Improving post-quantum cryptography through cryptanalysis

by

John M. Schanck

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Mathematics

Waterloo, Ontario, Canada, 2020

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:
Chris Peikert, Associate Professor,
Department of Computer Science and Engineering,
University of Michigan

Supervisor:
Michele Mosca, Professor,
Department of Combinatorics and Optimization,
University of Waterloo

Internal Member:
Alfred Menezes, Professor,
Department of Combinatorics and Optimization,
University of Waterloo

Internal-External Member:
Richard Cleve, Professor,
School of Computer Science,
University of Waterloo

Other Member(s):
Douglas Stebila, Associate Professor,
Department of Combinatorics and Optimization,
University of Waterloo

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of contributions

The introduction, chapter prefaces, Chapter 4, and the conclusion were authored solely by myself. Chapter 2 was jointly authored with Samuel Jaques. Chapter 3 was jointly authored with Martin Albrecht, Eamonn Postlethwaite, and Vlad Gheorghiu. More detailed statements of contributions precede Chapters 2 and 3.

Abstract

Large quantum computers pose a threat to our public-key cryptographic infrastructure. The possible responses are:

1. Do nothing; accept the fact that quantum computers might be used to break widely deployed protocols.
2. Mitigate the threat by switching entirely to symmetric-key protocols.
3. Mitigate the threat by switching to different public-key protocols.

Each user of public-key cryptography will make one of these choices, and we should not expect consensus. Some users will do nothing—perhaps because they view the threat as being too remote. And some users will find that they never needed public-key cryptography in the first place.

The work that I present here is for people who need public-key cryptography and want to switch to new protocols. Each of the three articles raises the security estimate of a cryptosystem by showing that some attack is less effective than was previously believed. Each article thereby reduces the cost of using a protocol by letting the user choose smaller (or more efficient) parameters at a fixed level of security.

In Part 1, I present joint work with Samuel Jaques in which we revise security estimates for the Supersingular Isogeny Key Exchange (SIKE) protocol. We show that known quantum claw-finding algorithms do not outperform classical claw-finding algorithms. This allows us to recommend 434-bit primes for use in SIKE at the same security level that 503-bit primes had previously been recommended.

In Part 2, I present joint work with Martin Albrecht, Vlad Gheorghiu, and Eamonn Postelthwaite that examines the impact of quantum search on sieving algorithms for the shortest vector problem. Cryptographers commonly assume that the cost of solving the shortest vector problem in dimension d is $2^{(0.265\ldots+o(1))d}$ quantumly and $2^{(0.292\ldots+o(1))d}$ classically. These are upper bounds based on a near neighbor search algorithm due to Becker–Ducas–Gama–Laarhoven. Naively, one might think that d must be at least $483 (\approx 128/0.265)$ to avoid attacks that cost fewer than 2^{128} operations. Our analysis accounts for terms in the $o(1)$ that were previously ignored. In a realistic model of quantum computation, we find that applying the Becker–Ducas–Gama–Laarhoven algorithm in dimension $d > 376$ will cost more than 2^{128} operations. We also find reason to believe that the classical algorithm will outperform the quantum algorithm in dimensions $d < 288$.

In Part 3, I present solo work on a variant of post-quantum RSA. The original pqRSA proposal by Bernstein–Heninger–Lou–Valenta uses terabyte keys of the form $n = p_1 p_2 p_3 p_4 \cdots p_i \cdots p_{2^{31}}$ where each p_i is a 4096-bit prime. My variant uses terabyte keys of the form $n = p_1^2 p_2^3 p_3^5 p_4^7 \cdots p_i^{\pi_i} \cdots p_{20044}^{225287}$ where each p_i is a 4096-bit prime and π_i is the i-th prime. Prime generation is the most expensive part of post-quantum RSA in practice, so the smaller number of prime factors in my proposal gives a large speedup in key generation. The repeated factors help an attacker identify an element of small order, and thereby allow the attacker to use a small-order variant of Shor's algorithm. I analyze small-order attacks and discuss the cost of the classical pre-computation that they require.

Acknowledgements

Thanks to Cary, my love, for her friendship, kindness, and strength; for sharing these years with me; for demonstrating honesty and courage, an unpretentious intellect, and the power of community. I am humbled by her love.

Thanks to my family. My mother, Nancy, for her encouragement, love, and enthusiasm. My father, John, for his good humor—and for at least one memorable lesson in mathematics. Thanks to Julie, David, Cady, Matt, Max, Reid, Rory, Sarah, Axton, Grace, Ian, Mary Catherine, and Laura for filling our holidays with good meals and good company.

Thanks to everyone who has mentored me over the course of my formal education, but especially: Steve Isaacs of Liberty Corner Computing; Bill Dewees and David Zimmerman of Conestoga High School; Lee Spector, Herb Bernstein, and Stephen Banzaert of Hampshire College; Scott Kaplan and Catherine McGeoch of Amherst College; William Whyte and Jeffrey Hoffstein of NTRU Cryptosystems; and Ian Goulden of the University of Waterloo. You have each given me precious gifts. I can only hope that I am able to pass these gifts forward to future generations of learners. If I do, it will be with you in mind as role models.

Thanks to everyone who has welcomed me into the crypto community. Thanks to Alfred Menezes and Douglas Stebila for many enjoyable conversations. Thanks to Michael Naehrig for career advice. Thanks to Peter Schwabe for hosting me in Nijmegen, Andreas Hülsing for hosting me in Eindhoven, and J. F. Biasse for hosting me in Tampa. Thanks to the organizers and attendees of the Dagstuhl seminar series on quantum cryptanalysis and the AIM workshop on quantum algorithms for analysis of public-key crypto.

Thanks to everyone who made this thesis possible. Thanks to Michele Mosca for advising me and supporting my research. Thanks to my co-authors: Martin Albrecht, Vlad Gheorghiu, Sam Jaques, and Eamonn Postlethwaite. Thanks to the anonymous conference reviewers (even #2). Thanks to Emma McKay, Travis Morrison, Sam Jaques, and Cary for comments on drafts. Thanks to the staff and volunteers of the Queen Street Commons cafe and the Kitchener Public Library—two true "palaces for the people" where many pages of this thesis were written. Thanks to the IQC staff, especially Chin Lee. Thanks to Dave Mackey of UW Counselling Services.

Thanks to everyone on May Place and nearby: Emma, Tamara, Gage and Evie, Katie, Rayne, Patt, and Barb. Thanks to everyone who came to one of our May Day parties. Thanks to "Casper the friendly dog" for trying so hard to live up to his name. Thanks to Vic for inviting us to a birthday party, Todd whose birthday it was, and Gordon whose house the party was at. Special thanks to Dan, Michelle, Violet, and Alex for treating us like family. Thanks to everyone who made Kitchener our home. ♡

Dedication

For rock and roll legend

Jimmy Schanck

Table of Contents

Chapter 1

Introduction

Thermodynamics arose out of attempts to understand the limits on the efficiency of steam engines. Information theory arose out of the effort to understand channel capacity limitations. These models have motivated an attempt to do the same thing for the computer.

Rolf Landauer (1981)

1 Information is physical, but computation is technological

Any algorithmic cryptosystem can be broken with the help of a sufficiently powerful computer. Part of our role as cryptographers is, therefore, to make predictions about which computers can be built and which computations can be performed. To do this we make physical, technological, and economic assumptions.

We should periodically re-evaluate these assumptions—especially when there are dramatic changes in our understanding of the physics or technology of computation.

To a crude approximation, which ignores the significant social component of our practice, the process by which we come to believe that an algorithmic cryptosystem is secure is:

1. we fix a machine model;
2. we determine the smallest machine (in that model) that breaks the system in a reasonable amount of time; and then
3. we argue that even this smallest machine cannot be built.

Our physical assumptions are inherent in our choice of machine model. Our technological assumptions arise from our need to assign costs, with physical units, to the components of our machine model. And our economic assumptions constrain the resources that our imagined adversaries might obtain.

This thesis presents several detailed cost estimates for attacks on public-key cryptosystems. These cost estimates track quantities that have been ignored previously, quantities that are small asymptotically, and quantities that (one could say) are tedious to keep track of. I have engaged in this work because I think that practical cryptography is important, and I think that these particular analyses are impactful. Naturally, I have searched for tools that might make my life easier—because no matter how important I think cryptography is, I do not want to do tedious work.

The tools that I have found are are not standard. To use them means to revise our machine models; to go back to Step 1 and speak frankly with each other about whether the assumptions we have made are good ones. The work below proposes some small revisions to quantum machine models. I hope to convince you that these revisions are reasonable.

1.1 Physical assumptions

If cryptographers do not agree on a machine model, then cryptography fractures into multiple competing theories. The field is currently divided into at least two theories; the notable point of rupture being Deutsch's 1985 introduction of the quantum-mechanical description of the *state* of a computation.

Cryptographers working in the "pre-quantum" theory judge the efficiency of a computation by the number of steps that it takes on a Turing machine. They are thereby working (perhaps implicitly) with the physical assumptions that were laid out by Turing in 1937 [35]. Cryptographers working in the "post-quantum" theory also count Turing machine steps, but the tapes of their Turing machines are quantum mechanical and their physical assumptions are (perhaps implicitly) those of Deutsch [15].

Fredkin and Toffoli list Turing's physical assumptions as **locality**: "the speed of propagation of information is bounded"; **finiteness**: "the amount of information which can be encoded in the state of a finite system is bounded"; and **reliability**: "it is possible to construct [physical devices] which perform in a recognizable and reliable way the logical functions AND, NOT, and FAN-OUT" [20].

Deutsch assumes locality and finiteness in precisely the same sense. He extends the notion of reliability to include certain quantum mechanical processes.

There are conflicts within the pre-quantum and post-quantum theories, and I will return to these later. What is more interesting, however, is how similar they are. Why have these two theories, these two sets of physical assumptions, risen to prominence and not others?

Post-reversible cryptography? Papers by Bennett [10], Fredkin–Toffoli [20], Benioff [7, 8], and Feynman [18] provided successively more realistic models of dissipationless computation. The later models suggest physically reversible processes that circumvent Landauer's bound [24] on the energetic cost of computation. In these models there is no sense in which the energy consumed by a computation (or even the time it takes[1]) must grow with the number of logical steps that it performs. Why did we not see cryptography cleave into "pre-reversible" and "post-reversible" theories?

There could have been lively debates between pre-reversible and post-reversible cryptographers. A notable exchange in the pages of Physical Review Letters shows us how

[1] As early as 1982 Deutsch wrote that "the laws of physics impose no fundamental bound on the rate at which information can be processed" [14]. This holds up today—the widely cited Margolus–Levitin theorem gives a limit on how quickly a quantum system can transition between orthogonal states [26], but Jordan has shown that this does not impose a limit on the rate of computation itself [23].

this might have gone. Porod–Grondin–Ferry–Porod [30] took the position of our imagined pre-reversible camp. They argued that protecting a computational system against thermal noise would require dissipation proportional to operation count. They were quickly rebuked.

- Toffoli: “the second principle of thermodynamics *per se* does not impose a lower bound [on dissipation per operation] higher than zero, and no bounds higher than zero have been compellingly suggested by anyone on fundamental physical grounds” [33].
- Benioff: “the idealized existence of [Hamiltonian models of Turing machines] demonstrates that one cannot use purely quantum mechanical arguments [...] to deny the existence of models which dissipate no energy” [9].

The debate continued. Porod *et al.* insisted that noise was inherent and not, as they accused Toffoli and Benioff of assuming, “a technological detail” [31]. They had a point. Shortly thereafter Zurek wrote a remarkable paper demonstrating the devastating effect of noise on the Fredkin–Toffoli model [38]. But in that same paper Zurek writes: “Reversible computation is compatible with the laws of physics. The quantum spin computers of Benioff and Feynman prove that it can be arbitrarily fast and dissipationless” [38].

The physical possibility of dissipationless and arbitrarily fast error-corrected quantum mechanical computers remains open.

The role of physical assumptions We might say that cryptographers are taking sides in the physicists’ debate by choosing not to develop a post-reversible theory. But an equally satisfying explanation is that cryptographers are comfortable dismissing reversible computation on practical, technological, grounds. They would not be alone. Even Toffoli admits that “no one that I know believes in practical, physical computation that is exactly dissipationless” [33], and Jordan admits that it “seems unlikely” that the universe allows for arbitrarily fast computation [23].

Cryptographers are not disciplinarily bound to use the most powerful machine model that physics allows. They are free to pick and choose from physical constraints and to supplement their reasoning with stronger assumptions. If it seems strange that a post-quantum theory has risen to prominence while a post-reversible theory has not, the reason is likely that the post-quantum theory is supported by (at least the prospect of) realistic computer technology.

1.2 Economic assumptions

As intellectually satisfying as it would be to reason from physical assumptions alone, cryptographers cannot yet do so. If cryptographers want to work on key encapsulation, digital signatures, pseudorandom generators, or one-way functions—and claim that their protocols have some relevance to communications security—then they will need economic and technological assumptions.

Their economic assumptions could be mild. Freeman Dyson famously speculated about the resources that might be available to technologically advanced civilizations [17]. He took our own solar system as a model: "The quantities of matter and energy which might conceivably become accessible to us within the solar system are [2^{100} grams] (the mass of Jupiter) and [2^{88} joules per second] (the total energy output of the Sun)." He suggested that Jupiter, re-arranged into a two-meter thick shell revolving about the sun at twice Earth's distance, could be used to harvest the solar radiation. Of course, building this device would be difficult: "the energy required to disassemble and rearrange a planet of the size of Jupiter is about [2^{123} joules], equal to the energy radiated by the sun in 800 years" [17].

The mass of the planets and the energy output of the sun are physical facts that are known to good precision. A cryptographer might put a mild economic assumption—that the solar resources are the only resources available to an attacker—to good use. E.g., for at least the next 800 years, there will be no 2^{100} gram memories and no 2^{88} watt attack machines. A less conservative cryptographer might base their predictions on the mass of the moon (2^{86} g) and the solar energy that strikes Earth's surface (2^{56} W [34]), or even smaller quantities.

Economic assumptions are not sufficient, though. Cryptographers need technological assumptions to translate "bits of memory" to grams and "operations" to energy, and this is messy business. The past century of technological progress has been exceptional in a way that presumably prohibits extrapolation. If we are going to make mistakes in setting assumptions, they will most likely be here.

We should learn from past failures, pay close attention to technological prospects, and take advantage of the few hard physical constraints of which we are aware.

1.3 Algorithmic assumptions

Cryptographers are careful to adorn their security claims with the phrase "against known algorithms." They rarely expand this to "against known technologies."

Algorithmic assumptions play an privileged role in theory: they help to establish the *existence* of problem instances that are easy to generate but hard to solve. From the perspective of practice, however, an algorithm is a technology and an algorithmic assumption is a technological assumption. While it is true that cryptosystems sometimes fall to stunning algorithmic advances, cryptosystems more often fall to a combination of new technologies. Let's consider some examples.

Claim In 1977, Rivest, Shamir, and Adleman suggested that the use of 663-bit composite numbers would provide "a margin of safety [for factoring based cryptosystems] against future developments" [32]. They assumed that factoring an integer n costs $E_1(n)$ operations where

$$E_1(n) = \exp\left(\sqrt{\ln n \ln\ln n}\right). \tag{1}$$

They assumed that one operation takes one microsecond. They concluded that factoring a 663-bit integer would take 3.8 billion years.

Attack In 2005, a 663-bit RSA modulus was factored by Bahr, Boehm, Franke, and Kleinjung [5]. They used the number field sieve, an algorithm whose cost is exponential in $(\ln n)^{1/3+o(1)}$.

Claim In 1978, McEliece suggested that a cryptosystem based on the difficulty of decoding certain binary linear codes of length 1024 and dimension 524 to distance 50 would be "quite secure" [27]. He estimated one attack as having "a work factor of [...] about $10^{19} \approx 2^{65}$".

Attack In 2008, Bernstein, Lange, and Peters claimed an improved attack with a cost of $2^{60.1}$ bit operations. They demonstrated their attack by performing the computation [12].

At first glance, these failures[2] are of very different kinds. McEliece underestimated the extent to which technology would improve. Rivest, Shamir, and Adleman were caught off guard by an extraordinary, superpolynomial, algorithmic advance.

[2] It is reasonable to ask whether these were failures at all. Putting aside advances in algorithms and other technologies, there would be no great mystery if a cryptosystem was called "quite secure" in 1978 and "hopelessly insecure" in 2020. The role of cryptography has changed, as have our expectations. Nevertheless, it can be instructive to apply the standards of today to the recommendations of the past. By today's standards, these were poor recommendations.

It is true that the number field sieve reduced the cost of factoring 663-bit numbers. But Rivest, Shamir, and Adleman also underestimated the extent to which technology would improve. To see how, we can apply their technological assumptions and reasoning to the cost function for the number field sieve suggested by NIST in 2019 [6, Appendix D],

$$E_2(n) = \exp\left(1.923(\ln n)^{1/3}(\ln\ln n)^{2/3} - 4.69\right). \tag{2}$$

Doing so, we estimate the time to factor a 663-bit number as 1.1 million years, a factor of $E_1(663)/E_2(663) \approx 2^{11.6}$ less than the 3.8 billion years estimated by Rivest, Shamir, and Adleman. The 663-bit factorization reported by Bahr–Boehm–Franke–Kleinjung took less than 100 years of CPU time. This is still $2^{13.4}$ times less than our revisionist estimate, and the difference is largely due to Rivest, Shamir, and Adleman's assumption of a 1 MHz clock rate. In other words, the failure of RSA-663 was as much a failure to predict technological advances as it was a failure to predict algorithmic advances.

Similarly, while technological advances certainly enabled the Bernstein–Lange–Peters attack, the failure of the original McEliece parameters was in a large part due to algorithmic advances as well. McEliece arrived at his "work factor of [...] 2^{65}" estimate by what we might now call a *conservative analysis*. In 1987, Adams and Meijer put the work factor of the same attack at $2^{80.7}$ [2]. In 1988, Lee and Brickell claimed an improved attack with a work factor of $2^{69.6}$ [25]. In 1998, Canteaut and Sendrier claimed a "certainly still infeasible" attack with a cost of $2^{64.2}$ bit operations [13]. And in 2008, Bernstein, Lange, and Peters improved this to $2^{60.1}$ bit operations. There was a million fold reduction from the Adams–Meijer estimate to the Bernstein–Lange–Peters estimate. All of it was due to algorithmic techniques that only yield polynomial factor improvements asymptotically.

It may well be that the algorithmic techniques that broke these systems could not have been predicted. The technological advances, on the other hand, were predictable. In fact, they were predicted.

1.4 Technological assumptions

When Adams and Meijer revisited McEliece's analysis in 1987, they concluded that the parameter set "appears to be fairly secure." They justified this by saying that the cost of the attack ($2^{80.7}$ bit operations) was "significantly higher" than the cost of an attack on the US Data Encryption Standard, DES. This comparison with DES is revealing. In 1977, Diffie and Hellman [16] had argued that the 56-bit DES cipher offered an inadequate level of security. With extraordinary attention to existing computer technology, they argued that for \$20 000 000 one could build a machine that would break DES in 12 hours. They projected a decrease in cost to \$200 000 by 1987.

Not all of Diffie and Hellman's assumptions have held up. Notably, they were writing before physically realistic models of reversible computing had been constructed, and they were of the opinion that "quantum mechanical and thermodynamic considerations rule out exhaustive searches on keys of several hundred bits" [16]. Yet their estimates were sound, as evidenced by public demonstrations of attacks on DES in the mid-90s. Their ultimate recommendation that "a 128-bit or larger key is needed to preclude exhaustive search" holds up even today.

Contemporary cryptosystems continue to use complex economic and technological assumptions to justify security at the 128-bit level. Let's take the widely deployed X25519 key exchange protocol as an example.

X25519 can be broken, using Pollard's rho algorithm, at a cost of about 2^{125} elliptic curve point additions [11]. In his 2006 security analysis of the scheme, Bernstein estimates technological facts like "a modern CPU costs about 2^6 dollars" and "one can fit [about 2^{10} parallel rho] circuits into the same amount of circuitry as a modern CPU." He calculates that a billion-dollar machine operating for a year could compute 2^{80} elliptic curve point additions. While this would suffice to compute a 160-bit elliptic curve discrete logarithm, Bernstein notes that it would only have a 2^{-90} chance of breaking X25519. He concludes that, while "one must adjust these estimates as chip technology improves," the X25519 "security level will remain comfortable for the foreseeable future" [11].

Bernstein's analysis is part of a long history of cost estimates for the rho method. In 1996, van Oorschot and Wiener presented a parallel variant of the rho method and gave detailed hardware designs for several applications [36]. They reason from a 1993 hardware design for curve arithmetic from Agnew–Mullin–Vanstone [3]. This design performs one point addition every 50 µs on a chip with an area of 1 mm^2. Van Oorschot and Wiener estimate that an attacker with a billion dollar budget could compute 2^{70} point additions per year.[3]

The technological assumptions made by van Oorschot–Wiener are quite modest, presumably because they were challenging a security claim rather than predicting the capabilities of future attackers. The Agnew–Mullin–Vanstone design assumes a 1.5 µm fabrication process

[3] I am extrapolating for easier comparison with Bernstein's estimate. The original estimate is 2^{60} point additions in 32 days with a ten million dollar hardware budget.

(a technology from 1981). Bernstein's estimate follows a hardware design from Franke–Kleinjung–Paar–Pelzl–Priplata–Šimka–Stahlke [19], which assumes a 130 nm fabrication process (a technology from 2001). Moore–Dennard scaling[4] predicts a factor of $2^{10.5}$ gap between the van Oorschot–Wiener and Bernstein estimates. A factor 2^7 of this is due to an increase in the number of transistors per millimeter at constant price and a factor $2^{3.5}$ is due to an increase in clock rate at constant power density.

2 Our technological prospects

We can use a Moore–Dennard model to update Bernstein's estimate for 2020 technology. Dennard-type frequency scaling slowed significantly at 65 nm processes. So I will assume Moore–Dennard scaling from 130 nm to 65 nm and then Moore scaling alone to 7 nm. Scaling the Franke–Kleinjung–Paar–Pelzl–Priplata–Šimka–Stahlke design to 65 nm gives a factor $\approx 2^3$ increase in point additions per second at a fixed hardware and power budget. Scaling further to a modern 7 nm process gives a $2^{6.5}$ factor increase in transistor count at a fixed hardware budget. This crude model suggests that an adversary who could buy a 2^{80} point addition per year device in 2001 could buy a $2^{89.5}$ point addition per year device in 2020. Whether they could power it is a different question—their energy expenditures would be $2^{6.5}$ times greater.

Are $2^{89.5}$ point additions per year plausible? The largest cryptanalytic machine in operation is, arguably, Bitcoin. It is composed of thousands of volunteer-run computers, which are commonly referred to as "mining rigs" or "miners." The overall architecture of the machine is strikingly similar to what a cryptanalyst would design for a parallel collision search: each miner iterates through inputs to SHA-256 looking for pre-images of certain distinguished strings. They report any pre-images that they find to a distributed database. The miners' method of iteration differs from what is done in the van Oorschot–Wiener algorithm, but not in a way that significantly affects circuit size.

As of January 1, 2020 Bitcoin computes approximately 2^{26} terahashes per second.[5] One terahash is roughly 2^{40} invocations of the SHA-256 hash function. One invocation of

[4] Moore-Dennard scaling is the silicon industry trend of exponentially increasing transistor count and clock rate at constant price ($) and constant power (W) per square millimeter. For sufficiently large devices, scaling linear features down by a factor of $\sqrt{2}$ will double transistor count per square millimeter and will reduce energy-per-switching by a factor of $\sqrt{2}$. For small devices this reduction in energy-per-switching breaks down.

[5] https://www.blockchain.com/charts/hash-rate

SHA-256 is roughly 2^{17} logic gates.[6] Ignoring growth in the network, it is reasonable to assume that Bitcoin will perform on the order of 2^{91} SHA-256 evaluations, or 2^{108} logic gates, in 2020.

If we assume that a circuit for SHA-256 is the same size as a circuit for an elliptic curve point addition, and we assume that Bitcoin enthusiasts have made a one billion dollar investment in hardware, then the current size of Bitcoin suggests that $2^{89.5}$, or even 2^{91}, point additions per year are indeed possible.

How many more point additions per year are plausible? In 2020, Bitcoin will use on the order of 77 TW h of electricity,[7] which is about one third of one percent of global electricity consumption. Its energy usage is a significant barrier to further scaling.

Its physical size is also a barrier. Bitcoin generates about 11 thousand tons ($2^{33.4}$ g) of electronic waste per year,[8] much of it old mining hardware that has been made obsolete. Since most of the mining is done by new machines, 11 thousand tons is a good estimate for the weight of a machine as powerful as Bitcoin.

Global electronic waste production is about 50 million tons ($2^{45.5}$ g) per year, and is only expected to grow to twice that by 2050.[9] The types of things we throw away lag a bit behind the types of things that we produce, but it's not so hard to imagine change. We could cut back on televisions, cell phones, and refrigerators and make Bitcoin mining hardware instead.

An 11 *million* ton machine, made of state-of-the art 2020 hardware, would perform on the order of 2^{101} SHA-256 evaluations per year, or 2^{118} logic gates per year. It would consume energy on par with the current global electricity supply. It would have a roughly 2^{-54} chance of finding a SHA-256 collision in the process. A similar amount of hardware would have a similar chance of breaking X25519 in one year.

Building and powering this 11 million ton machine would take a massive, shameful, and nauseating re-alignment of capital. We should not build this device, but we should also not assume that our adversaries are constrained by good morals. Nor should we assume that computing hardware will not improve.

[6] Based on Steven Goldfeder's MPC circuits: http://stevengoldfeder.com/projects/circuits/sha2circuit.html

[7] Based on the Cambridge Bitcoin Electricity Consumption Index, https://www.cbeci.org/. See also: https://digiconomist.net/bitcoin-energy-consumption.

[8] https://digiconomist.net/bitcoin-electronic-waste-monitor/

[9] Based on the 2019 World Economic Forum Global E-Waste Report: http://www3.weforum.org/docs/WEF_A_New_Circular_Vision_for_Electronics.pdf.

Future computer technologies The energy efficiency of mining hardware has improved dramatically. However, the gains have been modest since the first application-specific mining chips were introduced. The BITMAIN BM1382 chip, introduced in 2014 and built with a 28 nm fabrication process, consumes 770 J/THash. The newer BITMAIN BM1397 chip, introduced in 2019 and built with a 7 nm process, consumes 40 J/THash. This significantly outpaces silicon industry projections, and we can infer that most of the improvement is due to application-level design changes.

If one could divert the 2^{56} watts of solar energy that reach Earth's surface to BM1397 chips, with perfect efficiency, one could perform $2^{115.7}$ SHA-256 evaluations per year ($0.025\,\mathrm{THash/J} \times 2^{56}\,\mathrm{J/s} \times 2^{25}\,\mathrm{s}$). More efficient chips are needed to reach 2^{128} SHA-256 evaluations per year.

The IEEE International Roadmap for Devices and Systems' More Moore team projects an 18% generation-to-generation reduction in energy-per-switching for the six generations of fabrication processes scheduled between 2020 and 2034 [22, Section 4.3]. This would lead to an overall 3× improvement in efficiency by 2034. Hence, in the near future, we will have chips that could reach 2^{118} SHA-256 evaluations per year if given all 2^{56} watts of available solar energy.

Would a further thousand fold increase in efficiency render SHA-256 insecure?

No. The physical size of the device would be prohibitive, as would acquiring the energy. A machine weighing 11 *billion* tons and composed of 40 J/PHash mining rigs would only perform 2^{121} SHA-256 evaluations per year. This machine would weigh as much as a thousand years of electronic waste. It would consume an amount of electricity that is on par with global supply. And if its material substance was extracted from the Earth using the same processes that have brought us the BM1397, the human suffering and ecological devastation caused by its construction would be unspeakable. It would still only have a 2^{-14} chance of finding a SHA-256 collision in one year.

Even a machine that is a million times larger than Bitcoin and made out of chips that are a thousand times more efficient than the BM1397 would be unlikely to find a SHA-256 collision. Chips a thousand times more efficient than the BM1397 would still have switching energies that are orders of magnitude above Landauer's bound. Still, I know of no technology that promises a thousand fold improvement.

3 Our habits

The physicist David Mermin writes that “A bad habit is something you do, without being fully aware of it, that makes life harder than it needs to be. It is a bad habit of physicists to take their most successful abstractions to be real properties of our world” [28].

At the beginning of this chapter I mentioned that we, as cryptographers, have some freedom in choosing our physical assumptions. We do not have to discard the possibility of public key cryptography just because there are spacetimes with closed timelike curves (where $\mathsf{P} = \mathsf{PSPACE}$ [1, 4]). We do not even have to form an opinion about whether our spacetime has closed timelike curves. Technological and economic assumptions protect us from the rough edges of established physical theory, and they allow us to construct efficient cryptosystems. Making these assumptions is a good habit.

We have habits that make life difficult as well. We conflate abstract machines with real machines. We have held on, for too long, to a dream of making technology-independent predictions. And we have ignored some of the few hard physical constraints that are known.

As we revise our security analyses to deal with quantum adversaries, we would do well to incorporate new physical constraints into our machine models—even if these constraints, like constraints on reversibility, are conjectural.

We might start by re-examining the following features.

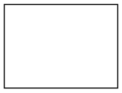
- **Zero-cost quantum storage**. The notion that wires in a quantum circuit incur no cost per unit time.
- **Unit-cost random access memory**. The notion that an arbitrary element of a memory can be queried at a cost that does not grow with the memory size.
- **Unit-cost quantum accessible classical memory (qRAM)**. The notion that an arbitrary superposition of the elements of a memory can be constructed at a cost that does not grow with the memory size.
- **Zero-cost classical computation**. The notion that classical computation can be ignored when costing quantum algorithms.

The papers below raise issues around the use of non-volatile quantum memory (Chapter 2), quantum accessible classical memory (Chapter 3), and free classical computation (Chapter 4). In Chapter , I conclude with several concrete recommendations that I hope will move us closer to more realistic machine models.

Before we move on, let's pause to consider the cost of large classical memories. It will be useful to have a sense for these in our later discussions of quantum memory and qRAM.

3.1 Large classical memories

Shopping for a high-end micro-SD card can give you a sense of the near-term limits of large memories. A micro-SD card is small and light: just 15 mm × 11 mm × 1 mm and 0.5 g. You could fit one comfortably in this rectangle:



But if you want to pack a lot of chips into a small area you will stand them on end, ▭. This orientation has a footprint of just $2^{3.5}$ mm^2.

The five boroughs of New York City lie on about 800 km^2 ($2^{49.5}$ mm^2) of land. A 15 mm thick shell of 2^{46} micro-SD cards would occupy about the same area. This is a large, but not inconceivable, amount of electronics, with a mass on par with the electronic waste produced world-wide in one year.

The high-end micro-SD cards available today hold an impressive terabyte of data (2^{43} bits). A New York City-sized memory of these chips would have a capacity of 2^{89} bits. I doubt that there is the political will to build a memory this large—even at one cent per chip the device would cost about a trillion dollars—but I see no obvious physical or technological barriers.

Large memory computations The idle power draw of a micro-SD card is on the order of milliwatts. The power draw for 2^{46} cards, if we imagine a parallel computation that requires all of the cards to be powered simultaneously, would be on the order of gigawatts.

Sorting is a good example of a memory intensive computation.[10] Wiring 2^{46} micro-SD cards to support a random access algorithm like quicksort would be expensive (cf. Wiener's notion of the "full cost" of a cryptanalytic attacks [37]). But it is worth noting that micro-SD cards already contain microcontrollers for reading and writing their contents. Adding circuitry for communication between neighboring cards would not add much cost. Adding circuitry for each card to perform some simple computation would also not add much cost.

[10] Sorting is also the bottleneck in simulating quantum circuits [21, Section 3.2-3.3].

The cost of sorting 2^{46} one-terabyte integers on a two-dimensional mesh is $\approx 2^{112}$ bit operations (roughly $2^{46} \cdot \sqrt{2^{46}}$ parallel comparison steps each of which is followed by a routing step that moves 2^{43} bits).

Bitcoin's power draw is also on the order of gigawatts and it performs a similar number of bit operations in one year. The bit operations in a large mesh sort, which push data across millimeters of wire, are slower and more energetically expensive than the bit operations in a hash function evaluation. So we can expect that sorting 2^{46} one terabyte integers, with current technology, would take longer than one year. Sorting 2^{89} bits worth of smaller integers would take even longer.

For the sake of comparison, the world record for energy efficient sorting is currently held by the KioxiaSort machine of Sano–Mahmoud–Suzuki.[11] KioxiaSort uses $2^{16.4}$ J to sort one terabyte worth of 110-byte values. The process takes 9 minutes. The energetic cost of sorting 2^{46} *separate* one-terabyte data sets is already 10 TW h. This could be performed in one year by running KioxiaSort 2^{16} times sequentially on 2^{30} parallel machines. The hardware cost would be tens or hundreds of billions of dollars (ignoring the cost of idle storage). The energetic cost and time do not include the cost of merging 2^{46} sorted one terabyte lists, which would be the stage at which data is moved over many *kilometers*.

Future memory technology Putting aside the cost of computing with a large memory, how large of a memory might we make in the future?

The flash memory in a micro-SD card is composed of cells. Each cell stores a small number of bits, say 4, in the threshold voltage state of a single transistor. There are a variety of cell designs, and any given design can be fabricated at a variety of sizes using different fabrication techniques. Older "two-dimensional" cells can be scaled down to about 15 nm on a side using current processes. They cannot be scaled down much further, as statistical fluctuations in the number of stored electrons on the transistor's gate would limit their reliability. The terabyte micro-SD cards on the market today are based on "three-dimensional" cells which are about 100 nm on a side. The benefit of these three-dimensional cells is that they can be layered. Current high-end chips have 96 layers.

The IRDS More Moore roadmap [22] is somewhat optimistic about future flash memory scaling: more layers may be possible, maybe 256; and some amount of area scaling may also occur, especially as the limits of vertical scaling are reached. Both lead to greater memory density. But this trend cannot continue forever. A petabyte (2^{53} bit) micro-SD card would push the physical limits of flash memory technology. It would require something

[11] See http://sortbenchmark.org/ and http://sortbenchmark.org/KioxiaSort2019.pdf.

like 1024 layers of cells that are 15 nm on a side, and these cells would have to store 8 bits each. There are emerging memory technologies that might replace flash, but nothing in the roadmap promises significantly greater density.

A New York City-sized memory made of petabyte micro-SD cards would only hold 2^{99} bits. I say "only" because cryptographers commonly consider much larger memories. For instance, the attack used to evaluate the security of Frodo-640 in [29] requires[12] a memory of at least $2^{120.6}$ bits. This parameter set is given the lowest of three security categories. A similar attack on the medium-security Frodo-976 parameter set requires at least $2^{168.2}$ bits.

The surface area of the Earth is about a million times larger than New York City, roughly $2^{70}\,\mathrm{mm}^2$. A 15 millimeter thick shell of petabyte micro-SD cards around the Earth would only hold 2^{119} bits. A 15 *kilometer* thick shell would store 2^{139} bits and have a mass comparable to that of the moon. Nothing this large can be built on Earth.

And in the solar system? An 800 year long project to disassemble Jupiter and turn it into a thin shell of petabyte micro-SD cards revolving about the sun would only yield a memory of 2^{153} bits.

Bibliography

[1] Scott Aaronson and John Watrous. Closed timelike curves make quantum and classical computing equivalent. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 465(2102):631–647, 2009.

[2] Carlisle M. Adams and Henk Meijer. Security-related comments regarding McEliece's public-key cryptosystem. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 224–228. Springer, Heidelberg, August 1988.

[3] Gordon B. Agnew, Ronald C. Mullin, and Scott A. Vanstone. An implementation of elliptic curve cryptosystems over $\mathbb{F}_{2^{155}}$. *IEEE Journal on Selected areas in Communications*, 11(5):804–813, 1993.

[4] Dave Bacon. Quantum computational complexity in the presence of closed timelike curves. *Physical Review A*, 70(3):032309, 2004.

[12] For a lattice in dimension d, it is reasonable to assume that sieving algorithms store $\left(I_{3/4}((d-1)/2, 1/2)\right)^{-1} = 2^{(0.2075\cdots+o(1))d}$ vectors where $I_x(a,b)$ is the regularized incomplete beta function. See Chapter 3. Each vector is at least d bits. For Frodo-640 take $d = 520$. For Frodo-976 take $d = 746$.

[5] F. Bahr, M. Böhm, J. Franke, and T. Kleinjung. Factorization of RSA-200. http://www.loria.fr/~zimmerma/records/rsa200, May 2005.

[6] Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, Richard Davis, and Scott Simon. Recommendation for pair-wise key establishment using integer factorization cryptography. NIST Special Publication 800-56B Rev. 2, 2019. https://csrc.nist.gov/publications/detail/sp/800-56b/rev-2/final.

[7] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of statistical physics*, 22(5):563–591, 1980.

[8] Paul Benioff. Quantum mechanical Hamiltonian models of Turing machines. *Journal of Statistical Physics*, 29(3):515–546, 1982.

[9] Paul Benioff. Comment on "Dissipation in computation". *Physical Review Letters*, 53(12):1203, 1984.

[10] Charles H Bennett. Logical reversibility of computation. *IBM journal of Research and Development*, 17(6):525–532, 1973.

[11] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, April 2006.

[12] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the McEliece cryptosystem. In Johannes Buchmann and Jintai Ding, editors, *Post-quantum cryptography, second international workshop, PQCRYPTO 2008*, pages 31–46. Springer, Heidelberg, October 2008.

[13] Anne Canteaut and Nicolas Sendrier. Cryptanalysis of the original McEliece cryptosystem. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT'98*, volume 1514 of *LNCS*, pages 187–199. Springer, Heidelberg, October 1998.

[14] David Deutsch. Is there a fundamental bound on the rate at which information can be processed? *Physical Review Letters*, 48(4):286, 1982.

[15] David Deutsch. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proc. R. Soc. Lond. A*, 400:97–117, 1985.

