, but are not) is given as $Y \sim \mathrm{Bin}(L-qL, H'/H)$.

Let $p = H'/H$. The fidelity of the measured state is $\frac{X}{X+Y}$, so to bound this, I use standard binomial distribution concentration bounds to constrain X and Y to within a factor of $1 \pm \delta$ of their means, except with probability $\epsilon := 2\exp(-\mathbb{E}[x]\delta^2/3) + 2\exp(-\mathbb{E}[y]\delta^2/3)$. I take $\delta = \sqrt{3\ln(L')/L'}$, and since $q < \frac{1}{2}$, this gives $\epsilon < \frac{5}{L'}$.

Because we already know $\frac{X}{X+Y} \leq 1$, we can use these bounds to show

$$E\left[\frac{X}{X+Y}\right] \leq \frac{1+\delta}{1-\delta} \frac{p(qL-1)+1}{p(qL-1)+1+p(L-qL)} + \epsilon \tag{E.5}$$

which can be rearranged to

$$q + \frac{q}{L'} + 2\sqrt{3}\sqrt{\frac{\ln(L')}{L}} + \frac{5}{L'} \tag{E.6}$$

which gives the bound for large enough L'. □

After each level of collimation, the phases should be uniform up to the height, and so the distribution of the sum of phases from two phase vectors should form a triangular distribution. This would give p_m in Equation (E.4). This does not work exactly because the phase multiples of each state in superposition are not independent, since they are formed as all possible sums of two other sets of phases. However, I will assume phases *are* independent and derive another bound.

Lemma E.1.1. *Suppose Eve collimates two phase vectors of length L and fidelity $q \leq \frac{\sqrt{2}-1}{2\sqrt{2}} \approx 0.146$. Modelling the phases of the joint distribution as i.i.d. triangular variables, then the probability that the collimated state has more than $\frac{q}{2}L$ states in superposition is at most $q^2 + O(\frac{1}{qL})$.*

Proof. I will estimate how many phases will end up in the collimated vector. The same logic holds as before, where if I let X_j be a randomly selected phase, I can jump to Equation (E.4),

$$\Pr[W = w] = \binom{N-1}{w-1} \sum_{m=1}^{s} p_m^w (1-p_m)^{N-w} \tag{E.7}$$

where N is the number of *joint* states in superposition (i.e., q^2L^2 for two phase vectors of length L and fidelity q).

From this I compute any moment of W as

$$\mathbb{E}[W^t] = \sum_{w=1}^{N} \binom{N-1}{w-1} \sum_{m=1}^{s} p_m^w (1-p_m)^{N-w} \tag{E.8}$$

$$(\text{let } w' = w-1,\ N' = N-1) \tag{E.9}$$

$$= \sum_{m=1}^{s} \sum_{w'=0}^{N'} \binom{N'}{w'} p_m^{w'+1} (1-p_m)^{N'-w'} \tag{E.10}$$

$$= \sum_{m=1}^{s} p_m \mu(N', p_m, t) \tag{E.11}$$

where $\mu(N, p, t)$ denotes the t^{th} moment of $\text{Bin}(N, p)$.

Let $h = \frac{H'}{H}$, the range of possible quotients from one of the input phase vectors. Then range of possible measurements s equals $2\frac{H'}{H}$. Under the assumption that p_m is triangular, i.e.

$$p_m = \begin{cases} \frac{m}{h^2}, & m \leq h \\ \frac{2h-m}{h^2}, & h < m \leq 2h \end{cases} \tag{E.12}$$

then I can compute the first two moments of W, noting that $\mu(1, N, p_m) = Np_m$ and $\mu(2, N, p_m) = N^2p_m^2 - Np_m^2 + Np_m$.

$$\mathbb{E}[W] = 1 + \sum_{m=0}^{2h-1} N' p_m^2 \tag{E.13}$$

$$= 1 + N \sum_{m=0}^{h} \frac{m^2}{h^4} + N \sum_{m=h+1}^{2h} \frac{(2h-m)^2}{h^4} \tag{E.14}$$

$$=1+(N-1)\frac{2h^3+h}{3h^4} \tag{E.15}$$

and

$$\mathbb{E}[W^2]=\sum_{m=0}^{s}(N'^2-N')p_m^3+N'p_m^2 \tag{E.16}$$

$$=(N^2-3N+2)\frac{h^2(h^2+1)}{2h^6}+(N-1)\frac{2h^3+h}{3h^4} \tag{E.17}$$

so

$$\mathrm{Var}[W]=(N-1)\frac{(N-10)h^6-3h^4+12h^7-(N-1)16h^2}{18h^8} \tag{E.18}$$

Here I note that $N=q^2L^2$ for two input phase vectors of fidelity q, and Eve will parameterise such that $\frac{2}{3}\frac{L^2}{h}=L$. Substituting these gives $\mathbb{E}[W]=1+q^2L-\frac{1}{L}+O(\frac{1}{L^3})$ and $\mathrm{Var}[W]=\frac{q^4L^2}{8}+O(q^2L)$.

