Quantum Computing Applications

"A good way of pumping funding into the building of an actual quantum computer would be to find an efficient quantum factoring algorithm!"[1]

THE risk of wide-scale cryptanalysis pervades narratives about quantum computing. We argue in this chapter that Feynman's vision for quantum computing will ultimately prevail, despite the discovery of Peter Shor's factoring algorithm that generated excitement about a use of quantum computers that people could understand – and dread.

To explain this outcome, we canvass the three primary applications that have been developed for quantum computing: Feynman's vision of simulating quantum mechanical systems, factoring, and search. The next chapter discusses today's quantum computing landscape.

For Feynman, a quantum computer was the only way that he could imagine to efficiently simulate the physics of quantum mechanical systems. Such systems are called *quantum simulators.*[2] Quantum simulation remains the likely first practical use of quantum comput-

[1]Berthiaume and Gilles Brassard, "Oracle Quantum Computing" (1994), written hours before Peter Shor discovered such an algorithm.

[2]The term *quantum simulators* is confusing, because it is also applied to programs running on conventional computers that simulate quantum physics. For this reason, some authors use the terms *Feynman simulators* or even *Schrödinger–Feynman simulators*.

ers. Oddly, this application is *not* responsible for most of the public interest in quantum computers, which has instead been fueled by the desire to make super-machines that can crack the world's strongest encryption algorithms. Since then, without dramatic demonstrations of other capabilities, and with the underlying complexity of achievements that have been made, many news articles cast quantum computing in a single, privacy-ending narrative.

We believe that prominence of cryptanalysis in public interest and government funding over the past two decades is because a working quantum computer that could run Shor's algorithm on today's code would give governments that owned it an incredible advantage to use over their adversaries: the ability to crack messages that had been collected and archives going back decades. But while this advantage may be responsible for early funding of quantum computing, we believe that the cryptanalytic capabilities of initial quantum computers will be limited and outshone by the ability of these machines to realize Feynman's vision. And Feynman's vision, unlike cryptanalysis, confers first-mover advantage, since a working quantum physics simulator can be used to build better quantum physics simulators. That is, quantum physics simulations are likely to create a virtuous circle, allowing the rate of technology change to increase over time.

The last section of this chapter turns to search, and explains the kinds of speedups quantum computers are likely to provide. Understanding those likely speedups further advances our prediction that the future of quantum computing will be Feynman's.

5.1 Simulating Physical Chemistry

In this section we explore how one might actually go about simulating physics with quantum computers. Despite the similarity of titles, this section is not an extended discourse on Feynman's articles. Instead, it is a discussion of how chemists actually simulate the physics of chemical reactions with classical computers today, and how they might do so with quantum computers tomorrow.

Classical computers – like the computers used to write and typeset this book – are designed to execute predetermined sequences of instructions without error and as reliably as possible. Computer engineers have made these machines steadily faster over the past 80 years, which makes it possible to edit this book with graphical editors and typeset its hundreds of pages in less than a minute. Both of those activities are fundamentally a sequence of operations applied to a sequence of bits, starting with an input stream of `0`s and `1`s, and possibly a character typed on a computer keyboard) and deterministically creating a single output stream (the PDF file that is displayed on the computer's screen).

Modeling molecular interactions is fundamentally different from word processing and typesetting. When your computer is running a word processing program and you press the H key, there is typically only one thing that is supposed to happen: an "H" appears at the cursor on the screen. But many different things can happen when two molecules interact: they might stick together, they might bounce, or an atom might transfer from one molecule to the other. The probability of each of these outcomes is determined by quantum physics.

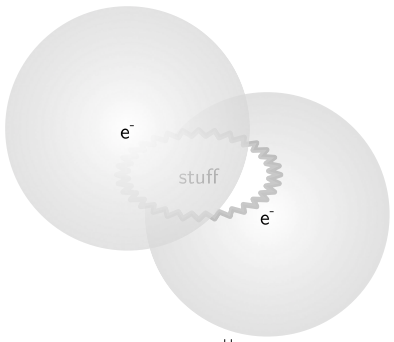
To explore how two molecules interact, the basic approach is to build a model of all the atomic nuclei and the electrons in the two-molecule system and then compute how the wave function for the system evolves over time. Such simulations quickly become unworkable, so scientists will consider a subset of the atoms and electrons, with the hope that others will stay more-or-less static. This hope was formalized in 1927 and today is known as the Born–Oppenheimer approximation, named after Max Born and J. Robert Oppenheimer who jointly proposed it. Other approximations exist, such as assuming that the nuclei are fixed in space and are point charges, rather than wave functions themselves. High school chemistry, which typically presents the electrons as little balls of charge spinning around the nuclei, is a further simplification.