[16] Whitfield Diffie and Martin E. Hellman. Exhaustive cryptanalysis of the NBS Data Encryption Standard. *IEEE Computer*, 1977. https://ee.stanford.edu/~hellman/publications/27.pdf.

[17] Freeman J. Dyson. Search for artificial stellar sources of infrared radiation. *Science*, 131(3414):1667–1668, 1960.

[18] Richard P. Feynman. Quantum mechanical computers. *Foundations of Physics*, 16:507–531, 1986.

[19] Jens Franke, Thorsten Kleinjung, Christof Paar, Jan Pelzl, Christine Priplata, Martin Šimka, and Colin Stahlke. An efficient hardware architecture for factoring integers with the elliptic curve method. *Special-Purpose Hardware for Attacking Cryptographic Systems–SHARCS*, 2005.

[20] Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21:219–253, 1982.

[21] Thomas Häner and Damian S Steiger. 0.5 petabyte simulation of a 45-qubit quantum circuit. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–10, 2017.

[22] IEEE International Roadmap for Devices and Systems. *More Moore*, 2018 edition. https://irds.ieee.org/editions/2018/more-moore.

[23] Stephen P. Jordan. Fast quantum computation at arbitrarily low energy. *Physical Rev. A*, 95, 2017.

[24] Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM journal of research and development*, 5(3):183–191, 1961.

[25] Pil Joong Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 275–280. Springer, Heidelberg, May 1988.

[26] Norman Margolus and Lev B. Levitin. The maximum speed of dynamical evolution. *Physica D: Nonlinear Phenomena*, 120(1-2):188–195, 1998.

[27] Robert J McEliece. A public-key cryptosystem based on algebraic coding theory. Jet Propulsion Laboratory DSN Progress Report, 1978. http://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF.

[28] N. David Mermin. What's bad about this habit. *Physics today*, 62(5):8–9, 2009.

[29] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[30] W. Porod, R. O. Grondin, D. K. Ferry, and G. Porod. Dissipation in computation. *Physical Review Letters*, 52(3):232, 1984.

[31] W. Porod, R. O. Grondin, D. K. Ferry, and G. Porod. Porod *et al.* respond. *Physical Review Letters*, 53(12):1206, 1984.

[32] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[33] Tommaso Toffoli. Comment on "Dissipation in computation". *Physical Review Letters*, 53(12):1204, 1984.

[34] Jeff Tsao, Nate Lewis, and George Crabtree. Solar FAQs. `https://www.sandia.gov/~jytsao/Solar%20FAQs.pdf`, 2006.

[35] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(1):230–265, 1937.

[36] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, January 1999.

[37] Michael J. Wiener. The full cost of cryptanalytic attacks. *Journal of Cryptology*, 17(2):105–124, March 2004.

[38] Wojciech H. Zurek. Reversibility and stability of information processing systems. *Physical Review Letters*, 53(4):391, 1984.

Chapter 2

SIKE

> *Mathematical models of computation are abstract constructions, by their nature unfettered by physical laws. However, if these models are to give indications that are relevant to concrete computing, they must somehow capture, albeit in a selective and stylized way, certain general physical restrictions to which all concrete computing processes are subjected.*
>
> Tommaso Toffoli (1980)

Context

Parallel collision search is one of the great success stories of cryptanalysis. Its theoretical analysis translates into real improvements in the cost of attacks. And, as we saw in the introduction, cryptanalysts have been reasonably successful in predicting the problem instances that can be solved with it.

Do quantum computers improve on this well understood and predictable attack?

In 1997, Brassard, Høyer, and Tapp gave an application of Grover's algorithm to collision search. Their algorithm finds a collision in a 2-to-1 function $f : [N] \to [N]$. It uses $N^{1/3+o(1)}$ bits of quantum accessible classical memory and $N^{1/3+o(1)}$ queries to f. In 2003, Grover and Rudolph questioned the practical significance of this algorithm. In 2009, Bernstein drew an explicit comparison with the cost of the van Oorschot–Wiener algorithm.

Parallel collision search can also be used to find a claw between a pair of functions, i.e. given $f, g : [N] \to [N]$ parallel collision search can be used to find some (x, y) such that $f(x) = g(y)$. A quantum algorithm for the claw finding problem was given by Tani. This algorithm uses $N^{1/3+o(1)}$ bits of quantum memory (rather than quantum accessible classical memory) and $N^{1/3+o(1)}$ queries to f and g.

Samuel Jaques and I question the practical significance of Tani's algorithm below. The argument we make is stronger than the argument made by Grover–Rudolph and by Bernstein, because quantum memory is a stronger resource than qRAM. We find that in the standard quantum circuit model known quantum collision search algorithms apply $N^{1/2+o(1)}$ gates.

The standard quantum circuit model assigns zero cost to the identity gate. This is inconsistent with current designs for quantum computers and with dissipative quantum error correction in general. Jaques and I therefore argue that a "depth-width" metric, which is closely related to the cost metrics that are used by quantum computing experimentalists, is favorable to a gate count metric. In the depth-width metric, our $N^{1/2+o(1)}$ claim is relatively easy to establish.

Why do we not use the depth-width metric for classical RAM model computation? The SRAM in a computer's CPU caches needs a continuous power supply. The DRAM in its main memory is effectively indistinguishable from a large parallel computer (as illustrated by "RowHammer" side-channel attacks,[1] and even more strikingly by the "ComputeDRAM"

[1] Kim, Yoongu, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. "Flipping bits in memory without accessing them: An experimental study

project[2]). But it is at least *possible* to build a classical computer that has only non-volatile memory. The paper tape of a Turing machine is non-volatile, as is the flash memory in a micro-SD card. So, other problems with the RAM model aside, zero-cost storage is neither physically nor technologically unrealistic.

Like dissipationless and arbitrarily fast computation, non-volatile quantum memory may be physically possible. However, its use in the quantum circuit model is an historical accident rooted in analogy rather than empiricism. Deutsch *did not* assume reliable non-volatile quantum memory when he proposed the quantum Turing machine. The assumption was introduced later, and it may represent a physical impossibility.

Does the choice between gate cost and depth-width cost matter? Jaques and I find a small gap between the gate and depth-width cost of Tani's algorithm. In a very recent paper,[3] Jaques and Schrottenloher raise the stakes substantially: they claim an exponential gap between the gate and depth-width costs of a different collision finding algorithm.

Detailed statement of contributions

Section 2 was primarily written by myself, with editorial contributions from Samuel Jaques. Sections 3 and 5 were primarily written by Samuel Jaques, with editorial contributions by myself. We made equal contributions to Section 4 and 6. Samuel Jaques contributed the TikZ diagrams of Figures 1 and 2. We made equal contributions to the software.

of DRAM disturbance errors." ACM SIGARCH Computer Architecture News 42, no. 3 (2014): 361-372. https://dl.acm.org/doi/pdf/10.1145/2678373.2665726

[2] Gao, Fei, Georgios Tziantzioulis, and David Wentzlaff. "ComputeDRAM: In-memory compute using off-the-shelf DRAMs." In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, pp. 100-113. 2019. https://dl.acm.org/doi/pdf/10.1145/3352460.3358260

[3] https://eprint.iacr.org/2020/424.pdf

Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE

Samuel Jaques John M. Schanck

Abstract We introduce models of computation that enable direct comparisons between classical and quantum algorithms. Incorporating previous work on quantum computation and error correction, we justify the use of the gate-count and depth-times-width cost metrics for quantum circuits. We demonstrate the relevance of these models to cryptanalysis by revisiting, and increasing, the security estimates for the Supersingular Isogeny Diffie–Hellman (SIDH) and Supersingular Isogeny Key Encapsulation (SIKE) schemes. Our models, analyses, and physical justifications have applications to a number of memory intensive quantum algorithms.

1 Introduction

The US National Institute of Standards and Technology (NIST) is currently standardising post-quantum cryptosystems. As part of this process, NIST has asked cryptographers to compare the security of such cryptosystems to the security of standard block ciphers and hash functions. Complicating this analysis is the diversity of schemes under consideration, the corresponding diversity of attacks, and stark differences in attacks on post-quantum schemes versus attacks on block ciphers and hash functions. Chief among the difficulties is a need to compare classical and quantum resources.

NIST has suggested that one quantum gate can be assigned a cost equivalent to $\Theta(1)$ classical gates [34, Section 4.A.5]. However, apart from the notational similarity between boolean circuits and quantum circuits, there seems to be little justification for this equivalence.

Even if an adequate cost function were defined, many submissions rely on proxies for quantum gate counts. These will need to be re-analyzed before comparisons can be made. Some submissions use query complexity as a lower bound on gate count. Other submissions use a non-standard circuit model that includes a unit-cost random access gate. The use of these proxies may lead to conservative security estimates. However,

1. they may produce severe security underestimates — and correspondingly large key size estimates — especially when they are used to analyze memory intensive algorithms; and

2. they lead to a proliferation of incomparable units.

We aim to provide cryptographers with tools for making justified comparisons between classical and quantum computations.

1.1 Contributions

In Section 2 we review the quantum circuit model and discuss the role that classical computers play in performing quantum gates and preserving quantum memories. We then introduce a model of computation in which a classical random access machine (RAM) acts as a controller for a *memory peripheral* such as an array of bits or an array of qubits. This model allows us to clearly distinguish between costly memory operations, which require the intervention of the controller, and free operations, which do not.

We then describe how to convert a quantum circuit into a parallel RAM (PRAM) program that could be executed by a collection of memory peripheral controllers. The complexity of the resulting program depends on the physical assumptions in the definition of the memory peripheral. We give two sets of assumptions that lead to two distinct cost metrics for quantum circuits. Briefly, we say that G quantum gates arranged in a circuit of depth D and width (number of qubits) W has a cost of

- $\Theta(G)$ RAM operations under the *G-cost metric*, which assumes that quantum memory is *passively corrected*; and
- $\Theta(DW)$ RAM operations under the *DW-cost metric*, which assumes that quantum memory is *actively corrected* by the memory peripheral controller.

These metrics allow us to make direct comparisons between quantum circuits and classical PRAM programs.

In the remainder of the paper we apply our cost metrics to algorithms of cryptographic significance. In Section 6 we review the known classical and quantum *claw-finding* attacks on the Supersingular Isogeny Key Encapsulation scheme (SIKE). Our analysis reveals an attack landscape that is shaped by numerous trade-offs between time, memory, and RAM operations. We find that attackers with limited memory will prefer the known quantum attacks, whereas attackers with limited time will prefer the known classical attacks. In terms of the SIKE public parameter p, there are low-memory quantum attacks that use $p^{1/4+o(1)}$ RAM operations, and there are low-depth classical attacks that use $p^{1/4+o(1)}$ RAM operations. Simultaneous time and memory constraints push the cost of all known claw-finding attacks higher. We are not aware of any attack that can be parameterized to use fewer than $p^{1/4+o(1)}$ RAM operations, although some algebraic attacks may also achieve this complexity.

We build toward our analysis of SIKE by considering the cost of prerequisite quantum data structures and algorithms. In Section 4 we introduce a new dynamic set data structure, which we call a Johnson vertex. In Section 5 we analyze the cost of quantum algorithms based on random walks on Johnson graphs. We find that data structure operations limit the range of time-memory trade-offs that are available in these algorithms. Previous analyses of SIKE [20, 21] ignore data structure operations and assume that time-memory trade-offs enable an attack of cost $p^{1/6+o(1)}$. After accounting for data structure operations, we find that the claimed $p^{1/6+o(1)}$ attack has cost $p^{1/3+o(1)}$.

In Section 6.3, we give non-asymptotic cost estimates for claw-finding attacks on SIKE-n (SIKE with an n-bit public parameter p). This analysis lends further support to the parameter recommendations of Adj et al. [1], who suggest that a 434-bit p provides 128-bit security and that a 610-bit p provides 192-bit security. Adj et al. base their recommendation on the cost of memory-constrained classical attacks. We complement this analysis by considering depth-constrained quantum attacks (with depth $< 2^{96}$). Under mild assumptions on the cost of some subroutines, we find that the best known depth-limited quantum claw-finding attack on SIKE-434 uses at least 2^{143} RAM operations. Likewise, we find that the best known depth-limited quantum claw-finding attack on SIKE-610 uses at least 2^{232} RAM operations.

Our methods have immediate applications to the analysis of other quantum algorithms that use large quantum memories and/or classical co-processors. We list some directions for future work in Section 7.

2 Machine models

We begin with some quantum computing background in Section 2.1, including the physical assumptions behind Deutsch's circuit model. We elaborate on the circuit model to construct *memory peripheral models* in Section 2.2. We specify classical control costs, with units of RAM operations, for memory peripheral models in Section 2.3. On a first read, the examples of memory peripherals given in Section 2.4 may be more informative than the general description of memory peripheral models in Section 2.2. Section 2.4 justifies the cost functions that are used in the rest of the paper.

2.1 Preliminaries on quantum computing

Quantum states and time-evolution. Let Γ be a set of observable configurations of a computer memory, e.g. binary strings. A quantum state for that memory is a unit vector $|\psi\rangle$ in a complex euclidean space $\mathcal{H} \cong \mathbb{C}^{\Gamma}$. Often Γ will have a natural cartesian product structure reflecting subsystems of the memory, e.g. an ideal n-bit memory has $\Gamma = \{0,1\}^n$. In such a case, $\mathcal{H}$ has a corresponding tensor product structure, e.g. $\mathcal{H} \cong (\mathbb{C}^2)^{\otimes n}$. The scalar product on $\mathcal{H}$ is denoted $\langle\cdot|\cdot\rangle$ and is Hermitian symmetric, $\langle\phi|\psi\rangle = \overline{\langle\psi|\phi\rangle}$. The notation $|\psi\rangle$ for unit vectors is meant to look like the right "half" of the scalar product. Dual vectors are denoted $\langle\psi|$. The set $\{|x\rangle \mid x \in \Gamma\}$ is the *computational basis* of $\mathcal{H}$. The *Hermitian adjoint* of a linear operator A is denoted $A^\dagger$. A linear operator is *self-adjoint* if $A = A^\dagger$ and *unitary* if $AA^\dagger = A^\dagger A = 1$.

One of the postulates of quantum mechanics is that the observable properties of a state correspond to self-adjoint operators. A self-adjoint operator can be written as $A = \sum_i \lambda_i P_i$ where $\lambda_i \in \mathbb{R}$ and P_i is a projector onto an eigenspace with eigenvalue λ_i. Measurement of a quantum state $|\psi\rangle$ with respect to A yields outcome λ_i with probability $\langle\psi| P_i |\psi\rangle$. The post-measurement state is an eigenvector of P_i.

Quantum computing is typically concerned with only two observables: the configurations of the memory, and the total energy of the system. The operator associated to the memory configuration has the computational basis vectors as eigenvectors; it can be written as $\sum_{x\in\Gamma} \lambda_x |x\rangle\langle x|$. If the state of the memory is given by $|\psi\rangle = \sum_{x\in\Gamma} \psi_x |x\rangle$, then measuring the memory configuration of $|\psi\rangle$ will leave the memory in configuration x with probability $|\langle x|\psi\rangle|^2 = |\psi_x|^2$. The total energy operator is called the *Hamiltonian* of the system and is denoted H. Quantum states evolve in time according to the Schrödinger equation[4]

$$\frac{\mathrm{d}}{\mathrm{d}t} |\psi(t)\rangle = -iH |\psi(t)\rangle . \tag{1}$$

Time-evolution for a duration δ yields $|\psi(t_0 + \delta)\rangle = U_\delta |\psi(t_0)\rangle$ where $U_\delta = \exp(-iH\delta)$. Note that since H is self-adjoint we have $U_\delta^\dagger = \exp(iH\delta)$ so U_δ is unitary. In general, the Hamiltonian of a system may vary in time, and one may write $H(t)$ in Eq. 1. The resulting time-evolution operator is also unitary. The Schrödinger equation applies only to closed systems. A time-dependent Hamiltonian is a convenient fiction that allows one to model an interaction with an external system without modeling the interaction itself.

Quantum circuits. Deutsch introduced the quantum circuit model in [15]. A quantum circuit is a collection of *gates* connected by *unit-wires*. Each wire represents

[4] Here we are taking Planck's constant equal to 2π, i.e. $\hbar = 1$.

the motion of a *carrier* (a physical system that encodes information). A carrier has both physical and logical (i.e. computational) degrees of freedom. External inputs to a circuit are provided by *sources*, and outputs are made available at *sinks*. The computation proceeds in time with the carriers moving from the sources to the sinks. A gate with k inputs represents a unitary transformation of the logical state space of k carriers. For example, if the carriers encode qubits, then a gate with k inputs is a unitary transformation of $(\mathbb{C}^2)^{\otimes k}$. Each gate takes some non-zero amount of time. Gates that act on disjoint sets of wires may be applied in parallel. The inputs to any particular gate must arrive simultaneously; wires may be used to delay inputs until they are needed.

Carriers feature prominently in Deutsch's description of quantum circuits [15, p. 79], as does time evolution according to an explicitly time-dependent Hamiltonian [15, p. 88]. However, while Deutsch used physical reasoning to justify his model, in particular his choice of gates, this reasoning was not encoded into the circuit diagrams themselves. The gates that appear in Deutsch's diagrams are defined entirely by the logical transformation that they perform. Gates, including the unit-wire, are deemed *computationally equivalent* if they enact the same logical transformation. Two gates can be equivalent even if they act on different carriers, take different amounts of time, etc. Computationally equivalent gates are given the same representation in a circuit diagram. Today it is common to think of quantum circuits as describing transformations of logical states alone.

2.2 Memory peripheral models

The *memory peripheral models* that we introduce in this section generalize the circuit model by making carriers explicit. We depart from the circuit model as follows:

1. We associate a carrier to each unit-wire and to each input and output wire of each gate. Wires can only be connected if they act on the same carrier.

2. We assume that the logical state of a computation emerges entirely from the physical state of its carriers.

3. Our unit-wire acts on its associated carrier by time evolution according to a given time-independent Hamiltonian for a given duration.

4. We interpret our diagrams as programs for classical controllers. Every gate (excluding the unit-wire) represents an intervention from the controller.

In sum, these changes allow us to give some physical justification for how a circuit is executed, and they allow us to assign different costs depending on the justification provided. In particular, they allow us to separate free operations — those that are due to natural time-independent evolution — from costly operations — those that are due to interventions from the classical controller.

Our model has some potentially surprising features. A unit-wire that acts on a carrier with a non-trivial Hamiltonian does not necessarily enact the logical identity transformation. Consequently, wires of different lengths may not be computationally equivalent in Deutsch's sense. In fact, since arbitrary computations can be performed *ballistically*, i.e. by time-independent Hamiltonians [16, 28, 24], the unit-wire can enact any transformation of the computational state. We do not take advantage of this in our applications; the unit-wires that we consider in Section 2.4 enact the logical identity transformation (potentially with some associated cost).

A carrier, in our model, is represented by a physical state space $\mathcal{H}$ and a Hamiltonian $H : \mathcal{H} \to \mathcal{H}$. To avoid confusion with Deutsch's carriers, we refer to $(\mathcal{H}, H)$ as a memory peripheral.

Definition 2.1. *A* memory peripheral *is a tuple* $\mathsf{A} = (\mathcal{H}, H)$ *where* $\mathcal{H}$ *is a finite dimensional state space and* H *is a Hermitian operator on* $\mathcal{H}$*. The operator* H *is referred to as the Hamiltonian of* A.

The reader may like to keep in mind the example of an ideal qubit memory $\mathsf{Q} = (\mathbb{C}^2, 0)$.

Parallel wires carry the parallel composition of their associated memory peripherals. The memory peripheral that results from parallel composition of A and B is denoted $\mathsf{A} \otimes \mathsf{B}$. The state space associated with $\mathsf{A} \otimes \mathsf{B}$ is $\mathcal{H}^{\mathsf{A}} \otimes \mathcal{H}^{\mathsf{B}}$, and the Hamiltonian is $H^{\mathsf{A}} \otimes I^{\mathsf{B}} + I^{\mathsf{A}} \otimes H^{\mathsf{B}}$. We say that A and B are *sub-peripherals* of $\mathsf{A} \otimes \mathsf{B}$. We say that a memory peripheral is *irreducible* if it has no sub-peripherals. The *width* of a memory peripheral is the number of irreducible sub-peripherals it contains.

A quantum circuit on n qubits may be thought of as a program for the memory peripheral $\mathsf{Q}^{\otimes n}$. Programs for other memory peripherals may involve more general memory operations.

Definition 2.2. *A* memory operation *is a morphism of memory peripherals* $f : \mathsf{A} \to \mathsf{B}$ *that acts as a quantum channel between* $\mathcal{H}^{\mathsf{A}}$ *and* $\mathcal{H}^{\mathsf{B}}$*, i.e. it takes quantum states on* $\mathcal{H}^{\mathsf{A}}$ *to quantum states on* $\mathcal{H}^{\mathsf{B}}$.

The *arity* of a memory operation is the number of irreducible sub-peripherals on which it acts. If there is no potential for ambiguity, we will refer to memory operations as gates. Examples of memory operations include: unitary transformations of a single state space, isometries between state spaces, state preparation, measurement, and changes to the Hamiltonian of a carrier.

In order to define state preparation and measurement it is convenient to introduce a void peripheral $\mathsf{1}$. State preparation is a memory operation of the form $\mathsf{1} \to \mathsf{A}$, and measurement is a memory operation of the form $\mathsf{A} \to \mathsf{1}$. The reader may assume that $\mathsf{1} = (\mathbb{C}, 0)$ in all of our examples.

Networks of memory operations can be represented by diagrams that are almost identical to quantum circuits. Memory peripherals must be clearly labelled, and times must be given for gates, but no other diagrammatic changes are necessary. An example is given in Figure 1.

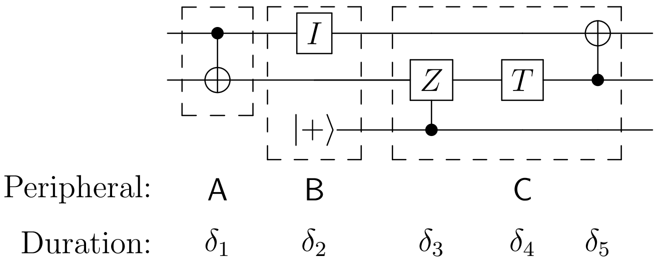
Just as it is useful to specify a gate set for quantum circuits, it is useful to define collections of memory peripherals that are closed under parallel composition and under sequential composition of memory operations. The notion of a symmetric monoidal category captures the relevant algebraic structure. The following definition is borrowed from [13, Definition 2.1] and, in the language of that paper, makes a memory peripheral model into a type of *resource theory*. The language of resource theories is not strictly necessary for our purposes, but we think this description may have future applications.

Definition 2.3. *A* memory peripheral model *is a symmetric monoidal category* $(\mathbf{C}, \circ, \otimes, \mathsf{1})$ *where*

- *the objects of* $\mathbf{C}$ *are memory peripherals,*
- *the morphisms between objects of* $\mathbf{C}$ *are memory operations,*
- *the binary operation* $\circ$ *denotes sequential composition of memory operations,*
- *the binary operation* $\otimes$ *denotes parallel composition of memory peripherals and of memory operations, and*
- *the void peripheral* $\mathsf{1}$ *satisfies* $\mathsf{A} \otimes \mathsf{1} = \mathsf{1} \otimes \mathsf{A} = \mathsf{A}$ *for all* $\mathsf{A} \in \mathbf{C}$*.*

2.3 Parallel RAM controllers for memory peripheral models

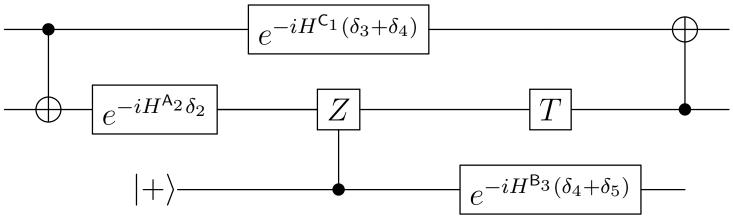
A memory peripheral diagram can be viewed as a program that tells a classical computer where, when, and how to interact with its memory. We will now specify a computer that executes these programs.



(a) A memory peripheral diagram with $\mathsf{A} = \mathsf{A}_1 \otimes \mathsf{A}_2$, $\mathsf{B} = \mathsf{A}_1 \otimes \mathsf{A}_2 \otimes \mathsf{B}_3$, and $\mathsf{C} = \mathsf{C}_1 \otimes \mathsf{A}_2 \otimes \mathsf{B}_3$ (numbered top to bottom). The memory peripheral changes from A to B in the second time step because of the new qubit. The change from B to C in the third time step is marked by a carrier change that enacts the logical identity gate. Note that truly instantaneous changes are not possible, but the δ_i could be very small.

Controller one	Controller two	Controller three
1. APPLY “CNOT” (1,2)	1. no op.	1. APPLY “INIT” (0,3)
2. no op.	2. STEP	2. no op.
3. APPLY “A_1 to C_1” (0,1)	3. no op.	3. no op.
4. no op.	4. no op.	4. STEP
5. no op.	5. APPLY “CZ” (3,2)	5. no op.
6. no op.	6. APPLY “T” (0,2)	6. no op.
7. APPLY “CNOT” (2,1)	7. no op.	7. no op.
8. no op.	8. no op.	8. STEP

(b) A PRAM program for the memory peripheral diagram of Fig. 1a using the notation of Section 2.3.


$e^{-iH^{\mathsf{C}_1}(\delta_3+\delta_4)}$
$e^{-iH^{\mathsf{A}_2}\delta_2}$
Z
T
$|+\rangle$
$e^{-iH^{\mathsf{B}_3}(\delta_4+\delta_5)}$

(c) A quantum circuit for the memory peripheral diagram of Fig. 1a. Note that the I gate is absent, and explicit time-evolution with respect to the wire Hamiltonians has been added.

Figure 1: Three representations of a quantum algorithm.

Following Deutsch, we have assumed that all gates take a finite amount of time, that each gate acts on a bounded number of subsystems, and that gates that act on disjoint subsystems can be applied in parallel. Circuits can be of arbitrary width, so a control program may need to execute an unbounded number of operations in a finite amount of time. Hence, we must either assume that the classical control computer can operate arbitrarily quickly or in parallel.

We opt to treat controllers as parallel random access machines (PRAMs). Several variants of the PRAM exist [26]. The exact details of the instruction set and concurrency model are largely irrelevant here. For our purposes, a PRAM is a collection of RAMs that execute instructions in synchrony. Each RAM executes (at most) one instruction per time step. At time step i each RAM can assume that the other RAMs have completed their step $i-1$ instructions. We assume that synchronization between RAMs and memory peripherals is free.

We assign a unique positive integer to each wire in a diagram, so that an ordered collection of k memory peripherals can be identified by a k-tuple of integers. We use a k-tuple to specify an input to a k-ary gate. The memory operations that are available to a controller are also assigned unique positive integers.

We add two new instructions to the RAM instruction set: `APPLY` and `STEP`. These instructions enable parallel and sequential composition of memory operations, respectively. `APPLY` takes three arguments: a k-tuple of addresses, a memory operation, and an (optional) k-tuple of RAM addresses in which to store measurement results. `STEP` takes no arguments; it is only used to impose a logical sequence on steps of the computation

When a processor calls `APPLY` the designated memory operation is scheduled to be performed during the next `STEP` call. In one layer of circuit depth, each RAM processor schedules some number of memory operations to be applied in parallel and then one processor calls `STEP`. If memory operations with overlapping addresses are scheduled for the same step, the behaviour of the memory peripheral is undefined and the controller halts. This ensures that only one operation is applied per subsystem per call to `STEP`.

A quantum circuit of width W can be converted into $O(W)$ RAM programs by assigning gates to processors according to a block partition of $\{1, \dots, W\}$. The blocks should be of size $O(1)$, otherwise a single processor could need to execute an unreasonable number of operations in a fixed amount of time. If a gate involves multiple qubits that are assigned to different processors, the gate is executed by the processor that is responsible for the qubit of lowest address. We have provided an example in Figure 1b.

To apply a multi-qubit gate, a RAM processor must be able to address arbitrary memory peripherals. This is a strong capability. However, each peripheral is involved in at most one gate per step, so this type of random access is analogous to the exclusive-read/exclusive-write random access that is typical of PRAMs.

The cost of a PRAM computation. Every RAM instruction has unit cost, except for the placeholder "no operation" instruction, `no op`, which is free. The cost of a PRAM computation is the total number of RAM operations executed.

2.4 Examples of memory peripheral models

Here we give three examples of memory peripheral models. Example 2.4.1 is classical and primarily an illustration of the model. It shows that our framework can accommodate classical memory without changing the PRAM costs. Example 2.4.2 is a theoretical self-corrected quantum memory that justifies the G-cost. Example 2.4.3 is a more realistic actively-corrected quantum memory that justifies the DW-cost.

2.4.1 Non-volatile classical memories.

A non-volatile bit-memory can store a bit indefinitely without periodic error correction or read/write cycles. As a memory peripheral, this can simulate other classical computation models and gives the expected costs.

Technologies. The historically earliest example of a non-volatile bit memory is the "core memory" of Wang and Woo [41]. A modern example is Ferroelectric RAM (FeRAM). The DRAM found in common consumer electronics requires a periodic read/write cycle, which should be included in a cost analysis. While there may be technological and economic barriers to using non-volatile memory at all stages of the computing process, there are no physical barriers.

Hamiltonian of a memory cell. A logical bit can be encoded in the net magnetization of a ferromagnet. A ferromagnet can be modelled as a collection of *spins*. Each spin is oriented up or down, and has state $|\uparrow\rangle$ or $|\downarrow\rangle$. The self-adjoint operator associated to the orientation of a spin is $\sigma_z = |\uparrow\rangle\langle\uparrow| - |\downarrow\rangle\langle\downarrow|$; measuring $|\uparrow\rangle$ with respect to σ_z yields outcome $+1$ with probability 1, and measuring $|\downarrow\rangle$ yields outcome -1 with probability 1.

In the d-dimensional Ising model of ferromagnetism, L^d spins are arranged in a regular square lattice of diameter L in d-dimensional space. The Ising Hamiltonian imposes an energy penalty on adjacent spins that have opposite orientations:

$$H_{Ising} = -\sum_{(i,j)} \sigma_z^{(i)} \otimes \sigma_z^{(j)}.$$

In 1936 [35] Peierls showed that the Ising model is *thermally stable* in dimensions $d \geq 2$. The two ground states, all spins pointing down and all spins pointing up, are energetically separated. The energy required to map the logical zero (all down) to logical one (all up) grows with L, and the probability of this happening (under a reasonable model of thermal noise) decreases with L. The phenomenon of thermal stability in dimensions 2 and 3 provides an intuitive explanation for why we are able to build classical non-volatile memories like core-memory (see also [14, Section X.A]).

Memory peripheral model. A single non-volatile bit, encoded in the net magnetization of an $L \times L$ grid of spins, can be represented by a memory peripheral $\mathsf{B}_L = ((\mathbb{C}^2)^{\otimes L^2}, H_{Ising})$. From a single bit we can construct w-bit word peripherals $\mathsf{W}_{L,w} = \mathsf{B}_L^{\otimes w}$.

Turing machines, boolean circuits, PRAMs, and various other classical models can be simulated by controllers for word memory peripheral models. The only strictly necessary memory operations are those for reading and writing individual words.

2.4.2 Self-correcting quantum memories.

A self-correcting quantum memory is the quantum analogue of a non-volatile bit memory. The Hamiltonian of the carrier creates a large energy barrier between logical states. At a sufficiently low temperature the system does not have enough energy for errors to occur.

The thermal stability of the Ising model in $d \geq 2$ spatial dimensions seems to have inspired Kitaev's search for geometrically local quantum stabilizer codes [27]. The two-dimensional toric code that Kitaev defined in [27] is not thermally stable [2]. However, a four-dimensional variant is thermally stable [14, 3]. The question of whether there exists a self-correcting quantum memory with a Hamiltonian that is geometrically local in < 4 spatial dimensions remains open.

In two spatial dimensions, various "no-go theorems" suggest that self-correcting quantum memories may not exist. For example, a stabilizer code defined on a two-dimensional lattice of qubits cannot self-correct [11]. Brown et al. [12] summarize generalizations of this no-go result and survey the remaining avenues toward self-correcting memory in low dimensions.

At present, a model of quantum computation that assumes non-volatile memory, i.e. a free identity gate, and < 4 spatial dimensions is making a physical assumption about the existence of two- or three-dimensional self-correcting memories. Here we will simply ignore geometric locality and write down a memory peripheral for the four-dimensional toric code. Because real devices are limited to three spatial dimensions, this is purely a theoretical example.

Memory peripheral model. The Hamiltonian for the four-dimensional toric code can be found in [14, Section X.B]. We will denote it H_{toric}. Like the four-dimensional Ising Hamiltonian it is defined on L^4 spins arranged in a square lattice. The memory peripheral $\mathsf{Q}_{toric} = (\mathbb{C}^{L^4}, H_{toric})$ can serve as a drop-in replacement for the ideal qubit memory peripheral Q for the purpose of describing the unit-wire.

To execute arbitrary quantum computations on a collection of logical qubits encoded in Q_{toric} peripherals, we need memory operations for a universal gate set, initialization, and measurement. Initialization and Clifford+T gates are described for the two-dimensional toric code in [14, Section IX] and the four-dimensional versions are similar. A measurement procedure for the four-dimensional toric code is in [14, Section X.B]. Treating any of these procedures as a single memory operation will mask some classical control cost that is polynomial in L. Treating the T gate as a single memory operation masks the use of an additional memory peripheral to hold a resource state.

Cost function. A quantum circuit on n qubits can be converted into a memory peripheral diagram for $\mathsf{Q}_{toric}^{\otimes n}$ and then interpreted as a PRAM program. In this way we can assign a cost, in units of RAM operations, to the quantum circuit itself. Each wire in the quantum circuit is assigned a length in the memory peripheral diagram. The quantum circuit and memory peripheral diagram are otherwise identical. Each gate in the diagram (including state-preparation and measurement gadgets, but not unit-wires) is expanded into at least one `APPLY` instruction. The wires themselves incur no RAM cost, but one `STEP` instruction is needed per layer of circuit depth for synchronization. The number of `STEP` instructions is no more than the number of `APPLY` instructions. The following cost function, the G-cost, is justified by assuming that each gate expands to $O(1)$ RAM operations.

Definition 2.4 (G-cost)**.** *A logical Clifford+T quantum circuit that uses G gates (in any arrangement) has a G-*cost *of $\Theta(G)$ RAM operations.*

Remark 2.1. *The depth and width of a circuit do not directly affect its G-cost, but these quantities are often relevant in practice. A PRAM controller for a circuit that* *uses G gates in an arrangement that is D gates deep and W qubits wide uses $O(W)$ RAM processors for $\Omega(D)$ time. Various G-cost-preserving trade-offs between time and number of processors may be possible. For example, a circuit can be re-written so that no two gates are applied at the same time. In this way, a single RAM processor can execute any G gate circuit in $\Theta(G)$ time. This trade-off is only possible because self-correcting memory allows us to assign an arbitrary duration to a unit-wire.*

2.4.3 Actively corrected quantum memories.

It should be possible to build quantum computers even if it is not possible to build self-correcting quantum memories. Active error correction strategies are nearing technological realizability; several large companies and governments are currently pursuing technologies based on the surface code.

Memory peripheral model. When using an active error correction scheme, a logical Clifford+T circuit has to be compiled to a physical circuit that includes active error correction. We may assume that the wires carry the ideal qubit memory peripheral Q. A more detailed analysis might start from the Hamiltonians used in circuit QED [9].

Memory operations. The compiled physical circuit will not necessarily use the Clifford+T gate set. The available memory operations will depend on the physical architecture, e.g. in superconducting nano-electronic architectures one typically has arbitrary single qubit rotations and one two-qubit gate [42].

Cost function. We can assume that every physical gate takes $\Theta(1)$ RAM operations to apply. This may mask a large constant; a proposal for a hardware implementation of classical control circuitry can be found in [32]. A review of active quantum error correction for the purpose of constructing memories can be found in [39].

An active error correction routine is applied, repeatedly, to all physical qubits regardless of the logical workload. If we assume that logical qubits can be encoded in a constant number of physical qubits, and that logical Clifford+T gates can be implemented with a constant number of physical gates, then the above considerations justify the DW-cost for quantum circuits.

Definition 2.5 (DW-cost)**.** *A logical Clifford+T quantum circuit that is D gates deep, W qubits wide, and uses any number of gates within that arrangement has a DW-*cost *of $\Theta(DW)$ RAM operations.*

Remark 2.2. *In contrast with the G-cost, there are no DW-cost preserving trade-offs between time and number of processors when constructing a PRAM program from a quantum circuit. A circuit of depth D and width W uses $\Theta(W)$ processors for time $\Theta(D)$.*

Technologies. Fowler et al. provide a comprehensive overview of the surface code [17]. Importantly, to protect a circuit of depth D and width W, the surface code requires $\Theta(\log^2(DW))$ physical qubits per logical qubit. The active error correction is applied in a regular cycle (once every $200ns$ in [17]). In each cycle a constant fraction of the physical qubits are measured and re-initialized. The measurement results are processed with a non-trivial classical computation [18]. The overall cost of surface code computation is $\Omega(\log^2(DW))$ RAM operations per logical qubit per layer of logical circuit depth. Nevertheless, future active error correction techniques may bring this more in line with the DW-cost.

3 Cost analysis: Quantum random access

Our memory peripheral models provide classical controllers with random access to individual qubits. A controller can apply a memory operation — e.g. a Clifford+T gate or a measurement — to any peripheral in any time step. However, a controller does not have quantum random access to individual qubits. A controller cannot call `APPLY` with a superposition of addresses. Quantum random access must be built from memory operations.