With Chebyshev's inequality, I can conclude

$$\Pr\left[W\geq\tfrac{q}{2}L\right]\leq\Pr\left[|W-\mathbb{E}[W]|\geq(\tfrac{1}{2}-q)qL+\frac{1}{L}+O(\tfrac{1}{L^3})\right] \tag{E.19}$$

$$\leq\frac{\mathrm{Var}[W]}{\frac{(1-2q)^2}{4}q^2L^2+O(q)} \tag{E.20}$$

$$\leq\frac{q^2}{2(1-2q)^2}+O(\tfrac{1}{qL}) \tag{E.21}$$

$$\tag{E.22}$$

If $q\leq\frac{1}{2}-\sqrt{\frac{1}{8}}$, then this is at most $q^2+O(\frac{1}{qL})$. □

F

Properties of Iterated Functions

Throughout this section I will repeatedly use Stirling's formula:

$$n! = \sqrt{\tau n}\left(\frac{n}{e}\right)^n e^{O(1/n)} = \sqrt{\tau n}\left(\frac{n}{e}\right)^n \left(1 + O(\tfrac{1}{n})\right). \tag{F.1}$$

Recall the definition of predecessors:

Definition F.1.1. *For a function h and an element $x \in X$, the set of* predecessors *of x is*

$$\mathcal{P}_x = \{y \in X | h^n(y) = x, n \geq 0\}. \tag{F.2}$$

Lemma F.1.1 (Lemma B.1, [JS21])**.** *The probability of choosing a random function $h : X \to X$ such that $P_x = t$ is given by*

$$Pr[P_x = t] = \frac{t^{t-1}}{e^t t!}\left(1 + O(\tfrac{1}{N})\right). \tag{F.3}$$

for $t = o(N)$. In particular, $Pr[P_x \geq t] = \Theta(1/\sqrt{t})$.

Proof. To prove this I will count all functions h such that $P_x = t$.

First, I can choose the set of predecessors of x. There will be $t-1$ elements which are not x, chosen out of the $N-1$ elements in X which are not x, so there are $\binom{N-1}{t-1}$ such elements. To form these into a tree, Cayley's formula tells us there are t^{t-2} undirected trees on t elements. Each tree can be uniquely directed so that

x is the root, and each defines a distinct action for the $t-1$ elements which are predecessors of x.

The remaining $N-t$ points must only map to themselves (or else they would also be predecessors of x). Thus, there are $(N-t)^{N-t}$ ways to do this. Finally, h maps x to some other point; there are N possibilities since any choice is distinct and does not change the number of predecessors of x.

Since there are N^N functions from X to X, this gives us a probability of

$$\frac{\binom{N-1}{t-1}t^{t-2}N(N-t)^{N-t}}{N^N} = \frac{t^{t-1}}{t!}\frac{N!}{N^N}\frac{(N-t)^{N-t}}{(N-t)!} \tag{F.4}$$

Applying Stirling's formula (F.1) to the terms with N in the above, I get

$$\frac{t^{t-1}}{t!}\sqrt{\tau N}\frac{N^N}{e^N\cdot N^N(1+O(1/N))}\frac{e^{N-t}(N-t)^{N-t}}{\sqrt{\tau(N-t)}(N-t)^{N-t}(1+O(1/(N+t))} \tag{F.5}$$

$$=\frac{t^{t-1}e^{-t}}{t!}\sqrt{\frac{N}{N-t}}\left(1+O(\tfrac{1}{N})\right) \tag{F.6}$$

We can see that $\sqrt{\frac{N}{N-t}} = 1+O(1/N)$, giving the first result.

For the specific case, I again use Stirling's formula with $t!$ to get

$$\frac{t^{t-1}e^t}{e^t t^t\sqrt{\tau t}(1+O(1/t))}(1+O(\tfrac{1}{N})) = \frac{1}{\sqrt{\tau t^3}}(1+O(\tfrac{1}{t})) \tag{F.7}$$

Approximating the sum of all $t \geq t_0$ with an integral, we find $\Pr[P_x \geq t] = \Theta(\frac{1}{\sqrt{t}})$. □

Lemma F.1.2 (Lemma B.2, [JS21])**.** *Fix distinct $x, y \in X$, and let h be a random function under the restriction that $h(x) = h(y)$. Then for $t, s \in o(N)$,*

$$Pr[P_x = t, P_y = s] = \frac{t^{t-1}}{e^t t!}\frac{s^{s-1}}{e^s s!}\left(1+O(\tfrac{1}{N})\right) \tag{F.8}$$

and the probability that x and y both have at least t predecessors is $\Theta(1/t)$.

Proof. Again, I will show this by counting functions with the required property. First note that x and y cannot be in a cycle, since $h(x) = h(y)$. Thus, either their predecessors are disjoint, or x is a predecessor of y. I assume $s \geq t$ without loss of generality, meaning that y cannot be a predecessor of x (since x is in its own set of predecessors, that would imply $P_x > P_y$).

Then I will analyse the case when the predecessors are disjoint. The logic is the same as Lemma F.1.1: I choose $t-1$ predecessors of x out of $N-2$ elements which are not x or y and form them into one of t^{t-2} trees, and then I choose $s-1$ predecessors of y out of $X \setminus \mathcal{P}_x \setminus \{y\}$, which has size $N-t-1$, and make one of s^{s-2} trees. At this point I must choose $h(x)$ (which is also $h(y)$), but I cannot choose any of the predecessors of y or x because I assumed the sets are disjoint. Thus, I have $N-s-t$ choices for $h(x)$. Finally, there are $(N-t-s)^{N-t-s}$ choices for the remainder of X.