Many of the chemistry discoveries of the twentieth century were possible because the Born–Oppenheimer approximation is largely correct, but it is not perfect. For example, it may not apply in exotic materials, such as graphene.[3] More generally, it may not apply to certain kinds of surface chemistry. "There is growing evidence that the usual approach to modelling chemical events at surfaces is incomplete – an important concern in studies of the many catalytic processes that involve surface reactions."[4]

Most chemistry can be understood working with the time-independent Schrodinger equation, in which the chemist simply looks for the most likely configuration of the atoms and electrons. Systems

[3]Pisana et al., "Breakdown of The Adiabatic Born – Oppenheimer Approximation in Graphene" (2007).

[4]Sitz, "Approximate Challenges" (2005).



Energy State		outcome
electron 1	electron 2	probability
low	low	2%
low	high	5%
high	low	90%
high	high	3%

Figure 5.1. The possible energy states of two electrons in a hypothetical quantum system.

that cannot be studied this way can be modeled using what's called Monte Carlo methods (or a Monte Carlo simulation), in which the chemist creates a probabilistic model and then runs the simulation multiple times, examining the range of possible outcomes.

We present a simplified Monte Carlo simulation in Figure 5.1. To keep things simple, we have assumed that there are only two electrons of interest, and that each will end up in either a *low* or *high* energy state. Facing this system, a scientist can use modeling software to determine the probability of each of these outcomes. Here our hypothetical scientist has used a conventional computer to run this experiment many times, tabulate the results, and report them in the rightmost column as an *outcome probability.*

Our scientist would take a fundamentally different approach to solve this problem on a quantum computer. Instead of modeling the probabilities, the scientist designs a *quantum circuit* that directly represents (or simulates) the chemistry in question. With most quantum computers today, the scientist would then turn on the quantum computer, placing each of its quantum bits (called *qubits*) into a superposition state. The quantum circuit plays through the quantum computer, changing how the qubits interact with each other over time. This "playing" of the quantum circuit is performed by a second computer – a classical computer – that controls the quantum computer. When the circuit is finished playing, the second computer measures each qubit, collapsing the superposition wave function and revealing its quantum state. At this point each qubit is *either* a `0` or a `1`.

In this example, each qubit might directly represent an energy state of an electron that was previously modeled. So if our scientist designed a quantum circuit and ran it on our hypothetical quantum computer, the result might look like this:

Trial	qubit 1	qubit 2
#1	`1`	`0`

It looks like the quantum computer has found the right answer instantly!

Actually, no. Because if the scientist ran the experiment a second time, the answer might be different:

Trial	qubit 1	qubit 2
#2	`1`	`1`

In an actual quantum computer, the experiment would run multiple times:

Trial	qubit 1	qubit 2
#3	`1`	`0`
#4	`1`	`0`
#5	`0`	`0`
#6	`1`	`0`
#7	`1`	`0`
#8	`1`	`0`
#9	`1`	`0`
#10	`1`	`0`

After these trials, the results are tabulated to get a distribution of possible answers. The statistics that are similar to those produced by the classical computer, but a little different:

qubit 1	qubit 2	Trial #s	Count	Probability
0	0	#5	1	10%
0	1	–	0	0%
1	0	#1, #3, #4, #6, #7, #8, #9, #10	8	80%
1	1	#2	1	10%

Notice that the quantum computer does not generally produce the same results as the classical computer. This may be because we did not run sufficiently many trials to get results with the same statistical distribution as the results produced by the classical computer. It might also be because the model run on the classical computer is incomplete. More likely, both models are incomplete, but incomplete in different ways. (Even if they were identical models, it's unlikely that identical statistics would emerge with just 10 runs.)

It is important to remember that in this simulation, as in real quantum systems, *there is no right answer*. Instead, there is a range of possible answers, with some more probable and some less probable.