In [4], Ambainis considers a data structure that makes use of a "random access gate." This gate takes an index i, an input b, and an R element array $A = (a_1, a_2, \ldots, a_R)$. It computes the XOR of a_i and b:

$$|i\rangle\,|b\rangle\,|A\rangle \mapsto |i\rangle\,|b \oplus a_i\rangle\,|A\rangle\,. \tag{2}$$

Assuming that each $|a_j\rangle$ is encoded in $O(1)$ irreducible memory peripherals, a random access gate has arity that grows linearly with R. If the underlying memory peripheral model only includes gates of bounded arity, then an implementation of a random access gate clearly uses $\Omega(R)$ operations. Beals et al. have noted that a circuit for random access to an R-element array of m-bit strings must have width $\Omega(Rm)$ and depth

$\Omega(\log R)$ [5, Theorem 4]. Here we give a Clifford+T construction that is essentially optimal[5].

Rather than providing a full circuit, we will describe how the circuit acts on $|i\rangle\,|0\rangle\,|A\rangle$. The address is $\log R$ bits and each register of A is m bits. We use two ancillary arrays $|A'\rangle$ and $|A''\rangle$, both initialized to 0. The array A' holds R address-sized registers and $O(R)$ additional qubits for intermediary results, a total of $O(R\log R)$ qubits. The array A'' is $O(Rm)$ qubits.

We use a standard construction of R-qubit fan-out and R-qubit parity due to Moore [33]. The fan-out is a tree of $O(R)$ CNOT gates arranged in depth $O(\log R)$. Parity is fan-out conjugated by Hadamard gates. We also use a $\log R$-bit comparison circuit due to Thapliyal, Ranganathan, and Ferreir [40]. This circuit uses $O(\log R)$ gates in depth $O(\log\log R)$.

Our random access circuit acts as follows:

1. *Fan-out address:* Fan-out circuits copy the address i to each register of A'. This needs a total of $\log R$ fan-outs, one for each bit of address. These can all be done in parallel.

2. *Controlled copy:* For each $1 \leq j \leq R$, the boolean value $A'[j] = j$ is stored in the scratch space associated to A'. The controller knows the address of each register, so it can apply a dedicated circuit for each comparison. Controlled-CNOTs are used to copy $A[j]$ to $A''[j]$ when $A'[j] = j$. Since $A'[j] = j$ if and only if $j = i$, this copies $A[i]$ to $A''[i]$ but leaves $A''[j] = 0$ for $j \neq i$.

3. *Parity:* Since $A''[j]$ is 0 for $j \neq i$, the parity of the low-order bit of all the A'' registers is equal to the low-order bit of just $A''[i]$. Likewise for the other $m - 1$ bits. So parallel R-qubit parity circuits can be used to copy $A''[i]$ to an m-qubit output register.

4. *Uncompute:* The controlled copy and fan-out steps are applied in reverse, returning A'', A', and the scratch space to zero.

The entire circuit can be implemented in width $O(Rm + R\log R)$. Step 1 dominates the depth and Step 2 dominates the gate cost. The comparison circuits use $O(R\log R)$ gates with depth $O(\log\log R)$. To implement the controlled-CNOTs used to copy $A[i]$ to $A''[i]$ in constant depth, instead of $O(m)$ depth, each of the R comparison results can be fanned out to $(m - 1)$ qubits in the scratch space of A''. This fan-out has depth $O(\log m)$.

[5] Actually, here and elsewhere, we use a gate set that includes Toffoli gates and controlled-swap gates. These can be built from $O(1)$ Clifford+T gates.

The total cost of random access is given in Cost 1. Observe that there is more than a constant factor gap between the G- and DW-cost.

Cost 1 Random access to R registers of m bits each.

- **Gates:** $O(Rm + R \log R)$
- **Depth:** $O(\log m + \log R)$
- **Width:** $O(Rm + R \log R)$

4 Cost analysis: The Johnson vertex data structure

We expect to find significant gaps between the G- and DW-costs of algorithms that use a large amount of memory. Candidates include quantum algorithms for element distinctness [4], subset-sum [7], claw-finding [38], triangle-finding [30], and information set decoding [25]. All of these algorithms are based on quantum random walks on Johnson graphs — graphs in which each vertex corresponds to a subset of a finite set.

In this section we describe a quantum data structure for representing a vertex of a Johnson graph. Essentially, we need a dynamic set that supports membership testing, uniform sampling from the encoded set, insertion, and deletion. These operations can be fine-tuned for quantum walk applications. In particular, insertion and deletion only need to be defined on inputs that would change the size of the encoded set. To avoid ambiguity, we will refer to these special cases as *guaranteed insertion* and *guaranteed deletion.*

4.1 History-independence

Fix a finite set $\mathcal{X}$. A quantum data structure for subsets of $\mathcal{X}$ consists of two parts: a presentation of subsets as quantum states, and unitary transformations representing set operations. The presentation must assign a *unique* quantum state $|\mathcal{A}\rangle$ to each $\mathcal{A} \subset \mathcal{X}$. Uniqueness is a strong condition, but it is necessary for quantum interference. Different sequences of insertions and deletions that produce the same set will only interfere if each sequence presents the output in exactly the same way. The set $\{0, 1\}$ cannot be stored as $|0\rangle\,|1\rangle$ or $|1\rangle\,|0\rangle$ depending on the order in which the elements were inserted. Some valid alternatives are to fix an order (e.g. always store $|0\rangle\,|1\rangle$) or to coherently randomize the order (e.g. always store $\frac{1}{\sqrt{2}}(|0\rangle\,|1\rangle + |1\rangle\,|0\rangle)$). Data structures that allow for interference between computational paths are called *history-independent*.

Ambainis describes a history-independent data structure for sets in [4]. His construction is based on a combined hash table and skip list. Bernstein, Jeffery, Lange, and Meurer [7], and Jeffery [22], provide a simpler solution based on radix trees. Both of these data structures use random access gates extensively. Our Johnson vertices largely avoid random access gates, and in Section 4.4 we show that our data structure is more efficient as a result.

4.2 Johnson vertices

The Johnson graph $J(X, R)$ is a graph whose vertices are R-element subsets of $\{1, \ldots, X\}$. Subsets $\mathcal{U}$ and $\mathcal{V}$ are adjacent in $J(X, R)$ if and only if $|\mathcal{U} \cap \mathcal{V}| = R - 1$. In algorithms it is often useful to fix a different base set, so we will define our data structure with this in mind: A *Johnson vertex* of capacity R, for a set of m-bit strings, is a data structure that represents an R-element subset of some set $\mathcal{X} \subseteq \{1, \ldots, 2^m - 1\}$. This implies $\log_2 R \leq m$.

In our implementation below, a subset is presented in lexicographic order in an array of length R. This ensures that every R element subset has a unique presentation.

We describe circuits parameterized by m and R for membership testing, uniform sampling, guaranteed insertion, and guaranteed deletion. Since R is a circuit parameter, our circuits cannot be used in situations where R varies between computational paths[6]. This is fine for quantum walks on Johnson graphs, but it prevents our data structure from being used as a generic dynamic set.

Memory allocation.

The set is stored in a length R array of m-bit registers that we call A. Every register is initialized to the m-bit zero string, $\perp$. The guaranteed insertion/deletion and membership testing operations require auxiliary arrays A' and A''. Both contain $O(Rm)$ bits and are initialized to zero. It is helpful to think of these as length R arrays of m-bit registers that each have some scratch space. We will not worry about the exact layout of the scratch space.

[6] One can handle a range of capacities using controlled operations, but the size of the resulting circuit grows linearly with the number of capacities it must handle.

Guaranteed insertion/deletion.

Let $\mathcal{U}$ be a set of m-bit strings with $|\mathcal{U}| = R - 1$, and suppose x is an m-bit string not in $\mathcal{U}$. The capacity $R - 1$ guaranteed insertion operation performs

$$|\mathcal{U}\rangle \, |\bot\rangle \, |x\rangle \mapsto |\mathcal{U} \cup \{x\}\rangle \, |x\rangle \, .$$

Capacity R guaranteed deletion is the inverse operation.

Figure 2 depicts the following implementation of capacity $R - 1$ guaranteed insertion. For concreteness, we assume that the correct position of x is at index k with $1 \leq k \leq R$. At the start of the routine, the first $R - 1$ entries of A represent a sorted list. Entry R is initialized to $|\bot\rangle = |0\rangle^{\otimes m}$.

(a). *Fan-out:* Fan-out the input x to the R registers of A' and also to $A[R]$, the blank cell at the end of A. The fan-out can be implemented with $O(Rm)$ gates in depth $O(\log R)$ and width $O(Rm)$.

(b). *Compare:* For i in 1 to R, flip all m bits of $A''[i]$ if and only if $A'[i] \leq A[i]$. The comparisons are computed using the scratch space in A''. Each comparison costs $O(m)$ gates, and has depth $O(\log m)$ and width $O(m)$ [40]. The single bit result of each comparison is fanned out to all m bits of $A''[i]$ using $O(m)$ gates in depth $O(\log m)$. The total cost is $O(Rm)$ gates, $O(\log m)$ depth.

(c). *First conditional swap:* For i in 1 to $R - 1$, if $A''[i]$ is $11\ldots1$ swap $A'[i+1]$ and $A[i]$. After this step, cells k through R of A hold copies of x. The values originally in $A[k], \ldots, A[R-1]$ are in $A'[k+1], \ldots, A'[R]$. Each register swap uses m controlled-swap gates. All of the swaps can be performed in parallel. The cost is $O(Rm)$ gates in $O(1)$ depth.

(d). *Second conditional swap:* For i in 1 to $R - 1$, if $A''[i]$ is $11\ldots1$ then swap $A'[i+1]$ and $A[i+1]$. After this step, the values originally in $A'[k+1], \ldots, A'[R]$ are in $A[k+1], \ldots, A[R]$. The cost is again $O(Rm)$ gates in $O(1)$ depth.

(e). *Clear comparisons:* Repeat the comparison step to reset A''.

(f). *Clear fan-out:* Fan-out the input x to the array A'. This will restore A' back to the all 0 state. Note that the fan-out does not include $A[R]$ this time.

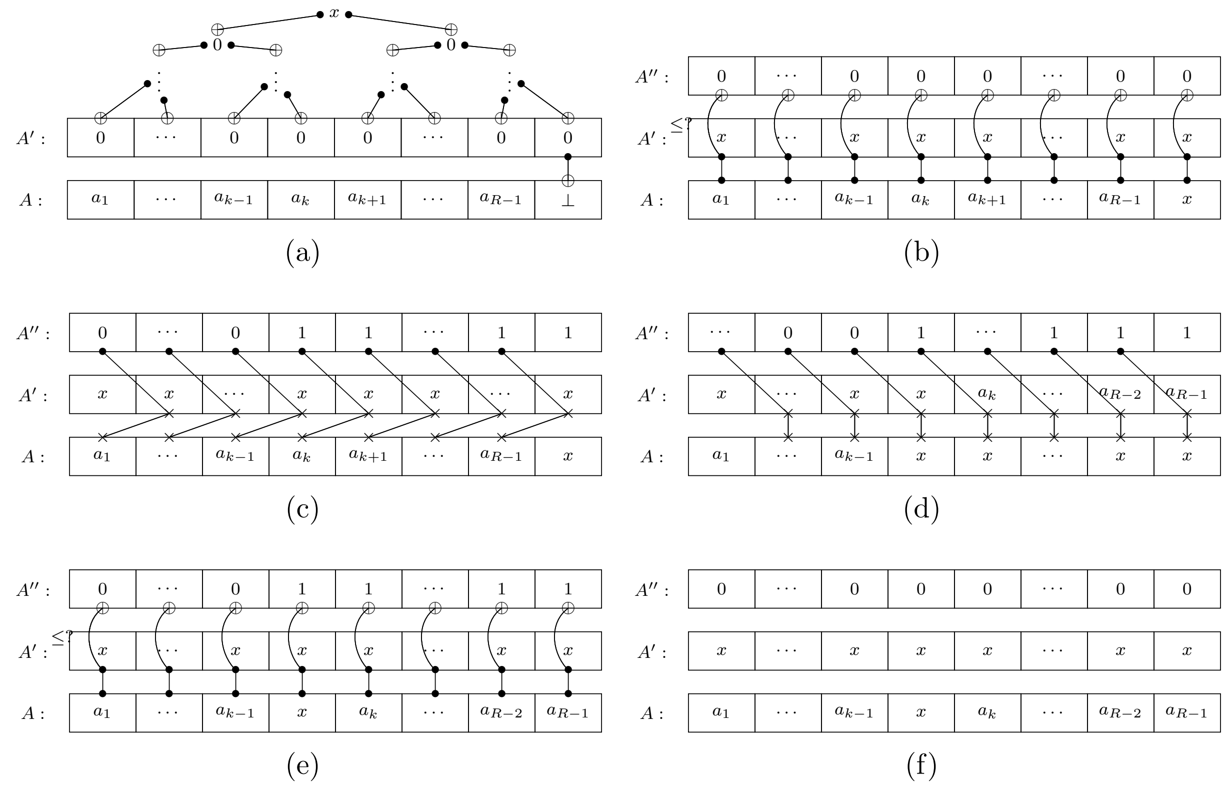

(a)
x
0
0
A' :
0 | $\cdots$ | 0 | 0 | 0 | $\cdots$ | 0 | 0
A :
a_1 | $\cdots$ | a_{k-1} | a_k | a_{k+1} | $\cdots$ | a_{R-1} | $\perp$
(b)
A'' :
0 | $\cdots$ | 0 | 0 | 0 | $\cdots$ | 0 | 0
$\leq ?$
A' :
x | $\cdots$ | x | x | x | $\cdots$ | x | x
A :
a_1 | $\cdots$ | a_{k-1} | a_k | a_{k+1} | $\cdots$ | a_{R-1} | x
(c)
A'' :
0 | $\cdots$ | 0 | 1 | 1 | $\cdots$ | 1 | 1
A' :
x | x | $\cdots$ | x | x | x | $\cdots$ | x
A :
a_1 | $\cdots$ | a_{k-1} | a_k | a_{k+1} | $\cdots$ | a_{R-1} | x
(d)
A'' :
$\cdots$ | 0 | 0 | 1 | $\cdots$ | 1 | 1 | 1
A' :
x | $\cdots$ | x | x | a_k | $\cdots$ | a_{R-2} | a_{R-1}
A :
a_1 | $\cdots$ | a_{k-1} | x | x | $\cdots$ | x | x
(e)
A'' :
0 | $\cdots$ | 0 | 1 | 1 | $\cdots$ | 1 | 1
$\leq ?$
A' :
x | $\cdots$ | x | x | x | $\cdots$ | x | x
A :
a_1 | $\cdots$ | a_{k-1} | x | a_k | $\cdots$ | a_{R-2} | a_{R-1}
(f)
A'' :
0 | $\cdots$ | 0 | 0 | 0 | $\cdots$ | 0 | 0
A' :
x | $\cdots$ | x | x | x | $\cdots$ | x | x
A :
a_1 | $\cdots$ | a_{k-1} | x | a_k | $\cdots$ | a_{R-2} | a_{R-1}

Figure 2: Insertion into a Johnson vertex. See text for full description.

Membership testing and relation counting.

The capacity R membership testing operation performs

$$|\mathcal{U}\rangle\,|x\rangle\,|b\rangle \mapsto \begin{cases} |\mathcal{U}\rangle\,|x\rangle\,|b \oplus 1\rangle & \text{if } x \in \mathcal{U} \\ |\mathcal{U}\rangle\,|x\rangle\,|b\rangle & \text{otherwise.} \end{cases}$$

As in guaranteed insertion/deletion, the routine starts with a fan-out followed by a comparison. In the comparison step we flip the leading bit of $A''[i]$ if and only if $A'[i] = A[i]$. This will put at most one 1 bit into the A'' array. Computing the parity of the A'' array will extract the result. The comparisons use $O(Rm)$ gates in depth $O(\log m)$ [40], as does the parity check [33]. Thus the cost of membership testing matches that of guaranteed insertion: $O(Rm)$ gates in depth $O(\log m + \log R)$.

The above procedure is easily modified to test other relations and return the total number of matches. In place of the parity circuit, we would use a binary tree of

Cost 2 Guaranteed insertion/deletion for a Johnson vertex of capacity R with m-bit elements.

- **Gates:** $O(Rm)$
- **Depth:** $O(\log m + \log R)$
- **Width:** $O(Rm)$

$O(\log R)$-bit addition circuits. With the adders of [37], the cost of the addition tree is $O(R \log R)$ gates in depth $O(\log^2 R)$. The ancilla bits for the addition tree do not increase the overall width beyond $O(Rm)$. As such, the gate cost of the addition tree is no more than a constant factor more than the cost of a guaranteed insertion. The full cost of relation counting will also depend on the cost of evaluating the relation.

Cost 3 Membership testing and relation counting for a Johnson vertex of capacity R with m-bit elements. The terms T_G, T_D, and T_W denote the gates, depth, and width of evaluating a relation.

	Membership testing	**Relation counting**
Gates:	$O(Rm)$	$O(Rm + R\mathsf{T}_G)$
Depth:	$O(\log m + \log R)$	$O(\log^2 R + \mathsf{T}_D)$
Width:	$O(Rm)$	$O(Rm + R\mathsf{T}_W)$

Uniform sampling.

The capacity R uniform sampling operation performs $|\mathcal{A}\rangle |0\rangle = |\mathcal{A}\rangle \left(\frac{1}{\sqrt{R}} \sum_{x \in \mathcal{A}} |x\rangle \right)$. We use a random access to the array A with a uniform superposition of addresses. By Cost 1, this uses $O(Rm)$ gates in depth $O(\log m + \log R)$.

4.3 Random replacement

A quantum walk on a Johnson graph needs a subroutine to replace $\mathcal{U}$ with a neighbouring vertex in order to take a step. Intuitively, this procedure just needs to delete $u \in \mathcal{U}$, sample $x \in \mathcal{X} \backslash \mathcal{U}$, then insert x. The difficulty lies in sampling x in such a way that it can be uncomputed even after subsequent insertion/deletion operations. The naive rejection sampling approach will entangle x with $\mathcal{U}$.

The applications that we consider below can tolerate a replacement procedure that leaves $\mathcal{U}$ unchanged with probability R/X. We first sample x uniformly from $\mathcal{X}$ and perform a membership test. This yields $\sqrt{1/X}\sum_{x\in\mathcal{X}}|\mathcal{U}\rangle\,|x\rangle\,|x\in\mathcal{U}\rangle$. Conditioned on non-membership, we uniformly sample some $u\in\mathcal{U}$, delete u, and insert x. Conditioned on membership, we copy x into the register that would otherwise hold u. The membership bit can be uncomputed using the "u" register. This yields $\sqrt{1/X}\sum_{x\in\mathcal{U}}|\mathcal{U}\rangle\,|x\rangle\,|x\rangle+\sqrt{1/RX}\sum_{\mathcal{V}\sim\mathcal{U}}|\mathcal{V}\rangle\,|x\rangle\,|u\rangle$. The cost of random replacement is $O(1)$ times the cost of guaranteed insertion plus the cost of uniform sampling in $\mathcal{X}$.

4.4 Comparison with quantum radix trees

In [7] a quantum radix tree is constructed as a uniform superposition over all possible memory layouts of a classical radix tree. This solves the problem of history-dependence, but relies heavily on random access gates. The internal nodes of a radix tree store the memory locations of its two children. In the worst case, membership testing, insertion, and deletion follow paths of $\Theta(m)$ memory locations. Because a quantum radix tree is stored in all possible memory layouts, these are genuine random accesses to an R register array. Note that a radix tree of m-bit strings cannot have more than 2^m leaves. As such, $\log R = O(m)$ and Cost 1 matches the lower bound for random access gates given by Beals et al. [5]. Cost 4 is obtained by using Cost 1 for each of the $O(\log R)$ random accesses. The lower bound in Cost 4 exceeds the upper bound in Cost 2.

Cost 4 Membership testing, insertion, and deletion for quantum radix trees.

Gates: $\Omega(Rm^2)$
Depth: $\Omega(m\log m + m\log R)$
Width: $\Omega(Rm)$

5 Cost analysis: Claw-finding by quantum walk

5.1 Quantum walk based search algorithms

Let $\mathcal{S}$ be a finite set with a subset $\mathcal{M}$ of "marked" elements. We focus on a generic search problem: to find some $x\in\mathcal{M}$. A simple approach is to repeatedly guess elements of $\mathcal{S}$. This can be viewed as a random walk. At each step, one transitions from the current guess to another with uniform probability. The random walk starts with a *setup* routine that produces an initial element $\mathcal{S}$. It then repeats a loop of

1) *checking* if the current element is marked, and 2) *walking* to another element. Of course, one need not use the uniform distribution. In a Markov chain, the transition probabilities can be arbitrary, so long as they only depend on the current guess. The probability of transitioning from a guess of u to a guess of v can be viewed as a weighted edge in a graph with vertex set $\mathcal{S}$. The weighted adjacency matrix of this graph is called the *transition matrix* of the Markov chain.

Quantum random walks perform analogous operations. The elements of $\mathcal{S}$ are encoded into pairwise orthogonal quantum states. A setup circuit produces an initial superposition of these states. A check circuit applies a phase to marked elements. An additional *diffusion* circuit amplifies the probability of success. It uses a walk circuit, which samples a new element of $\mathcal{S}$.

Grover's algorithm is a quantum walk with uniform transition probabilities. It finds a marked element after $\Theta(\sqrt{|\mathcal{S}| \,/\, |\mathcal{M}|})$ check steps. Szegedy's algorithm can decide whether or not $\mathcal{M}$ is empty for a larger class of Markov chains [36]. Magniez, Nayak, Roland, and Santha (MNRS) generalize Szegedy's algorithm to admit even more general Markov chains [31]. They also describe a routine that can find a marked element [31, "Tolerant RAA" algorithm]. We will not describe these algorithms in detail; we will only describe the subroutines that applications of quantum walks must implement. We do not present these in full generality.

Quantum walk subroutines.

Szegedy- and MNRS-style quantum walks use circuits for the following transformations. The values u and v are elements of $\mathcal{S}$, and $\mathcal{M}$ is the subset of marked elements. The values p_{vu} are matrix entries of the transition matrix of a Markov chain P. We assume $p_{vu} = p_{uv}$, and that the corresponding graph is connected.

$$\textbf{Set-up:}\, |0\cdots 0\rangle \mapsto \frac{1}{\sqrt{|\mathcal{S}|}} \sum_{u\in\mathcal{S}} |u\rangle\, |0\rangle\,. \tag{3}$$

$$\textbf{Check:}\, |u\rangle\, |v\rangle \mapsto \begin{cases} -\,|u\rangle\, |v\rangle & \text{if } u \in \mathcal{M}, \\ |u\rangle\, |v\rangle & \text{otherwise.} \end{cases} \tag{4}$$

$$\textbf{Update:}\, |u\rangle\, |0\rangle \mapsto \sum_{u\in\mathcal{S}} \sqrt{p_{vu}}\, |u\rangle\, |v\rangle \tag{5}$$

$$\textbf{Reflect:}\, |u\rangle\, |v\rangle \mapsto \begin{cases} |u\rangle\, |v\rangle & \text{if } v = 0, \\ -\,|u\rangle\, |v\rangle & \text{otherwise.} \end{cases} \tag{6}$$

The walk step applies $(\text{Update})^{-1}(\text{Reflect})(\text{Update})$. After this, it swaps $|u\rangle$ and $|v\rangle$, repeats $(\text{Update})^{-1}(\text{Reflect})(\text{Update})$, then swaps the vertices back.

Following MNRS, we write S for the cost of the Set-up circuit, U for the cost of the Update and C for the cost of the check. The reflection cost is insignificant in our applications. The cost of a quantum walk also depends on the fraction of marked elements, $\epsilon = |\mathcal{M}|/|\mathcal{S}|$, and the spectral gap of P. With our assumptions, the spectral gap is $\delta(P) = 1 - |\lambda_2(P)|$ where $\lambda_2(P)$ is the second largest eigenvalue of P, in absolute value.

Szegedy's algorithm repeats the check and walk steps for $O(1/\sqrt{\epsilon\delta})$ iterations. MNRS uses $O(1/\sqrt{\epsilon\delta})$ iterations of the walk step, but then only $O(1/\sqrt{\epsilon})$ iterations of the check step. MNRS also uses $O(\log(1/\epsilon\delta))$ ancilla qubits. Cost 5 shows the costs of both algorithms.

Cost 5 Quantum Random Walks. The tuples S, C, and U are the costs of random walk subroutines, ϵ is the fraction of marked vertices, and δ is the spectral gap of the underlying transition matrix.

	Szegedy	**MNRS**
Gates:	$O\left(\mathsf{S}_G + \frac{1}{\sqrt{\epsilon\delta}}\left(\mathsf{U}_G + \mathsf{C}_G\right)\right)$	$O\left(\mathsf{S}_G + \frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U}_G + \mathsf{C}_G\right)\right)$
Depth:	$O\left(\mathsf{S}_D + \frac{1}{\sqrt{\epsilon\delta}}\left(\mathsf{U}_D + \mathsf{C}_D\right)\right)$	$O\left(\mathsf{S}_D + \frac{1}{\sqrt{\epsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U}_D + \mathsf{C}_D\right)\right)$
Width:	$O\left(\max\{\mathsf{S}_W, \mathsf{U}_W, \mathsf{C}_W\}\right)$	$O\left(\max\{\mathsf{S}_W, \mathsf{U}_W + \log\left(\frac{1}{\epsilon\delta}\right), \mathsf{C}_W + \log\left(\frac{1}{\epsilon\delta}\right)\}\right)$

5.2 The claw-finding problem

We will now consider a quantum walk algorithm with significant cryptanalytic applications. The claw-finding problem is defined as follows.

Problem 5.1 (Claw Finding). *Given finite sets $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ and functions $f : \mathcal{X} \to \mathcal{Z}$ and $g : \mathcal{Y} \to \mathcal{Z}$ find $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ such that $f(x) = g(y)$.*

In a so-called *golden* claw-finding problem the pair (x, y) is unique.

Tani applied Szegedy's algorithm to solve the decisional version of the claw-finding problem (detecting the presence of a claw) [38]. He then applied a binary search strategy to solve the search problem. As noted in [38], the MNRS algorithm can solve the claw-finding problem directly. The core idea is the same in either case. Parallel walks are taken on Johnson graphs $J(X, R_f)$ and $J(Y, R_g)$, and the checking step looks for claws.

There are a few details to address. First, since the claw property is defined in terms of the set $\mathcal{Z}$, we will need to augment the base sets with additional data. Second, we need to formalize the notion of parallel walks. Fortunately, this does not require any new machinery. Tani's algorithm perfoms a walk on the graph product $J(X, R_f) \times J(Y, R_g)$. A graph product $G_1 \times G_2$ is a graph with vertex set $V(G_1) \times V(G_2)$ which includes an edge between (v_1, v_2) and (u_1, u_2) if and only if v_1 is adjacent to u_1 in G_1 and v_2 is adjacent to u_2 in G_2. Our random replacement routine adds self-loops to both Johnson graphs.

5.3 Tracking claws between a pair of Johnson vertices.

In order to track claws we will store Johnson vertices over the base sets $\mathcal{X}_f = \{(x, f(x)) : x \in \mathcal{X}\}$ and $\mathcal{Y}_g = \{(y, g(y)) : y \in \mathcal{Y}\}$. Alongside each pair of Johnson vertices for $\mathcal{U} \subset \mathcal{X}_f$ and $\mathcal{V} \subset \mathcal{Y}_g$, we will store a counter for the total number of claws between $\mathcal{U}$ and $\mathcal{V}$.

This counter can be maintained using the relationship counting routine of Section 4. Before a guaranteed insertion of $(x, f(x))$ into $\mathcal{U}$ we count the number of $(y, g(y))$ in $\mathcal{V}$ with $f(x) = g(y)$. Evaluating the relation costs no more than equality testing and so the full relation counting procedure uses $O(R_g m)$ gates in depth $O(\log m + \log^2 R_g)$. Assuming that $R_f \approx R_g$, counting claws before insertion into $\mathcal{U}$ is the dominant cost. We maintain the claw counter when deleting from $\mathcal{U}$, inserting into $\mathcal{V}$, and deleting from $\mathcal{V}$.

5.4 Analysis of Tani's claw-finding algorithm

We will make a few assumptions in the interest of brevity. We assume that elements of $\mathcal{X}_f$ and $\mathcal{Y}_g$ have the same bit-length m. We write $X = |\mathcal{X}|$, $Y = |\mathcal{Y}|$, and $R = \max\{R_f, R_g\}$. We also assume that the circuits for f and g are identical; we write E_G, E_D, and E_W for the gates, depth, and width of either.

In Tani's algorithm a single graph vertex is represented by two Johnson vertex data structures. Szegedy's algorithm and MNRS store a pair of adjacent graph vertices, so here we are working with two pairs of adjacent Johnson vertices $\mathcal{U}_{\mathcal{X}} \sim \mathcal{V}_{\mathcal{X}}$ and $\mathcal{U}_{\mathcal{Y}} \sim \mathcal{V}_{\mathcal{Y}}$. The main subroutines are as follows.

Set-up.

The Johnson vertices $\mathcal{U}_\mathcal{X}$ and $\mathcal{U}_\mathcal{Y}$ are populated by sampling R elements of $\mathcal{X}$ and inserting these while maintaining the claw counter. We defer the full cost as it is essentially $O(R)$ times the update cost.

Update.

The update step applies the random replacement of Section 4.3 to each of the Johnson vertices. The insertions and deletions within the replacement routine must maintain the claw counter, so relation counting is the dominant cost of either. Replacement has a cost of $O(1)$ guaranteed insertion/deletions (from the larger of the two sets) and $O(1)$ function evaluations. Based on Cost 3 and the cost of evaluating f, the entire procedure uses $O(Rm + \mathsf{E}_G)$ gates in a circuit of depth $O(\log m + \log^2 R + \mathsf{E}_D)$ and width $O(Rm + \mathsf{E}_W)$.

Check.

A phase is applied if the claw-counter is non-zero, with negligible cost.

Walk parameters.

Let P be the transition matrix for a random walk on $J(X, R_f)$, formed by normalizing the adjacency matrix. The second largest eigenvalue of P is $\lambda_2 = O(1 - \frac{1}{R_f})$, and is positive. Our update step introduces self-loops with probability R/X into the random walk. The transition matrix with self-loops is $P' = \frac{R}{X}I + (1 - \frac{R}{X})P$. The second-largest eigenvalue of P' is $\lambda'_2 = \frac{R}{X} + (1 - \frac{R}{X})\lambda_2$. Since λ_2 is positive, the spectral gap of the walk with self-loops is $\delta'_f = 1 - |\lambda'_2| = \Omega\left(\frac{1}{R_f} - \frac{1}{X}\right)$. In general, the spectral gap of a random walk on $G_1 \times G_2$ is the minimum of the spectral gap of a walk on G_1 or G_2. Thus the spectral gap of our random walk on $J(X, R_f) \times J(Y, R_g)$ is

$$\delta = \Omega\left(\frac{1}{R} - \frac{1}{X}\right).$$

The marked elements are vertices $(\mathcal{U}_\mathcal{X}, \mathcal{U}_\mathcal{Y})$ that contain a claw. In the worst case there is one claw between the functions and

$$\epsilon = \frac{R_f R_g}{XY}.$$

The walk step will then be applied $1/\sqrt{\epsilon\delta} \geq \sqrt{XY/R}$ times.

In Cost 6 we assume $R \leq (XY)^{1/3}$. This is because the query-optimal parameterization of Tani's algorithm uses $R \approx (XY)^{1/3}$ [38], and the set-up routine dominates the cost of the algorithm when $R > (XY)^{1/3}$. The optimal values of R for the G- and DW-cost will typically be much smaller than $(XY)^{1/3}$. The G-cost is minimized when $R = \mathsf{E}_G/m$, and the DW-cost is minimized when $R = \mathsf{E}_W/m$.

Cost 6 Claw-finding using Tani's algorithm with $|\mathcal{X}_f| = X$; $|\mathcal{Y}_g| = Y$; $R = \max\{R_f, R_g\} \leq (XY)^{1/3}$; m large enough to encode an element of $\mathcal{X}_f$ or $\mathcal{Y}_g$; and E_G, E_D, and E_W the gates, depth, and width of a circuit to evaluate f or g.

- **Gates:** $O\left(m\sqrt{XYR} + \mathsf{E}_G\sqrt{\frac{XY}{R}}\right)$
- **Depth:** $O\left(\log m\sqrt{\frac{XY}{R}} + \log^2 R\sqrt{\frac{XY}{R}} + \mathsf{E}_D\sqrt{\frac{XY}{R}}\right)$
- **Width:** $O\left(Rm + \mathsf{E}_W\right)$

5.5 Comparison with Grover's Algorithm

Cost 7 gives the costs of Grover's algorithm applied to claw-finding. It requires $O(\sqrt{XY})$ Grover iterations. Each iteration evaluates f and g, and we assume this is the dominant cost of each iteration. Note that the cost is essentially that of Tani's algorithm with $R = 1$.

Grover's and Tani's algorithms have the same square root relationship to XY. Tani's algorithm can achieve a slightly lower cost when the functions f and g are expensive.

Cost 7 Claw-finding using Grover's algorithm with the notation of Cost 6.

Gates: $O\left(\mathsf{E}_G\sqrt{XY}\right)$
Depth: $O\left(\mathsf{E}_D\sqrt{XY}\right)$
Width: $O\left(\mathsf{E}_W\right)$

5.6 Effect of parallelism

The naive method to parallelise either algorithm over P processors is to divide the search space into P subsets, one for each processor. For both algorithms, parallelising will reduce the depth and gate cost for *each* processor by $1/\sqrt{P}$. Accounting for costs across all P processors shows that parallelism increases the total cost of either algorithm by a factor of $\sqrt{P}$. This is true in both the G- and the DW-cost metric. This is optimal for Grover's algorithm [43], but may not be optimal for Tani's algorithm. The parallelisation strategy of Jeffery et al. [23] is better, but uses substantial communication between processors in the check step. A detailed cost analysis would need to account for the physical geometry of the processors, which we leave for future work.

6 Application: Cryptanalysis of SIKE

The Supersingular Isogeny Key Encapsulation (SIKE) scheme [20] is based on Jao and de Feo's Supersingular Isogeny Diffie–Helman (SIDH) protocol [21]. In this section we describe the G- and DW-costs of an attack on SIKE. Our analysis can be applied to SIDH as well.

SIKE has public parameters p and E where p is a prime of the form $2^{e_A}3^{e_B} - 1$ and E is a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$. Typically e_A and e_B are chosen so that $2^{e_A} \approx 3^{e_B}$; we will assume this is the case. For each prime $\ell \neq p$, one can associate a graph, *the ℓ-isogeny graph*, to the set of supersingular elliptic curves defined over $\mathbb{F}_{p^2}$. This graph has approximately $p/12$ vertices. Each vertex represents an equivalence class of elliptic curves with the same *j-invariant*. Edges between vertices represent *degree-ℓ isogenies* between the corresponding curves[7]. A SIKE public key is a curve E_A, and a private key is a path of length e_A that connects E and E_A in the 2-isogeny graph; only one path of this length is expected to exist.

The ℓ-isogeny graph is $(\ell + 1)$-regular. So the set of paths of length c that start at some fixed vertex in the 2-isogeny graph is of size $3 \cdot 2^{c-1}$. This suggests the following golden claw-finding problem. Let $\mathcal{X}$ be the set of paths of length $\lfloor e_A/2 \rfloor$ that start at E, and let $\mathcal{Y}$ be the set of paths of length $\lceil e_A/2 \rceil$ that start at E_A. Let $f : \mathcal{X} \to \mathbb{F}_{p^2}$ and $g : \mathcal{Y} \to \mathbb{F}_{p^2}$ be functions that compute the j-invariant corresponding to the curve reached by a path. Recovering the private key corresponding to E_A is no more difficult

[7] We are being slightly imprecise, as the ℓ-isogeny graph is actually directed. However, if there is an edge from u to v corresponding to an isogeny ϕ, then there is an edge from v to u corresponding to the dual isogeny $\hat{\phi}$.

than finding a claw between f and g. With the typical parameterisation of $2^{e_A} \approx 3^{e_B}$, both $\mathcal{X}$ and $\mathcal{Y}$ are of size approximately $p^{1/4}$.

We will fix these definitions of $\mathcal{X}$, $\mathcal{Y}$, f, and g for the remainder. We will also assume that E_G, E_D, and E_W — the gates, depth, and width of a circuit for evaluating f or g — are all $p^{o(1)}$.

6.1 Quantum claw-finding attacks

Let us first consider a parallel Grover search with P quantum processors using the parallelisation strategy of Section 5.6. Processor i performs a Grover search on $\mathcal{X}_i \times \mathcal{Y}_i$ where $\mathcal{X}_i$ is a subset of $\mathcal{X}$ of size $p^{1/4}/\sqrt{P}$, and $\mathcal{Y}_i$ is a subset of $\mathcal{Y}$ of size $p^{1/4}/\sqrt{P}$. Based on Cost 7 the circuit for all P processors uses $p^{1/4+o(1)}\sqrt{P}$ gates, has depth $p^{1/4+o(1)}/\sqrt{P}$, and has width $p^{o(1)}P$. The only benefit to using more than 1 processor is a reduction in depth. The G- and the DW-cost both increase with P.

Tani's algorithm admits time vs. memory trade-offs using both the Johnson graph parameter R and the number of parallel instances P. With any number of instances, both the G- and the DW-cost are minimized when $R = p^{o(1)}$. Based on Cost 6 the circuit for P processors uses $p^{1/4+o(1)}\sqrt{P}$ gates, has depth $p^{1/4+o(1)}/\sqrt{P}$, and has width $p^{o(1)}P$. This is identical to Grover search up to the $p^{o(1)}$ factors. However, there may be a benefit to using $R > 1$ if function evaluations are sufficiently expensive.

6.2 Classical claw-finding attacks

In a recent analysis of SIKE, Adj et al. [1] conclude that the best known classical claw-finding attack on the scheme is based on the van Oorschot–Wiener (VW) parallel collision search algorithm. We defer to [1] for a full description of the attack. The VW method uses a PRAM with P processors and M registers. Each register must be large enough to store an element of $\mathcal{X}$ or $\mathcal{Y}$ and a small amount of additional information.