The total number of functions such that $h(x) = h(y)$ is N^{N-1}: pick any output for all elements except y, then set $h(y)$ to equal $h(x)$. This gives a probability of such a function as

$$\frac{\binom{N-2}{t-1} t^{t-2} \binom{N-t-1}{s-1} s^{s-2} (N-t-s)(N-t-s)^{N-t-s}}{N^{N-1}} \tag{F.9}$$

which I can rearrange and simplify to

$$\frac{t^{t-1}}{t!} \frac{s^{s-1}}{s!} \frac{N!}{N^N} \frac{(N-t-s)^{N-t-s}}{(N-t-s)!} \frac{N-t-s}{N-1}. \tag{F.10}$$

Just as in Lemma F.1.1, Stirling's formula (F.1) takes this to

$$\frac{t^{t-1}}{e^t t!} \frac{s^{s-1}}{e^s s!} \left(1 + O(\tfrac{1}{N})\right). \tag{F.11}$$

Thus, I must now show that the case where x is a predecessor of y is an $O(1/N)$ term.

If x is a predecessor of y, I choose the other $s-2$ predecessors of y out of $N-2$, then choose $t-1$ of *those* to be predecessors of x. I form a tree behind x (t^{t-2} choices), and a disjoint tree behind y ($(s-t)^{s-t-2}$ choices). This gives me $s-t$ choices for $h(x)$, which must "attach" to the tree behind y somewhere in order for x to be a predecessor of y. This also gives me $h(y)$, since $h(y) = h(x)$. As usual, the $N-s$ other elements map to themselves. The total probability is thus

$$\frac{\binom{N-2}{s-2} \binom{s-2}{t-1} t^{t-2} (s-t)^{s-t-2} (N-s)^{N-s}}{N^{N-1}} \tag{F.12}$$

and I re-arrange to

$$\frac{t^{t-1}}{t!}\frac{(s-t)^{s-t}}{(s-t)!}\frac{N!}{N^N}\frac{(N-s)^{N-s}}{(N-s)!}\frac{1}{N-1} \tag{F.13}$$

Stirling's formula (F.1) again gives

$$\frac{t^{t-1}}{t!e^t}\frac{(s-t)^{s-t}}{e^{s-t}(s-t)!}\frac{1}{N-1}\left(1+O(\tfrac{1}{N}\right) \tag{F.14}$$

and further gives

$$\frac{t^{t-1}}{t!e^t}\frac{s^{s-1}}{s!e^s}\frac{\sqrt{\tau s}(1+O(1/s))}{\sqrt{\tau(s-t)}(1+O(\frac{1}{s+t}))}\frac{1}{N-1}\left(1+O(\tfrac{1}{N}\right) \tag{F.15}$$

which fits into the $O(1/N)$ term in Equation (F.11).

The special case, showing the probability that x and y both have predecessors of that size, can be obtained by separate integral approximations of the sums over s and t. □

Lemma F.1.3 (Lemma B.3, [JS21]). *Let $x, y \in X$ and let n_x be the largest integer such that there there is some point $p \in X$, $p \neq x$, with n_x as the minimum value such that $h^{n_x}(p) = x$. Define n_y similarly. Suppose x has t predecessors and y has s predecessors. For any $c > 1$, the probability that $n_x > c\sqrt{\tau t}$ or $n_y > c\sqrt{\tau s}$ is at most*

$$\frac{\tau-6}{3(c-1)^2}\left(1+O(\tfrac{s}{N}\right). \tag{F.16}$$

Proof. As in Lemma F.1.2, the probability that x and y share predecessors can fit in the $O(\frac{s}{N})$ term, so we assume their predecessors are disjoint.

By [RS67], the height of a random tree on t vertices has expected value $\sqrt{\tau t}$, with variance $\frac{\tau(\tau-6)t}{6}$. Chebyshev's inequality tells us that $n_x > c\sqrt{\tau t}$ with probability at most $\frac{\tau-6}{6(c-1)^2}$, and same for $n_y > c\sqrt{\tau s}$. The union bound gives the result. □

Lemma B.4 in [JS21] claims that there will be about u^2 predecessors of distance at most u, even if there are more than u^2 predecessors total; however, I believe there is a flaw in the proof. Specifically, we argued by selecting a random subset of size u, forming a tree from them, then forming the rest of the predeceesor tree by

adding other vertices to that set. However, this double-counts cases where the tree's depth is small, but these are precisely the sets we want to give a lower bound for!

Instead, in the next theorem, I simply argue that having too many predecessors is unlikely.

Theorem F.1.1 (Theorem B.1, [JS21])**.** *Let h be a function such that $h(x) = h(y)$, where x has t predecessors and the largest trail leading to x has $n_x \leq u$ points. Let y, s, and n_y be defined analogously. Then if Eve iterated h for u iterations in each step in the quantum random walk, using sets of size R, the fraction of marked vertices is*

$$\Theta\left(\frac{R^2ts}{N^2}\right). \tag{F.17}$$

Proof. Define the u-predecessors of x as

$$\mathcal{P}_u(x) = \{p \in X | h^m(p) = x, 0 \leq m \leq u\}. \tag{F.18}$$

A set in the random walk has R random points from X, and it is marked if and only if it contains exactly one point from $\mathcal{P}_u(x)$ and one from $\mathcal{P}_u(y)$. By assumption, $n_x \leq u$ and $n_y \leq u$, so the sizes of these sets are t and s, respectively.