In practice, efficient quantum computing algorithms are designed so that "correct" or desired answers tend to generate constructive interference on the quantum computing circuits, while answers that are not desired tend to cancel each other out with destructive interference. This is possible because what quantum computers actually do is to evolve carefully constructed probability waves in space and time. These waves "collapse" when the final measurement is made by the scientist (or, more specifically, by the classical computer that is controlling the quantum computer). For a discussion of quantum mechanics and probability, please see Appendix B.

The advantage of a quantum computer becomes clear as the scale increases. Exploring the interaction of 32 electrons, each of which could be in two states, requires exploring a maximum of 4 Gi[5] combinations. A classical computer would need to explore *all* of those combinations one-by-one. Exponential growth is really something: simply printing out those 4 Gi combinations at 6 lines per inch would consume 11 297 linear miles of paper. Today for certain problems, quantum computing scientists have discovered algorithms that run more efficiently on quantum computers than the equivalent classical

[5] 4 Gi means 4 Gigi, which is the SI prefix that denotes powers-of-two rather than powers-of-ten counting. 4 Gi is $4 \times 1024 \times 1024 \times 1024 = 2^{32} = 4\,294\,967\,296$, or roughly 4.2 billion.

algorithms that exist to solve the problems on conventional computers. Generally speaking, the more qubits a quantum computer has, the more complex a system it can simulate.

Approaches for programming quantum computers are still in their infancy. Because the machines are small – with dozens of qubits, rather than millions – programmers need to concern themselves with individual qubits and gates. In some notable cases quantum computers are being constructed to solve specific problems.[6] This is reminiscent of the way that the first computers were built and programmed in the 1940s, before the invention of stored programs and computer languages: in England the Colossus computers were built to crack the Germans' Lorentz code, while in the US the ENIAC was created to print artillery tables. Programming quantum computers will get easier as scientists shift from single-purpose to general machines and as the machines themselves get larger.

In addition to the number of qubits, the second number that determines the usefulness of a modern quantum computer is the stability of its qubits. Stability is determined by many things, including the technology on which the qubits are based, the purity of the materials from which the qubits are manufactured, the degree of isolation between the qubits and the rest of the universe, and possibly other factors. Qubits that are exceedingly stable could be used to compute complex, lengthy quantum programs. Such qubits do not currently exist. In fact, an entire research field explores ways to shorten quantum algorithms so that they are compatible with short-lived qubits.

Quantum engineers use the word noise to describe the thing that makes qubits less stable. *Noise* is a technical term that engineers use to describe random signals. The reason we use this term is that random signals fed into a speaker literally sound like a burst of noise, like the crackle between stations on an AM radio, or the sound of crashing waves. Noise in the circuit does not help the quantum computer achieve the proper distributions of randomness and uncertainty described by quantum mechanics. Instead, noise collapses the wave functions and scrambles the quantum computations, similar to the way that jamming the relay contacts in the Harvard's Mark II computer caused it to compute the wrong numbers on September 9, 1947.[7] Early computers only became useful after computer engineers

[6] Zhong et al., "Quantum Computational Advantage Using Photons" (2020).

[7] A moth was found pinned between the contacts of Relay #70 Panel F. Grace Hopper, a developer and builder of the Mark II, taped the insect into her laboratory

Quantum Error Correction

The quantum computing applications that we discuss in this chapter all assume the existence of a working, reliable quantum computer with sufficient qubits, able to run quantum circuits with sufficient size and complexity for a sufficiently long period of time.

Although an absolutely reliable quantum computer is a useful theoretical construct for thinking about quantum computing algorithms, actual quantum computers will probably need to use some form of *quantum error correction*, in which multiple noisy qubits are used to simulate a smaller number of qubits that have less noise.

Although quantum error correction is powerful, today's techniques do not appear to be up to the task of sustaining a single quantum computation for time periods that would be long enough to pose a threat to modern cryptographic systems.

learned how to design circuits that reduced noise to the point of irrelevance. They did this using an engineering technique called *digital discipline* that is still used today (see p. 84), but that approach won't work with quantum computers.

Instead, companies like Google, IBM, and Rigetti have created machines that have noisy qubits. As a result, most quantum programs today are small and designed to run quickly. Looking towards the future, many noisy qubits can be combined to simulate cleaner qubits using an error-correcting technique called surface codes,[8] but today's machines do not have enough sufficient noisy qubits for this to be practical. Another approach is to use a quantum computing media that is largely immune to noise; that's the approach being taken by Microsoft with its so-called *topological qubits*, although other approaches using photonic qubits or ion traps might produce similar noise-free results. But for today, noise significantly limits the complexity of computations that can be done on quantum computers, even if we could build machines with hundreds or thousands of noisy qubits.

notebook with the notation "first actual case of bug being found."