From [1], a claw-finding attack on SIKE using VW on a PRAM with 1 processor and M registers of memory performs

$$\max\left\{\frac{p^{3/8+o(1)}}{M^{1/2}},\ p^{1/4+o(1)}\right\} \tag{7}$$

RAM operations. The $o(1)$ term hides the cost of evaluating f, g, and a hash function. The algorithm parallelizes perfectly so long as $P < M \leq p^{1/4}$. This restriction is to avoid a backlog of operations on the shared memory. The algorithm performs $p^{1/4+o(1)}$ shared memory operations in total, and $M^{1/2}P/p^{1/8+o(1)}$ shared memory operations simultaneously. Using memory $M > p^{1/4+o(1)}$ does not reduce the total number of shared memory operations, hence the second term in Equation 7.

It is natural to treat the P processors in this attack as a memory peripheral controller for M registers of non-volatile memory. Each processor needs an additional $p^{o(1)}$ bits of memory for its internal state, and each of the M registers are of size $p^{o(1)}$. Unlike the quantum claw-finding attacks that we have considered, the RAM operation cost of the VW method decreases as the amount of available hardware increases.

The query-optimal parameterisation of Tani's algorithm has $p^{1/6+o(1)}$ qubits of memory. In our models this implies $p^{1/6+o(1)}$ classical processors for control with a combined $p^{1/6+o(1)}$ bits of classical memory. A RAM operation for these processors is equivalent to a quantum gate in cost and time. Repurposed to run VW, these processors would solve the claw-finding problem in time $p^{1/8+o(1)}$ with $p^{7/24+o(1)}$ RAM operations. Our conclusion is that an adversary with enough quantum memory to run Tani's algorithm with the query-optimal parameters could break SIKE faster by using the classical control hardware to run van Oorschot–Wiener.

6.3 Non-asymptotic cost estimates

The claw-finding attacks that we have described above can all break SIKE in $p^{1/4+o(1)}$ RAM operations. However, they achieve this complexity using different amounts of time and memory. Both quantum attacks achieve their minimal cost in time $p^{1/4+o(1)}$ on a machine with $p^{o(1)}$ qubits. The van Oorschot–Wiener method achieves its minimal cost in time $p^{o(1)}$ on a machine with $p^{1/4+o(1)}$ memory and processors. A more thorough accounting of the low order terms could identify the attack (and parameterization) of least cost, but real attackers have resource constraints that might make this irrelevant.

We use SIKE-n to denote a parameterisation of SIKE using an n-bit prime. We focus on SIKE-434 and SIKE-610, parameters introduced as alternatives to the original submission to NIST [1]. Figure 3 depicts the attack landscape for SIKE-434. Figure 4 gives the cost of breaking SIKE-434 and SIKE-610 under various constraints. These cost estimates are based on assumptions that we describe below.

Cost of function evaluations

The functions f and g involve computing isogenies of (2-smooth) degree approximately $p^{1/4}$. We assume that the cost of evaluating f is equal to the cost of evaluating g, and we let E_G, E_D, and E_W denote the gate-count, depth, and width of a circuit for either.

We assume that the classical and quantum gate counts are equal, which may lead us to underestimate the quantum cost.

The SIKE specification describes a method for computing a degree-2^e isogeny that uses approximately $e \log e$ curve operations [20]. Each operation is either a point doubling or a degree-2 isogeny evaluation. We assume that it costs the attacker at least $4 \log p \log \log p$ gates to compute either curve operation. This is a very conservative estimate given that both operations involve multiplication in $\mathbb{F}_{p^2}$, and a single multiplication in $\mathbb{F}_{p^2}$ involves 3 multiplications in $\mathbb{F}_p$. Based on this, we assume that computing an isogeny of degree $\approx p^{1/4}$ costs the attacker at least $(\log p)^2 \left((\log \log p)^2 - 2 \log \log p\right)$ gates. We assume that the attacker's circuit has width $2 \log p$, which is just enough space to represent its output. We assume that the gates parallelize perfectly so that $\mathsf{E}_D = \mathsf{E}_G / \mathsf{E}_W$.

For an attack on SIKE-434 our assumptions give $\mathsf{E}_G = 2^{23.4}$, $\mathsf{E}_D = 2^{13.7}$, and $\mathsf{E}_W = 2^{9.8}$. For an attack on SIKE-610, they give $\mathsf{E}_G = 2^{24.6}$, $\mathsf{E}_D = 2^{14.3}$, and $\mathsf{E}_W = 2^{10.3}$. We assume that elements of $\mathcal{X}_f$ and $\mathcal{Y}_g$ can be represented in $m = (\log p)/2$ bits.

Grover

Each Grover iteration computes two function evaluations. However, to avoid the issue of whether these evaluations are done in parallel or in series, we only cost a single evaluation. We ignore the cost of the diffusion operator. We partition the search space into P parts and distribute the subproblems to P processors. Each processor performs approximately $p^{1/4}/\sqrt{P}$ Grover iterations. This gives a total gate count of at least $p^{1/4}\sqrt{P}\mathsf{E}_G$, depth of at least $p^{1/4}\mathsf{E}_D/\sqrt{P}$, and width of at least $P\mathsf{E}_W$.

For depth-constrained computations we use the smallest P that is compatible with the constraint. For memory-constrained computations we take P large enough to use all of the available memory.

Tani

A single instance of Tani's algorithm stores two lists of size R and needs scratch space for computing two function evaluations. We only cost a single function evaluation. We assume that only $2Rm + \mathsf{E}_W$ qubits are needed.

We parallelise the gate-optimal parameterisation, i.e. we take $R = \mathsf{E}_G/m$. We partition the search space into P parts and distribute subproblems to P processors. Each processor performs roughly $p^{1/4}/\sqrt{RP}$ walk iterations. Each walk iteration performs at least one guaranteed insertion with claw-tracking and at least one function evaluation. Each insertion costs at least Rm gates. Each function evaluation has depth E_D and width E_W. The total gate cost across all P processors is at least $p^{1/4}\sqrt{P/R}(Rm+\mathsf{E}_G) = p^{1/4}\sqrt{2m\mathsf{E}_G P}$ gates in depth at least $p^{1/4}\mathsf{E}_D/\sqrt{RP} = p^{1/4}\mathsf{E}_D\sqrt{m/P\mathsf{E}_G}$ and uses width at least $P(2Rm + \mathsf{E}_W) = P(2\mathsf{E}_G + \mathsf{E}_W)$.

For depth-constrained computations we use the smallest P that is compatible with the constraint. For memory-constrained computations we take P large enough to use all of the available memory. If the parallelisation is such that $R = \mathsf{E}_G/m \geq (p^{1/2}/P)^{1/3}$, which would cause the setup cost to exceed the cost of the walk iteration, we decrease R.

van Oorschot–Wiener

Each processor iterates a cycle of computing a function evaluation and storing the result. We only cost a single function evaluation per iteration. Our quantum machine models assume a number of RAM controllers that is proportional to memory. We make the same assumption here. When the attacker has M bits of memory we assume they also have $P = M/(\mathsf{E}_W + m)$ processors. Intuitively, each processor needs space to evaluate a function and is responsible for one unit of shared memory. This gives a total gate count of at least $(p^{3/8}/M^{1/2})\mathsf{E}_G$, a depth of at least $(p^{3/8}/M^{3/2})(\mathsf{E}_W + m)\mathsf{E}_D$, and a width of M.

For depth constrained-computations we use the smallest amount of memory that satisfies the constraint. Unlike the quantum attacks, the gate cost of VW decreases with memory use, so Tables 4a and 4b do not show the best gate count that VW can achieve with a depth constraint. For memory-constrained computations we use the maximum amount of memory allowed.

7 Conclusions and future work

7.1 Impact of our work on the NIST security level of SIKE

The SIKE submission recommends SIKE-503, SIKE-751, and SIKE-964 for security matching AES-128, AES-192, and AES-256, respectively. NIST suggests that an attack on AES-128 costs 2^{143} classical gates (in a non-local boolean circuit model). NIST also suggests that attacks on AES-192 and AES-256 cost 2^{207} and 2^{272} classical gates, respectively. We have used "RAM operations" throughout to refer to non-local bit/qubit operations; our G-cost is directly comparable with these estimates.

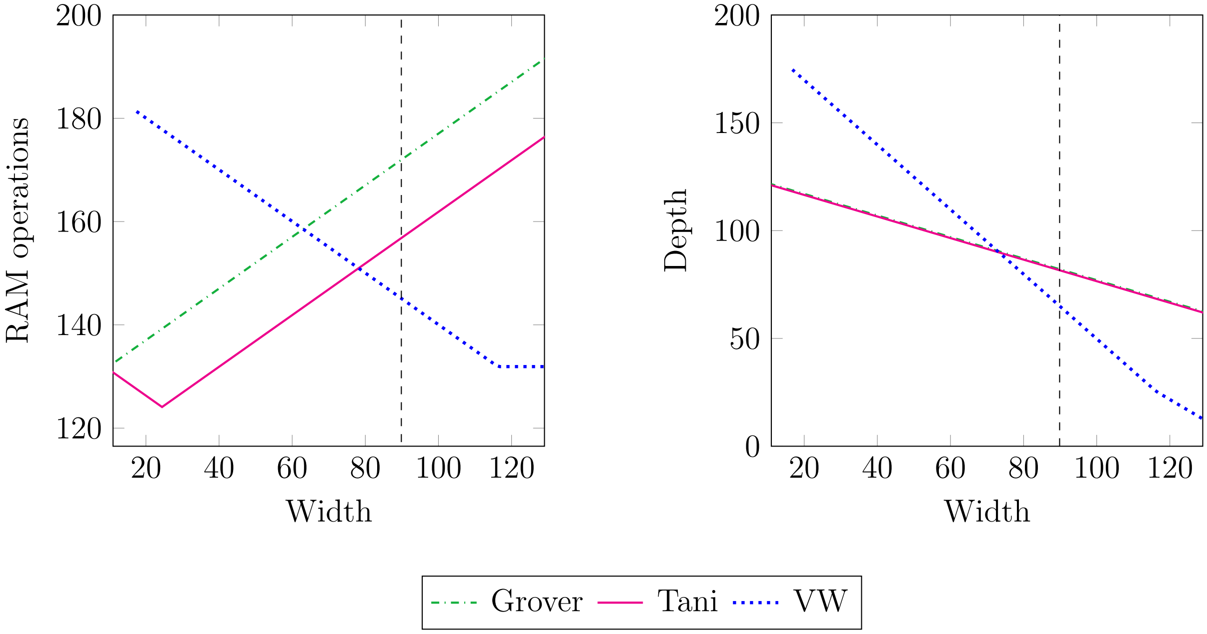


Figure 3: G-cost and depth of claw-finding attacks on SIKE-434, with the isogeny costs of Section 6.3. The vertical dashed lines are at the width of the query-optimal parameterisation including storage, $(p^{1/6}\log p)/2$. Axes are in base-2 logarithms.

Adj et al. [1] recommend slightly smaller primes: SIKE-434 for security matching AES-128 and SIKE-610 for security matching AES-192. Their analysis is based on the cost of van Oorschot–Wiener with less than 2^{80} registers of memory. NIST's recommended machine model does not impose a limit on classical memory, but it does impose a limit on the depth of quantum circuits. Our cost estimates (Figure 4) suggests that known quantum attacks do not break SIKE-434 using less than 2^{143} classical gates, or SIKE-610 using less than 2^{207} classical gates, when depth is limited to 2^{96}. We agree with the conclusions of Adj et al., and believe that NIST's machine model should include a width constraint.

We caution that claw-finding attacks may not be optimal. Biasse, Jao, and Sankar [8] present a quantum attack that exploits the algebraic structure of supersingular curves defined over $\mathbb{F}_p$. This attack uses $p^{1/4+o(1)}$ quantum gates and $2^{O(\sqrt{\log p})}$ qubits of memory. Given our analysis of Tani's algorithm, this attack may be competitive with other quantum attacks.

7.2 Further applications of our memory peripherals

Our analysis should be immediately applicable to other cryptanalytic algorithms that use quantum walks on Johnson graphs. These include algorithms for subset sum [7], information set decoding [25], and quantum Merkle puzzles [10].

The G- and DW-cost metrics have applications to classical algorithms that use quantum subroutines, such as the quantum number field sieve [6], and to quantum algorithms that use classical subroutines, such as Shor's algorithm.

Our analysis of quantum random access might affect memory-intensive algorithms like quantum lattice sieving [29]. However, we only looked at quantum access to quantum memory. There may be physically realistic memory peripherals that enable inexpensive quantum access to classical memory (e.g. [19]).

7.3 Geometrically local memory peripherals

Neither of our memory peripheral models account for communication costs. We allow non-local quantum communication in the form of long-range CNOT gates. We allow non-local classical communication in the controllers. The distributed computing model of Beals et al. [5] might serve as a useful guide for eliminating non-local quantum communication. Note that the resulting circuits are, at present, only compatible with the DW-cost metric. The known self-correcting qubit memories are built out of physical qubit interactions that cannot be implemented locally in 3 dimensional space.

Bibliography

[1] G. Adj, D. Cervantes-Vázquez, J.-J. Chi-Domínguez, A. Menezes, and F. Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. In *Selected Areas in Cryptography – SAC 2018*, pages 322–343. LNCS 11349.

[2] R. Alicki, M. Fannes, and M. Horodecki. On thermalization in Kitaev's 2d model. *J. Physics A*, 42:065303, 2009.

[3] R. Alicki, M. Horodecki, P. Horodecki, and R. Horodecki. On thermal stability of topological qubit in Kitaev's 4d model. *Open Systems & Information Dynamics*, 17:1–20, 2010.

[4] A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM J. Computing*, 37:210–239, 2007.

[5] R. Beals, S. Brierley, O. Gray, A. W. Harrow, S. Kutin, N. Linden, D. Shepherd, and M. Stather. Efficient distributed quantum computing. *Proc. Royal Soc. London A: Mathematical, Physical and Engineering Sciences*, 469, 2013.

[6] D. J. Bernstein, J.-F. Biasse, and M. Mosca. A low-resource quantum factoring algorithm. In *Post-Quantum Cryptography – PQCrypto 2017*, pages 330–346. LNCS 10346.

[7] D. J. Bernstein, S. Jeffery, T. Lange, and A. Meurer. Quantum algorithms for the subset-sum problem. In *Post-Quantum Cryptography – PQCrypto 2013*, pages 16–33. LNCS 7932.

[8] J.-F. Biasse, D. Jao, and A. Sankar. A quantum algorithm for computing isogenies between supersingular elliptic curves. In *Progress in Cryptology – INDOCRYPT 2014*, pages 428–442. LNCS 8885.

[9] A. Blais, R.-S. Huang, A. Wallraff, S. M. Girvin, and R. J. Schoelkopf. Cavity quantum electrodynamics for superconducting electrical circuits: An architecture for quantum computation. *Physical Rev. A*, 69, 2004.

[10] G. Brassard, P. Høyer, K. Kalach, M. Kaplan, S. Laplante, and L. Salvail. Merkle puzzles in a quantum world. In *Advances in Cryptology – CRYPTO 2011*, pages 391–410.

[11] S. Bravyi and B. Terhal. A no-go theorem for a two-dimensional self-correcting quantum memory based on stabilizer codes. *New J. Physics*, 11, 2009.

[12] B. J. Brown, D. Loss, J. K. Pachos, C. N. Self, and J. R. Wootton. Quantum memories at finite temperature. *Rev. Modern Physics*, 88, Nov 2016.

[13] B. Coecke, T. Fritz, and R. W Spekkens. A mathematical theory of resources. *Information and Computation*, 250:59–86, 2016.

[14] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill. Topological quantum memory. *J. Mathematical Physics*, 43, 2002.

[15] D. E. Deutsch. Quantum computational networks. *Proc. R. Soc. Lond. A*, 425:73–90, 1989.

[16] R. P Feynman. Quantum mechanical computers. *Foundations of Physics*, 16:507–531, 1986.

[17] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Rev. A*, 86, Sep 2012.

[18] A. G. Fowler, A. C. Whiteside, and L. C. L. Hollenberg. Towards practical classical processing for the surface code. *Physical Rev. letters*, 108, 2012.

[19] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Architectures for a quantum random access memory. *Physical Rev. A*, 78, Nov 2008.

[20] D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. De Feo, B. Hess, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes, V. Soukharev, and D. Urbanik. Supersingular isogeny key encapsulation. *Submission to NIST post-quantum project*, November 2017. available at https://sike.org/#nist-submission.

[21] D. Jao and L. De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Post-Quantum Cryptography – PQCrypto 2011*, pages 19–34. LNCS 7071.

[22] S. Jeffery. *Frameworks for Quantum Algorithms*. PhD thesis, University of Waterloo, 2014.

[23] S. Jeffery, F. Magniez, and R. De Wolf. Optimal parallel quantum query algorithms. *Algorithmica*, 79(2):509–529, Oct 2017.

[24] S. P Jordan. Fast quantum computation at arbitrarily low energy. *Physical Rev. A*, 95, 2017.

[25] G. Kachigar and J.-P. Tillich. Quantum information set decoding algorithms. In *Post-Quantum Cryptography – PQCrypto 2017*, LNCS 10346, pages 69–89. Springer.

[26] R. M. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machines. Technical Report UCB/CSD-88-408, EECS Department, University of California, Berkeley, Mar 1988.

[27] A. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303:2–30, 2003.

[28] A. Kitaev, A. Shen, M. N. Vyalyi, and M. N. Vyalyi. *Classical and quantum computation*. Number 47. American Mathematical Soc., 2002.

[29] T. Laarhoven, M. Mosca, and J. van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77:375–400, Dec 2015.

[30] F. Le Gall and S. Nakajima. Quantum algorithm for triangle finding in sparse graphs. *Algorithmica*, 79:941–959, Nov 2017.

[31] F. Magniez, A. Nayak, J. Roland, and M. Santha. Search via quantum walk. *SIAM Journal on Computing*, 40:142–164, 2011.

[32] R. McDermott, M. G. Vavilov, B. L. T. Plourde, F. K. Wilhelm, P. J. Liebermann, O. A. Mukhanov, and T. A. Ohki. Quantum–classical interface based on single flux quantum digital logic. *Quantum science and technology*, 3, 2018.

[33] C. Moore. Quantum circuits: Fanout, parity, and counting. arXiv preprint, 1999. available at https://arxiv.org/abs/quant-ph/9903046.

[34] National Institute of Standards and Technology. Submission requirements and evaluation criteria or the post-quantum cryptography standardization process. 2017. available at https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[35] R. Peierls. On Ising's model of ferromagnetism. In *Mathematical Proc. Cambridge Philosophical Society*, volume 32, pages 477–481. Cambridge University Press, 1936.

[36] M. Szegedy. Quantum speed-up of markov chain based algorithms. In *2004 IEEE Symposium on Foundations of Computer Science*, pages 32–41, Oct.

[37] Y. Takahashi, S. Tani, and N. Kunihiro. Quantum addition circuits and unbounded fan-out. *Quantum Info. Comput.*, 10:872–890, September 2010.

[38] S. Tani. An improved claw finding algorithm using quantum walk. In *Mathematical Foundations of Computer Science – MFCS 2007*, pages 548–558. LNCS 4708.

[39] B. M. Terhal. Quantum error correction for quantum memories. *Rev. Modern Physics*, 87:307, 2015.

[40] H. Thapliyal, N. Ranganathan, and R. Ferreira. Design of a comparator tree based on reversible logic. In *2010 IEEE International Conference on Nanotechnology*, pages 1113–1116.

[41] A. Wang and W. D. Woo. Static magnetic storage and delay line. *J. Applied Physics*, 21:49–54, 1950.

[42] G. Wendin. Quantum information processing with superconducting circuits: a review. *Reports on Progress in Physics*, 80, 2017.

[43] C. Zalka. Grover's quantum searching algorithm is optimal. *Physical Rev. A*, 60, Oct 1999.

	SIKE-434			SIKE-610		
Attack	G	D	W	G	D	W
Grover	190	64	127	280	64	216
Tani	175	63	126	264	64	216
VW	145	64	91	189	63	136

(a) `MAXDEPTH` $= 2^{64}$

	SIKE-434			SIKE-610		
Attack	G	D	W	G	D	W
Grover	158	96	63	248	96	152
Tani	143	95	62	232	96	152
VW	155	95	70	200	95	115

(b) `MAXDEPTH` $= 2^{96}$

	SIKE-434			SIKE-610		
Attack	G	D	W	G	D	W
Grover	159	95	64	204	140	64
Tani	144	94	64	188	140	64
VW	158	104	64	225	172	64

(c) `MAXMEMORY` $= 2^{64}$

	SIKE-434			SIKE-610		
Attack	G	D	W	G	D	W
Grover	175	79	96	220	124	96
Tani	160	78	96	204	124	96
VW	142	56	96	209	124	96

(d) `MAXMEMORY` $= 2^{96}$

	SIKE-434			SIKE-610		
Attack	G	D	W	G	D	W
Grover	132	122	10	177	167	10
Tani	124	114	25	169	159	25
VW	132	14	128	177	14	173

(e) G-cost optimal

	SIKE-434			SIKE-610		
Attack	G	D	W	G	D	W
Grover	132	122	10	177	167	10
Tani	131	122	10	177	166	10
VW	132	14	128	177	14	173

(f) DW-cost optimal

Figure 4: Cost estimates for claw finding attacks on SIKE. All numbers are expressed as base-2 logarithms. The source code and data for this table can be found at https://jmschanck.info/code/20200701-schanck-thesis-supplemental.tar.gz. A backup has also been uploaded to the Internet Archive: https://archive.org/details/20200701-schanck-thesis-supplemental.

Chapter 3

NTRU and LWE

A galactic algorithm is an algorithm that is wonderful in its asymptotic behavior, but is never used to actually compute anything.

Richard Lipton (2010)

Context

I have been involved with the design of several lattice based cryptosystems. Current practice assigns one of those systems, ntruhrss701, a 123-bit security level. As such, my co-authors and I recommended ntruhrss701 to NIST at the lowest of their five security categories. However, I believe that ntruhrss701 is significantly more secure than this categorization suggests. The paper below represents my best effort to date to solidify this belief.

Using the Core-SVP methodology of Alkim–Ducas–Pöppelmann–Schwabe, the cost of breaking ntruhrss701 is approximated by the cost of solving the shortest vector problem in dimension ≈ 464. The cost of solving the shortest vector problem in dimension ≈ 464 is approximated by the cost of one call to the Becker–Ducas–Gama–Laarhoven (BDGL) near neighbour search algorithm. In dimension 464, current practice estimates

- the cost of BDGL as $2^{135.5}$ RAM operations;
- the cost of quantum search assisted BDGL as $2^{123.0}$ qubit operations; and
- the memory usage of both classical and quantum BDGL as $2^{101.8}$ vectors.

The memory usage is worth dwelling on. The number of bits per vector will be roughly 2^{10}. Assuming a memory composed of one petabyte micro-SD cards, a $2^{111.8}$ bit memory with a footprint the size of the five boroughs of New York City would stand over 100 meters tall. The quantum variant needs all of this memory to be stored in quantum accessible classical memory (qRAM).

When one looks closely at the memory usage of BDGL in dimension 464, one sees that the algorithm makes many small searches inside “buckets” that each contain roughly one petabyte worth of vectors. The paper below focuses on the cost of searching a single bucket. We work in a RAM model for classical computation and in a quantum circuit model for quantum computation. Our quantum circuits have unit-cost qRAM gates.

There are problems with assuming unit-cost RAM / qRAM for petabyte memories. Good programmers restructure RAM model programs to make efficient use of their (real) computer’s caches. They know that access to a register is less expensive than access to main memory, that access to main memory is less expensive than access to disk, and so on. They know that without cache-awareness a random access to an N-bit memory that is embedded in a 3-dimensional space takes $\Omega(N^{1/3})$ seconds.

Good programmers will, likewise, restructure quantum circuits to minimize gate latency on their (real) quantum computers. A qRAM gate, however, can be used to construct an arbitrary superposition over the entries of a classical memory. The process relies on quantum interference and cannot be optimized through cache-awareness. Hence, a qRAM gate has the latency of a worst-case memory access. In other words, a qRAM gate takes $\Omega(N^{1/3})$ seconds.

The problems with qRAM appear to be more severe than the problems with RAM. But the problems with RAM cannot be discounted. Both the classical and the quantum versions of the algorithms considered below must route input data (a large list of vectors) to the appropriate buckets. This process is entirely classical.

A full accounting of data movement costs alone would likely call into question the estimate of $2^{135.5}$ RAM operations for BDGL in dimension 464. Bernstein, Chungsatiansup, Lange, and van Vredendaal have made an argument to this effect in the NTRU Prime submission document.

Our results below paint a slightly different picture: We estimate the cost of BDGL in dimension 464 as $2^{163.8}$ RAM operations. We estimate the depth-width cost of quantum search assisted BDGL as $2^{152.8}$ qubit-cycles. We further estimate that the cost of quantum error correction is $2^{158.5}$ RAM operations. With our estimates, the combined cost of parallel near neighbor searches could easily dwarf the cost of routing input data to appropriate buckets.

Further work will be needed to determine whether the modest quantum cost reduction—from $2^{163.8}$ to $2^{158.5}$ RAM operations—survives the translation out of the RAM/qRAM model.

Detailed statement of contributions

I am solely responsible for Section 3. Eamonn Postlethwaite, Vlad Gheorghiu, and I made equal contributions to Section 4. Vlad Gheorghiu contributed Figure 1. Eamonn Postlethwaite and I made equal contributions to Sections 5 and 6. The appendices on the cost of generalized Toffoli gates and our choice of addition circuit, which are referenced in Section 4 were written by Eamonn Postlethwaite and Vlad Gheorghiu. All authors contributed significantly to the remainder of the paper. Martin Albrecht, Eamonn Postlethwaite, and I made equal contributions to the software.

Estimating quantum speedups for lattice sieves

Martin R. Albrecht Vlad Gheorghiu Eamonn W. Postlethwaite John M. Schanck

Abstract Quantum variants of lattice sieve algorithms are often used to assess the security of lattice based cryptographic constructions. In this work we provide a heuristic, non-asymptotic, analysis of the cost of several algorithms for near neighbour search on high dimensional spheres. These algorithms are used in lattice sieves. We design quantum circuits for near neighbour search algorithms and provide software that numerically optimises algorithm parameters according to various cost metrics. Using this software we estimate the cost of classical and quantum near neighbour search on spheres. For the most performant near neighbour search algorithm that we analyse we find a small quantum speedup in dimensions of cryptanalytic interest. Achieving this speedup requires several optimistic physical and algorithmic assumptions.

1 Introduction

Sieving algorithms for the shortest vector problem (SVP) in a lattice have received a great deal of attention recently [1, 2, 9, 19, 35, 44]. The attention mostly stems from lattice based cryptography, as many attacks on lattice based cryptographic constructions involve finding short lattice vectors [3, 40, 43].

Lattice based cryptography is thought to be secure against quantum adversaries. None of the known algorithms to solve SVP (to a small approximation factor) do so in subexponential time, but this is not to say that there is no gain to be had given a large quantum computer. Lattice sieve algorithms use near neighbour search (NNS) as a subroutine; near neighbour search algorithms use black box search as a subroutine; and Grover's quantum search algorithm [27] gives a square root improvement to the query complexity of black box search. A black box search that is expected to take $\Theta(N)$ queries on classical hardware will take $\Theta(\sqrt{N})$ queries on quantum hardware using Grover's algorithm.

Previous work has analysed the effect of quantum search on the query complexity of lattice sieves [36, 37]. Of course, one must implement the queries efficiently in order to realise the improvement in practice. Recent work has given concrete quantum resource estimates for the black box search problems involved in key recovery attacks on AES [25, 30] and preimage attacks on SHA-2 and SHA-3 [5]. In this work, we give explicit quantum circuits that implement the black box search subroutines of several quantum lattice sieves. Our quantum circuits are efficient enough to yield a cost improvement in dimensions of cryptanalytic interest. However, for the most performant sieve that we analyse the cost improvement is small and several barriers stand in the way of achieving it.

Outline and Contributions. We start with some preliminaries in Section 2. In particular, we discuss the "XOR and Population Count" operation (henceforth `popcount`), which is our primary optimisation target. The `popcount` operation is used to identify pairs of vectors that are likely to lie at a small angle to each other. It is typically less expensive than a full inner product computation.

In Section 3 we introduce and analyse a filtered quantum search procedure. We present our quantum circuit for `popcount` in Section 4. In Section 5 we provide a heuristic analysis of the probability that `popcount` successfully identifies pairs of vectors that are close to each other. This analysis may be of independent interest; previous work [2, 19] has relied largely on experimental data for choosing `popcount` parameters.

In Section 6, we rederive the overall cost of the NNS subroutines of three lattice sieves. Our cost analysis exposes the impact of the `popcount` parameters so that we can numerically optimise these in parallel with the sieve parameters. We have chosen to profile the Nguyen–Vidick sieve [44], the `bgj1` specialisation [2] of the Becker–Gama–Joux sieve [10], and the Becker–Ducas–Gama–Laarhoven sieve [9]. We have chosen these three sieves as they are, respectively, the earliest and most conceptually simple, the most performant yet implemented, and the fastest known asymptotically.

Finally, we optimise the cost of classical and quantum search under various cost metrics to produce Figures 2 and 3 of Section 7. We conclude by discussing barriers to obtaining the reported quantum advantages in NNS, the relationship between SVP and NNS, and future work. We provide the source code used to compute all data in this work in Appendix E.We consider our software a contribution in its own right; it is documented, easily extensible and allows for the inclusion of new nearest neighbour search strategies and cost models.

Interpretation. Quantum computation seems to be more difficult than classical computation. As such, there will likely be some minimal dimension, a crossover point, below which classical sieves outperform quantum ones. Our estimates give non-trivial crossover points for the sieves we consider. Yet, our results do not rule out the relevance of quantum sieves to lattice cryptanalysis. The crossover points that we estimate are well below the dimensions commonly thought to achieve 128 bits of security against quantum adversaries. However, our initial logical circuit level analysis (labelled “q: depth-width” in Figures 2 and 3) is optimistic. It ignores the costs of quantum random access memory and quantum error correction.

To illustrate the potential impact of error correction, we apply a cost model developed by Gidney and Ekerå to our quantum circuits. The Gidney–Ekerå model was developed as part of a recent analysis of Shor’s algorithm [22]. In the Gidney–Ekerå model, the crossover point for the NNS algorithm underlying the Becker–Ducas–Gama–Laarhoven sieve [9] is dimension 304. In this dimension, the classical and quantum variants both perform $2^{115.6}$ operations and need at least $2^{76.6}$ bits of (quantum accessible) random access memory. A large cost improvement is obtained asymptotically, but for cryptanalytically relevant dimensions the improvement is tenuous. Between dimensions 352 and 824 our estimate for the quantum cost grows from appoximately 2^{128} to approximately 2^{256}. In dimension 352 this is an improvement of a factor of $2^{1.3}$ over our estimate for the classical cost. In dimension 824 the improvement is by a factor of $2^{14.0}$.

We caution that a memory constraint would significantly reduce the range of cryptanalytically relevant dimensions. For instance, an adversary with no more than 2^{128} bits of quantum accessible classical memory is limited to dimension 544 and below. In these dimensions we estimate a cost improvement of no more than a factor of $2^{13.2}$ at the logical circuit level and no more than $2^{6.8}$ in the Gidney–Ekerå metric.

A depth constraint would also reduce the range of cryptanalytically relevant dimensions. The quantum algorithms that we consider would be more severely affected by a depth constraint than their classical counterparts, due to the poor parallisability of Grover’s algorithm.

2 Preliminaries

2.1 Models of computation

We describe quantum algorithms as circuits using the Clifford+T gate set, but we augment this gate set with a table lookup operation (qRAM). We describe classical algorithms as programs for RAM machines (random access memory machines).

Clifford+T+qRAM quantum circuits. Quantum circuits can be described at the *logical layer*, wherein an array of n qubits encodes a unit vector in $(\mathbb{C}^2)^{\otimes n}$, or at the *physical layer*, wherein the state space may be much larger. Ignoring qubit initialisation and measurement, a circuit is a sequence of unitary operations, one per unit time. Each unitary in the sequence is constructed by parallel composition of gates. At most one gate can be applied to each qubit per time step. The Clifford+T gate set

$$\mathbf{H} = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix}, \quad \mathbf{CNOT} = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}, \quad \mathbf{T} = \begin{pmatrix}1 & 0\\ 0 & e^{i\pi/4}\end{pmatrix},$$

is commonly used to describe circuits at the logical layer due to its relationship with some quantum error correcting codes. This gate set is universal for quantum computation when combined with qubit initialisation (of $|0\rangle$ and $|1\rangle$ states) and measurement in the computational basis.

In addition to Clifford+T gates, we allow unit cost table lookups in the form of qRAM (quantum access to classical RAM). The difference between RAM and qRAM is that qRAM can construct arbitrary superpositions of table entries. Suppose that $(R_0, \ldots, R_{2^n-1})$ are registers of a classical RAM and that each register encodes an ℓ bit binary string. We allow our Clifford+T circuits access to these registers in the form of an $(n+\ell)$ qubit qRAM gate that enacts

$$\sum_{j=0}^{2^n-1} \alpha_j |j\rangle |x\rangle \overset{qRAM}{\longrightarrow} \sum_{j=0}^{2^n-1} \alpha_j |j\rangle |x \oplus R_j\rangle . \tag{1}$$

Here $\sum_j \alpha_j |j\rangle$ is a superposition of addresses and x is an arbitrary ℓ bit string.

Quantum access to classical RAM is a powerful resource, and the algorithms we describe below fail to achieve an advantage over their classical counterparts when qRAM is not available. We discuss qRAM at greater length in Section 7.

RAM machines. We describe classical algorithms in terms of random access memory machines. For comparability with the Clifford+T gate set, we will work with a limited instruction set, e.g. {NOT, AND, OR, XOR, LOAD, STORE}. For comparability with qRAM, LOAD and STORE act on ℓ bit registers.

Cost. The cost of a RAM program is the number of instructions that it performs. One can similarly define the *gate cost* of a quantum circuit to be the number of gates that it performs. Both metrics are reasonable in isolation, but it is not clear how one should compare the two. Jaques and Schanck recommend that quantum circuits be assigned a cost in the unit of RAM instructions to account for the role that classical computers play in dispatching gates to quantum memories [31]. They also recommend that the identity gate be assigned unit cost to account for error correction. The *depth-width cost* of a quantum circuit is the total number of gate operations that it performs when one includes identity gates in the count.

2.2 Black box search

A predicate on $\{0, 1, \ldots, N-1\}$ is a function $f : \{0, 1, \ldots, N-1\} \to \{0, 1\}$. The kernel, or set of roots, of f is $\mathrm{Ker}(f) = \{x : f(x) = 0\}$. We write $|f|$ for $|\mathrm{Ker}(f)|$. A black box search algorithm finds a root of a predicate without exploiting any structure present in the description of the predicate itself. Of course, black box search algorithms can be applied when structure is known, and we will often use structure such as "f has M roots" or "f is expected to have no more than M roots" in our analyses. We will also use the fact that the set of predicates on any given finite set can be viewed as a Boolean algebra. We write $f \cup g$ for the predicate with kernel $\mathrm{Ker}(f) \cup \mathrm{Ker}(g)$ and $f \cap g$ for the predicate with kernel $\mathrm{Ker}(f) \cap \mathrm{Ker}(g)$.

Exhaustive search. An exhaustive search evaluates $f(0)$, $f(1)$, $f(2)$, and so on until a root of f is found. The order does not matter so long as each element of the search space is queried at most once. If f is a uniformly random predicate with M roots, then this process has probability $1 - \binom{N-M}{j}/\binom{N}{j} \geq 1 - (1 - M/N)^j$ of finding a root during j evaluations of f. This is true even if M is not known.

Filtered search. If f is expensive to evaluate, we may try to decrease the cost of exhaustive search by applying a search filter. We say that a predicate g is a filter for f if $f \neq g$ and $|f \cap g| \geq 1$. We say that g recognises f with a false positive rate of

$$\rho_f(g) = 1 - \frac{|f \cap g|}{|g|},$$

and a false negative rate of

$$\eta_f(g) = 1 - \frac{|f \cap g|}{|f|}.$$

A filtered search evaluates $g(0), f(0), g(1), f(1), g(2), f(2)$, and so on until a root of $f \cap g$ is found. The evaluation of $f(i)$ can be skipped when i is not a root of g, which may reduce the cost of filtered search below that of exhaustive search.

Quantum search. Grover's quantum search algorithm is a black box search algorithm that provides a quadratic advantage over exhaustive search in terms of query complexity. Suppose that f is a predicate with M roots. Let $\mathbf{D}$ be any unitary transformation that maps $|0\rangle$ to $\frac{1}{\sqrt{N}} \sum_i |i\rangle$, let $\mathbf{R}_0 = \mathbf{I}_N - 2|0\rangle\langle 0|$ and let $\mathbf{R}_f$ be the unitary $|x\rangle \mapsto (-1)^{f(x)}|x\rangle$. Measuring $\mathbf{D}|0\rangle$ yields a root of f with probability M/N. Grover's quantum search algorithm amplifies this to probability ≈ 1 by repeatedly applying the unitary $\mathbf{G}(f) = \mathbf{D}\mathbf{R}_0\mathbf{D}^{-1}\mathbf{R}_f$ [27]. Suppose that j repetitions are applied.

The analysis in [27] shows that measuring the state $\mathbf{G}(f)^j\mathbf{D}|0\rangle$ yields a root of f with probability $\sin^2((2j+1)\cdot\theta)$ where $\sin^2(\theta) = M/N$. Assuming $M \ll N$, the probability of success is maximised at $j \approx \frac{\pi}{4}\sqrt{N/M}$ iterations. Boyer, Brassard, Høyer, and Tapp (BBHT) show that a constant success probability can be obtained after $O(\sqrt{N/M})$ iterations.

The same complexity can be obtained when M is not known. One simply runs the algorithm repeatedly with j chosen uniformly from successively larger intervals. The following lemma contains the core observation.