Constructing R is like sampling without replacement from X, but since $N \gg R$, I can approximate it as sampling *with* replacement as a multinomial distribution:

$$\binom{R}{2}\frac{t}{N}\frac{s}{N}\left(1 - \frac{t+s}{N}\right)^{R-2} = \Theta\left(\frac{R^2ts}{N}\right). \tag{F.19}$$

□

Theorem F.1.2 (Theorem B.3, [JS21])**.** *Suppose h is a random function such that x has at least t predecessors and y has at least s predecessors. Then with probability at least*

$$\frac{\tau - 6}{3(c-1)^2}\left(1 - O(\tfrac{\sqrt{t}+\sqrt{s}}{u})\right), \tag{F.20}$$

the fraction of marked vertices is at least

$$\Omega\left(\frac{R^2 \min\{u^2, t\} \min\{u^2, s\}}{N^2}\right). \tag{F.21}$$

Proof. I first consider the case where x has exactly $k_x \geq t$ predecessors (and k_y for y). If $k_x \leq u^2$ and $k_y \leq u^2$, then by Lemma F.1.3, all k_x predecessors will be within distance at most u (with the given probability).

The probability that $k_x > u^2$ is $O(1/u)$ by Lemma F.1.1, so the probability that $k_x > u$ is at most $O(\frac{\sqrt{t}}{u})$, since have already assumed that h has at least t predecessors. The union bound with $k_y > u$ gives the result. □

Bibliography

[AAB+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, October 2019.

[AB99] Mikhail J. Atallah and Marina Blanton. *Algorithms and Theory of Computation Handbook*, volume 1. CRC Press LLC, 1999.

[ACVCD+] Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javir Chi-Domínguez, Alfred Menezes, and Francisco Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In *Selected Areas in Cryptography – SAC 2018*, pages 322–343. LNCS 11349.

[AGPS20] Martin R. Albrecht, Vlad Gheorghiu, Eamonn W. Postlethwaite, and John M. Schanck. Estimating quantum speedups for lattice sieves. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 583–613, Cham, 2020. Springer International Publishing.

[Amb07] A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM J. Computing*, 37:210–239, 2007.

[AMD22] AMD. AMD Instinct™ MI250X accelerator, 2022. `https://www.amd.com/en/products/server-accelerators/instinct-mi250x`.

[AS04] S. Aaronson and Y. Shi. Quantum lower bounds for the collision and the element distinctness problems. *J. ACM*, 51(4):595–605, July 2004.

[Bab86] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, March 1986.

[BBG+13] Robert Beals, Stephen Brierley, Oliver Gray, Aram W. Harrow, Samuel Kutin, Noah Linden, Dan Shepherd, and Mark Stather. Efficient distributed quantum computing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 469(2153):20120686, May 2013.

[BBJ+09] E. B. Barker, W. E. Burr, A. C. Jones, W. T. Polk, S. W. Rose, M. E. Smid, and Q. H. Dang. Recommendation for key management, part 3. Technical report, National Institute of Standards and Technology, 2009.

[BBSS20] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved classical and quantum algorithms for subset-sum. In *Advances in Cryptology – ASIACRYPT 2020*, pages 633–666. Springer International Publishing, 2020.

[BBvHL20] Gustavo Banegas, Daniel J. Bernstein, Iggy van Hoof, and Tanja Lange. Concrete quantum cryptanalysis of binary elliptic curves. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):451–472, Dec. 2020.

[BDK+18] Joppe Bos, Leo Ducas, Eike Kiltz, T. Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. CRYSTALS - Kyber: A CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, April 2018.

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989.

[Ber01] Daniel J. Bernstein. Circuits for integer factorization: a proposal, 2001. `https://cr.yp.to/papers.html#nfscircuit`.

[Ber09] Daniel J. Bernstein. Cost analysis of hash collisions: Will quantum computers make sharcs obsolete? *Workshop Record of SHARCS'09: Special-purpose Hardware for Attacking Cryptographic Systems*, 2009.

[BFLS20] Daniel J. Bernstein, Luca De Feo, Antonin Leroux, and Benjamin Smith. Faster computation of isogenies of large prime degree. *Open Book Series*, 4(1):39–55, December 2020.

[BGB+18] Ryan Babbush, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. Encoding electronic spectra in quantum circuits with linear T complexity. *Phys. Rev. X*, 8:041015, Oct 2018.

[BGG+20] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. Comparing the difficulty of factorization and discrete logarithm: A 240-digit experiment. In *Advances in Cryptology – CRYPTO 2020*, pages 62–91. Springer International Publishing, 2020.

[BGM+19] 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, December 2019.

[BGMW92] Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, and David Bruce Wilson. Fast exponentiation with precomputation (extended abstract). In *Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings*, volume 658 of *Lecture Notes in Computer Science*, pages 200–207. Springer, 1992.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

[BHMT02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In *Quantum computation and information*, pages 53–74. American Mathematical Society, 2002.

[BK22a] Nouédyn Baspin and Anirudh Krishna. Connectivity constrains quantum codes. *Quantum*, 6:711, May 2022.

[BK22b] Nouédyn Baspin and Anirudh Krishna. Quantifying nonlocality: How outperforming local quantum codes is expensive. *Phys. Rev. Lett.*, 129:050505, Jul 2022.

[BKP20] Ward Beullens, Shuichi Katsumata, and Federico Pintore. Calamari and falafl: Logarithmic (linkable) ring signatures from isogenies and lattices. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 464–492, Cham, 2020. Springer International Publishing.

[BKS22] Michael Beverland, Vadym Kliuchnikov, and Eddie Schoute. Surface code compilation via edge-disjoint paths. *PRX Quantum*, 3:020342, May 2022.