[8]Fowler et al., "Surface Codes: Towards Practical Large-Scale Quantum Computation" (2012).

Even so, some companies are eager to get a head start, and are having their scientists and engineers learn to program these machines today. As a result, IBM is able to generate revenue with its "quantum experience" by giving free access over the Internet to machines with only a few qubits, and renting time to institutions who want access to IBM's larger machines. Likewise, Amazon Web Services has started making small quantum computers built by other companies available through its "Bracket" cloud service. However, the power of these machines is dwarfed by Amazon's conventional computing infrastructure.

Finally, there is an important point that we need to make: there is no mathematical *proof* that a quantum computer will be able to simulate physics faster than a classical computer. The lack of such a proof reflects humanity's fundamental ignorance on one of the great mathematical problems of time, *NP completeness* (see Section 3.5.4, "NP-Complete and NP-Hard" (p. 110)). What we do know is that *today's* quantum simulation algorithms get exponentially slower as the size of the problem being simulated increases in size, and the simulation algorithms that we have designed for quantum computers do not. But this may reflect the limits of our knowledge, rather than the limits of classical computers. It might be that work on quantum computing leads to a breakthrough in mathematics that allows us to create dramatically faster algorithms to run on today's classical computers. Or it may be that work on quantum computing allows us to prove that quantum computers really are fundamentally more powerful than classical computers, which would help us to solve the great mathematical question of NP completeness. What we know today is that quantum computers can take advantage of quantum physics to run so-called *BQP algorithms*, and that today's BQP algorithms run more efficiently than the fastest algorithms that we know of to run on classical computers. (See Section 3.5.4 (p. 110) and Section 3.5.6 (p. 116) for a more in-depth discussion of these topics.)

5.1.1 Nitrogen Fixation, without Simulation

To put efforts to develop a quantum computer into context, this section explores how such a machine might help develop more efficient approaches for "fixing" nitrogen.

Nitrogen, in the form of organic nitrates, is both vital for biological life and in surprisingly short supply. The productivity of pre-industrial agriculture was often limited by the lack of nitrogen, rather than limitations of water or sunlight. Industrial agriculture has solved this problem through the industrial production of nitrogen-based fertilizers.

The need for added nitrogen is surprising given the fact that plants are surrounded by nitrogen in the form of air. Nearly 80 percent of dry air is nitrogen. The problem is that nitrogen in the air is N_2, also written $N{\equiv}N$, with a triple chemical bond between the two nitrogen atoms. This triple bond has the charge of six electrons, making it difficult to break. As a result, the nitrogen in air is inaccessible to most plants.

Nitrogen fixation is the process of taking N_2 and turning it into a more usable form, typically ammonia (NH_3). The overall chemical reaction is not very complex:

$$\mathit{Energy} + \mathrm{N_2} + 3\,\mathrm{H_2} \longrightarrow 2\,\mathrm{NH_3} \tag{1}$$

Most of the natural nitrogen fixation on Earth happens in the roots of alfalfa and other legumes, where nitrogen-fixing bacteria live in a symbiotic relationship with the plant host.[9] Instead of hydrogen gas, biological nitrogen fixation uses ATP (adenosine triphosphate) produced by photosynthesis, some spare electrons, and some hydrogen ions (present in acid) that just happen to be floating around. The products are ammonia (containing the fixed nitrogen), hydrogen gas, ADP (adenosine diphosphate), and inorganic potassium (written as Pi below):

$$\mathrm{N_2} + 16\,\mathrm{ATP} + 8\,\mathrm{e^-} + 8\,\mathrm{H^+} \longrightarrow 2\,\mathrm{NH_3} + \mathrm{H_2} + 16\,\mathrm{ADP} + 16\,\mathrm{Pi} \tag{2}$$

The plant then uses photosynthesis and sunlight to turn the ADP back into ATP.