Lemma 2.1 (Lemma 2 of [13])**.** *Suppose that measuring $\mathbf{D}|0\rangle$ would yield a root of f with probability $\sin^2(\theta)$. Fix a positive integer m. Let j be chosen uniformly from $\{0,\ldots,m-1\}$. The expected probability that measuring $\mathbf{G}(f)^j\mathbf{D}|0\rangle$ yields a root of f is $\frac{1}{m}\sum_{j=0}^{m-1}\sin^2((2j+1)\cdot\theta) = \frac{1}{2} - \frac{\sin(4m\theta)}{4m\sin(2\theta)}$. If $m > 1/\sin(2\theta)$ then this quantity is at least $1/4$.*

The complete strategy is made precise by [13, Theorem 3].

Amplitude amplification. Brassard, Høyer, Mosca, and Tapp observed that the $\mathbf{D}$ subroutine of Grover's algorithm can be replaced with any algorithm that finds a root of f with positive probability [14]. This generalisation of Grover's algorithm is called amplitude amplification. Let $\mathbf{A}$ be a quantum algorithm that makes no measurements and let p be the probability that measuring $\mathbf{A}|0\rangle$ yields a root of f. Let $\mathbf{G}(\mathbf{A}, f) = \mathbf{A}\mathbf{R}_0\mathbf{A}^{-1}\mathbf{R}_f$, where $\mathbf{R}_0$ and $\mathbf{R}_f$ are as in Grover's algorithm. Let θ be such that $\sin^2(\theta) = p$. Suppose that j iterations of $\mathbf{G}(\mathbf{A}, f)$ are applied to $\mathbf{A}|0\rangle$. The analysis in [14] shows that measuring the state $\mathbf{G}(\mathbf{A}, f)^j\mathbf{A}|0\rangle$ yields a root of f with probability $\sin^2((2j+1)\cdot\theta)$. The BBHT strategy for handling an unknown number of roots generalises to an unknown p.

2.3 Lattice sieving and near neighbour search on the sphere

A Euclidean lattice of rank m and dimension d is an abelian group generated by integer sums of $m \le d$ linearly independent vectors in $\mathbb{R}^d$. In this paper we only consider full rank lattices, i.e. $m = d$. The shortest vector problem in a lattice Λ is the problem of finding a non-zero $v \in \Lambda$ of minimal Euclidean norm. Norms in this work are Euclidean and denoted $\|\cdot\|$. The angular distance of $u, v \in \mathbb{R}^d$ is denoted $\theta(u, v) = \arccos(\langle u, v\rangle/\|u\|\|v\|)$, $\arccos(x) \in [0, \pi]$.

A lattice sieve takes as input a list of lattice points, $L \subset \Lambda$, and searches for integer combinations of these points that are short. If the initial list is sufficiently large, SVP can be solved by performing this process recursively. Each point in the initial list can be sampled at a cost polynomial in d [33]. Hence the initial list can be sampled at a cost of $|L|^{1+o(1)}$.

Sieves that combine k points at a time are called k-sieves. The sieves that we consider in this paper are 2-sieves. They take integer combinations of the form $u \pm v$ with $u, v \in L$ and $u \neq \pm v$. If $\|u \pm v\| \geq \max\{\|u\|, \|v\|\}$ then we say that (u, v) is a reduced pair, else it is a reducible pair.

We analyse 2-sieves under the heuristic that the points in L are independent and identically distributed (i.i.d.) uniformly in a thin spherical shell. This heuristic was introduced by Nguyen and Vidick in [44]. As a further simplification, we assume that the shell is very thin and normalise such that $L \subset \mathcal{S}^{d-1}$, the unit sphere in $\mathbb{R}^d$. As such, (u, v) are reducible if and only if $\theta(u, v) < \pi/3$. The `popcount` filter, introduced in Section 2.4, acts as a first approximation to $\theta(\cdot\,, \cdot)$.

When we model L as a subset of $\mathcal{S}^{d-1}$, we can translate some lattice sieves into the language of (angular) near neighbour search on the sphere. For example, the Nguyen–Vidick sieve [44], which checks all pairs in L for reducibility, becomes[1] Algorithm 8 with $\theta = \pi/3$.

Algorithm 8 AllPairSearch

Require: A list $L = (v_1, v_2, \ldots v_N) \subset \mathcal{S}^{d-1}$ of N points. Parameter $\theta \in (0, \pi/2)$.
Ensure: A list of pairs $(u, v) \in L \times L$ with $\theta(u, v) \leq \theta$.

```
function AllPairSearch($L; \theta$)
    $L' \leftarrow \emptyset$
    for $1 \leq i < N$ do
        $L_i \leftarrow (v_{i+1}, \ldots, v_N)$
        Search $L_i$ for any number of $u$ that satisfy $\theta(u, v_i) \leq \theta$.
        For each such $u$ found, add $(u, v_i)$ to $L'$.
        If $|L'| \geq N$, return $L'$.
    return $L'$
```

[1] This is slightly imprecise. The analogy with the Nguyen–Vidick sieve is completed only when Algorithm 8 is wrapped in a procedure that takes each $(u, v) \in L'$ and maps it to $(u \pm v)/\|u \pm v\|$, and then recurses.

2.4 The popcount filter

Charikar's locality sensitive hashing (LSH) scheme [16] is a family of hash functions $\mathcal{H}$, defined on $\mathcal{S}^{d-1}$, for which

$$\Pr_{h \leftarrow \mathcal{H}}[h(u) = h(v)] = 1 - \frac{\theta(u,v)}{\pi}. \tag{2}$$

The hash function family is defined by

$$\mathcal{H} = \left\{ u \mapsto \operatorname{sgn}(\langle r, u \rangle) : r \in \mathcal{S}^{d-1} \right\},$$

where $\operatorname{sgn}(x) = 1$ if $x \geq 0$ and $\operatorname{sgn}(x) = 0$ if $x < 0$. Equation 2 follows from the fact that $\theta(u,v)/\pi$ is the probability that uniformly random u and v lie in opposite hemispheres.

Charikar observed that one can estimate $\theta(u,v)/\pi$ by choosing a random hash function $h = (h_1, \ldots, h_n) \in \mathcal{H}^n$ and measuring the Hamming distance between $h(u) = (h_1(u), \ldots, h_n(u))$ and $h(v) = (h_1(v), \ldots, h_n(v))$. Each bit $h_i(u) \oplus h_i(v)$ is Bernoulli distributed with parameter $p = \theta(u,v)/\pi$. In the limit of large n, the normalised Hamming weight $wt(h(u) \oplus h(v))/n$ converges to a normal distribution with mean p and standard deviation $\sqrt{p(1-p)/n}$.

In the sieving literature, the process of filtering a $\theta(\cdot,\cdot)$ test using a threshold on the value of $wt(h(u) \oplus h(v))$ is known as the "XOR and population count trick" [2, 19, 20]. Functions in $\mathcal{H}^n$ are also used in Laarhoven's HashSieve [35]. We write $\texttt{popcount}_{k,n}(u, v; h)$ for a search filter of this type

$$\texttt{popcount}_{k,n}(u, v; h) = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} h_i(u) \oplus h_i(v) \leq k, \\ 1 & \text{otherwise.} \end{cases}$$

When the n hash functions are fixed we write $\texttt{popcount}_{k,n}(u, v)$. The threshold, k, is chosen based on the desired false positive and false negative rates. Heuristically, if one's goal is to detect points at angle at most θ, one should take $k/n \approx \theta/\pi$. If $k/n \ll \theta/\pi$ then the false negative rate will be large, and many neighbouring pairs will be missed. An important consequence of missing potential reductions is that the N required to iterate Algorithms 8, 10, 11 increases. In Section 6 this increase is captured in the quantity $\ell(k, n)$. If $k/n \gg \theta/\pi$ then the false positive rate will be large, and the full inner product test will be applied often. We calculate these false positive and negative rates in Section 5. These calculations and the fact that popcount is significantly cheaper than an inner product makes popcount a good candidate for use as a filter under the techniques of Section 2.2. Furthermore it is the filter used in the most performant sieves to date [2, 19].

2.5 Geometric figures on the sphere

Our analysis of the `popcount` filter requires some basic facts about the size of some geometric figures on the sphere. We measure the volume of subsets of $\mathcal{S}^{d-1} = \{v \in \mathbb{R}^d \colon \|v\| = 1\}$ using the $(d-1)$ dimensional spherical probability measure[2] μ^{d-1}. The spherical cap of angle θ about $u \in \mathcal{S}^{d-1}$ is $\mathcal{C}^{d-1}(u, \theta) = \{v \in \mathcal{S}^{d-1} : \theta(u, v) \leq \theta\}$. The measure of a spherical cap is

$$C_d(u, \theta) := \mu^{d-1}(\mathcal{C}^{d-1}(u, \theta)) = \frac{1}{\sqrt{\pi}} \frac{\Gamma(\frac{d}{2})}{\Gamma(\frac{d-1}{2})} \int_0^\theta \sin^{d-2}(t) \, \mathrm{d}t.$$

We will often interpret $C_d(u, \theta)$ as the probability that v drawn uniformly from $\mathcal{S}^{d-1}$ satisfies $\theta(u, v) \leq \theta$. We denote the density of the event $\theta(u, v) = \theta$ by

$$A_d(u, \theta) := \frac{\partial}{\partial \theta} C_d(u, \theta) = \frac{1}{\sqrt{\pi}} \frac{\Gamma(\frac{d}{2})}{\Gamma(\frac{d-1}{2})} \sin^{d-2}(\theta).$$

Note that $C_d(u, \theta)$ does not depend on u, so we may write $C_d(\theta)$ and $A_d(\theta)$ without ambiguity. The wedge formed by the intersection of two caps is $\mathcal{W}^{d-1}(u, \theta_u, v, \theta_v) = \mathcal{C}^{d-1}(u, \theta_u) \cap \mathcal{C}^{d-1}(v, \theta_v)$. The measure of a wedge only depends on $\theta = \theta(u, v)$, θ_u, and θ_v, so we denote it

$$W_d(\theta, \theta_u, \theta_v) = \mu^{d-1}(\mathcal{W}^{d-1}(u, \theta_u, v, \theta_v)).$$

We will often interpret $W_d(\theta, \theta_u, \theta_v)$ as the probability that w drawn uniformly from $\mathcal{S}^{d-1}$ satisfies $\theta(u, w) \leq \theta_u$ and $\theta(v, w) \leq \theta_v$. Note that $\theta \geq \theta_u + \theta_v \Rightarrow W_d(\theta, \theta_u, \theta_v) = 0$. An integral representation of $W_d(\theta, \theta_u, \theta_v)$ is given in Appendix A.

3 Filtered quantum search

A filter can reduce the cost of a search because a classical computer can branch to avoid evaluating an expensive predicate. A quantum circuit cannot branch inside a Grover search in this way. Nevertheless, a filter can be used to reduce the cost of a quantum search.

The idea is to apply amplitude amplification to a Grover search. The inner Grover search prepares the uniform superposition over roots of the filter, g. The outer amplitude amplification searches for a root of f among the roots of g. We present pseudocode for this strategy in Algorithm 9.

[2] By "probability measure" we mean that $\mu^{d-1}(\mathcal{S}^{d-1}) = 1$.

If $|g|$ and $|f \cap g|$ are known, then we can choose the number of iterations of the inner Grover search and the outer amplitude amplification optimally. When these quantities are not known, we can attempt to guess them as in the BBHT algorithm. In our applications, we have some information about $|g|$ and $|f \cap g|$, which we can use to fine-tune a BBHT-like strategy.

Proposition 3.1 gives the cost of Algorithm 9 when we know *1.* a lower bound, Q, on the size of $|f \cap g|$, and *2.* the value of $|g|$ up to relative error γ. In essence, when a filter with a low false positive rate is used to search a space with few true positives, Algorithm 9 can be tuned such that it finds a root of f with probability at least $1/14$ and at a cost of roughly $\frac{\gamma}{2}\sqrt{N/Q}$ iterations of $\mathbf{G}(g)$.

Algorithm 9 FilteredQuantumSearch

Require: A predicate f and a filter g defined on $\{0, \ldots, N-1\}$. Integer parameters m_1 and m_2.

Ensure: A root of f or $\perp$.

```
function FilteredQuantumSearch($f, g; m_1, m_2$)
    Sample integers $j$ and $k$ with $0 \leq j < m_1$ and $0 \leq k < m_2$ uniformly at random.
    Let $\mathbf{A}_j = \mathbf{G}(g)^j \mathbf{D}$.
    Let $\mathbf{B}_k = \mathbf{G}(\mathbf{A}_j, f \cap g)^k$.
    Prepare the state $|\psi\rangle = \mathbf{B}_k \mathbf{A}_j |0\rangle$.
    Let $r$ be the result of measuring $|\psi\rangle$ in the computational basis.
    if $f(r) = 0$ then
        return $r$
    return $\perp$
```

If we know that the the inner Grover search succeeds with probability $x < 1$, we can compensate with a factor of $\sqrt{1/x}$ more iterations of the outer amplitude amplification. We do not know x. However, in our applications, we do know that the value of θ for which $\sin^2(\theta) = |g|/N$ will be fairly small, e.g. $\theta < 1/10$. The following technical lemma shows that, when θ is small, we may assume that $x = 1/5$ with little impact on the overall cost of the search.

Let j and $\mathbf{A}_j$ be as in Algorithm 9. Let $p_\theta(j)$ be the probability that measuring $\mathbf{A}_j|0\rangle$ would yield a root of g. For any $x \in (0,1)$, there is some probability $q_x(m_1)$ that the choice of j is insufficient, i.e. that $p_\theta(j) < x$. We expect to repeat Algorithm 9 a total of $(1 - q_x(m_1))^{-1}$ times to avoid this type of failure.

Lemma 3.1. *Fix $\theta \in [0, \pi/2]$ and $x \in [0,1)$. Let $p_\theta, q_x : \mathbb{R} \to \mathbb{R}$ be defined by $p_\theta(j) = \sin^2((2j+1)\cdot\theta)$ and $q_x(m) = \frac{1}{m}|\{j \in \mathbb{Z} : 0 \leq j < m, p_\theta(j) < x\}|$. If $m > \frac{\pi}{4\theta}$,* *then*

$$q_x(m) < \frac{3\arcsin(\sqrt{x})}{\pi - \arcsin(\sqrt{x})} + \frac{6\theta}{\pi}.$$

Proof. Observe that $p_\theta(j) < x$ when $|(2j+1)\theta \bmod \pi| < \arcsin(\sqrt{x})$. Let I_0 be the interval $[0, \arcsin(\sqrt{x}))$. For integers $t \geq 1$ let $I_t = (t\pi - \arcsin(\sqrt{x}), t\pi + \arcsin(\sqrt{x}))$. Let $c = c(m)$ be the largest integer for which $[0, (2m-1)\cdot\theta)$ intersects I_c. The quantity $mq_x(m)$ counts the number of non-negative integers $i < m$ for which $(2i+1)\cdot\theta$ lies in $I_0 \cup I_1 \cup \cdots \cup I_c$. This is no more than $(c+1) + \lfloor (2c+1)\arcsin(\sqrt{x})/(2\theta) \rfloor$. It follows that $q_x(m) < (c+1)/m + (2c+1)\arcsin(\sqrt{x})/2m\theta$. Note that $2m\theta > (2m-1)\theta > c\pi - \arcsin(\sqrt{x})$ and $(c+1)/m < 2\theta/\pi + 1/m$. Hence $q_x(m) < (2c+1)\arcsin(\sqrt{x})/(c\pi - \arcsin(\sqrt{x})) + 2\theta/\pi + 1/m$. Moreover, $q_x(m) > q_x(m-1)$ when $(2m-1)\cdot\theta$ lies in I_c, and $q_x(m) < q_x(m-1)$ otherwise. The upper bound on $q_x(m)$ that we have derived is decreasing as a function of c. Hence the claim holds when $c \geq 1$. Finally, when $m = \frac{\pi}{4\theta}$ and $c = 0$ we have $q_x(m) < 2\arcsin(\sqrt{x})/\pi + 4\theta/\pi$ and $q_x(m)$ is decreasing until $c = 1$. □

There are situations in which filtering is not effective, e.g. when the false positive rate of g is very high, when evaluting g is not much less expensive than evaluating f, or when f has a very large number of roots. In these cases, other algorithms will outperform Algorithm 9. We remark on these below. Proposition 3.1 optimises the choice of m_1 and m_2 in Algorithm 9 for a large class of filters that are typical of our applications.

Proposition 3.1. *Suppose that f and g are predicates on a domain of size N and that g is a filter for f. Let $Q \in \mathbb{R}$ be such that $1 \leq Q \leq |f \cap g|$. Let P and γ be real numbers such that $P/\gamma \leq |g| \leq \gamma P$. If $\gamma P/N < 1/100$ and $\gamma Q/P < 1/4$, then there are parameters m_1 and m_2 for Algorithm 9 such that Algorithm 9 finds a root of f with probability at least $1/14$ and has a cost that is dominated by $\approx \frac{\gamma}{2}\sqrt{N/Q}$ times the cost of $\mathbf{G}(g)$ or by $\approx \frac{2}{3}\sqrt{\gamma P/Q}$ times the cost of $\mathbf{R}_{f\cap g}$.*

Proof. Fix $x \in (0,1)$. We will analyse Algorithm 9 with respect to the parameters $m_1 = \left\lceil \frac{\pi}{4}\sqrt{\gamma N/P} \right\rceil$ and $m_2 = \left\lceil \sqrt{\gamma P/3xQ} \right\rceil$. Let θ_g be such that $\sin^2(\theta_g) = |g|/N$. Let j and k be chosen as in Algorithm 9. Let $p = p_{\theta_g}(j)$ and $q = q_x(m_1)$ be defined as in Lemma 3.1. Note that since $|g|/N < \gamma P/N < 1/100$ we can use $6\theta_g/\pi < 1/5$ in applying Lemma 3.1. Let $\theta_h(j)$ be such that $\sin^2(\theta_h(j)) = p \cdot |f \cap g| / |g|$. With probability at least $1 - q$ we have $p \geq x$, which implies that $\sin(\theta_h(j)) > \sqrt{xQ/\gamma P}$. Since $\gamma Q/P < 1/4 \Rightarrow \sin^2(\theta_h(j)) < 1/4$, then $\cos(\theta_h(j)) > \sqrt{3/4}$. Thus $1/\sin(2\theta_h(j)) < \sqrt{\frac{\gamma P}{3xQ}} \leq m_2$. By Lemma 2.1 measuring $\mathbf{G}(\mathbf{A}_j, f \cap g)^k \mathbf{A}_j|0\rangle$ yields a root of $f \cap g$ with probability at least $1/4$. It follows that Algorithm 9 succeeds with probability at least $(1-q)/4$.

The algorithm evaluates $\mathbf{G}(g)$ exactly $k \cdot j + 1$ times and evaluates $\mathbf{G}(g)^{-1}$ exactly $k \cdot j$ times. The expected value of $2kj+1$ is $c_1(x) \cdot \gamma \cdot \sqrt{N/Q}$ where $c_1(x) \approx (\pi/8)/\sqrt{3x}$. Likewise the algorithm evaluates $\mathbf{R}_{f\cap g}$ exactly k times, which is $c_2(x) \cdot \sqrt{\gamma P/Q}$ in expectation where $c_2(x) \approx (1/2)/\sqrt{3x}$. Taking $x = 1/5$, and applying the upper bound on $q_x(m_1)$ from Lemma 3.1, we have $(1 - q_x(m_1))/4 \geq 1/14$, $c_1(x) \approx 1/2$ and $c_2(x) \approx 2/3$. □

Remark 3.1. *When $\gamma P/N \geq 1/100$ or $\gamma Q/P \geq 1/4$ there are better algorithms. If both inequalities hold then classical search finds a root of f quickly. If $\gamma Q/P \geq 1/4$ then finding a root of f is not much harder than finding a root of g, so one can search on g directly. If $\gamma P/N \geq 1/100$ then the filter has little effect and one can search on f directly.*

Remark 3.2. *It is helpful to understand when we can ignore the cost of $\mathbf{R}_{f\cap g}$ in Proposition 3.1. Roughly speaking, if evaluating f is c times more expensive than evaluating g, then the cost of calls to $\mathbf{G}(g)$ will dominate when $N > c^2\,|g|$. In a classical filtered search the cost of evaluating g dominates when $N > c\,|g|$.*

4 Circuits for popcount

Consider a program for $\texttt{popcount}_{k,n}(u, v)$. This program loads u and v from specified memory addresses, computes $h(u)$ and $h(v)$, computes the Hamming weight of $h(u) \oplus h(v)$, and checks whether it is less than or equal to k. Recall $h(u)$ is defined by n inner products. If the popcount procedure is executed many times for each u, then it may be reasonable to compute $h(u)$ once and store it in memory. Moreover, if u is fixed for many sequential calls to the procedure, then it may be reasonable to cache $h(u)$ between calls. The algorithms that we consider in Section 6 use both of these optimisations.

In this section we describe RAM programs and quantum circuits that compute $\texttt{popcount}_{k,n}(u, \cdot)$ for a fixed u. These circuits have the value of $h(u)$ hard-coded. They load $h(v)$ from memory, compute the Hamming weight of $h(u) \oplus h(v)$, and check whether the Hamming weight is less than or equal to k. We ignore the initial, one time, cost of computing $h(u)$ and $h(v)$.

4.1 Quantum circuit for `popcount`

Loading $h(v)$ costs a single qRAM gate. Computing $h(u) \oplus h(v)$ can then be done in-place using a sequence of $\mathbf{X}$ gates that encode $h(u)$. The bulk of the effort is in computing the Hamming weight. For that we use a tree of in-place adders. The final comparison is also computed with an adder, although only one bit of the output is needed. See Figure 1 for a full description of the circuit.

We use the Cuccaro–Draper–Kutin–Petrie adder [17], with "incoming carry" inputs, to compute the Hamming weight. We argue in favour of this choice of adder in Appendix C. We use the Häner–Roetteler–Svore [28] carry bit circuit for implementing the comparison.

We will later use `popcount` within filtered quantum searches by defining predicates of the form $g(i) = \texttt{popcount}_{k,n}(u, v_i)$, $i \in \{1, \ldots, N\}$. To simplify that later discussion, we cost the entire Grover iteration $\mathbf{G}(g) = \mathbf{D}\mathbf{R}_0\mathbf{D}^{-1}\mathbf{R}_g$ here. In Appendix B we introduce the (possibly multiply controlled) Toffoli gate and discuss the Toffoli count for $\mathbf{G}(g)$, which in turn gives the $\mathbf{T}$ count for $\mathbf{G}(g)$.

The cost of $\mathbf{R}_g$. The $\mathbf{R}_g$ subroutine is computed by running the `popcount` circuit in Figure 1 and then uncomputing the addition tree and $\mathbf{X}$ gates. The circuit uses in-place i bit adders[3] for $i \in \{1, \ldots, \ell - 1\}$. The width of the circuit is given in Appendix B. The depth of the circuit is

$$\text{depth} = 2 + d(\text{CARRY}) + \sum_{i=1}^{\ell-1} 2 \cdot d(\text{ADD}_i), \tag{3}$$

where $d(\cdot)$ denotes the depth of its argument. The factor of 2 accounts for uncomputation of the ADD_i circuits. The CARRY circuit is only cost once as the carry bit is computed directly into the $|-\rangle$ state during the CARRY circuit itself. The summand 2 accounts for the $\mathbf{X}$ gates used to compute, and later uncompute, $h(u) \oplus h(v)$.

The cost of $\mathbf{D}\mathbf{R}_0\mathbf{D}^{-1}$. Recall that $\mathbf{D}$ can be any circuit that maps $|0\rangle$ to the uniform distribution on the domain of the search predicate. While there is no serious difficulty in sampling from the uniform distribution on $\{0, \ldots, N-1\}$ for any integer N, when costing the circuit we assume that N is a power of two. In this case $\mathbf{D}$ is simply $\log_2 N$ parallel $\mathbf{H}$ gates. The reflection $\mathbf{R}_0$ is implemented as a multiply

[3] An in-place i bit quantum adder takes two i bit inputs, initialises an ancilla qubit in the $|0\rangle$ state, and returns the addition result in an $i+1$ bit register that includes the new ancilla and overlaps with i bits of the input.

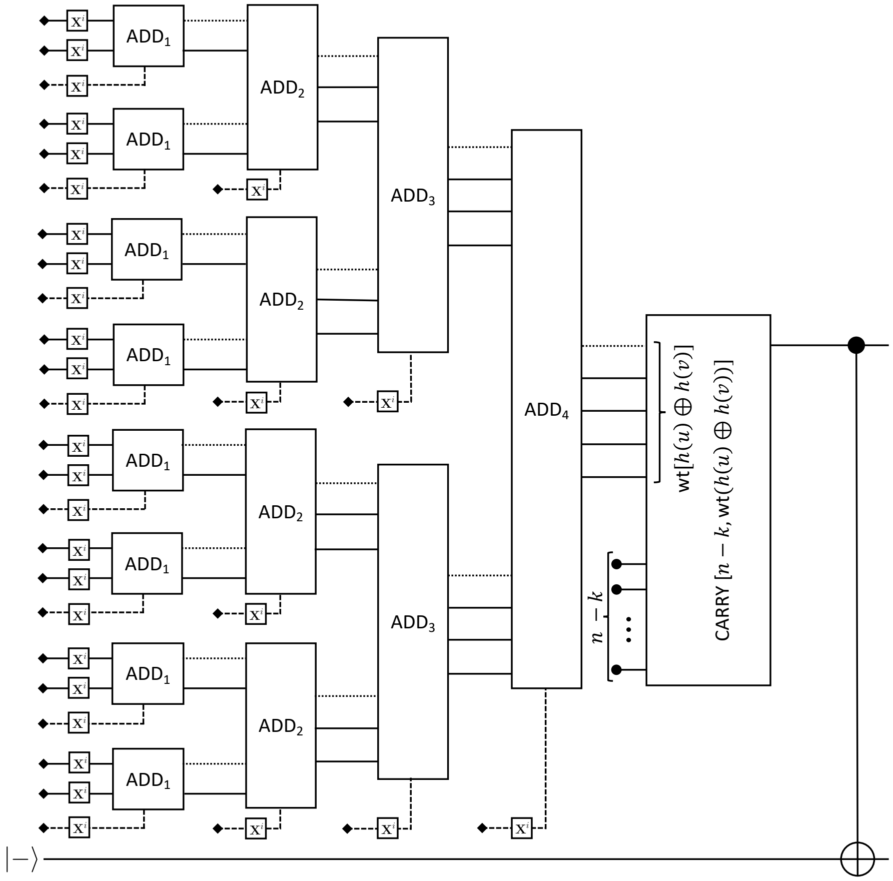

X^i
ADD$_1$
ADD$_2$
ADD$_3$
ADD$_4$
$\mathrm{wt}[h(u) \oplus h(v)]$
CARRY $[n-k, \mathrm{wt}(h(u) \oplus h(v))]$
$n-k$
$|-\rangle$

Figure 1: A quantum circuit for `popcount`. This circuit computes $h(u) \oplus h(v)$ for a fixed n bit $h(u)$, computes the Hamming weight of $h(u) \oplus h(v)$, and checks whether the Hamming weight is less than or equal to k. Here $n = 2^\ell - 1 = 31$. The input qubits are represented as lines ending with a black diamond. The dashed lines represent incoming carry inputs, and the dotted lines represent carry outputs. Not all of the output wires are drawn. For space efficiency, some of the input qubits are fed into the incoming carry qubits of the adders (dashed lines). The $\mathbf{X}^i$ mean that gate $\mathbf{X}$ is applied to input qubit i if bit i of $h(u)$ is 1. The circuit uses a depth $\ell - 1$ binary tree of full bit adders from [17], where ADD$_i$ denotes an i bit full adder. The output $wt(h(u) \oplus h(v))$ from the tree of adders together with the binary representation of the number $n - k$ are finally fed into the input of the CARRY circuit from [28], which computes the carry bit of $n - k + wt(h(u) \oplus h(v))$ (the carry bit will be 0 if $wt(h(u) \oplus h(v)) \leq k$, and 1 otherwise). The final **CNOT** is for illustration only. In actuality, the carry bit is computed directly into an ancilla that is initialised in the $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$ state, so we can obtain the needed phase kickback. The tree of adders and the initial $\mathbf{X}$ gates, but not the CARRY circuit, are run in reverse to clean up scratch space and return the inputs to their initial state. The uncomputation step is not depicted here.

controlled Toffoli gate (Appendix B) that targets an ancilla initialised in the $|-\rangle$ state. We use Maslov's multiply controlled Toffoli from [41]. The depth and width of $\mathbf{D}\mathbf{R}_0\mathbf{D}^{-1}$ are both $O(\log N)$; our software calculates the exact value.

4.2 RAM program for popcount

Recall that we use a RAM instruction set that consists of simple bit operations and table lookups. A Boolean circuit for popcount is schematically similar to Figure 1. Let $\ell = \lceil \log_2 n \rceil$. Loading $h(v)$ has cost 1. Computing $h(v) \oplus h(w)$ takes n XOR instructions and has depth 1. Following [45, Table. II], with $c_{FA} = 5$ the number of instructions in a full adder, $(n - \ell - 1)c_{FA} + \ell$ lower bounds the instruction cost of computing the Hamming weight and comparing it with a fixed k. This has depth $(\ell - 1)(\delta_{\text{sum}} + \delta_{\text{carry}}) + 1$. We assume $\delta_{\text{sum}} = \delta_{\text{carry}} = 1$. Thus, the overall instruction count is $6n - 4\ell - 5$ and the overall depth is 2ℓ.

4.3 Cost of inner products

The optimal popcount parameters will depend on the cost of a computing an inner product in dimension d. The cost of one inner product is amortised over many popcounts, and a small change in the popcount parameters will quickly suppress the ratio of inner products to popcounts (see Remark 3.2). Hence we only need a rough estimate for the cost of an inner product. We assume 32 bits of precision are sufficient. We then assume schoolbook multiplication is used for scalar products, which costs approximately 32^2 AND instructions. We then assume the cost of a full inner product is approximately $32^2\,d$, i.e. we ignore the cost of the final summation, assuming it is dwarfed by the ANDs.[4]

5 The accuracy of popcount

Here we give an analysis of the popcount technique based on some standard simplifying assumptions. We are particularly interested in the probability that a popcount filter identifies a random pair of points as potential neighbours. We are also interested in the probability that a pair of actual neighbours are not identified as potential neighbours, i.e. the false negative rate. Our software computes all of the quantities in this section to high precision.

[4] We also tested the effect of assuming 8-bit inner products are sufficient. As expected, this reduces all costs by a factor of two to four and thus does not substantially alter our relative results.

Let $P_{k,n}(u,v)$ be the probability that $\texttt{popcount}_{k,n}(u,v;h) = 0$ for a uniformly random h (recall $\texttt{popcount}_{k,n}(u,v;h) = 0$ if u, v pass the filter). In other words, let $h = (h_1, \ldots, h_n)$ be a collection of independent random variables that are distributed uniformly on the sphere, and define

$$P_{k,n}(u,v) = 1 - \mathbb{E}\left[\texttt{popcount}_{k,n}(u,v;h)\right].$$

The hyperplane defined by h_i separates u and v with probability $\theta(u,v)/\pi$, and $\texttt{popcount}_{k,n}(u,v;h) = 0$ if no more than k of the hyperplanes separate u and v. Hence,

$$P_{k,n}(u,v) = \sum_{i=0}^{k} \binom{n}{i} \cdot \left(\frac{\theta(u,v)}{\pi}\right)^{i} \cdot \left(1 - \frac{\theta(u,v)}{\pi}\right)^{n-i}.$$

Note that $P_{k,n}(u,v)$ depends only on the angle between u and v, so it makes sense to define $P_{k,n}(\theta)$. The main heuristic in our analysis of $\texttt{popcount}$ is that $P_{k,n}(u,v)$ is a good approximation to the probability that $\texttt{popcount}_{k,n}(u,v;h) = 0$ for *fixed* h and *varying* u and v. Under this assumption, all of the quantities in question can be determined by integrating $P_{k,n}(u,v)$ over different regions of the sphere.

Let $\hat{P}_{k,n}$ denote the event that $\texttt{popcount}_{k,n}(u,v;h) = 0$ for uniformly random u, v, and h. Let $\hat{R}_\theta$ be the event that $\theta(u,v) \leq \theta$. Recall that $\Pr[\hat{R}_\theta] = C_d(\theta)$, and observe that $\Pr[\hat{R}_\theta]$ is a cumulative distribution with associated density $A_d(\theta) = \frac{\partial}{\partial\theta} C_d(\theta)$. We find, letting $\mathcal{S} = \mathcal{S}^{d-1}$ for some implicit d,

$$\begin{aligned}
\Pr[\hat{P}_{k,n}] &= \int_{\mathcal{S}} \int_{\mathcal{S}} P_{k,n}(u,v)\ \mathrm{d}\mu(v)\ \mathrm{d}\mu(u) \\
&= \int_{\mathcal{S}} \left(\int_0^{\pi} P_{k,n}(\theta) \cdot A_d(\theta)\ \mathrm{d}\theta \right)\ \mathrm{d}\mu(u) \\
&= \int_0^{\pi} P_{k,n}(\theta) \cdot A_d(\theta)\ \mathrm{d}\theta. \qquad (4)
\end{aligned}$$

Let u, v such that $\theta(u,v) \leq \varphi$ be neighbours. The false negative rate is $1 - \Pr[\hat{P}_{k,n} \mid \hat{R}_\varphi]$. The quantity $\Pr[\hat{P}_{k,n} \wedge \hat{R}_\varphi]$ can be calculated by changing the upper limit of integration in Equation 4. It follows that

$$1 - \Pr[\hat{P}_{k,n} \mid \hat{R}_\varphi] = 1 - \frac{1}{C_d(\varphi)} \int_0^{\varphi} P_{k,n}(\theta) \cdot A_d(\theta)\ \mathrm{d}\theta. \qquad (5)$$

In Section 6 we consider u and v that are uniformly distributed in a cap of angle $\beta < \pi/2$, rather than the uniformly distributed on the sphere. Let $\hat{B}_{w,\beta}$ be the event that u and v are uniformly distributed in a cap of angle β about w. We have

$$\begin{aligned}\Pr[\hat{B}_{w,\beta}] &= \int_{\mathcal{S}}\int_{\mathcal{S}} \mathbb{1}\left\{w \in \mathcal{W}^{d-1}(u,\beta,v,\beta)\right\}\ \mathrm{d}\mu(v)\ \mathrm{d}\mu(u) \\ &= \int_0^{2\beta} W_d(\theta,\beta,\beta)\cdot A_d(\theta)\ \mathrm{d}\theta. \end{aligned} \tag{6}$$

In the second line we have used the fact that $\beta < \pi/2$ and $W(\theta,\theta_1,\theta_2)$ is zero when $\theta \geq \theta_1 + \theta_2$. The quantity $\Pr[\hat{B}_{w,\beta} \wedge \hat{R}_\varphi]$ can be computed by changing the upper limit of integration in Equation 6 from 2β to $\min\{2\beta,\varphi\}$. We note that $\hat{B}_{w,\beta}$ has no dependence on w and therefore may also be written $\hat{B}_\beta$. The conditional probability that $\texttt{popcount}_{k,n}(u,v;h) = 0$, given that u, v are uniformly distributed in a cap B_β, $\Pr[\hat{P}_{k,n} \mid \hat{B}_\beta]$, can be computed using Equation 6 and

$$\Pr[\hat{P}_{k,n} \wedge \hat{B}_\beta] = \int_0^{2\beta} P_{k,n}(\theta)\cdot W_d(\theta,\beta,\beta)\cdot A_d(\theta)\ \mathrm{d}\theta. \tag{7}$$

The quantity $\Pr[\hat{P}_{k,n} \wedge \hat{B}_\beta \wedge \hat{R}_\varphi]$ can be computed by changing the upper limit of integration in Equation 7 from 2β to $\min\{2\beta,\varphi\}$. The false negative rate for popcount when restricted to a cap is $1 - \Pr[\hat{P}_{k,n} \mid \hat{B}_{w,\beta} \wedge \hat{R}_\varphi]$.

6 Tuning popcount for NNS

We now use the circuit sizes from Section 4 and the probabilities from Section 5 to optimise popcount for use in NNS algorithms. Our analysis is with respect to points sampled independently from the uniform distribution on the sphere. We further restrict our attention to *list-size preserving* parameterisations, which take an input list of size N and return an output list of (expected) size N.

We use the notation for events introduced in Section 5. In particular, we write $\hat{R}_\theta$ for the event that a uniformly random pair of vectors are neighbours, i.e. that they lie at angle less than or equal to θ of one another; $\hat{P}_{k,n}$ for the event that popcount identifies a uniformly random pair of vectors as potential neighbours; $\hat{B}_\beta$ for the event that a uniformly random pair of vectors lie in a uniformly random cap of angle β; and $\hat{B}_{w,\beta}$ for the same event except we highlight the cap is centred on w. Throughout this section we use $\texttt{popcount}_{k,n}(u,\cdot)$, for various fixed u, as a filter for the search predicate $\theta(u,\cdot) \leq \theta$. We write $\eta(k,n)$ for the false negative rate of popcount. We assume that $\theta(u,v) \leq \theta$ is computed using an inner product test. Throughout this section, c_1 represents the instruction cost of the inner product test from Section 4.3, $c_2(k,n)$ the instruction cost of popcount from Section 4.2, q_1 the quantum cost of the reflection $\mathbf{R}_{f\cap g}$, and $q_2(k,n)$ the quantum cost of $\mathbf{G}(g)$ from Section 4.1. We note that c_1, q_1 have a dependence on d that we suppress. We write $q_0(m)$ for the number of $\mathbf{G}(g)$ iterations that are applied during a quantum search on a set of size m.