[BKV19] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 227–247. Springer, 2019.

[BL13] Daniel J. Bernstein and Tanja Lange. Non-uniform cracks in the concrete: The power of free precomputation. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, pages 321–340, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BLMP19] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, and Lorenz Panny. Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 409–441. Springer, 2019.

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

[BLP+16] Benjamin J. Brown, Daniel Loss, Jiannis K. Pachos, Chris N. Self, and James R. Wootton. Quantum memories at finite temperature. *Reviews of Modern Physics*, 88(4), November 2016.

[BMFK05] F. Bahr, M. Moehm, J. Franke, and T. Kleinjung. We have factored RSA640 by GNFS, 2005. `https://mathworld.wolfram.com/news/2005-11-08/rsa-640/`.

[BMN+21] Ryan Babbush, Jarrod R. McClean, Michael Newman, Craig Gidney, Sergio Boixo, and Hartmut Neven. Focus beyond quadratic speedups for error-corrected quantum advantage. *PRX Quantum*, 2:010103, Mar 2021.

[BMR+20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[BP95] G. Bilardi and F.P. Preparata. Horizons of parallel computation. *Journal of Parallel and Distributed Computing*, 27(2):172–182, 1995.

[BRS06] Stephen D. Bartlett, Terry Rudolph, and Robert W. Spekkens. Dialogue concerning two views on quantum coherence: Factist and fictionist. *International Journal of Quantum Information*, 04(01):17–43, February 2006.

[BS20] Xavier Bonnetain and André Schrottenloher. Quantum security analysis of CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020-Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 493–522. Springer, 2020.

[CD97] Matt Curtin and Justin Dolske. A brute force search of DES keyspace. *Interhack*, 1997. `http://web.interhack.com/publications/des-key-crack`.

[CD23] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH, 2023.

[CDEH+21] Kevin C. Chen, Wenhan Dai, Carlos Errando-Herranz, Seth Lloyd, and Dirk Englund. Heralded quantum random access memory in a scalable photonic integrated circuit platform. In *Conference on Lasers and Electro-Optics*. Optica Publishing Group, 2021.

[CDS+22] B. David Clader, Alexander M. Dalzell, Nikitas Stamatopoulos, Grant Salton, Mario Berta, and William J. Zeng. Quantum resources required to block-encode a matrix of classical data, 2022.

[CGGI19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, April 2019.

[CGH+17] Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *Proceedings of the IEEE*, 105(9):1666–1704, 2017.

[CH10] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, page 21, USA, 2010. USENIX Association.

[CHI+18] Carlo Ciliberto, Mark Herbster, Alessandro Davide Ialongo, Massimiliano Pontil, Andrea Rocchetto, Simone Severini, and Leonard Wossnig. Quantum machine learning: a classical perspective. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2209):20170551, January 2018.

[Cho22] Charles Q. Choi. The beating heart of the world's first exascale supercomputer. IEEE Spectrum, 2022. `https://spectrum.ieee.org/frontier-exascale-supercomputer`.

[CL84] H. Cohen and H. W. Lenstra. Heuristics on class groups of number fields. In Hendrik Jager, editor, *Number Theory Noordwijkerhout 1983*, pages 33–62, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg.

[CLM+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 -Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018.

[CMN+18] Andrew M. Childs, Dmitri Maslov, Yunseong Nam, Neil J. Ross, and Yuan Su. Toward the first quantum simulation with quantum speedup. *Proceedings of the National Academy of Sciences*, 115(38):9456–9461, 2018.

[CNPS17] André Chailloux, María Naya-Plasencia, and André Schrottenloher. An efficient quantum collision search algorithm and implications on symmetric cryptography. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 211–240, Cham, 2017. Springer International Publishing.

[CSCDJRH21] Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. The SQALE of CSIDH: sublinear vélu quantum-resistant isogeny action with low exponents. *Journal of Cryptographic Engineering*, August 2021.

[DKL+18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, February 2018.

[DKRS04] Thomas G. Draper, Samuel A. Kutin, Eric M. Rains, and Krysta M. Svore. A logarithmic-depth quantum carry-lookahead adder. June 2004. arXiv: quant-ph/0406142.

[DP10] D. Dong and I.R. Petersen. Quantum control theory and applications: a survey. *IET Control Theory & Applications*, 4(12):2651–2671, December 2010.

[DSP+16] Subo Dong, B. J. Shappee, J. L. Prieto, S. W. Jha, K. Z. Stanek, T. W.-S. Holoien, C. S. Kochanek, T. A. Thompson, N. Morrell, I. B. Thompson, U. Basu, J. F. Beacom, D. Bersier, J. Brimacombe, J. S. Brown, F. Bufano, Ping Chen, E. Conseil, A. B. Danilet, E. Falco, D. Grupe, S. Kiyota, G. Masi, B. Nicholls, F. Olivares E., G. Pignata, G. Pojmanski, G. V. Simonian, D. M. Szczygiel, and P. R. Woźniak. ASASSN-15lh: A highly super-luminous supernova. *Science*, 351(6270):257–260, 2016.

[DSvW21] Léo Ducas, Marc Stevens, and Wessel van Woerden. Advanced lattice sieving on GPUs, with tensor cores. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 249–279, Cham, 2021. Springer International Publishing.

[Fey85] Richard P. Feynman. Quantum mechanical computers. *Optics News*, 11(2):11–20, Feb 1985.

[FMMC12] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86:032324, Sep 2012.