In 1909, the German chemist Fritz Haber discovered an inorganic approach to nitrogen fixation using high pressure and the chemical element osmium, which somehow helps the electrons to rearrange. Chemists say that osmium *catalyzes* the reaction. Haber was awarded the Nobel Prize in Chemistry in 1918, "for the synthesis of ammonia from its elements."[10]

[9]There is also a small amount of nitrogen fixation that results from lightning.

[10]Haber is also known as the "father of chemical warfare" for his work weaponizing the production and delivery of chlorine gas as part of Germany's efforts during World War I, and for his institute's development of Zyklon A. Despite this service

Haber sold his discovery to the German chemical firm BASF, which assigned Carl Bosch the job of making the process commercially viable. Osmium has 76 electrons that are exquisitely arranged, which presumably is the reason for its catalytic prowess, but it is also one of the rarest chemicals on the planet, so Bosch and his colleague looked for a cheaper catalyst. They discovered that uranium also worked, but settled on a catalyst made by treating iron with potassium. (Iron is in the same column of the periodic table as Osmium because they have the same arrangement of "outer" electrons, with the result that they have some similar chemical properties.) Today modern industrial catalysts for nitrogen fixation include mixtures of aluminum oxide (Al_2O_3), potassium oxide (K_2O), zirconium dioxide (ZrO_2), and silicon oxide (SiO_2). For this work, Carl Bosch received the 1931 Nobel Prize in Chemistry, which he shared with Friedrich Bergius, another BASF employee.

Chemically, the modern Haber–Bosch process looks something like this:

$$\mathit{Energy} + \mathrm{N_2} + 3\,\mathrm{H_2}^- \xrightarrow{\mathrm{Fe, Fe_3O_4, Al_2O_3}} \mathrm{NH_3} + \mathrm{H_2} \tag{3}$$

The energy comes from temperatures in the range from 750°F to 3000°F, with pressures as great as 350 times atmospheric pressure at sea-level, and the hydrogen comes from natural gas. Today the world is so hungry for nitrogen that the Haber–Bosch process is responsible for 3 percent of the world's carbon emissions and consumes roughly 3 percent of the world's natural gas. Not surprisingly, scientists are constantly looking for ways to improve nitrogen fixation. Areas of current research including finding better catalysts[11] and how biolog-

to his country and the fact that he had converted from Judaism to Christianity, Haber was considered a Jew by the Nazi regime, and fled to England after the Nazis rose to power. "[S]cientists there shunned him for his work with chemical weapons. He traveled Europe, fruitlessly searching for a place to call home, then suffered heart failure in a hotel in Switzerland in 1934. He passed away shortly thereafter at the age of 65, but not before repenting for devoting his mind and his talents to wage war with poison gasses." See King, "Fritz Haber's Experiments in Life and Death" (2012). Zyklon A ultimately led to the development of Zyklon B, the gas that was used in the Nazi extermination camps.

[11] Ashida et al., "Molybdenum-Catalysed Ammonia Production with Samarium Diiodide and Alcohols or Water" (2019).

https://doi.org/10.1017/9781108883719.009 Published online by Cambridge University Press

ical systems work.[12,13,14] After all, alfalfa is able to fix nitrogen at room temperature with just air, water, sunlight, and some clever microbes.

5.1.2 Modeling Chemical Reactions

One way for industry to develop improved nitrogen fixation catalysts would be to better understand what is happening at the atomic level when nitrogen gas becomes ammonia inside those microbes. Chemists think of this process in terms of some chemical bonds being broken while new chemical bonds are created. Much of modern chemistry is devoted to describing and predicting the behavior of such chemical bonds.

Except there is really no such thing as a chemical bond! While students in high school chemistry class learn to visualize bonds as little black lines connecting letters (e.g., $N{\equiv}N$), "bonds" and indeed our entire model of chemical reactions are really just approximations for Schrödinger wave equations that evolve over time and describe the probability that a collection of mass, charge and spin will interact with our measuring devices. It is just far too hard to write down such wave equations, let alone solve them. Meanwhile, the mental models of chemical bonds and other approximations developed over the past 150 years all work pretty well, especially with ongoing refinements, and so chemists continue to use these approximations.[15]

More accurate models that do a better job incorporating the underlying quantum physics would let chemists create more accurate predictions of how these things we call atoms rearrange during the course of a chemical reaction. Highly accurate models would let chemists design and try out catalyst candidates in a computer, with-

[12] Molteni, "With Designer Bacteria, Crops Could One Day Fertilize Themselves" (2017).