Our goal is to minimise the cost of list-size preserving NNS algorithms as a function of the input list size, the `popcount` parameters k and n, and the other NNS parameters. In a list of N points there are $\binom{N}{2}$ ordered pairs. We expect $\binom{N}{2} \cdot \Pr[\hat{R}_\theta] = \binom{N}{2} \cdot C_d(\theta)$ of these to be neighbours, and we expect a $1 - \eta(k,n)$ fraction of neighbours to be detected by `popcount`. List-size preserving parmaterisations that use a `popcount` filter must therefore take an input list of size at least

$$\ell(k,n) = \frac{2}{1 - \eta(k,n)} \cdot \frac{1}{C_d(\theta)}. \tag{8}$$

The optimised costs reported in Figures 2, 3, and 4 typically use popcount parameters for which $\ell(k,n) \in (2/C_d(\pi/3), 4/C_d(\pi/3))$. Here we assume that list-size preserving parameterisations take $N = \ell(k,n)$. Note that $\eta(k,n) = 1 - \Pr[\hat{P}_{k,n} \mid \hat{R}_\theta]$ when the search is over a set of points uniformly distributed on the sphere, and $\eta(k,n) = 1 - \Pr[\hat{P}_{k,n} \mid \hat{R}_\theta \wedge \hat{B}_\beta]$ when the search is over a set of points uniformly distributed in a cap of angle β (left implicit).

In each of the quantum analyses, we apply Proposition 3.1 with $\gamma = 1$, $P = |g|$ and $Q = 1$ to estimate $q_0(m)$. We assume that filtered quantum search succeeds with probability 1 instead of probability at least $1/14$, as guaranteed by Proposition 3.1. In practice, one will not know $|g|$ and one will therefore take $\gamma > 1$. Our use of $\gamma = 1$ is a systematic underestimate of the true cost of the search. There may be searches where our lower bound of $Q = 1$ on $|f \cap g|$ is too pessimistic. However, the probability of success in filtered quantum search decreases quadratically with $Q / |f \cap g|$ if $Q > |f \cap g|$. In Sections 6.1 and 6.3 we expect $|f \cap g| \approx 2$ so the effect of taking $Q = 1$ is negligible. In Section 6.2, where Q may be larger, an optimistic analysis using the expected value of Q makes negligible savings in dimension 512 and small savings in dimension 1024. This analysis does not decrement Q when a neighbour is found in, then removed from, a search space and ignores the quadratic decrease in success probability.

6.1 AllPairSearch

As a warmup, we optimise AllPairSearch. Asymptotically its complexity is $2^{(0.415\ldots+o(1))d}$ classically and $2^{(0.311\ldots+o(1))d}$ quantumly. We describe implementations of Line 5 of Algorithm 8 based on filtered search and filtered quantum search, and optimise `popcount` relative to these implementations.

Filtered search.

Suppose that Line 5 applies $\texttt{popcount}_{k,n}(v_i, \cdot)$ to each of v_{i+1} through v_N and then applies an inner product test to each vector that passes. With an input list of size $N = \ell(k, n)$, we expect this implementation to test all $\binom{N}{2}$ pairs before finding N neighbouring pairs. Moreover, we expect the popcount filter to identify $\binom{N}{2} \cdot \Pr[\hat{P}_{k,n}]$ potential neighbours, and to perform an equal number of inner product tests. The optimal parameters are obtained by minimising

$$\left(c_1 \cdot \Pr[\hat{P}_{k,n}] + c_2(k,n)\right) \cdot \binom{\ell(k,n)}{2}. \tag{9}$$

Filtered quantum search.

Suppose that Line 5 is implemented using the search routine Algorithm 9. Specifically, we take the predicate f to be $\theta(v_i, \cdot) \leq \theta$ with domain L_i. We take the filter g to be $\texttt{popcount}_{k,n}(v_i, \cdot)$. Each call to the search routine returns at most one neighbour of v_i. To find all detectable neighbours of v_i in L_i we must repeat the search $|f \cap g|$ times. This is expected to be $|L_i| \cdot \Pr[\hat{P}_{k,n} \wedge \hat{R}_\theta]$. Known neighbours of v_i can be removed from L_i to avoid a coupon collector scenario. We consider an implementation in which searches are repeated until a search fails to find a neighbour of v_i.

We expect to call the search subroutine $|L_i| \cdot \Pr[\hat{P}_{k,n} \wedge \hat{R}_\theta] + 1$ times in iteration i. Proposition 3.1 with $P = |L_i| \cdot \Pr[\hat{P}_{k,n}]$, $Q = 1$, and $\gamma = 1$ gives $q_0\left(|L_i|\right) = \frac{1}{2}\sqrt{|L_i|}$ iterations of $\mathbf{G}(g)$. As i ranges from 1 to $N-1$ the quantity $|L_i|$ takes each value in $\{1, \ldots, N-1\}$. Our proposed implementation therefore performs an expected

$$\sum_{j=1}^{N-1} \frac{1}{2}\sqrt{j}\left(j \cdot \Pr[\hat{P}_{k,n} \wedge \hat{R}_\theta] + 1\right)$$
$$= \Pr[\hat{P}_{k,n} \wedge \hat{R}_\theta]\left(\frac{1}{5}N^{5/2} + \frac{1}{4}N^{3/2}\right) + \frac{1}{3}N^{3/2} + O(\sqrt{N}) \tag{10}$$

applications of $\mathbf{G}(g)$; the expansion is obtained by the Euler–Maclaurin formula. When $N = \ell(k, n)$ we expect $N \cdot \Pr[\hat{P}_{k,n} \wedge \hat{R}_\theta] = 2 + O(1/N)$. The right hand side of Equation 10 is then $\frac{11}{15}N^{3/2} + O(\sqrt{N})$.

Proposition 3.1 also provides an estimate for the rate at which reflections about the true positives, $\mathbf{R}_{f\cap g}$ are performed. With P and Q as above, we find that $\mathbf{R}_{f\cap g}$ is performed at roughly $p(k, n) = \sqrt{\Pr[\hat{P}_{k,n}]}$ the rate of calls to $\mathbf{G}(g)$. The optimal `popcount` parameters (up to some small error due to the $O(\sqrt{N})$ term in Equation 10) are obtained by minimising the total cost

$$\frac{11}{15}\left(q_1 p(k,n) + q_2(k,n)\right) \cdot \ell(k,n)^{3/2}. \tag{11}$$

6.2 RandomBucketSearch

One can improve AllPairSearch by *bucketing* the search space such that vectors in the same bucket are more likely to be neighbours [35]. For example, one could pick a hemisphere H and divide the list into $L_1 = L \cap H$ and $L_2 = L \backslash L_1$. These lists would be approximately half the size of the original and the combined cost of AllPairSearch within L_1 and then within L_2 would be half the cost of an AllPairSearch within L. However, this strategy would fail to detect the expected θ/π fraction of neighbours that lie in opposite hemispheres.

Becker, Gama, and Joux [10] present a very efficient generalisation of this strategy. They propose bucketing the input list into subsets of the form $\{v \in L : \texttt{popcount}_{k,n}(0, v; h) = 0\}$ with varying choices of h. This bucketing strategy is applied recursively until the buckets are of a minimum size. Neighbouring pairs are then found by an AllPairSearch.

A variant of the Becker–Gama–Joux algorithm that uses buckets of the form $L \cap \mathcal{C}^{d-1}(f, \theta_1)$, with randomly chosen f and fixed θ_1, was proposed and implemented in [2]. This variant is sometimes called `bgj1`. Here we call it RandomBucketSearch. This algorithm has asymptotic complexity $2^{(0.349\ldots+o(1))d}$ classically [2] and $2^{(0.301\ldots+o(1))d}$ quantumly.[5] This is worse than the Becker–Gama–Joux algorithm, but RandomBucketSearch is conceptually simple and still provides an enormous improvement over AllPairSearch. Pseudocode is presented in Algorithm 10.

Description of Algorithm 10.

The algorithm takes as input a list of N points uniformly distributed on the sphere. A random bucket centre f is drawn uniformly from $\mathcal{S}^{d-1}$ in each of the t iterations of the outer loop. The choice of f defines a bucket in Line 5, $L_f = L \cap \mathcal{C}^{d-1}(f, \theta_1)$, which is of expected size $N \cdot C_d(\theta_1)$. For each $v_j \in L_f$, the inner loop searches a set $L_{f,j} \subset L_f$ for neighbours of v_j. The quantity $|L_{f,j}|$ takes each value in $\{1, \ldots, |L_f| - 1\}$ as v_j

[5] The asymptotic quantum complexity is calculated, similarly to the classical complexity [2], using the asymptotic value of $W_d(\theta, \theta_1, \theta_1)$ given in [9]. Let $N = 1/C_d(\pi/3)$ and $t(\theta_1) = 1/W_d(\pi/3, \theta_1, \theta_1)$. The exponent $0.3013\ldots$ is obtained by minimising $t(\theta_1)\left(N + (NC_d(\theta_1))^{3/2}\right)$ with respect to θ_1.

Algorithm 10 RandomBucketSearch

Require: A list $L = (v_1, v_2, \dots v_N) \subset \mathcal{S}^{d-1}$ of N points. Parameters $\theta, \theta_1 \in (0, \pi/2)$ and $t \in \mathbb{Z}_+$.

Ensure: A list of pairs $(u, v) \in L \times L$ with $\theta(u, v) \leq \theta$.

```
 1: function RandomBucketSearch($L; \theta, \theta_1, t$)
 2:     $L' \leftarrow \emptyset$
 3:     for $1 \leq i \leq t$ do
 4:         Sample $f$ uniformly on $\mathcal{S}^{d-1}$
 5:         $L_f \leftarrow L \cap \mathcal{C}^{d-1}(f, \theta_1)$
 6:         for $j$ such that $v_j \in L_f$ do
 7:             $L_{f,j} \leftarrow \{v_k \in L_f : j < k \leq N\}$
 8:             Search $L_{f,j}$ for any number of $u$ that satisfy $\theta(v_j, u) \leq \theta$
 9:             For each such $u$ found, add $(v_j, u)$ to $L'$.
10:             If $|L'| \geq N$, return $L'$.
11:     return $L'$
```

ranges over L_f. The inner loop is identical to the loop in AllPairSearch apart from indexing and the fact that elements of L_f are known to be in the cap $\mathcal{C}^{d-1}(f, \theta_1)$.

A bucket L_f is expected to contain $\binom{N}{2} \cdot \Pr[\hat{R}_\theta \wedge \hat{B}_{f,\theta_1}]$ neighbouring pairs. Only a $1 - \eta(k, n)$ fraction of these are expected to be identified by the popcount filter. When $\theta_1 > \theta$ it is reasonable to assume that $\Pr[\hat{R}_\theta \wedge \hat{B}_{f,\theta_1}] \approx C_d(\theta) \cdot W_d(\theta, \theta_1, \theta_1)$. We use this approximation. The expected number of neighbouring pairs in L_f that are detected by the popcount filter is therefore approximately $\binom{N}{2} \cdot (1 - \eta(k, n)) \cdot C_d(\theta) \cdot W_d(\theta, \theta_1, \theta_1)$. When $N = \ell(k, n)$ this is $N \cdot W_d(\theta, \theta_1, \theta_1)$. If all detectable neighbours are found by the search routine then the algorithm is list-size preserving when $N = \ell(k, n)$ and $t = 1/W_d(\theta, \theta_1, \theta_1)$.

We can now derive optimal popcount parameters for various implementations of Line 8.

Filtered search.

Suppose that Line 8 of Algorithm 10 applies $\texttt{popcount}_{k,n}(v_j, \cdot)$ to each element of $L_{f,j}$ and then applies an inner product test to each vector that passes. This implementation applies popcount tests to all $\binom{|L_f|}{2} \approx \binom{N \cdot C_d(\theta_1)}{2}$ pairs of elements in L_f and finds all of the neighbouring pairs that pass. In the process it applies inner product tests to a $p(\theta_1, k, n) = \Pr[\hat{P}_{k,n} \mid \hat{B}_{f,\theta_1}]$ fraction of pairs. The cost of populating buckets in one iteration of Line 5 is $c_1 \cdot \ell(k, n)$. The cost of all searches on Line 8 is

$(c_1 \cdot p(\theta_1, k, n) + c_2(k, n)) \cdot \binom{NC_d(\theta_1)}{2}$. With the list-size preserving parameters N and t given above, the optimal θ_1, k, and n can be obtained by minimising the total cost

$$\frac{c_1 \cdot \ell(k,n) + (c_1 \cdot p(\theta_1, k, n) + c_2(k,n)) \cdot \binom{\ell(k,n) \cdot C_d(\theta_1)}{2}}{W_d(\theta, \theta_1, \theta_1)}. \tag{12}$$

Filtered quantum search.

Suppose that Line 8 is implemented using the search routine Algorithm 9. We take the predicate f to be $\theta(v_j, \cdot) \leq \theta$ with domain $L_{f,j}$. We take the filter g to be $\texttt{popcount}_{k,n}(v_j, \cdot)$. Each call to the search routine returns at most one neighbour of v_j. To find all detectable neighbours of v_j in $L_{f,j}$ we must repeat the search several times. Known neighbours of v_j can be removed from $L_{f,j}$ to avoid a coupon collector scenario. Proposition 3.1 with $P = |L_{f,j}| \cdot \Pr[\hat{P}_{k,n} \mid \hat{B}_{f,\theta_1}]$, $Q = 1$, and $\gamma = 1$ gives us that the number of $\mathbf{G}(g)$ iterations in a search on a set of size $|L_{f,j}|$ is $q_0\left(|L_{f,j}|\right) = \frac{1}{2}\sqrt{|L_{f,j}|}$.

We consider an implementation of Line 8 in which searches are repeated until a search fails to find a neighbour of v_j. With $N = \ell(k, n)$, the set L_f is of expected size $\ell(k,n) \cdot C_d(\theta_1)$ and contains an expected $\ell(k, n) \cdot W_d(\theta, \theta_1, \theta_1)$ neighbouring pairs detectable by $\texttt{popcount}$. The set $L_{f,j}$ is expected to contain a proportional fraction of these pairs. As such, we expect to call the search subroutine $|L_{f,j}| \cdot r(\theta_1, k, n) + 1$ times in iteration j where

$$r(\theta_1, k, n) = \frac{N \cdot W_d(\theta, \theta_1, \theta_1)}{\binom{|L_f|}{2}} \approx \frac{2\, W_d(\theta, \theta_1, \theta_1)}{\ell(k,n) \cdot C_d(\theta_1)^2}.$$

The inner loop makes an expected

$$\sum_{j=1}^{|L_f|-1} \frac{1}{2}\sqrt{j}\,(j \cdot r(\theta_1, k, n) + 1)$$

applications of $\mathbf{G}(g)$. This admits an asymptotic expansion similar to that of Equation 10. If we assume that $|L_f|$ takes its expected value of $\ell(k, n) \cdot C_d(\theta_1)$, then the inner loop makes

$$q_3(\theta_1, k, n) \cdot (\ell(k,n) \cdot C_d(\theta_1))^{3/2}$$

applications of $\mathbf{G}(g)$, where

$$q_3(\theta_1, k, n) = \frac{2\, W_d(\theta, \theta_1, \theta_1)}{5\, C_d(\theta_1)} + \frac{1}{3}.$$

Proposition 3.1 also provides an estimate for the rate at which reflections about the true positives, $\mathbf{R}_{f \cap g}$ are performed. With P and Q as above, we find that $\mathbf{R}_{f \cap g}$ is applied at roughly $p(\theta_1, k, n) = \sqrt{\Pr[\hat{P}_{k,n} \mid \hat{B}_{f,\theta_1}]}$ the rate of $\mathbf{G}(g)$ iterations. The total cost of searching for neighbouring pairs in L_f is therefore

$$s(\theta_1, k, n) = (q_1 \cdot p(\theta_1, k, n) + q_2(k, n)) \cdot q_3(\theta_1, k, n) \cdot \big(\ell(k, n) \cdot C_d(\theta_1)\big)^{3/2}. \tag{13}$$

Populating L_f has a cost of $c_1 \cdot \ell(k, n)$. With the list-size preserving t given above, the optimal parameters θ_1, k, and n can be obtained by minimising the total cost

$$\frac{c_1 \cdot \ell(k, n) + s(\theta_1, k, n)}{W_d(\theta, \theta_1, \theta_1)}. \tag{14}$$

6.3 ListDecodingSearch

The optimal choice of θ_1 in RandomBucketSearch balances the cost of $N \cdot t$ cap membership tests against the cost of all calls to the search subroutine. It can be seen that reducing the cost of populating the buckets would allow us to choose a smaller θ_1, which would reduce the cost of searching within each bucket.

Algorithm 11, ListDecodingSearch, is due to Becker, Ducas, Gama, and Laarhoven [9]. Its complexity is $2^{(0.292\ldots+o(1))d}$ classically and $2^{(0.265\ldots+o(1))d}$ quantumly [36, 37]. Like RandomBucketSearch, it computes a large number of list-cap intersections. However, these list-cap intersections involve a structured list—the list-cap intersections in RandomBucketSearch involve the inherently unstructured input list.

Description of Algorithm 11.

The algorithm first samples a t point *random product code* F. See [9] for background on random product codes. In our analysis, we treat F as a list of uniformly random points on $\mathcal{S}^{d-1}$. Some justification for this heuristic is given in [9, Appendix B].

The first loop populates t buckets that have as centres the points f of F. Bucket L_f stores elements of L that lie in the cap of angle θ_2 about f. Each bucket is of expected size $N \cdot C_d(\theta_2)$.

The second loop iterates over $v_j \in L$ and searches for neighbours of v_j in the disjoint union of buckets with centres within an angle θ_1 of v_j. The set F_j constructed on Line 8 contains an expected $t \cdot C_d(\theta_1)$ bucket centres. The disjoint union of certain elements from the corresponding buckets, denoted $L_{F,j}$, is of expected size $(N - j) \cdot C_d(\theta_2) \cdot t \cdot C_d(\theta_1)$. We note that by simplifying and assuming the expected size of $L_{F,j}$ is $N \cdot C_d(\theta_2) \cdot t \cdot C_d(\theta_1)$ the costs given below are never wrong by more than a factor of two.

Algorithm 11 ListDecodingSearch

Require: A list $L = (v_1, v_2, \dots v_N) \subset \mathcal{S}^{d-1}$ of N. Parameters $\theta, \theta_1, \theta_2 \in (0, \pi/2)$ and $t \in \mathbb{Z}_+$.

Ensure: A list of pairs $(u, v) \in L \times L$ with $\theta(u, v) \leq \theta$.

```
function ListDecodingSearch($L; \theta, \theta_1, \theta_2, t$)
    Sample a random product code $F$ of size $t$
    Initialise an empty list $L_f$ for each $f \in F$
    for $1 \leq i \leq N$ do
        $F_i \leftarrow F \cap \mathcal{C}^{d-1}(v_i, \theta_2)$
        Add $v_i$ to $L_f$ for each $f$ in $F_i$
    for $1 \leq j < N$ do
        $F_j \leftarrow F \cap \mathcal{C}^{d-1}(v_j, \theta_1)$
        for $f \in F_j$ do
            $L_{f,j} \leftarrow \{v_k \in L_f : j < k \leq N\}$
        $L_{F,j} \leftarrow \coprod_{f \in F_j} L_{f,j}$     (disjoint union)
        Search $L_{F,j}$ for any number of $u$ that satisfy $\theta(v_j, u) \leq \theta$
        For each such $u$ found, add $(v_j, u)$ to $L'$.
        If $|L'| \geq N$, return $L'$.
    return $L'$
```

Suppose that w is a neighbour of v_j, so $\theta(v_j, w) \leq \theta$. The measure of the wedge formed by a cap of angle θ_1 about v_j and a cap of angle θ_2 about w is at least $W_d(\theta, \theta_1, \theta_2)$. Assuming that the points of a random product code are indistinguishable from points sampled uniformly on the sphere, the probability that some $f \in F_j$ contains w is at least $t \cdot W_d(\theta, \theta_1, \theta_2)$.

The second loop is executed N times. Iteration j searches $L_{F,j}$ for neighbours of v_j. With $N = \ell(k, n)$ there are expected to be N detectable neighbouring pairs in L. With $t = 1/W_d(\theta, \theta_1, \theta_2)$ we expect that each neighbouring pair is of the form (v_j, w) with $w \in L_{F,j}$.

The angles θ_1, θ_2 relate to the spherical cap parameters α, β respectively in [9], and are such that $\theta_1 \geq \theta_2$. Optimal time complexity is achieved when $\theta_1 = \theta_2$.

We have omitted the list decoding mechanism by which list-cap intersections are computed. In our analysis we assume that the cost of a list-cap intersection such as $F_i = F \cap \mathcal{C}^{d-1}(v_i, \theta_1)$ is proportional to $|F_i|$, but independent of $|F|$. Specifically, we assume that the cost is equal to $|F_i|$ inner product tests. A cost of $|F_i|^{1+o(1)}$ is justified by [9, Section 5]. A cursory examination of that method shows the $o(1)$ term masks a cost much larger than one inner product test, but there may yet be improvements in this area.

Filtered search.

Suppose that the implementation of Line 12 of Algorithm 11 applies $\texttt{popcount}_{k,n}(v_j, \cdot)$ to each element of $L_{F,j}$ and then applies an inner product test to each vector that passes. This implementation applies popcount tests to all $N \cdot C_d(\theta_2) \cdot t \cdot C_d(\theta_1)$ elements of $L_{F,j}$ and finds all of the neighbours of v_j that pass. Note that $w \in L_{F,j}$ implies that there exists some $f \in F$ such that both v_j and w lie in a cap of angle θ_1 around f. Inner product tests are applied to a $p(\theta_1, k, n) \geq \Pr[\hat{P}_{k,n} \mid \hat{B}_{f,\theta_1}]$ fraction of all pairs.[6]

The cost of preparing all t buckets in the first loop is $c_1 \cdot N \cdot t \cdot C_d(\theta_2)$. The cost of constructing the search spaces in the second loop is $c_1 \cdot N \cdot t \cdot C_d(\theta_1)$. Each search has a cost of $|L_{F,j}|$ popcount tests and $|L_{F,j}| \cdot p(\theta_1, k, n)$ inner product tests. With the list-size preserving parameterisation given above, the optimal θ_1, θ_2, k, and n can be obtained by minimising the total cost

$$\frac{\ell(k,n)}{W_d(\theta,\theta_1,\theta_2)}\Big(c_1 \cdot C_d(\theta_1) + c_1 \cdot C_d(\theta_2) \\ + \big(c_1 \cdot p(\theta_1,k,n) + c_2(k,n)\big) \cdot \ell(k,n) \cdot C_d(\theta_1) \cdot C_d(\theta_2)\Big). \quad (15)$$

Filtered quantum search.

Suppose that Line 12 is implemented using Algorithm 9. We take the predicate f to be $\theta(v_j, \cdot) \leq \theta$ with domain $L_{F,j}$. We take the filter g to be $\texttt{popcount}_{k,n}(v_j, \cdot)$. Each call to the search routine returns at most one neighbour of v_j. Known neighbours of v_j can be removed from $L_{F,j}$ to avoid a coupon collector scenario. Proposition 3.1 with $P = |L_{F,j}| \cdot \Pr[\hat{P}_{k,n} \mid \hat{B}_{f,\theta_2}]$, $Q = 1$, and $\gamma = 1$ gives us that the number of $\mathbf{G}(g)$ iterations in a search on a set of size $|L_{F,j}|$ is $q_0\left(|L_{F,j}|\right) \approx \frac{1}{2}\sqrt{|L_{F,j}|}$.

Assuming that computing $F_j = F \cap C(v_j, \theta_1)$ has a cost of $c_1\,|F_j|$, the N iterations of Lines 5 and 8 have a total cost of

$$c_1 \cdot N \cdot t \cdot (C_d(\theta_1) + C_d(\theta_2)) \quad (16)$$

Each search applies an expected

$$q_0\left(|L_{F,j}|\right) \approx \frac{1}{2}\sqrt{N \cdot C_d(\theta_1) \cdot t \cdot C_d(\theta_2)}$$

[6] The inequality is because v_j and w may be contained in multiple buckets, $L_{f,j}$.

applications of $\mathbf{G}(g)$. Reflections about the true positives, $\mathbf{R}_{f\cap g}$, are performed at roughly $p(\theta_1, k, n) = \sqrt{\Pr[\hat{P}_{k,n} \mid B_{f,\theta_1}]}$ the rate of $\mathbf{G}(g)$ iterations. We consider an implementation of Line 8 in which searches are repeated until a search fails to find a neighbour of v_j. With the list-size preserving parameters given above, we expect to perform two filtered quantum searches per iteration of the second loop. The optimal parameters can be obtained by minimising the total cost

$$\ell(k,n)\left(c_1\frac{C_d(\theta_1)+C_d(\theta_2)}{W_d(\theta,\theta_1,\theta_2)} + (q_1 p(\theta_1,k,n) + q_2(k,n))\sqrt{\frac{\ell(k,n)C_d(\theta_1)C_d(\theta_2)}{W_d(\theta,\theta_1,\theta_2)}}\right).$$

7 Cost estimates

Our software numerically optimises the cost functions in Sections 6.1, 6.2 and 6.3 with respect to several classical and quantum cost metrics. The classical cost metrics that we consider are: c *(unit cost)*, which assigns unit cost to `popcount`; c *(RAM)*, which uses the classical circuits of Section 4. The quantum cost metrics that we consider are: q *(unit cost)*, which assigns unit cost to a Grover iteration; q *(depth-width)*, which assigns unit cost to every gate (including the identity) in the quantum circuits of Section 4; q *(gates)*, which assigns unit cost only to the non-identity gates; q *(T count)*, which assigns unit cost only to T gates; and q *(GE19)*, which is described in Section 7.1.

We stress that our software, and Figures 2, 3, and 4, give *estimates* for the cost of each algorithm. These estimates are neither upper bounds nor lower bounds. As we mention above, we have systematically omitted and underestimated some costs. For instance, we have omitted the list decoding mechanism in our costing of Algorithm 11. We have approximated other costs. For instance, the cost that we assign to an inner product in Section 4.3. We have also not explored the entire optimisation space. We only consider values of the popcount parameter n that are one less than a power of two. Moreover, following the discussion in Section 2.4, we set $k = \lfloor n/3 \rfloor$.

While we have omitted and approximated some costs, we have tried to ensure that these omissions and approximations will ultimately lead our software to *underestimate* of the total cost of the algorithm. For instance, if our inner product cost is accurate, our optimisation procedure ensures that we satisfy Remark 3.2 and can ignore costs relating to $\mathbf{R}_{f\cap g}$.

Our results are presented in Figures 2, 3, and 4. We also plot the leading term of the asymptotic complexity of the respective algorithms as these are routinely referred to in the literature.

We give the source code to produce our figures in Appendix E.

Figure 2: Quantum ("q") and classical ("c") resource estimates for RandomBucketSearch. The three plots, respectively, show crossover points in dimension ≤ 64 under the gate cost metric, dimension 88 under the depth-width metric, and dimension 288 under the Gidney–Ekerå cost metric.

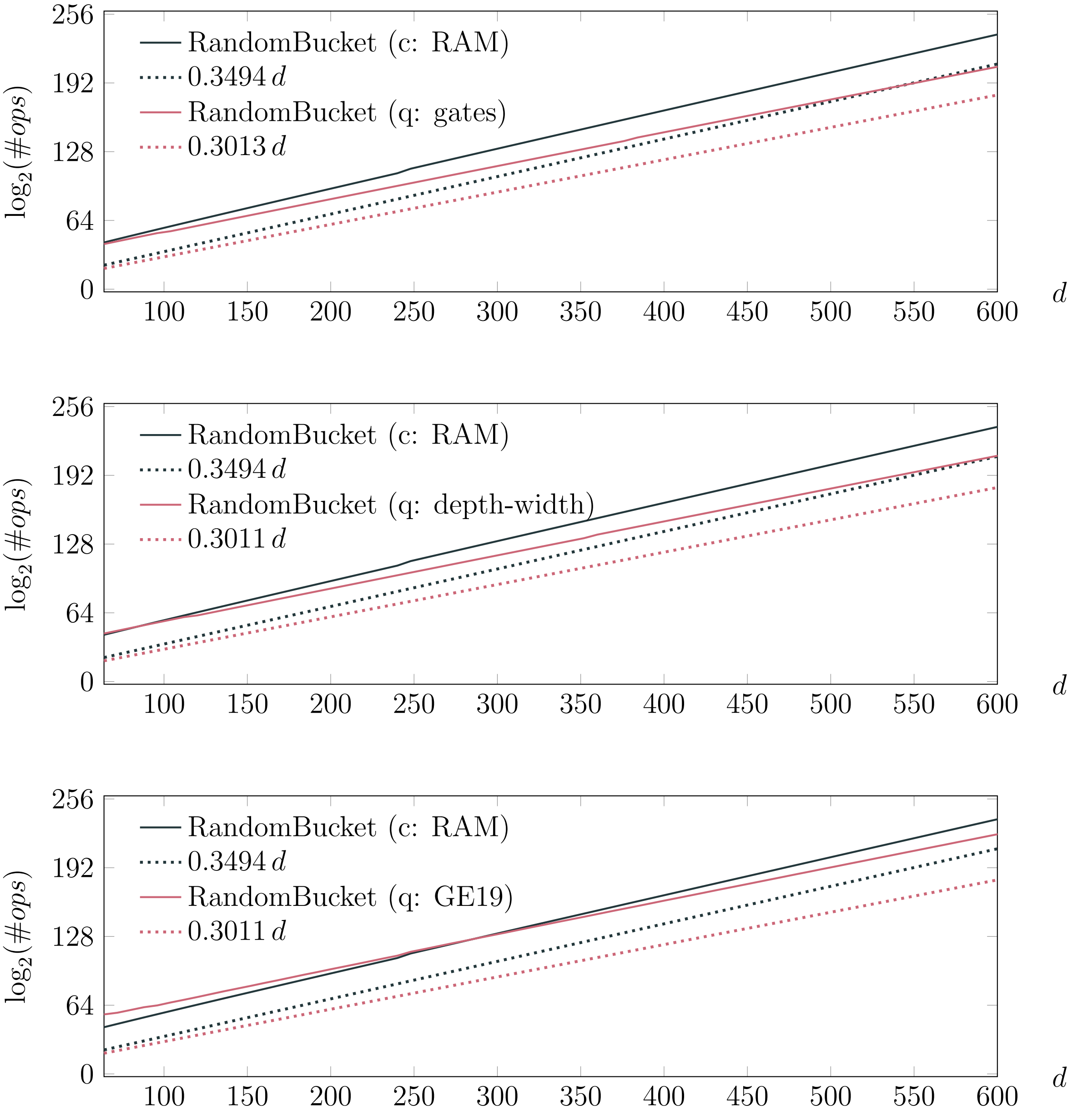
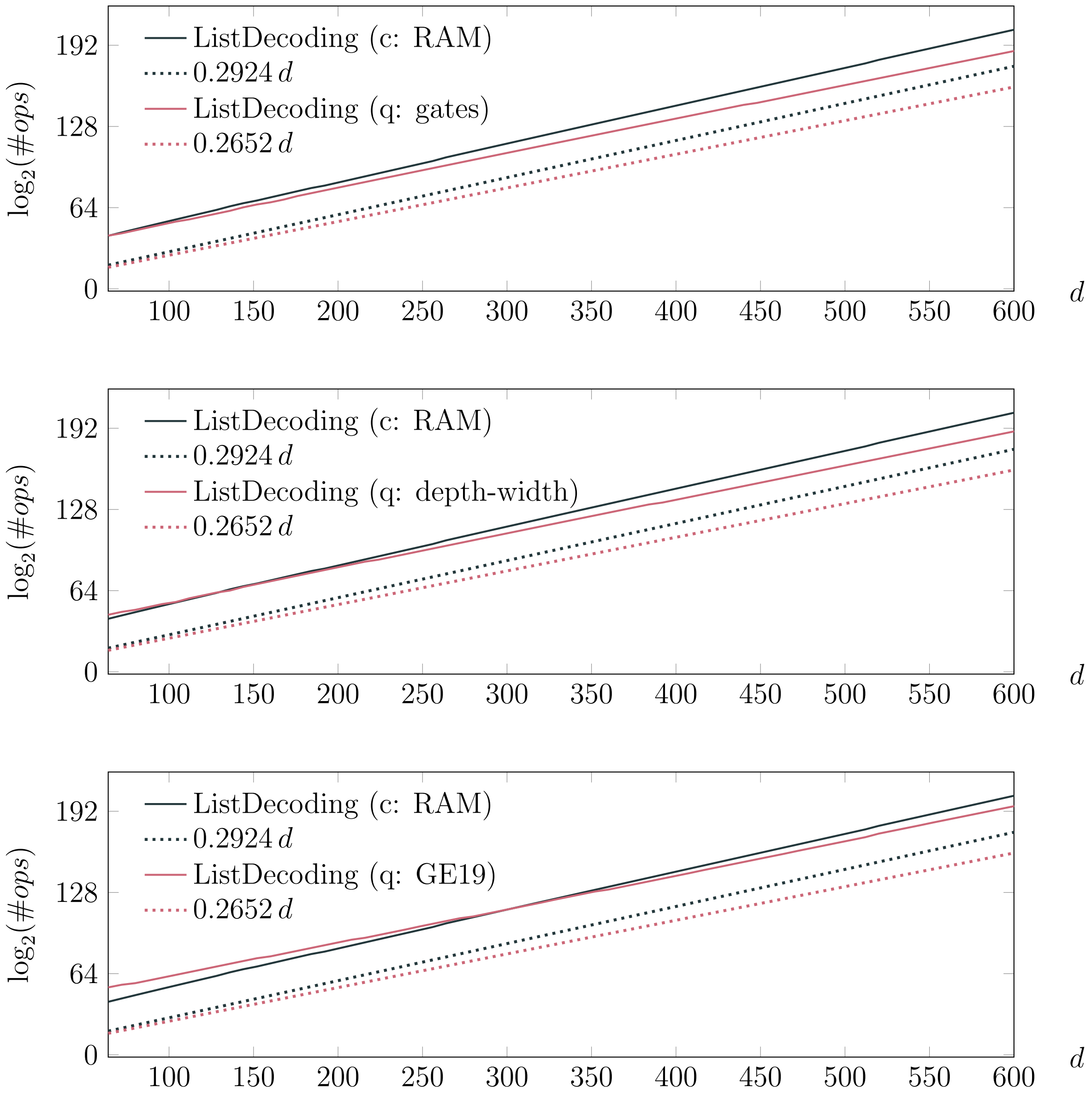


Figure 3: Quantum ("q") and classical ("c") resource estimates for ListDecodingSearch. The three plots, respectively, show crossover points in dimension 72 under the gate cost metric, dimension 136 under the depth-width metric, and dimension 304 under the Gidney–Ekerå cost metric.



7.1 Barriers to a quantum advantage

As expected, the plots indicate that quantum search provides a substantial savings over classical search asymptotically. Our plots fully contain the range of costs from 2^{128} to 2^{256} that are commonly thought to be cryptanalytically interesting. Modest cost improvements are attained in this range.

The range of parameters in which a sieve could conceivably be run, however, is much narrower. If one assumes a memory density of one petabyte per gram (2^{53} bits per gram), a 2^{140} bit memory would have a mass comparable with that of the Moon. Supposing that a 2-sieve stores $1/C_d(\pi/3)$ vectors, and that each vector is $\log_2(d)$ bits, an adversary with a 2^{140} bit memory could only run a sieve in dimension 608 or lower. The potential cost improvement in dimension 608 is smaller than the potential cost improvement in, say, dimension 1000. The potential cost improvement that can be actualised is likely smaller still.

We expect that our cost estimates are underestimates. However, the quantum advantage could grow, shrink, or even be eliminated if our underestimates do not affect quantum and classical costs equally. In this section, we list several reasons to think that the advantage might shrink or disappear.

Error correction overhead.

By using the depth-width metric for quantum circuits, we assume that dispatching a logical gate to a logical qubit costs one RAM instruction. In practice, however, the cost depends on the error correcting code that is used for logical qubits. This cost may be significant.

Gidney and Ekerå have estimated the resources required to factor a 2048 bit RSA modulus using Shor's algorithm on a surface code based quantum computer [22]. Under a plausible assumption on the physical qubit error rate, they calculate that a factoring circuit with $2^{12.6}$ logical qubits and depth 2^{31} requires a distance $\delta = 27$ surface code. Each logical qubit is encoded in $2\,\delta^2 = 1458$ physical qubits, and the error tracking routine applies at least $\delta^2 = 729$ bit instructions, per logical qubit per layer of logical circuit depth, to read its input.

In general, a circuit of depth D and width W requires a distance $\delta = \Theta(\log(DW))$ surface code. To perform a single logical gate, classical control hardware dispatches several instructions to each of the $\Theta(\log^2(DW))$ physical qubits. The classical control hardware also performs a non-trivial error tracking routine between logical gates, which takes measurement results from half of the physical qubits as input.[7] Consequently, the cost of surface code computation grows like $\Omega(DW \log^2(DW))$.

We have adapted scripts provided by Gidney and Ekerå to estimate δ for our circuits. The third plot in Figures 2 and 3 shows the cost of ListDecodingSearch when every logical gate (including the identity) is assigned a cost of δ^2. For ListDecodingSearch the cost in the Gidney–Ekerå metric grows from 2^{128} to 2^{256} between dimensions 352 and 824, and we calculate a 2^{128} bit memory is sufficient to run in dimension 544. We find that the advantage of quantum search over classical search is a factor of $2^{1.3}$ in dimension 352, a factor of $2^{6.8}$ in dimension 544, and a factor of $2^{14.0}$ in dimension 824. Compare this with the naïve estimate for the advantage, $2^{0.292d-0.265d}$, which is a factor of $2^{9.5}$ in dimension 352, a factor of $2^{14.7}$ in dimension 544, and a factor of $2^{22.5}$ in dimension 824.

One should also note that error correction for the surface code sets a natural clock speed, which Gidney and Ekerå estimate at one cycle per microsecond. Gidney and Ekerå estimate that their factoring circuit, the cost of which is dominated by a single modular exponentiation, would take 7.44 hours to run. This additional overhead in terms of time is not refelected in the instrution count.

On the positive side, the Gidney–Ekerå cost model is specific to the surface code architecture. Significant improvements may be possible. Gottesman has shown that an overhead of $\Theta(1)$ physical qubits per logical qubit is theoretically possible [24]. Whether this technique offers lower overhead than the surface code in practice is yet to be seen.

Dependence on qRAM.