[Fow12] Austin G. Fowler. Proof of finite surface code threshold for matching. *Phys. Rev. Lett.*, 109:180502, Nov 2012.

[FP21] Tako Boris Fouotsa and Christophe Petit. SimS: A simplification of SiGamal. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography*, pages 277–295, Cham, 2021. Springer International Publishing.

[GE21] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, April 2021.

[GF19a] Craig Gidney and Austin G. Fowler. Efficient magic state factories with a catalyzed |CCZ> to 2|T> transformation. *Quantum*, 3:135, April 2019.

[GF19b] Craig Gidney and Austin G. Fowler. Flexible layout of surface code computations using AutoCCZ states, 2019. `https://arxiv.org/abs/1905.08916`.

[Gid18] Craig Gidney. Halving the cost of quantum addition. *Quantum*, 2:74, June 2018.

[Gid19a] Craig Gidney. Asymptotically efficient quantum Karatsuba multiplication, 2019. `https://arxiv.org/abs/1904.07356`.

[Gid19b] Craig Gidney. Spooky pebble games and irreversible uncomputation, 2019. `https://algassert.com/post/1905`.

[Gid19c] Craig Gidney. Windowed quantum arithmetic, 2019. `arXiv:1905.07682`.

[Gid23] Craig Gidney. Inplace access to the surface code Y basis, 2023. `https://arxiv.org/abs/2302.07395`.

[Gio17] R. Gioiosa. Resilience for extreme scale computing. In *Rugged Embedded Systems*, pages 123–148. Elsevier, 2017.

[GLM08a] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Architectures for a quantum random access memory. *Phys. Rev. A*, 78:052310, Nov 2008.

[GLM08b] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, Apr 2008.

[GN96] Robert B. Griffiths and Chi-Sheng Niu. Semiclassical fourier transform for quantum computation. *Phys. Rev. Lett.*, 76:3228–3231, Apr 1996.

[Goo22] Google. System architecture, 2022. `https://cloud.google.com/tpu/docs/system-architecture-tpu-vm`.

[Goo23] Google Quantum AI. Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614(7949):676–681, February 2023.

[Gos17] Jeremi M. Gosney. 8x Nvidia GTX 1080 Hashcat benchmarks, 2017. `https://gist.github.com/epixoip/ace60d09981be09544fdd35005051505`.

[Got98] Daniel Gottesman. The Heisenberg representation of quantum computers. 1998. `https://arxiv.org/abs/quant-ph/9807006`.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC '96*. ACM, 1996.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[hBO] (https://stats.stackexchange.com/users/173082/ben) Ben O'Neill. Distribution of urns for non-uniform distribution. Cross Validated (https://stats.stackexchange.com/q/463916). (version: 2020-05-06).

[HC21] Stephen Holmes and Liqun Chen. Assessment of quantum threat to bitcoin and derived cryptocurrencies. Cryptology ePrint Archive, Paper 2021/967, 2021. https://eprint.iacr.org/2021/967.

[HFDM12] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, December 2012.

[HJN+20] Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler, and Mathias Soeken. Improved quantum circuits for elliptic curve discrete logarithms. In *Post-Quantum Cryptography*, pages 425–444. Springer International Publishing, 2020.

[HLGJ21] Connor T. Hann, Gideon Lee, S.M. Girvin, and Liang Jiang. Resilience of quantum random access memory to generic noise. *PRX Quantum*, 2(2), April 2021.

[HRS17] Thomas Häner, Martin Roetteler, and Krysta M. Svore. Factoring using 2n + 2 qubits with Toffoli based modular multiplication. *Quantum Info. Comput.*, 17(7–8):673–684, jun 2017.

[IBM23] IBM Quantum. Compute resources, 2023. Available at https://quantum-computing.ibm.com/services/resources?services=systems, retrieved 31 March 2023.

[JAC+17] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular isogeny key encapsulation. *Submission to NIST post-quantum project*, November 2017. Available at https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

[Jaq19] Samuel Jaques. Quantum cost models for cryptanalysis of isogenies. Master's thesis, University of Waterloo, 2019.

[JH22] Samuel Jaques and Thomas Häner. Leveraging state sparsity for more efficient quantum simulations. *ACM Transactions on Quantum Computing*, 3(3), jun 2022.

[JNRV20] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing grover oracles for quantum key search on AES and LowMC. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, volume 12106 of *Lecture Notes in Computer Science*, pages 280–310. Springer, 2020.

[JNRV23] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing grover oracles for quantum key search on AES and LowMC. In preparation; to appear as a revision to

the Cryptology ePrint Archive Paper 2019/1146, 2023. https://eprint.iacr.org/2019/1146.

[Joh98] Thomas R. Johnson. *United States Cryptologic History*, volume 5. Center for Cryptologic History, National Security Agency, 1998.

[Jor17] Stephen P. Jordan. Fast quantum computation at arbitrarily low energy. *Physical Review A*, 95(3), March 2017.

[JPC+19] N. Jiang, Y.-F. Pu, W. Chang, C. Li, S. Zhang, and L.-M. Duan. Experimental realization of 105-qubit random access quantum memory. *npj Quantum Information*, 5(1), April 2019.

[JR23] Samuel Jaques and Arthur G. Rattew. QRAM: A survey and critique, 2023. https://arxiv.org/abs/2305.10310.

[JS19] Samuel Jaques and John M. Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019*, volume 11692 of *Lecture Notes in Computer Science*, pages 32–61. Springer, 2019.