Quantum accessible classical memories are used in many quantum algorithms. For example, they are used in black box search algorithms [27], in collision finding algorithms [15], and in some algorithms for the the dihedral hidden subgroup problem [34]. The use of qRAM is not without controversy [12, 26]. Previous work on quantum lattice sieve algorithms [36, 37] has noted that constructing practical qRAM seems challenging.

Morally, looking up an ℓ bit value in a table with 2^n entries should have a cost that grows at least with $n+\ell$. Recent results [6, 7, 42] indicate that realistic implementations of qRAM have costs that grow much more quickly than this. When ancillary qubits are kept to a minimum, the best known Clifford+T implementation of a qRAM has a **T** count of $4 \cdot (2^n - 1)$ [7]. While it is conceivable that a qRAM could be constructed

[7] For a thorough introduction to how logical gates are performed on the surface code see [21], and for more advanced techniques see e.g. [29].

at lower cost on a different architecture, as has been suggested in [23], a unit cost qRAM gate should be seen as a powerful, and potentially unrealistic, resource.

One can argue that classical RAMs also have a large cost. This is not to say that classical and quantum RAMs have the same cost. A qRAM can be used to construct an arbitrary superposition over the elements of a memory. This process relies on quantum interference and necessarily takes as long as a worst case memory access time. This is in contrast with classical RAM, where careful programming and attention to a computer's caches can mask the fact that accessing an N bit memory laid out in a 3-dimensional space necessarily takes $\Omega(N^{1/3})$ time.

If the cost of a qRAM gate is equivalent to $\Theta(N^{1/3})$ Clifford+T gates, then the asymptotic cost of quantum AllPair search is $2^{(0.380\ldots+o(1))d}$, the asymptotic cost of quantum RandomBucket search is $2^{(0.336\ldots+o(1))d}$, and the asymptotic cost of quantum ListDecoding search is $2^{(0.284\ldots+o(1))d}$. If memory is constrained to two dimensions, and qRAM costs $\Theta(N^{1/2})$ Clifford+T gates, the quantum asymptotics match the classical RAM asymptotics.

Quantum sampling routines.

We have assumed that $\mathbf{D}$ in Section 4.1 (the uniform sampling subroutine in Grover's algorithm) is implemented using parallel $\mathbf{H}$ gates. This is the smallest possible circuit that might implement $\mathbf{D}$, and may be a significant underestimate. In Line 12 of Algorithm 11 we must construct a superposition (ideally uniform) over $\{k : v_k \in L_{F,j}\}$. The set $L_{F,j}$ is presented as a disjoint union of smaller sets. Copying the elements of these smaller sets to a flat array would be more expensive than our estimate for the cost of search. While we do not expect the cost of sampling near uniformly from $L_{F,j}$ to be large, it could easily exceed the cost of popcount.

7.2 Relevance to SVP

The NNS algorithms that we have analysed are closely related to lattice sieves for SVP. While the asymptotic cost of NNS algorithms are often used as a proxy for the asymptotic cost of solving SVP, we caution the reader against making this comparison in a non-asymptotic setting. On the one hand, our estimates might lead one to underestimate the cost of solving SVP:

- the costs given in Figures 2 and 3 represent one iteration of NNS within a sieve, while sieve algorithms make $\mathsf{poly}(d)$ iterations;

- the costs given in Figures 2 and 3 do not account for all of the subroutines within each NNS algorithm.

On the other hand, our estimates might lead one to overestimate the cost of solving SVP:

- it is a mistake to conflate the cost of NNS in dimension d with the cost of SVP in dimension d. The "dimensions for free" technique of [19] can be used to solve SVP in dimension d by calling an NNS routine polynomially many times in dimension $d' < d$. Our analysis seamlessly applies to dimension d';
- there are heuristics that exploit structure present in applications to SVP not captured in our general setting, e.g. the vector space structure allowing both $\pm u$ to be tested for the cost of u, and keeping the vectors sorted by length.

7.3 Future Work

The sieving techniques considered here are not exhaustive. While it would be relatively easy to adapt our software to other 2-sieves, like the cross polytope sieve [11], future work might consider k-sieves such as [8, 32].

Future work might also address the barriers to a quantum advantage discussed in Section 7.1. Two additional barriers are worth mentioning here. First, as Grover search does not parallelise well, one might consider depth restrictions for classical and quantum circuits. Second, our estimates might be refined by including some of the classical subroutines, present in both the classical and quantum variants of the same sieve, that we have ignored, e.g. the cost of sampling lattice vectors or the cost of list-decoding in Algorithm 11. Any cost increase will reduce the range of cryptanalytically relevant dimensions, giving fewer dimensions to overcome quantum overheads.

Finally, our estimates should be checked against experiments. Our analysis of Algorithm 10 recommends a database of size $N(d) \approx 2/C_d(\pi/3)$, while the largest sieving experiments to date [2] runs Algorithm 10 with a database of size $N'(d) = 3.2 \cdot 2^{0.2075d}$ up to dimension $d = 127$. There is a factor of 8 gap between $N'(127)$ and $N(127)$. A factor of two can be explained by the fact that [2] treats each database entry u as $\pm u$. It is possible that the remaining factor of four can be explained by the other heuristics used in [2]. As d increases, $N(d)$ and $N'(d)$ continue to diverge, so future work could attempt to determine more accurately the required list size.

Bibliography

[1] Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: 33rd ACM STOC. pp. 601–610. ACM Press (Jul 2001)

[2] Albrecht, M.R., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E.W., Stevens, M.: The general sieve kernel and new records in lattice reduction. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 717–746. Springer, Heidelberg (May 2019)

[3] Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - A new hope. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 327–343. USENIX Association (Aug 2016)

[4] Amy, M., Maslov, D., Mosca, M., Roetteler, M.: A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 32(6), 818–830 (June 2013)

[5] Amy, M., Matteo, O.D., Gheorghiu, V., Mosca, M., Parent, A., Schanck, J.M.: Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In: Avanzi, R., Heys, H.M. (eds.) SAC 2016. LNCS, vol. 10532, pp. 317–337. Springer, Heidelberg (Aug 2016)

[6] Arunachalam, S., Gheorghiu, V., Jochym-O'Connor, T., Mosca, M., Srinivasan, P.V.: On the robustness of bucket brigade quantum RAM. New Journal of Physics 17(12), 123010 (2015), http://stacks.iop.org/1367-2630/17/i=12/a=123010

[7] Babbush, R., Gidney, C., Berry, D.W., Wiebe, N., McClean, J., Paler, A., Fowler, A., Neven, H.: Encoding electronic spectra in quantum circuits with linear T complexity. Phys. Rev. X 8, 041015 (Oct 2018), https://link.aps.org/doi/10.1103/PhysRevX.8.041015

[8] Bai, S., Laarhoven, T., Stehlé, D.: Tuple lattice sieving. LMS Journal of Computation and Mathematics 19(A), 146–162 (2016)

[9] Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Krauthgamer, R. (ed.) 27th SODA. pp. 10–24. ACM-SIAM (Jan 2016)

[10] Becker, A., Gama, N., Joux, A.: Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. Cryptology ePrint Archive, Report 2015/522 (2015), http://eprint.iacr.org/2015/522

[11] Becker, A., Laarhoven, T.: Efficient (ideal) lattice sieving using cross-polytope LSH. In: Pointcheval, D., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 16. LNCS, vol. 9646, pp. 3–23. Springer, Heidelberg (Apr 2016)

[12] Bernstein, D.J.: Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete? Workshop Record of SHARCS'09: Special-purpose Hardware for Attacking Cryptographic Systems (2009), http://cr.yp.to/papers.html#collisioncost

[13] Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte der Physik 46(4-5), 493–505 (1998), https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291521-3978%28199806%2946%3A4/5%3C493%3A%3AAID-PROP493%3E3.0.CO%3B2-P

[14] Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. Contemporary Mathematics 305, 53–74 (2002), https://arxiv.org/abs/quant-ph/0005055

[15] Brassard, G., Høyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. SIGACT News 28(2), 14–19 (Jun 1997), http://doi.acm.org/10.1145/261342.261346

[16] Charikar, M.: Similarity estimation techniques from rounding algorithms. In: 34th ACM STOC. pp. 380–388. ACM Press (May 2002)

[17] Cuccaro, S.A., Draper, T.G., Kutin, S.A., Moulton, D.P.: A new quantum ripple-carry addition circuit (2004), arXiv:quant-ph/0410184

[18] Draper, T., Kutin, S., Rains, E., Svore, K.M., Svore, K.M.: A logarithmic-depth quantum carry-lookahead adder. Quantum Info. Comput. pp. 351–369 (January 2006), https://www.microsoft.com/en-us/research/publication/a-logarithmic-depth-quantum-carry-lookahead-adder/, lANL ArXiv: quant-ph/0406142.

[19] Ducas, L.: Shortest vector from lattice sieving: A few dimensions for free. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 125–145. Springer, Heidelberg (Apr / May 2018)

[20] Fitzpatrick, R., Bischof, C.H., Buchmann, J., Dagdelen, Ö., Göpfert, F., Mariano, A., Yang, B.Y.: Tuning GaussSieve for speed. In: Aranha, D.F., Menezes, A. (eds.) LATINCRYPT 2014. LNCS, vol. 8895, pp. 288–305. Springer, Heidelberg (Sep 2015)

[21] Fowler, A.G., Mariantoni, M., Martinis, J.M., Cleland, A.N.: Surface codes: Towards practical large-scale quantum computation. Phys. Rev. A 86, 032324 (Sep 2012), https://link.aps.org/doi/10.1103/PhysRevA.86.032324

[22] Gidney, C., Ekerå, M.: How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits (2019), https://arxiv.org/abs/1905.09749, arXiv:1905.09749

[23] Giovannetti, V., Lloyd, S., Maccone, L.: Quantum random access memory. Phys. Rev. Lett. 100, 160501 (Apr 2008), http://link.aps.org/doi/10.1103/PhysRevLett.100.160501

[24] Gottesman, D.: Fault-tolerant quantum computation with constant overhead (2013), https://arxiv.org/abs/1310.2984, arXiv:1310.2984

[25] Grassl, M., Langenberg, B., Roetteler, M., Steinwandt, R.: Applying Grover's algorithm to AES: Quantum resource estimates. In: Takagi, T. (ed.) Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016. pp. 29–43. Springer, Heidelberg (2016)

[26] Grover, L., Rudolph, T.: How significant are the known collision and element distinctness quantum algorithms. Quantum Info. Comput. 4, 201–206 (May 2004)

[27] Grover, L.K.: Quantum mechanics helps in searching for a needle in a haystack. Phys. Rev. Lett. 79, 325–328 (Jul 1997), http://link.aps.org/doi/10.1103/PhysRevLett.79.325

[28] Häner, T., Roetteler, M., Svore, K.M.: Factoring using 2n + 2 qubits with Toffoli based modular multiplication. Quantum Info. Comput. 17(7-8), 673–684 (Jun 2017), http://dl.acm.org/citation.cfm?id=3179553.3179560

[29] Horsman, C., Fowler, A.G., Devitt, S., Meter, R.V.: Surface code quantum computing by lattice surgery. New Journal of Physics 14(12), 123011 (2012), http://stacks.iop.org/1367-2630/14/i=12/a=123011

[30] Jaques, S., Naehrig, M., Roetteler, M., Virdia, F.: Implementing Grover oracles for quantum key search on AES and LowMC. Cryptology ePrint Archive, Report 2019/1146 (2019), https://eprint.iacr.org/2019/1146

[31] Jaques, S., Schanck, J.M.: Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 32–61. Springer, Heidelberg (Aug 2019)

[32] Kirshanova, E., Mårtensson, E., Postlethwaite, E.W., Moulik, S.R.: Quantum algorithms for the approximate k-list problem and their application to lattice sieving. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019. pp. 521–551. Springer International Publishing, Cham (2019)

[33] Klein, P.N.: Finding the closest lattice vector when it's unusually close. In: Shmoys, D.B. (ed.) 11th SODA. pp. 937–941. ACM-SIAM (Jan 2000)

[34] Kuperberg, G.: Another subexponential-time quantum algorithm for the Dihedral Hidden Subgroup Problem. In: Theory of Quantum Computation, Communication and Cryptography – TQC 2013. pp. 20–34. LIPIcs 22 (2013), http://drops.dagstuhl.de/opus/volltexte/2013/4321

[35] Laarhoven, T.: Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 3–22. Springer, Heidelberg (Aug 2015)

[36] Laarhoven, T.: Search problems in cryptography: from fingerprinting to lattice sieving. Ph.D. thesis, Department of Mathematics and Computer Science (2 2016), proefschrift

[37] Laarhoven, T., Mosca, M., van de Pol, J.: Solving the shortest vector problem in lattices faster using quantum search. In: Gaborit, P. (ed.) Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013. pp. 83–101. Springer, Heidelberg (Jun 2013)

[38] Lee, Y., Kim, W.C.: Concise formulas for the surface area of the intersection of two hyperspherical caps. Tech. rep., KAIST (February 2014), https://ie.kaist.ac.kr/uploads/professor/tech_file/Concise+Formulas+for+the+Surface+Area+of+the+Intersection+of+Two+Hyperspherical+Caps.pdf

[39] Li, S.: Concise formulas for the area and volume of a hyperspherical cap. Asian Journal of Mathematics and Statistics 4(1), 66–70 (2011)

[40] Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 319–339. Springer, Heidelberg (Feb 2011)

[41] Maslov, D.: Advantages of using relative-phase Toffoli gates with an application to multiple control Toffoli optimization. Physical Review A 93(2), 022311 (2016)

[42] Matteo, O.D., Gheorghiu, V., Mosca, M.: Fault tolerant resource estimation of quantum random-access memories (2019), arXiv:1902.01329v1

[43] Micciancio, D., Regev, O.: Lattice-based cryptography. In: Bernstein, D.J., Buchmann, J., Dahmen, E. (eds.) Post-Quantum Cryptography, pp. 147–191. Springer Berlin Heidelberg, Berlin, Heidelberg (2009), https://doi.org/10.1007/978-3-540-88702-7_5

[44] Nguyen, P.Q., Vidick, T.: Sieve algorithms for the shortest vector problem are practical. J. of Mathematical Cryptology 2(2) (2008)

[45] Parhami, B.: Efficient Hamming weight comparators for binary vectors based on accumulative and up/down parallel counters. IEEE Trans. on Circuits and Systems 56-II(2), 167–171 (2009), https://doi.org/10.1109/TCSII.2008.2010176

[46] Takahashi, Y., Kunihiro, N.: A fast quantum circuit for addition with few qubits. Quantum Info. Comput. 8(6), 636–649 (Jul 2008), http://dl.acm.org/citation.cfm?id=2016976.2016981

A Caps and wedges

The expression of $C_d(\theta)$ in terms of the regularised incomplete beta function, $C_d(\theta) = \frac{1}{2}I_{\sin^2(\theta)}\left(\frac{d-1}{2}, \frac{1}{2}\right)$, is perhaps not as well known as it should be.[8] Apart from being concise, this representation makes it clear that $C_d(\theta)$ has a hypergeometric representation (see https://dlmf.nist.gov/8.17#E7). Computer algebra systems often provide routines for computing hypergeometric functions that are more robust than generic numerical integration routines.

An exact expression for $W_d(\theta, \theta_u, \theta_v)$ involves a subtle case analysis [38]. Fortunately, we only need the following case.

Fact A.1 ([38, Case 8]). *Let $\theta, \theta_u, \theta_v$ be real numbers is in $(0, \pi/2)$ with $\theta < \theta_u + \theta_v$ and $(\cos(\theta_v) - \cos(\theta_u)\cos(\theta))(\cos(\theta_v)\cos(\theta) - \cos(\theta_u)) < 0$. Define $\theta^* \in (0, \pi/2)$ by $\cos(\theta_v)\sec(\theta^*) = \cos(\theta_u)\sec(\theta - \theta^*)$. For any $u, v \in \mathcal{S}^{d-1}$ with $\langle u, v\rangle = \cos(\theta)$, the wedge $\mathcal{W}^{d-1}(u, \theta_u, v, \theta_v)$ has μ^{d-1} measure*

$$W_d(\theta, \theta_u, \theta_v) = J_d\left(\theta^*, \theta_v\right) + J_d\left(\theta - \theta^*, \theta_u\right)$$

[8] We learnt this from [39] via [38]. The identity is easy to check

$$\begin{aligned}\frac{1}{2}I_{\sin^2(\theta)}\left(\tfrac{d-1}{2}, \tfrac{1}{2}\right) &= \frac{1}{2\sqrt{\pi}}\frac{\Gamma(\frac{d}{2})}{\Gamma(\frac{d-1}{2})}\int_0^{\sin^2(\theta)} t^{\frac{d-1}{2}-1}(1-t)^{-\frac{1}{2}}\ \mathrm{d}t \\ &= \frac{1}{\sqrt{\pi}}\frac{\Gamma(\frac{d}{2})}{\Gamma(\frac{d-1}{2})}\int_0^{\theta} \sin^{d-2}(t)\ \mathrm{d}t.\end{aligned}$$

where

$$J_d(t_1, t_2) = \frac{1}{\sqrt{\pi}} \frac{\Gamma\left(\frac{d}{2}\right)}{\Gamma\left(\frac{d-1}{2}\right)} \int_{t_1}^{t_2} \sin^{d-2}(\varphi) C_{d-1}\left(\arccos\left(\frac{\tan(t_1)}{\tan(\varphi)}\right)\right) \mathrm{d}\varphi.$$

While we use fixed dimension in this work, asymptotics for $C_d(\theta)$ and $W_d(\theta, \theta_u, \theta_v)$, as a function of d, are occasionally useful. For fixed θ it follows immediately from the integral representation of $C_d(\theta)$ above that $C_d(\theta) = (\sin\theta)^{(1+o(1))d}$. The expression for W_d is slightly more complicated. For fixed θ, θ_u, and θ_v let θ^* be such that $\cos\theta_u \sec\theta^* = \cos\theta_v \sec(\theta - \theta^*)$, and let $\varphi = \arccos(\cos\theta_u \sec\theta^*)$. Note that $\theta^* = \theta/2$ when $\theta_u = \theta_v$. The argument in [9, Appendix A] establishes that $W_d(\theta, \theta_u, \theta_v) = (\sin\varphi)^{(1+o(1))d}$.

Remark A.1. *The constraint* $(\cos(\theta_v) - \cos(\theta_u)\cos(\theta))(\cos(\theta_v)\cos(\theta) - \cos(\theta_u)) < 0$ *in the statement of Fact A.1 ensures that u and v lie on opposite sides of the hyperplane orthogonal to $w = u/\cos(\theta_u) - v/\cos(\theta_v)$. The quantity* $\cos(\theta_v)\sec(\theta^*)$ *that appears in the statement of Fact A.1 is equal to the quantity γ that appears in the statement of [9, Lemma 2.2]. It is the norm of the shortest $x \in \mathbb{R}^n, \|x\| \leq 1$ with $\langle x, u\rangle = \cos(\theta_u)$, $\langle x, v\rangle = \cos(\theta_v)$, and $\langle x, w\rangle = 0$. The entire wedge is contained in a cap of angle* $\arccos(\|x\|)(= \arccos(\gamma))$ *about this shortest x. The proof of [9, Lemma 2.2] establishes that the wedge is essentially a factor of $\sqrt{d}$ smaller than this cap asymptotically, i.e. $W_d(\theta, \theta_u, \theta_v) = \Theta\left(\frac{1}{d^2}(1-\gamma^2)^{d/2}\right)$.*

B Toffoli counts and circuit width

Let the notation be as in Section 4.1. The Toffoli gate is often used in circuit design. Here we count the Toffoli (or $\mathbf{T}$) gates required to implement $\mathbf{G}(g)$ when $g(i) = \texttt{popcount}_{k,n}(u, v_i)$. Recall $\mathbf{G}(g) = \mathbf{D}\mathbf{R}_0\mathbf{D}^{-1}\mathbf{R}_g$ and that $\mathbf{R}_0$ is implemented with a "multiply controlled" Toffoli. For some $m \geq 3$ the m bit Toffoli gate is $TOF^m \colon \{0,1\}^m \to \{0,1\}^m, (x_1, \ldots, x_{m-1}, x_m) \mapsto (x_1, \ldots, x_{m-1}, (x_1 \wedge \cdots \wedge x_{m-1}) \oplus x_m)$. The $m = 3$ case is referred to as a Toffoli gate, rather than, as for $m \geq 4$, a "multiply controlled" Toffoli gate. The Toffoli count is a pertinent quantity because it determines the $\mathbf{T}$ count. The $\mathbf{T}$ count plays a large part in determining the distance required for error correction, and hence the cost of error correction. In the $m \geq 4$ case, i.e. for $\mathbf{R}_0$, more efficient decompositions into $\mathbf{T}$ gates are possible [41] and therefore our software does not count Toffoli gates, but rather directly counts $\mathbf{T}$ gates.

The Toffoli Count of $\mathbf{R}_g$. The Toffoli count of the full circuit described in Fig 1, i.e. including the uncomputation, is

$$c(\text{CARRY}) + 2\sum_{i=0}^{\ell-2} 2^i c(\text{ADD}_{\ell-i-1}), \tag{17}$$

where $c(\cdot)$ denotes the Toffoli count of its argument. The factor of 2 accounts for uncomputation and, as explained in Section 4.1, the CARRY circuit is only cost once. The i bit full adder ADD_i of [17] has a Toffoli count of $2i - 1$. The CARRY circuit of [28] has a Toffoli count of $4(\ell - 2) + 2$. A routine calculation gives the overall Toffoli count as $3n - 5$. When decomposing Toffoli gates ($m = 3$) into $\mathbf{T}$ gates we make use of [4]. Roughly, a Toffoli gate costs 7 $\mathbf{T}$ gates.

The Toffoli Count of $\mathbf{D}\mathbf{R}_0\mathbf{D}^{-1}$. As explained in Section 4.1, $\mathbf{D}$ and $\mathbf{D}^{-1}$ comprise solely of $\mathbf{H}$ gates. We decompose the multiply controlled Toffoli into $\mathbf{T}$ gates via [41, Table I]. In particular we use the bottommost "Ours" row for TOF^4 and $TOF^m, m \geq 5$.

The circuit width of $\mathbf{R}_g$. The first layer of adders in Figure. 1 requires $3 \times 2^{\ell-2}$ qubits. The subsequent layers of adders reuse the qubits from their "parent" layers, e.g. the adders ADD_2 in the second layer reuse the qubits from the layer consisting of ADD_1 adders, with the exception of the input carry for each adder. Hence the total number of qubits required by the adders is simply equal to the number of input lines (that end with a black diamond), i.e. $2^\ell - 1$.

The CARRY in Fig. 1 requires 2ℓ input qubits and one dirty ancilla, ℓ of which are reused from the last adder.

Therefore, the total number of qubits required by $\mathbf{R}_g$ is

$$\text{width} = (2^\ell - 1) + \ell + 1 = 2^\ell + \ell. \tag{18}$$

C The choice of adder

Let the notation be as in Section 4.1. The i bit adders of [18, 46] function similarly to [17] but instead require $O(i)$ and $O(i/\log i)$ ancillae, respectively. While these adders require more ancillae, they have an optimal depth of $O(\log i)$ compared to the depth $O(i)$ of the [17] adder. We argue below to not consider [18, 46] when costing our circuit.

The largest n we consider is 32767 and therefore $\ell = 15$ and the largest adder we require has $i = 14$ bits. Such an in-place adder, with incoming carry bit, of [17, Table. 1] has depth 34, width 29 and requires 27 Toffoli gates. Such an in-place adder, with incoming carry bit, of [18, Table. 2] has depth 18, width 50 and requires 102 Toffoli gates. The adder of [46] does not allow an incoming carry, and hence Figure 1 would require $2^{\ell-1} = n + 1$, i.e. $\ell = 16$ and the largest adder to have $i = 15$ bits. Following the discussion at the end of [46, Section. 3.4], while ultimately the depth-width is $O(i)$ compared to $O(i \log i)$ of [18], for our small i these circuits are larger—ignoring the asymptotic subtracted terms and requiring $i = 15$, we get depth approximately 90, width approximately 45 and require approximately 435 Toffoli gates.

Therefore, while we have not explicitly constructed the relevant circuits, we expect any gain due to the smaller depths of [18, 46] would be minimal at best for our parameters, given the extra ancillae. Certainly, if one were to consider error correction, such a gain would be outweighed entirely by their higher Toffoli count.

D Additional figures

Figure 4 provides estimates for AllPairSearch from Section 6.1.

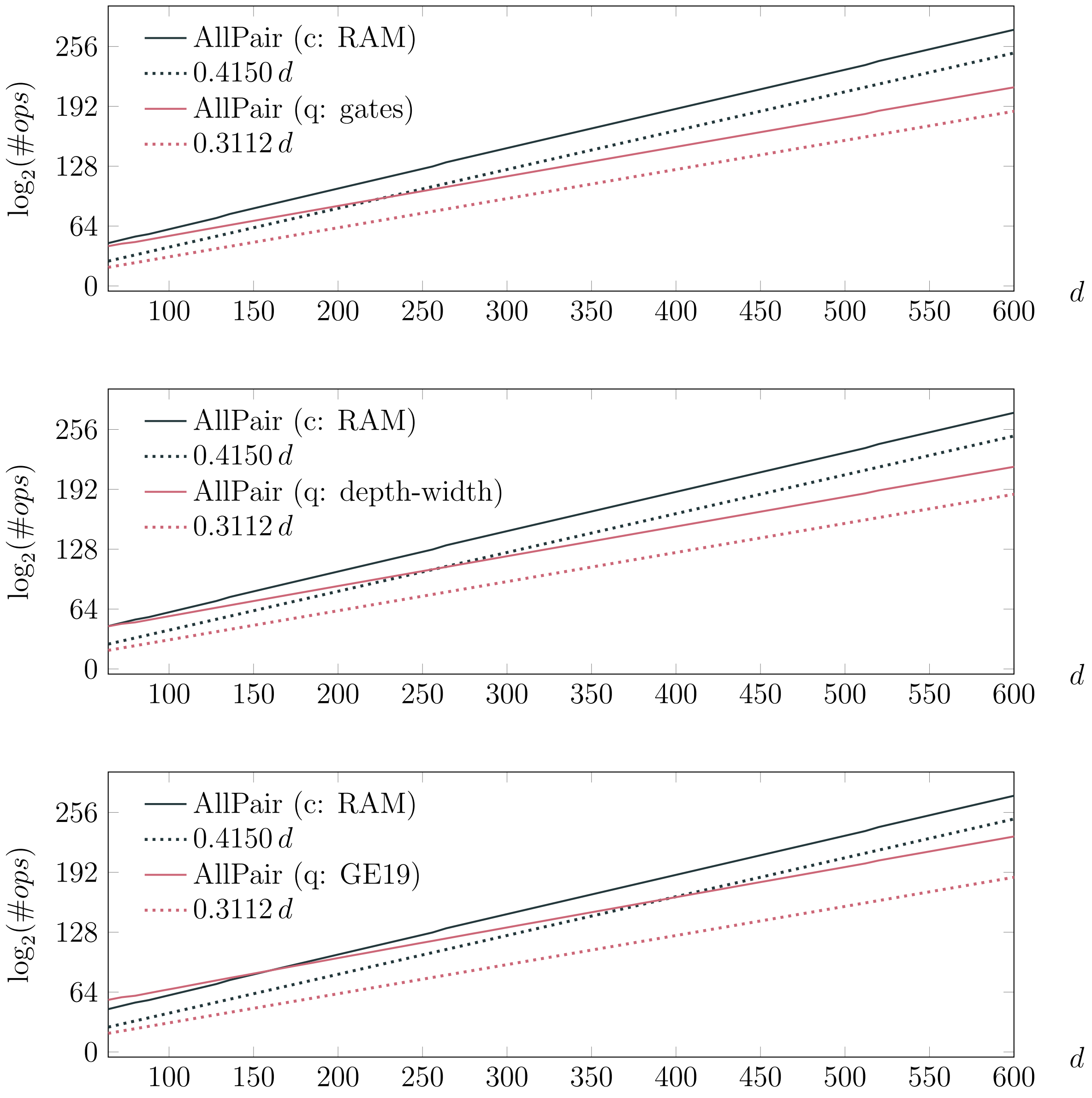
E Location of supplemental materials

The source code and data for this paper can be found on
https://jmschanck.info/code/20200701-schanck-thesis-supplemental.tar.gz.

A backup has also been uploaded to the Internet Archive:
https://archive.org/details/20200701-schanck-thesis-supplemental.

Other versions of this paper are available at:
https://eprint.iacr.org/2019/1161.

Figure 4: Quantum (“q”) and classical (“c”) resource estimates for AllPairSearch. The three plots, respectively, show crossover points in dimension ≤ 64 under the gate cost metric, dimension ≤ 64 under the depth-width metric, and dimension 160 under the Gidney–Ekerå cost metric.



Chapter 4

RSA

It is clear that the standard size of 512 bits no longer provides adequate protection, and should be substantially increased.

Adi Shamir (1995)

Context

In celebrating the successful factorization of $2^{95} + 1$ by mechanical computer, Derrick N. Lehmer wrote: “the mathematician reaches out with his high-powered machines and his high-powered theorems and investigates the internal structure of his distant bodies much as the astronomer inquires into the structure of some distant star. [...] For over fifty years this number has stood on the outposts of the number system, a challenge to explorers in this field of thought.”

How much farther will we see with quantum computers?

The answer depends, of course, on the form of the factorization. Shor’s algorithm can split generic composites. Its asymptotic complexity suggests that we might some day factor generic integers that are as large as the Mersenne numbers that can be tested currently with the Lucas–Lehmer primality test.[1] Variants of Shor’s algorithm might see farther for numbers of a special form.

In the paper below I examine the use of Shor’s algorithm to factor numbers of a special form. For any number N of the form that I consider, the element $3^N \bmod N$ is of small multiplicative order. Given $3^N \bmod N$, the cost of Shor’s algorithm is dramatically reduced. However, if one equates bit operations and quantum gates, then computing $3^N \bmod N$ (classically) becomes the dominant cost.

With quantum computers, the outposts of the number system will recede into the distance. But to predict how far, we will need to determine the relative cost of quantum and classical operations.

[1] The largest known Mersenne prime is 27.5 megabytes when fully expressed in base 2.

Multi-power Post-quantum RSA

John M. Schanck

Abstract Special purpose factoring algorithms have discouraged the adoption of multi-power RSA, even in a post-quantum setting. We revisit the known attacks and find that a general recommendation against repeated factors is unwarranted. We find that one-terabyte RSA keys of the form $n = p_1^2 p_2^3 p_3^5 p_4^7 \cdots p_i^{\pi_i} \cdots p_{20044}^{225287}$, with π_i the i-th prime, are competitive with one-terabyte RSA keys of the form $n = p_1 p_2 p_3 p_4 \cdots p_i \cdots p_{2^{31}}$. Prime generation can be made to be a factor of 100 000 times faster at a loss of at least 1 but not more than 17 bits of security against known attacks. The range depends on the relative cost of bit and qubit operations under the assumption that qubit operations cost 2^c bit operations for some constant c.

1 Introduction

An RSA modulus is a "publicly specified product, n, of two large secret prime numbers p and q" [17]. This – a direct quote from the paper that introduced RSA – is an uncontroversial and historically accurate definition. But, while $n = pq$ is the most common form for RSA moduli in deployment, it is not clear that things *should* be this way. A lot has changed since [17] was written. The bit-length of n that provide "adequate security" has increased decade by decade. Why hasn't the number of factors? Or their multiplicity? Assuming modest, or even dramatic, reductions in the cost of factoring, what shape should RSA moduli take in the long term?

If you adhere to some dictum like *the right form is that which maximizes security as a function of the bit length of* n, then $n = pq$ is quite plausibly the right form. However, security is not the only measure of a cryptosystem. Users consider computational efficiency, compactness, resistance to side-channel attacks, and intellectual property claims, among other attributes. So there may be room for alternative key forms.

Indeed, a number of alternatives to $n = pq$ have been proposed. The fact that it is possible to use $n = p_1^{e_1} p_2^{e_2} \cdots p_\ell^{e_\ell}$ was mentioned in the RSA patent [18]. Other proposals in the literature tend to focus on improving efficiency.

- Takagi [22] has suggested $n = p^r q$. This form admits a fast decryption algorithm based on Takagi's earlier work [21] on "multi-block" RSA with modulus $n^k = (pq)^k$.

- Lim, Kim, Yie, and Lee [11] have suggested $n = p^r q^s$. They claim that $n = p^2 q^3$ is 15 times faster than $n = pq$ for 8192-bit n. They consider, but dismiss, the use of more than two primes.

- Shamir [19] has suggested unbalanced moduli, $n = pq$, with q much larger than p. He has also proposed efficiency enhancements so that large n can be used – large enough that the system is "likely to provide long term security even to professional paranoids."

- Collins, Hopkins, Langford, and Sabin [4] have patented efficient methods for using moduli of the form $n = p_1 p_2 \cdots p_\ell$ where each prime is $\approx (\lg n)/\ell$ bits and $\ell > 2$ is a constant.

- Bernstein, Heninger, Lou, and Valenta [1] have suggested $n = p_1 p_2 \cdots p_\ell$ where each prime is of $(\lg \lg n)^{2+o(1)}$ bits and ℓ grows proportionally with n. They propose key generation, encryption, and decryption routines that each cost $(\lg n)(\lg \lg n)^{O(1)}$ bit operations.

All of these proposals sacrifice the security of the standard RSA key form for greater efficiency – even Shamir's "paranoids" appear to have limited patience for slow cryptography. The efficiency gains are largely in decryption and are due to the decryption algorithms of Quisquater and Couvreur [16] and Takagi [21]. The cost of these decryption algorithms, at least for moduli with a constant number of prime factors, depends on the size of the private primes rather than the size of n.

Special purpose factoring algorithms limit the extent to which one can use small primes in RSA. Lenstra's elliptic curve method (ECM) heuristically finds a prime p that divides n in $\exp\left((\log p)^{1/2+o(1)}\right)$ bit operations. Hence, a system designer who tries to maximize performance for a given bit length must weigh the cost of ECM against the cost of the best general purpose factoring algorithm in his attack model.

The number field sieve, a pre-quantum general purpose factoring algorithm, factors n in $\exp\left((\log n)^{1/3+o(1)}\right)$ bit operations. As such, small private primes weaken security when $\min_{p|n} \log p < (\log n)^{2/3}$, at least asymptotically. The non-asymptotic analysis is somewhat more delicate: for $\lg n \approx 2048$ no more than 3 equally sized private primes should be used, and for $\lg n \approx 4096$ no more than 4 equally sized private primes should be used [8].

Shor's quantum factoring algorithm dramatically reconfigures the relationship between special and general purpose factoring algorithms. Shor's algorithm factors arbitrary n with high probability using only $(\lg n)^{2+o(1)}$ qubit operations. For some n the cost can even be reduced to $(\log n)^{1+o(1)}$ qubit operations. Faced with Shor's algorithm, one has to consider whether honest parties are left with any advantage over attackers.

This advantage can be formalized as a "cost/performance ratio" – the cost of the best attack on the system divided by the cost of using the system.

The proposal of Bernstein, Heninger, Lou, and Valenta [1], which we will call "multi-prime post-quantum RSA" or "multi-prime pqRSA", is an attempt to maximize the cost/performance ratio of RSA against Shor's algorithm. The use of small primes is still a liability in a post-quantum setting, but their use is also a necessity. The cost/performance ratio of RSA with $n = pq$ is constant (assuming qubit operations and bit operations have equal cost), while the cost/performance ratio of multi-prime pqRSA is $(\lg n)^{1+o(1)}$.

Could a different key form provide an even larger post-quantum cost/performance ratio? There is some evidence that other forms are more efficient in practice. Boneh and Shacham [3] have compared multi-prime RSA, $n = p_1p_2p_3$, with Takagi's variant, $n = p_1^2p_2$, and have found that decryption in Takagi's system is faster. This suggests that a multi-power post-quantum RSA, with $n = p_1^{e_1}p_2^{e_2}\dots p_\ell^{e_\ell}$, might have better performance than multi-prime post-quantum RSA. However, attackers might also have better cost.

Bernstein, Heninger, Lou, and Valenta dismiss post-quantum parameterizations of Takagi's system due to worrisome structure. In [1] they write,

> One can try to further accelerate key generation using Takagi's idea [22] of choosing n as $p^{k-1}q$. We point out two reasons that this is worrisome. The first reason is lattice attacks [2]. The second reason is that any nth power modulo n has small order, namely some divisor of $(p-1)(q-1)$; Shor's algorithm finds the order at relatively high speed once the nth power is computed.

We will elaborate on these two concerns in the remainder of this article.

1.1 Structure of this document.

In Sections 2 and 3 we review lattice based factoring and its cost. In Sections 4 and 5 we review Shor's algorithm and its cost. In Section 6 we propose a particular exponent sequence for multi-power post-quantum RSA and examine the implications of this choice. Readers familiar with Coppersmith's technique and Shor's algorithm are encouraged to skip to Section 6 and refer to Sections 3 and 5 as needed.

1.2 Notation.

We use standard notation, $o(\cdot)$, $O(\cdot)$, $\omega(\cdot)$, $\Omega(\cdot)$, for asymptotic growth rates. The natural logarithm is denoted log and the logarithm to base 2 is denoted lg. Euler's totient function is denoted $\phi(n)$. The bit operation cost of $(\lg n)$-bit (modular) multiplication is denoted $\mathcal{M}(\lg n)$.

2 Factoring with Coppersmith's Technique

The first concern, "lattice attacks," refers to Boneh, Durfee, and Howgrave-Graham's work [2] on factoring numbers of the form $n = p^k q$. Their algorithm is based on Coppersmith's technique, and outperforms Lenstra's elliptic curve method for large k. Coppersmith's technique is a general method for finding small modular roots of univariate polynomials. More formally, it solves some instances of the following problem.

Problem 2.1. *Given a rational polynomial*

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0,$$

and positive integers R and P, enumerate all integers x_0 such that $f(x_0) \equiv 0 \pmod{P}$ and $|x_0| < R$.

The difficulty of Problem 2.1 varies dramatically with R and P. The Berlekamp-Zassenhaus-van Hoeij algorithm solves Problem 2.1 in randomized polynomial time when the factorization of P is known. Coppersmith's technique solves some instances even when the factorization of P is not known.