[JS21] Samuel Jaques and André Schrottenloher. Low-gate quantum golden collision finding. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography*, pages 329–359, Cham, 2021. Springer International Publishing.

[Kit96] Alexei Kitaev. Quantum measurements and the abelian stabilizer problem. *Electronic Colloquium on Computational Complexity*, 3, 1996.

[KL20] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. Chapman & Hall/CRC Cryptography and Network Security Series. CRC Press, Boca Raton, FL, 3 edition, December 2020.

[KMPW21] Péter Kutas, Simon-Philipp Merz, Christophe Petit, and Charlotte Weitkämper. One-way functions and malleability oracles: Hidden shift attacks on isogeny-based protocols. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 242–271, Cham, 2021. Springer International Publishing.

[Kni92] Emanuel Knill. An analysis of Bennett's pebble game, 1992. https://arxiv.org/abs/math/9508218.

[KSS21] Niels Kornerup, Jonathan Sadun, and David Soloveichik. The spooky pebble game, 2021. https://arxiv.org/abs/2110.08973.

[KSV02] A. Yu. Kitaev, A. Shen, and M. N. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, USA, 2002.

[Kup13] G. Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In *TQC 2013*, LIPIcs 22, pages 20–34, 2013.

[Lan89] Rolf Landauer. Dissipation in computation. *Nature*, 340(6236):681–681, August 1989.

[LKS18] Guang Hao Low, Vadym Kliuchnikov, and Luke Schaeffer. Trading T-gates for dirty qubits in state preparation and unitary synthesis, 2018.

[LT16] Seth Lloyd and Barbara M Terhal. Adiabatic and hamiltonian computing on a 2d lattice with simple two-qubit interactions. *New Journal of Physics*, 18(2):023042, February 2016.

[McK06] Craig McKay. German teleprinter traffic and Swedish wartime intelligence. In *Colossus*. Oxford University Press, February 2006.

[ML98] Norman Margolus and Lev B. Levitin. The maximum speed of dynamical evolution. *Physica D: Nonlinear Phenomena*, 120(1-2):188–195, September 1998.

[MMP+23] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. A direct key recovery attack on SIDH, 2023.

[MNRS11] F. Magniez, A. Nayak, J. Roland, and M. Santha. Search via quantum walk. *SIAM Journal on Computing*, 40:142–164, 2011.

[Mø97] Klaus Mølmer. Optical coherence: A convenient fiction. *Phys. Rev. A*, 55:3195–3203, Apr 1997.

[Moh10] Vidyabhushan Mohan. *Modelling the physical characteristics of NAND flash memory*. PhD thesis, School of Engineering and Applied Science, University of Virginia, 2010.

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

[MP21] Michele Mosca and Marco Piani. Quantum threat timeline report 2020. Technical report, Global Risk Institute, 2021.

[MTT20] Tomoki Moriya, Katsuyuki Takashima, and Tsuyoshi Takagi. Group key exchange from CSIDH and its application to trusted setup in supersingular isogeny cryptosystems. In Zhe Liu and Moti Yung, editors, *Information Security and Cryptology*, pages 86–98, Cham, 2020. Springer International Publishing.

[Nat09] National Center for Atmospheric Research Computational and Information Systems Lab. CRAY 1-a: 1977-1989, 2009. Avarilable at `https://web.archive.org/web/20150607021044/http://www.cisl.ucar.edu/computers/gallery/cray/cray1.jsp`. Inflation adjustment from `https://www.usinflationcalculator.com/`.

[NC00] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, UK, 2000.

[Neu98] Karl-Dietrich Neubert. The Flashsort1 algorithm. 1998.

[NPS20] María Naya-Plasencia and André Schrottenloher. Optimal merging in quantum k-xor and k-sum algorithms. In *Advances in Cryptology – EUROCRYPT 2020*, pages 311–340. Springer International Publishing, 2020.

[PDV21] Carlos A Perez-Delgado and Sai Vinjanampathy. Coherent parallelization of universal classical computation. *New Journal of Physics*, 23(12):123015, December 2021.

[Pei20] Chris Peikert. He gives c-sieves on the CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 463–492. Springer, 2020.

[PH78] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms overgf(p)and its cryptographic significance (corresp.). *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[PK21] Pavel Panteleev and Gleb Kalachev. Degenerate quantum LDPC codes with good finite length performance. *Quantum*, 5:585, November 2021.

[PK22] Pavel Panteleev and Gleb Kalachev. Asymptotically good quantum and locally testable classical LDPC codes. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 375–388, New York, NY, USA, 2022. Association for Computing Machinery.

[PWLK22] Dedy Septono Catur Putranto, Rini Wisnu Wardhani, Harashta Tatimma Larasati, and Howon Kim. Another concrete quantum cryptanalysis of binary elliptic curves. Cryptology ePrint Archive, Paper 2022/501, 2022. `https://eprint.iacr.org/2022/501`.

[PY15] Fernando Pastawski and Beni Yoshida. Fault-tolerant logical gates in quantum error-correcting codes. *Phys. Rev. A*, 91:012305, Jan 2015.

[PZ03] John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. January 2003. `http://arxiv.org/abs/quant-ph/0301141`.

[RC18] Rich Rines and Isaac Chuang. High Performance Quantum Modular Multipliers. January 2018. `http://arxiv.org/abs/1801.01081`.

[RD82] Dorothy Elizabeth Robling Denning. *Cryptography and Data Security*. Addison-Wesley Longman Publishing Co., Inc., USA, 1982.