We will describe a formulation of Coppersmith's technique due to Howgrave-Graham [9]. Our emphasis will be on factoring integers of the form $n = p^k q$ with p and q of known bit length and k large. Howgrave-Graham's version of Coppersmith's technique solves a generalization of Problem 2.1 in which only a multiple of P, rather than P itself, is given.

Problem 2.2. *Given a rational polynomial*

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0,$$

positive integers R and N, a real number β, and a promise that N has a divisor P of size N^β, enumerate all integers x_0 such that $f(x_0) \equiv 0 \pmod{P}$ and $|x_0| < R$.

Both variants of the problem have applications to factoring and the cryptanalysis of RSA; Alexander May has compiled an extensive survey of these applications [12].

Suppose that $n = p^k q$ and that we know some bits of p. For example, suppose that we know the high bits of p; or equivalently, that we know an integer u for which $p - u$ is a small (unknown) integer r. Then the rational polynomial

$$f(x) = (u + x)^k/n \tag{1}$$

evaluates to $1/q$ at $x = r$. Likewise $f^2(r) = 1/q^2$, $f^3(r) = 1/q^3$, and every polynomial of the form

$$h(\vec{a}; x) = \sum_{i=0}^{m-1} \sum_{j=0}^{k-1} a_{ik+j} f(x)^i x^j, \tag{2}$$

with $\vec{a} \in \mathbb{Z}^{km}$, satisfies $h(\vec{a}; r) \in q^{-(m-1)}\mathbb{Z}$. In other words, $q^{m-1}h(\vec{a}; r)$ is an integer for every $\vec{a} \in \mathbb{Z}^{km}$. Coppersmith's technique constructs an $\vec{a}$ for which $|h(\vec{a}; r)| < q^{-(m-1)}$ and, consequently, $h(\vec{a}; r) = 0$.

Remarkably, such an $\vec{a}$ can be constructed in polynomial time whenever sufficiently many bits of p are known. This is made precise by Theorem 2.1.

Theorem 2.1. *Let $N = P^k Q$ for positive integers P, Q, and k with $k > 2$. Let $f(x) = a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + a_0$ with $\gcd\{N, a_0, a_1, \ldots, a_k\} = 1$. Let R be such that*

$$R < \frac{1}{2} \left(P^k/Q\right)^{1/(k+1)}.$$

Then all integers x_0 satisfying $f(x_0) \equiv 0 \pmod{P^k}$ and $|x_0| < R$ can be enumerated at a cost of

$$k^{11+o(1)} \lg N + k^{10+o(1)} \lg^2 N$$

bit operations.

2.1 Constructing a short polynomial for fixed R.

The set of rational polynomials of degree at most d is denoted $\mathbb{Q}[x]_{\leq d}$. Observe that $\mathbb{Q}[x]_{\leq d}$ is a $\mathbb{Q}$-vector space of dimension $d + 1$. We equip this space with the usual coefficient norm,

$$\left\| c_0 + c_1 x + \cdots + c_d x^d \right\| = \sqrt{c_0^2 + c_1^2 + \cdots + c_d^2},$$

so that we can view the integer span of a finite collection of elements of $\mathbb{Q}[x]_{\leq d}$ as a lattice.

A basis for a full rank lattice $L \subset \mathbb{Q}[x]_{\leq d}$ is a set of $d+1$ linearly independent polynomials in L. The determinant, or co-volume, of L is $\det L := |\det B|$ where B is the matrix of coefficient vectors of a basis of L.

For a rational number t the evaluation map $x \mapsto t$ is a linear functional on $\mathbb{Q}[x]_{\leq d}$, which one can think of as an inner product of a coefficient vector with $(1, t, t^2, \ldots, t^d)$. Thus, for $g(x) \in \mathbb{Q}[x]_{\leq d}$ we have

$$|g(t)| \leq \sqrt{C_t}\, \|g(x)\| \tag{3}$$

with $C_t = \sum_{i=0}^{d} t^{2i}$. More conveniently, when $|t| \leq 1$ we have

$$|g(t)| < \sqrt{d+1}\, \|g(x)\|\,. \tag{4}$$

This simple fact, the Cauchy-Schwarz inequality, proves to be quite useful. Preserving the motivation and definition of $f(x)$ in Eq. 1, recall that we are looking for a small integer r. The definition of small is made precise by introducing a number R such that $|r| < R$. Suppose $s \in \mathbb{Z}$ and $|s| < R$, then Equation 4 can be applied, with $t = s/R$, to polynomials of the form

$$\hat{h}(\vec{a}; x) = \sum_{i=0}^{m-1} \sum_{j=0}^{k-1} a_{ik+j} f(xR)^i (xR)^j, \quad \text{with } \vec{a} \in \mathbb{Z}^{km}.$$

These polynomials form a lattice in $\mathbb{Q}[x]_{\leq km-1}$. The parameter $\vec{a}$ expresses $\hat{h}(\vec{a}; x)$ in the basis

$$H = \left\{ \begin{array}{rrrcr} 1, & (xR), & (xR)^2, & \ldots, & (xR)^{k-1}, \\ f(xR), & f(xR)(xR), & f(xR)(xR)^2, & \ldots, & f(xR)(xR)^{k-1}, \\ f(xR)^2, & f(xR)^2(xR), & f(xR)^2(xR)^2, & \ldots, & f(xR)^2(xR)^{k-1}, \\ & & \cdots & & \\ f(xR)^{m-1}, & f(xR)^{m-1}(xR), & f(xR)^{m-1}(xR)^2, & \ldots, & f(xR)^{m-1}(xR)^{k-1} \end{array} \right\}.$$

Note that H contains exactly one element of each degree from 0 to $km-1$. So H is in fact a basis for a lattice of rank km. Moreover, the corresponding matrix of coefficient vectors is triangular, so we can easily compute $|\det H|$. The leading coefficient of $f(xR)^i (xR)^j$ is R^{ik+j}/n^i, hence

$$\begin{aligned} |\det H| &= \prod_{i=0}^{m-1} \prod_{j=0}^{k-1} R^{ik+j}/n^i \\ &= R^{km(km-1)/2} n^{-km(m-1)/2}. \end{aligned} \tag{5}$$

The lattice reduction algorithm of Lenstra, Lenstra, and Lovász (LLL) can be used to find "short" elements of a lattice in polynomial time. The exact meaning of "short" is provided by the following fact.

Fact 2.1 (LLL [10]). *Let* $(\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_d)$ *be an* LLL-reduced *basis of a lattice* L*. Then*

$$\|\vec{b}_1\| \leq 2^{(d-1)/4}(\det L)^{1/d}.$$

We omit the precise definition of LLL-reduced; for now it suffices to know that an LLL-reduced basis can be produced, in polynomial time, by applying the LLL algorithm to a basis of L.

If $\hat{h}(x)$ is the first vector obtained by applying LLL to H, then

$$\|\hat{h}(x)\| \leq 2^{(km-1)/4} \left|\det H\right|^{1/km} \tag{6}$$

The question now is whether this is small enough to ensure that $\hat{h}(r/R) = 0$.

2.2 Optimizing R.

We will determine the largest R for which

$$|\hat{h}(s/R)| < q^{-(m-1)} \tag{7}$$

for all $s \in \mathbb{Z}$ with $|s| < R$. When Eq. 7 is satisfied, we are able to recover r, and factor n, by computing the rational roots of $\hat{h}(x)$. Equations 4, 5, and 6 imply

$$\begin{aligned}|\hat{h}(s/R)| &< \sqrt{km}\ \|\hat{h}(x)\| \\ &= \sqrt{km}\ 2^{(km-1)/4} R^{(km-1)/2} n^{-(m-1)/2}.\end{aligned}$$

Observe that $\sqrt{km}\ 2^{(km-1)/4} < 2^{(km-1)/2}$ for integers k and m with $km > 7$. So Equation 7 is satisfied, for $km > 7$, when

$$(2R)^{(km-1)/2} < n^{(m-1)/2} q^{-(m-1)}. \tag{8}$$

Taking logarithms we have

$$\lg 2R < \frac{m-1}{km-1}\left(\lg n - 2\lg q\right). \tag{9}$$

The bound on R in the statement of Theorem 2.1 is recovered by taking $m = k$ and writing $\lg n = k \lg p + \lg_p q$.

3 The cost of lattice attacks.

We will now consider the cost of using Theorem 2.1 to factor n of the form $n = p_1^{e_1} p_2^{e_2} \dots p_L^{e_L}$. For simplicity we will assume that the prime factors of n are of equal bit length and that the e_i are in decreasing order.

Let $E = \sum_{i=1}^{L} e_i$. Suppose $E \geq 5$, so that there exist integers P, Q, and k with $k > 2$ and $n = P^k Q$. Let R be as in Theorem 2.1. An attacker has to guess $\lg P - \lg R$ bits of P to construct an $f(x)$, as in Equation 1, to which Theorem 2.1 can be applied. With the upper bound on R from Theorem 2.1 we have

$$\begin{aligned} \lg P - \lg R &> \lg P - \frac{1}{k+1}(k \lg P - \lg Q) \\ &= \frac{1}{k+1}(\lg P + \lg Q). \end{aligned} \tag{10}$$

The cost of checking a guess is dominated by LLL. Thus, factoring $n = P^k Q$ with Coppersmith's technique costs

$$2^{\lg P - \lg R} \cdot \mathrm{poly}(k, N) > (PQ)^{1/(k+1)} \cdot \mathrm{poly}(k, N) \tag{11}$$

bit operations, where $\mathrm{poly}(k, N)$ is the cost of lattice reduction. The expected cost of guessing $\lg P - \lg R$ bits of P will be comparable to the cost of ECM when $\lg P - \lg R \approx \sqrt{\lg p_1}$. Note that $\lg P + \lg Q$ is at least $(\lg p_1)L$, since each prime dividing n must occur to some power in P or Q. If L is constant with respect to n, as it is for $n = p^k q$, the condition to beat ECM is $k = \Omega(\sqrt{\lg p_1})$, as was reported in [2]. However, if L grows with n, then one needs $k = \Omega(L\sqrt{\lg p_1})$.

The LLL algorithm costs $d^{5+o(1)}(\lg B)^{2+o(1)}$ bit operations where d is the lattice rank and $\lg B$ is the number of bits needed to represent the longest vector of the input basis. The $o(1)$ terms assume fast multiplication techniques.

Much of the algorithmic research on lattice reduction focuses on improving the constant $2^{(d-1)/4}$ in Fact 2.1. In Equation 8 we discarded any impact these algorithms could have, with little consequence.

On the other hand, Stehlé and Nguyen's L^2 variant of LLL [13] does seem to help. They use (carefully analyzed) floating point arithmetic to produce an LLL-reduced basis in $d^{5+o(1)} \lg B + d^{4+o(1)} \lg^2 B$ bit operations. The complexity claim in the statement of Theorem 2.1 is recovered with $m = k$, $d = k^2$, and $\lg B = O(k \lg N)$.

The cost can also be improved by using Novocin, Stehlé, and Villard's $\tilde{L}^1$ variant of LLL [14]. That algorithm costs $d^{5+o(1)} \lg B + d^{\omega+1+o(1)}(\lg B)^{1+o(1)}$ bit operations, where $\omega < 2.373$ is the matrix multiplication exponent.

4 Shor's Algorithm.

We turn now to quantum attacks. We will describe Shor's algorithm without the usual machinery of unitary transformations on complex euclidean space. Instead we break the algorithm into four steps: initialization, randomization, modular exponentiation, and the quantum Fourier transform. We give a high level description of each step, the cost of each step, and the probability of events that we label "$R_j^{(i)} = k$." The event "$R_j^{(i)} = k$" should be read as "terminating Shor's algorithm after step i and reading qubit register j yields outcome k."

The computation involves two qubit registers. The first register represents an element of $\mathbb{Z}/S\mathbb{Z}$, for an S that our analysis will determine later. The second register represents an element of $\mathbb{Z}/n\mathbb{Z}$, where n is the number to be factored. Each step of the algorithm may use "scratch" registers as well, but we assume that these scratch registers are statistically independent from the data registers in between steps[2]. The cost of each step is given as the depth of a quantum circuit that performs that step. Circuit depth is measured in elementary single- and two-qubit gates, hereafter "qubit operations."

Shor's algorithm computes the multiplicative order of 3 modulo n, i.e the smallest r such that $3^r \equiv 1 \pmod{n}$. If r is even, and $3^{r/2} \not\equiv -1 \pmod{n}$, then $\gcd(3^{r/2} - 1,\ n)$ is a non-trivial factor of n. If r is odd, or $3^{r/2} \equiv -1 \pmod{n}$, then one tries again with a different constant in place of 3. The algorithm proceeds as follows.

Step 1: The registers are initialized to 0, i.e.

$$\Pr[R_1^{(1)} = 0 \wedge R_2^{(1)} = 0] = 1.$$

The cost of this step is omitted.

Step 2: The first register is transformed so that it represents the uniform distribution on $\mathbb{Z}/S\mathbb{Z}$:

$$\Pr[R_1^{(2)} = x \wedge R_2^{(2)} = 0] = 1/S$$

for all $x \in \mathbb{Z}/S\mathbb{Z}$. When S is a power of 2 the circuit for this operation – parallel Hadamard gates on each qubit of the first register – has depth 1.

Step 3: The value $\mathsf{modexp}(3, x, n) := 3^x \bmod n$ is calculated and stored in the second register. The transformation $(x, 0) \mapsto (x, \mathsf{modexp}(3, x, n))$ is a bijection, so this does not affect the probability of finding x in the first register, i.e.

$$\Pr[R_1^{(3)} = x \ \wedge \ R_2^{(3)} = \mathsf{modexp}(3, x, n)] = 1/S$$

[2] Standard reversible computing techniques make this a benign assumption, though it is slightly stronger than what is needed.

for all $x \in \mathbb{Z}/S\mathbb{Z}$. This operation costs $(\lg S)\cdot\mathcal{M}(\lg n)$ qubit operations, where $\mathcal{M}(\lg n)$ is the cost of $(\lg n)$-bit modular multiplication. It is reasonable to expect that quantum modular multiplication will cost $(\lg n)^{1+o(1)}$ qubit operations, matching the classical bit operation cost[3].

Step 4: The (approximate) $\mathbb{Z}/S\mathbb{Z}$ quantum Fourier transform is applied to the first register. This costs $(\lg S)^{1+o(1)}$ qubit operations [5]. The effect of this transformation is best described by the distribution of $R_1^{(4)}$ conditioned on $R_2^{(4)} = 3^k \mod n$. Since 3 is of multiplicative order r we may assume $0 \le k < r$. Then for $y \in \mathbb{Z}/S\mathbb{Z}$ Shor shows that

$$\Pr[R_1^{(4)} = y \mid R_2^{(4)} = 3^k \bmod n] = \left| \frac{\sqrt{r}}{S} \sum_{b=0}^{\left\lfloor \frac{S-k-1}{r} \right\rfloor} \exp\left(2\pi i b \tfrac{ry}{S}\right) \right|^2 . \tag{12}$$

We will say that y is *good* if there exists an integer d such that

$$\left| \frac{y}{S} - \frac{d}{r} \right| \le \frac{1}{2S}. \tag{13}$$

Shor shows that for good y the right hand side of Equation 12 is at least $1/3r$. If $S > 2r$ then there are at least $\phi(r)$ pairs (d, y) for which d/r is in lowest terms and y is good. Hence, for $S > 2r$, the total probability assigned to the set of good y is at least $\phi(r)/3r = O(1/\log\log r)$.

The algorithm terminates after step 4. Shor's strategy for recovering r from the known quantities y and S is to compute the convergents of the continued fraction expansion of y/S. If $S > r^2$ then Equation 13 is a sufficient condition (well known in the theory of Diophantine approximation) for d/r to appear among the convergents. For S of this size there are also techniques that amplify the success probability from $O(1/\log\log r)$ to $O(1)$ through classical post-processing alone [20]. We ignore the bit operation cost of post-processing, including the cost of computing $\gcd(3^{r/2} - 1, n)$.

5 The cost of period finding.

The $(\lg S)\cdot\mathcal{M}(\lg n)$ qubit operations in step 3 are the dominant cost of Shor's algorithm. For general integers, Shor recommends taking S to be the power of two between n^2 and $2n^2$. However, for certain integers one can take S slightly smaller.

This brings us to the second reason that Bernstein, Heninger, Lou, and Valenta find $n = p^k q$ to be worrisome. Since $\phi(n) = p^{k-1}(p-1)(q-1)$ the multiplicative

[3] State of the art quantum circuits for mod-n multiplication cost $32(\lg n)^2 + O(\lg n)$ qubit operations [7], but there are no fundamental barriers to implementing fast multiplication.

order of any nth power in $(\mathbb{Z}/n\mathbb{Z})^\times$ divides $(p-1)(q-1)$. Hence, we may take $S = ((p-1)(q-1))^2 \approx n^{4/(k+1)}$ so long as we by replace the constant 3 in Step 3 with $3^n \bmod n$. The cost of computing $3^n \bmod n$ is $(\lg n)\cdot\mathcal{M}(\lg n)$ bit operations. Following this, the cost of Step 3 of Shor's algorithm is $O(\frac{\lg n}{k+1}) \cdot \mathcal{M}(\lg n)$ qubit operations. In other words, "Shor's algorithm finds the order at relatively high speed once the nth power is computed" [1].

The same argument applies to more general multi-power moduli $n = p_1^{e_1} p_2^{e_2} \dots p_L^{e_L}$. The adversary can take $S = n^{2L/E}$ where $E = \sum_i e_i$. The asymptotic cost of factoring such n by Shor's algorithm is $(\lg n) \cdot \mathcal{M}(\lg n)$ bit operations plus $O(\frac{L}{E} \lg n) \cdot \mathcal{M}(\lg n)$ qubit operations. If $E - L$ is large, and qubit operations are much more expensive than bit operations, this could be a dramatic improvement.

6 Multi-power pqRSA

The multi-prime post-quantum RSA system proposed in [1] uses fast multiplication techniques to reduce the cost of users' operations when $n = p_1 p_2 \cdots p_\ell$ and ℓ grows as $\lg n/(\lg\lg n)^{2+o(1)}$. The multi-power post-quantum RSA system we propose here applies similar techniques to reduce the cost of users' operations when $n = p_1^{e_1} p_2^{e_2} \cdots p_L^{e_L}$ and L grows as $(\lg n)^{1/(2+o(1))}$. Our decryption procedure follows [22] and [11] in using a Newton iteration to compute a cube root modulo $p_i^{e_i}$ from a cube root modulo p_i.

The constraints on parameter selection for multi-prime pqRSA come from ECM and Shor's algorithm. When repeated factors are allowed we also need to ensure that lattice attacks cost more than ECM and that special purpose variants of Shor's algorithm do not eliminate our cost/performance ratio.

6.1 Parameters.

We take

$$n = p_1^2 p_2^3 p_3^5 p_4^7 \cdots p_i^{\pi_i} \cdots p_L^{\pi_L}$$

where π_i is the ith prime. As in [1], we take each prime factor of n to be of bit length $(\lg\lg n)^{2+o(1)}$. We also take $p_i \equiv 2 \pmod 3$ for $i = 1, \dots, L$ so that encryption exponent 3 can be used.

We define $E(L) = \sum_{i \le L} \pi_i$. The prime number theorem suggests that $E(L) \sim \frac{1}{2} L^2 \log L$. To obtain keys of a specified bit-length we choose L such that $E(L) = \lg n/(\lg\lg n)^{2+o(1)}$.

For concreteness we will focus on 1-terabyte keys, $\lg n = 2^{43}$. We follow [1] and fix the bit length of the p_i at 4096 bits. We then take $L = 20044$, which gives $E(20044) = 2^{31.0001\ldots}$.

6.2 Security.

Small factor attacks.

The asymptotic security analysis of multi-power pqRSA with respect to trial division, Pollard's $p-1$ method, Williams' $p+1$ method, and Lenstra's elliptic curve method is identical to multi-prime pqRSA. This is also true for variants of these algorithms that use Grover's algorithm as a subroutine. We refer to [1] for a detailed discussion of why these attacks are not likely to succeed at a cost of less than 2^{140} bit operations when 4096-bit primes are used.

Note that Peralta and Okamoto [15] describe a method to speed up ECM for moduli of the form $n = p^2 q$. This method does not improve on the asymptotic $\exp((\log p)^{1/2+o(1)})$ cost estimate for ECM that is used here.

Lattice attacks.

The lattice attack in Section 2 is competitive with ECM when the attacker needs to guess fewer than $\sqrt{\lg p_1} = (\lg \lg n)^{1+o(1)}$ bits. We will argue that there do not exist integers P, Q, and k, with $n = P^k Q$, for which the lattice attack is competitive with ECM. From Equation 10, we see that P, Q, and k must satisfy

$$\frac{1}{k+1}(\lg P + \lg Q) \leq (\lg \lg n)^{1+o(1)}. \tag{14}$$

Observe that $p_1^2 p_2^3 \cdots p_j^{\pi_j} \mid Q$ for j such that $\pi_j < k$. The sum of the primes less than k grows at least as fast as the sum of the first $k/\log k$ integers. In particular the sum dominates k. Hence $\lg Q = \omega((\lg \lg n)^2 k)$ and we see that Eq. 14 cannot be satisfied. It is a simple exercise to show that replacing Equation 14 with a condition based on Equation 9 does not improve the situation.

Even if a satisfying k did exist, the cost of LLL would be prohibitive. Theorem 2.1 suggests a cost of well over $k^{10} \lg n$ bit operations. Taking k smaller than L is unrealistic, as $\lg P + \lg Q$ grows super-linearly with L. With $\lg n = 2^{43}$ and $L = 20044$ we have $L^{10} \lg n > 2^{185}$.

The cost of lattice reduction becomes an enormous obstacle if one attempts to guess bits of P using Grover search.

Period finding.

As described in Section 5, one can reduce the cost of period finding by replacing the constant 3 in Shor's algorithm with $3^n \bmod n$. The resulting cost is $(\lg n) \cdot \mathcal{M}(\lg n)$ bit operations plus $(\lg S) \cdot \mathcal{M}(\lg n)$ qubit operations where $S = n^{2L/E}$. For our multi-power pqRSA moduli $\lg S = (\lg n)^{1/2+o(1)}$, so the cost of Shor's algorithm is $(\lg n) \cdot \mathcal{M}(\lg n)$ bit operations plus $(\lg n)^{1.5+o(1)} \cdot \mathcal{M}(\lg n)$ qubit operations.

Bernstein, Heninger, Lou, and Valenta [1] estimate $\mathcal{M}(\lg n)$ using contemporary speed records for multiplication in $\mathbb{F}_{2^{60}}$. Assuming that integer multiplication can be made as fast as binary polynomial multiplication, that reduction modulo n is free, and that bit operations have the same cost as qubit operations, they suggest that $\mathcal{M}(2^{43}) \approx 2^{56}$ bit (or qubit) operations.

For our 1-terabyte multi-power pqRSA parameters $S = (2^{4096L})^2 \approx 2^{2^{27}}$. Consequently, Shor's algorithm costs $(\lg n) \cdot \mathcal{M}(2^{43}) \approx 2^{99}$ bit operations plus $(\lg S) \cdot \mathcal{M}(2^{43}) \approx 2^{83}$ qubit operations. Compare this with the cost of attacking 1-terabyte multi-prime pqRSA: with $S = n^2$ Shor's algorithm costs $(\lg n) \cdot \mathcal{M}(2^{43}) \approx 2^{100}$ qubit operations.

6.3 Efficiency.

Apart from decryption, the user's computations in multi-power pqRSA are identical to the user's computations in multi-prime pqRSA. We refer the reader to [1] for the justification that key generation and encryption take $(\lg n)^{1+o(1)}$ bit operations.

Multi-power pqRSA decryption may be somewhat faster than multi-prime pqRSA decryption, but we do not expect a significant asymptotic improvement. To decrypt a ciphertext c, one first uses a remainder tree to compute $c \bmod p_i^{\pi_i}$ for $i = 1, \ldots, L$. One then computes the cube root of c modulo each p_i. The decryption algorithm differs from multi-prime pqRSA in that there are two distinct interpolation steps. The first, as in [22, 11], lifts the cube root of c modulo p_i to a cube root of c modulo $p_i^{\pi_i}$ for $i = 1, \ldots, L$. The second uses a CRT tree to construct the cube root of c modulo n. If there is any speedup for multi-power pqRSA decryption, it is likely due to the fact that fewer cube roots need to be computed. The two interpolation steps together are likely as expensive as the CRT tree in multi-prime pqRSA. Regardless, decryption will cost $(\lg n)^{1+o(1)}$ bit operations.

We have not implemented multi-power pqRSA, and can only estimate its non-asymptotic efficiency. Bernstein, Heninger, Lou, and Valenta report a prime generation rate of between 750 and 1585 primes per core-hour. They also report that a set of 2^{31} primes was generated in 1 975 000 core-hours [1]. At the same rate, generating the 20 044 primes needed for a 1-terabyte multi-power pqRSA key would take only 18.5

core-hours. Keep in mind that the full key generation procedure also involves a large product tree.

Cost/performance ratio.

Multi-prime pqRSA key generation, encryption, and decryption all cost $(\lg n)^{1+o(1)}$ bit operations. The best attack costs $(\lg n)^{2+o(1)}$ qubit operations. As there is no reason to believe that qubit operations are *less* expensive than bit operations, we can safely say that there is a quadratic gap between the attacker's cost and the user's cost. The gap could be larger if qubit operations are fundamentally more expensive than bit operations.

Multi-power pqRSA key generation, encryption, and decryption, likewise, cost $(\lg n)^{1+o(1)}$ bit operations. The best attack costs $(\lg n)^{2+o(1)}$ bit operations plus $(\lg n)^{1.5+o(1)}$ qubit operations. If qubit operations cost the same as bit operations, then multi-power pqRSA users still enjoy a quadratic advantage against attackers. This remains true when qubit operations cost C bit operations, for any constant C. The quadratic advantage is only threatened when the cost of qubit operations grows with n.

7 Conclusion

One-terabyte multi-prime pqRSA key generation is quite costly. In one test, prime generation took "four months running on spare compute capacity of a 1,400-core cluster" and evaluating the product tree took "about four days" [1]. A user with a comparable machine should be able to generate a multi-power pqRSA key in just 5 days. The speedup comes with some loss in security, but quantifying this loss presents challenges.

In Section 6.2 we saw that if bit operations and qubit operations have equal cost, then 1-terabyte multi-power pqRSA users suffer a 1 bit security loss. However, if qubit operations are much more expensive than bit operations, these users suffer a 17 bit security loss—albeit from a higher initial security level.

The security gap could be amplified in other cost models. In [6] the cost of a quantum circuit is expressed in "surface code cycles." Each surface code cycle involves error correction that can be assigned a cost in bit operations. For a logical circuit of depth D on W logical qubits, the error correction for the surface code costs $\operatorname{poly}\lg DW$ bit operations per surface code cycle. This is $O(\lg\lg n)$ bit operations per surface code cycle for attacks on both multi-prime and multi-power pqRSA, but the constant will be smaller for multi-power pqRSA.

Questions about the relative cost of bit and qubit operations will need to be answered whether users choose multi-prime pqRSA, multi-power pqRSA, or some other system. Even if users decide that no variant of pqRSA is fast enough for their needs, a robust quantum cryptanalysis of RSA will greatly improve our understanding of the post-quantum security of other systems.

Bibliography

[1] Daniel J. Bernstein, Nadia Heninger, Paul Lou, and Luke Valenta. Post-quantum RSA. In *International Workshop on Post-Quantum Cryptography*, pages 311–329. Springer, 2017.

[2] Dan Boneh, Glenn Durfee, and Nick Howgrave-Graham. Factoring $N = p^r q$ for large r. In *Annual International Cryptology Conference*, pages 326–337. Springer, 1999.

[3] Dan Boneh and Hovav Shacham. Fast variants of RSA. RSA Laboratories' CryptoBytes, vol 5, no 1, pp 1–9, 2002.

[4] T. Collins, D. Hopkins, S. Langford, and M. Sabin. Public key cryptographic apparatus and method. US Patent #5,848,159, Jan. 1997.

[5] Don Coppersmith. An approximate Fourier transform useful in quantum factoring. *arXiv preprint quant-ph/0201067*, 2002.

[6] 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.

[7] Thomas Häner, Martin Roetteler, and Krysta M. Svore. Factoring using 2n+ 2 qubits with Toffoli based modular multiplication. *arXiv preprint arXiv:1611.07995*, 2016.

[8] M. Jason Hinek. On the security of multi-prime RSA. *Journal of Mathematical Cryptology*, 2(2):117–147, 2008.

[9] Nicholas Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *IMA International Conference on Cryptography and Coding*, pages 131–142. Springer, 1997.

[10] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[11] Seongan Lim, Seungjoo Kim, Ikkwon Yie, and Hongsub Lee. A generalized Takagi-cryptosystem with a modulus of the form $p^r q^s$. In Bimal Roy and Eiji Okamoto, editors, *Progress in Cryptology —INDOCRYPT 2000*, pages 283–294. Springer, 2000.

[12] Alexander May. Using LLL-reduction for solving RSA and factorization problems. In *The LLL algorithm*, pages 315–348. Springer, 2009.

[13] Phong Q Nguyen and Damien Stehlé. An LLL algorithm with quadratic complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009.

[14] Andrew Novocin, Damien Stehlé, and Gilles Villard. An LLL-reduction algorithm with quasi-linear time complexity: Extended abstract. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 403–412, New York, NY, USA, 2011. ACM.

[15] René Peralta and Eiji Okamoto. Faster factoring of integers of a special form. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 79(4):489–493, 1996.

[16] J-J Quisquater and Chantal Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics letters*, 18(21):905–907, 1982.

[17] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[18] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. Cryptographic communications system and method. US Patent #4,405,829, Dec. 1977.

[19] Adi Shamir. RSA for paranoids. RSA Laboratories' CryptoBytes, vol 1, no 3, pp 1–4, 1995.

[20] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[21] Tsuyoshi Takagi. Fast RSA-type cryptosystems using n-adic expansion. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 372–384. Springer, 1997.

[22] Tsuyoshi Takagi. Fast RSA-type cryptosystem modulo $p^k q$. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, pages 318–326. Springer, 1998.

Chapter 5

Conclusion

1 Assumptions redux

My co-authors and I have worked within standard models where possible. Each of the above chapters includes an analysis in the standard circuit model, or an analysis in the circuit model with qRAM. Each chapter fills in details that were missing from previous analyses and draws new conclusions. To re-cap:

1. There is no known attack of cost $p^{1/6+o(1)}$ on SIKE—the best known is $p^{1/4+o(1)}$.
2. The $2^{0.265d}$ estimate for the cost of lattice sieving is a severe underestimate. I know of no algorithm for list-size preserving angular near neighbor search in dimension $d = 376$ that can claim a cost of less than $2^{127.7}$ Clifford+T+qRAM gates or less than $2^{137.6}$ RAM operations.
3. Multi-power RSA is either roughly as secure as multi-prime RSA, or multi-power RSA is less secure than multi-prime RSA and multi-prime RSA is more secure than it is currently claimed to be.

However, I began by asking that we re-examine several features of the quantum machine models that we use. Alongside our analyses in standard models, my co-authors and I have shown that the following features have practical consequences for the security of SIKE, lattice-based schemes, and RSA.

Zero-cost quantum storage If one assumes unit-cost quantum storage (i.e. the depth-width cost) rather than zero-cost quantum storage (i.e. the gate cost), then the claimed $p^{1/6+o(1)}$ attack on SIKE is immediately revealed to be a $p^{1/3+o(1)}$ attack. Optimizing the parameters of the attack only reduces its cost to $p^{1/4+o(1)}$. This matches the best known classical attack up to terms in the $o(1)$.

Unit-cost quantum accessible classical memory By the principle of locality, a qRAM gate *necessarily* has a cost that is at least proportional to the time to access an N bit classical memory. This impacts the asymptotic advantage that quantum search provides to angular near neighbor search algorithms. Under an assumption of unit-cost qRAM the asymptotic complexity of the Becker–Ducas–Gama–Laarhoven algorithm is $2^{(0.265\ldots+o(1))d}$. If the memory is embedded in a three dimensional space (at finite integration density), and one assumes that the cost of memory access matches the lower bound of $\Omega(N^{1/3})$, then the asymptotic complexity of the BDGL algorithm is $2^{(0.284\ldots+o(1))d}$. If one assumes a two dimensional embedding of the memory, and that the cost of memory access matches the lower bound of $\Omega(N^{1/2})$, then the asymptotic complexity of the BDGL algorithm is $2^{(0.292\ldots+o(1))d}$. Its classical RAM model complexity is also $2^{(0.292\ldots+o(1))d}$.

Zero-cost classical computation If one assumes that quantum gates and classical gates have equal cost, then the asymptotic complexity of Shor's algorithm matches the complexity of modular exponentiation. This presents an obvious barrier to the small order attack of Chapter 4.

2 Recommendations for quantum cryptanalysis

The tools that are needed by cryptanalysis are not, entirely, contained in the theory of the analysis of algorithms. Nor are they entirely contained in the theory of computational complexity. Nor are they contained in the union of these theories. What cryptanalysts need is a theory of *computational economy*—a theory of how to compute best given fixed finite resources.

For practitioners of computational economy, I have some concrete recommendations:

1. We should assume that active error correction is an essential part of quantum computation. This assumption comes with some risk, but it is a risk similar to that posed by non-dissipative computation.
2. We should cost classical pre-processing, co-routines, and post-processing in the same model of computation that we use to cost quantum error correction.
3. We should fix a technologically motivated absolute limit on the size of a memory. I suggest 2^{140} bits—a Lunar mass memory with a density of one petabyte per gram.

My goal in making these recommendations is to simplify the attack landscape of post-quantum cryptosystems without compromising real-world security.

Physicists may eventually show that self-correcting quantum memories do not exist. They may eventually show that robust computing systems must dissipate energy. They may eventually find computational speed limits. They may eventually show that closed timelike curves do not exist. Computer scientists may even have a role to play in exploring these limits. But cryptographers do not have to wait for future developments. We can continue to use economic and technological assumptions to shore up our analyses.

Appendix: Other work by the author

The three articles presented above are some of the work that I have been doing to make post-quantum cryptography more secure, more usable, and easier to deploy. The following is a short summary, organized by topic, of the other work that I have been doing.

Quantum cryptanalysis With Amy, Di Matteo, Gheorghiu, Mosca, and Parent, I studied the security of the SHA-2 and SHA-3 hash functions against generic quantum pre-image attacks. This work is very much within the theme of this thesis, but was partially completed during my Master's.

- Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John M. Schanck. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In Roberto Avanzi and Howard M. Heys, editors, *SAC 2016*, volume 10532 of *LNCS*, pages 317–337. Springer, Heidelberg, August 2016.

Key encapsulation mechanisms I have participated in the design of two lattice-based key encapsulation mechanisms: NTRU and Kyber. Both of these were submitted to NIST's post-quantum standardization effort, and both have advanced to the second round of this process. I am the principal author on the NTRU submission.

NTRU, specifically the ntruhrss701 parameter set, was profiled in an experiment conducted by Google[1] and Cloudflare[2]. That experiment compared ntruhrss701 against another submission to NIST's standardization effort called SIKEp434. The conclusion was that "[ntruhrss701] looks like the more promising algorithm for use in TLS." Google Chrome clients, starting with version 83, can negotiate a TLS 1.3 connection using ntruhrss701. This

[1] https://www.imperialviolet.org/2019/10/30/pqsivssl.html

[2] https://blog.cloudflare.com/the-tls-post-quantum-experiment/

likely makes NTRU the most widely used piece of post-quantum public-key cryptography on the Internet.

- Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hülsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, and Zhenfei Zhang. NTRU. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.
- Joppe Bos, Leo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals – Kyber: A CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367, April 2018.
- Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 232–252. Springer, Heidelberg, September 2017.
- John M. Schanck, Andreas Hulsing, Joost Rijneveld, and Peter Schwabe. NTRU-HRSS-KEM. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.
- John M. Schanck. A comparison of NTRU variants. Cryptology ePrint Archive, Report 2018/1174, 2018. https://eprint.iacr.org/2018/1174.
- Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS–KYBER. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.
- Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS–KYBER. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

Lattice KEM decryption failures Some post-quantum KEMs, like ntruhrss701, are essentially drop-in replacements for their pre-quantum counterparts. Others, like Kyber, differ from widely deployed pre-quantum KEMs in that they can fail to decrypt an honestly generated ciphertext. This occurs very rarely in Kyber, but decryption failures are a serious security risk and require careful analysis.

Nina Bindel and I have studied correlations between decryption failure events and the information revealed by *successful* decryptions. Separately, I have studied the risk of a decryption failures for a fixed user of the NewHope scheme—previous analyses of NewHope studied the expected rate of failures over all keys and ciphertexts.

- Nina Bindel and John M. Schanck. Decryption failure is more likely after success. Cryptology ePrint Archive, Report 2019/1392, 2019. (To appear at PQCrypto 2020). https://eprint.iacr.org/2019/1392.
- John M. Schanck. An upper bound on the decryption failure rate of static-key NewHope Cryptology ePrint Archive, Report 2020/326, 2020. https://eprint.iacr.org/2020/326.

Deployment of post-quantum cryptography With Douglas Stebila, I proposed a method for incorporating an additional key exchange into TLS 1.3. This method allows one to define "hybrid" ciphersuites that combine a pre-quantum and post-quantum KEM. With William Whyte and Zhenfei Zhang, I proposed a hybrid ciphersuite for the Tor anonymity network and proved that it achieves a notion of forward secrecy that is defined relative to an adversary who acquires a quantum computer after the session has been established.

- John M. Schanck and Douglas Stebila. A Transport Layer Security (TLS) Extension For Establishing An Additional Shared Secret. Internet Draft, April 2017. https://tools.ietf.org/html/draft-schanck-tls-additional-keyshare-00
- John M. Schanck, William Whyte, and Zhenfei Zhang. Circuit-extension handshakes for Tor achieving forward secrecy in a quantum world. *PoPETs*, 2016(4):219–236, October 2016.