[RDK+22] P. Roberge, B. Daly, N. Kettle, M. Barker, J. Gage, and L. Francis. Hyperledger Ursa code review. Technical report, Digital Identity Laboratory of Canada, 2022. `https://www.hyperledger.org/wp-content/uploads/2022/05/URSA-IDLab-Code-Review.pdf`.

[RNSL17] Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin E. Lauter. Quantum resource estimates for computing elliptic curve discrete logarithms. In *ASIACRYPT 2017*, volume 10625 of *Lecture Notes in Computer Science*, pages 241–270. Springer, 2017.

[Rob23] Damien Robert. Breaking SIDH in polynomial time, 2023.

[RS67] A. Rényi and G. Szekeres. On the height of trees. *Journal of the Australian Mathematical Society*, 7(4):497–507, November 1967.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, feb 1978.

[Sch85] René Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of Computation*, 44(170):483–494, 1985.

[Sch20] John M. Schanck. *Improving post-quantum cryptography through cryptanalysis*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 2020. `https://jmschanck.info/papers/20200703-phd-thesis.pdf`.

[Sch21] John M. Schanck. via Twitter, Jun 2021. `https://twitter.com/susurrusus/status/1532131390338215937`.

[SCS+22] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 36479–36494. Curran Associates, Inc., 2022.

[SDS+01] Erich Strohmaier, Jack Dongarra, Horst Simon, Martin Meuer, and Hans Meuer. Top 500, june 2001, 2001. `https://www.top500.org/lists/top500/2001/06/`.

[SDS+09] Erich Strohmaier, Jack Dongarra, Horst Simon, Martin Meuer, and Hans Meuer. Top 500, june 2009, 2009. `https://www.top500.org/lists/top500/2009/06/`.

[SG10] M. Schneider and N. Gama. SVP challenge, 2010. `https://www.latticechallenge.org/svp-challenge/index.php`.

[SGT+18] Krysta M. Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher E. Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#: Enabling scalable quantum computing and development with a high-level DSL. In *RWDSL@CGO 2018*, 2018.

[Sha49] Claude. E. Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28(1):59–98, 1949.

[Sho94] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[Smi18] Benjamin Smith. Pre- and post-quantum Diffie–Hellman from groups, actions, and isogenies. In *Arithmetic of Finite Fields*, pages 3–40. Springer International Publishing, 2018.

[SNN+20] L. J. Stephenson, D. P. Nadlinger, B. C. Nichol, S. An, P. Drmota, T. G. Ballance, K. Thirumalai, J. F. Goodwin, D. M. Lucas, and C. J. Ballance. High-rate, high-fidelity entanglement of qubits across an elementary quantum network. *Phys. Rev. Lett.*, 124:110501, Mar 2020.

[SPW11] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. DRAM errors in the wild: A large-scale field study. *Commun. ACM*, 54(2):100–107, feb 2011.

[Ste16] Damien Steiger. Racing in parallel: Quantum versus classical, 2016. `https://pirsa.org/16080019`.

[Tan07] Seiichiro Tani. An improved claw finding algorithm using quantum walk. In *Mathematical Foundations of Computer Science 2007*, pages 536–547. Springer Berlin Heidelberg, 2007.

[Tat17] Stephen R. Tate. The advanced encryption standard, 2017. `https://home.uncg.edu/cmp/faculty/srtate/580.s17/aeshandout.php`.

[Tez04] Tezzaron Semiconductor, January 2004. `https://tezzaron.com/media/soft_errors_1_1_secure.pdf`.

[U.S01] U.S. Department of Commerce/National Institute of Standards and Technology. Advanced Encryption Standard (AES). FIPS-197, 2001. `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf`.

[vOW99] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.

[VP17] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in WPA2. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1313–1328, New York, NY, USA, 2017. Association for Computing Machinery.

[WLQ+21] Pengfei Wang, Chun-Yang Luan, Mu Qiao, Mark Um, Junhua Zhang, Ye Wang, Xiao Yuan, Mile Gu, Jingning Zhang, and Kihwan Kim. Single ion qubit with estimated coherence time exceeding one hour. *Nature Communications*, 12(1), January 2021.

[WV21] David Warburton and Sander Vinberg. The 2021 TLS telemetry report, 2021. `https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report`.

[WWL22] Ze-Guo Wang, Shi-Jie Wei, and Gui-Lu Long. A quantum circuit design of AES requiring fewer quantum qubits and gate operations. *Frontiers of Physics*, 17(4), February 2022.

[YZ22] Pei Yuan and Shengyu Zhang. Optimal QRAM and improved unitary synthesis by quantum circuits with any number of ancillary qubits, 2022.

[Yzm07] Yzmo. Helium atom qm.svg. Wikimedia Commons, 2007. Licensed under a Creative Commons Attribution-Share Alike 3.0 Unported license. `https://commons.wikimedia.org/wiki/File:Helium_atom_QM.svg`.

[Zal99] Christof Zalka. Grover's quantum searching algorithm is optimal. *Phys. Rev. A*, 60:2746–2751, Oct 1999.

[ZDM+19] Darko Zivanovic, Pouya Esmaili Dokht, Sergi Moré, Javier Bartolome, Paul M. Carpenter, Petar Radojković, and Eduard Ayguadé. DRAM errors in the field: A statistical approach. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '19, page 69–84, New York, NY, USA, 2019. Association for Computing Machinery.