[13] *Biological Nitrogen Fixation: Research Challenges – A Review of Research Grants Funded by The US Agency for International Development* (1994).

[14] Manglaviti, "Exploring Greener Approaches to Nitrogen Fixation" (2018).

[15] A current textbook about the chemical bond reminds its readers that there are no electrons spinning around the atoms, only a "charge wave surrounding the nucleus." (I. D. Brown, *The Chemical Bond in Inorganic Chemistry: The Bond Valence Model, 2nd ed.* (2016), Chapter 2.) (See Figure 5.2 in this book, p. 186.) Nevertheless, the author continues, "chemists have largely rejected this simple wave picture of the atom in favor of a hybrid view in which the charge is composed of a collection of electrons that are not waves but small particles, [with the] density of the charge wave merely represent[ing] the probability that an electron will be found at a given location."

out having to go to the trouble of actually synthesizing them in a lab. This is the world of *computational chemistry*, also called quantum chemistry, or even computational quantum chemistry, which uses the math of quantum mechanics to answer questions about the chemical nature of the world around us.

Wave equations describe probabilities, so predicting the behavior of atoms at the quantum level requires programs that explore probability distributions. One way to do this is with a Monte Carlo simulation (see the sidebar "The Monte Carlo Method" on page 189). Simulations take exponentially longer to run as the number of electrons in the system increases – a good rule of thumb is that each additional electron doubles the simulation's running time.

In the Haber–Bosch nitrogen fixation equation presented above, there are 14 electrons among the two nitrogen atoms and 6 hydrogen electrons for a total of 20 electrons. But do not forget that all-important catalyst: that is where the chemical dance of the electrons is happening. Iron has 26 electrons per atom, while Fe_3O_4 has 110, and Al_2O_3 has 50. There must be some extraordinarily complex chemistry happening at the interface of the gaseous nitrogen and the solid catalyst.

To understand that complex chemistry, a computational chemist creates a simulation of the electrons and nuclei. Into the simulation the chemist programs physical constants that have been measured over the decades as well as mathematical functions that represent the laws of quantum mechanics. The more electrons and nuclei, the more complex the simulation.

The math of quantum physics is based on probability, so all of those probabilistic interactions – many coin flips – become inputs to the simulation. For example, some of the random draws might have less electron charge in a particular location between the two nitrogen nuclei and more charge between the nitrogen and iron nuclei that are interacting with some oxygen. This might sometimes push the two nitrogen nuclei slightly further apart – their electrostatic charges repel, after all – which might sometimes cause the charge probability to rearrange a little more, and then all of a sudden … *wham!* … the two nitrogen nuclei can now pick up some free floating protons, and the physics simulation has converted simulated nitrogen into simulated ammonia!

Running this simulation with a classical computer requires many random draws, many crunchings of quantum mathematics, and a lot

Figure 5.2. McMaster University Professor Emeritus I. David Brown observes: “An electron is the smallest quantum of charge that can have an independent existence, but the free electrons that are attracted to a nucleus in order to form a neutral atom cease to exist the moment they are captured by the nucleus. They are absorbed into the charge wave and, like Lewis Carroll’s (1865) Cheshire Cat that disappears leaving only its smile behind, the electron disappears bequeathing only its conserved properties: charge, mass and spin, to the charge wave surrounding the nucleus.” I. D. Brown, *The Chemical Bond in Inorganic Chemistry: The Bond Valence Model, 2nd ed.* (2016), chapter 2.

of matrix mathematics. Remember, classical computers are *deterministic by design.* To explore what happens when four random variables encounter each other, the computer takes random draws on each four variables and crunches the math. One cannot simply explore what happens when the most-probable value of each variable happens, because there might be some important outcome when three of the variables are in a low-probability configuration.

If it takes 10 seconds to simulate a single random variable, it will take on the order of $10 \times 10 \times 10 \times 10 = 10^4 = 1000$ seconds to simulate four random variables. With 10 random variables (and without any optimization), it will take 10^{10} seconds or 115 740 days – roughly 317 years.

These days, a computation that takes 317 years is not a big deal, provided that the computation consists of many individual problems that can be run in parallel. Good news: quantum simulations are such a problem! As we write this book in 2021, cloud providers will rent a computer with 96 cores for roughly $5/hour. One can rent 100 of those computers for $500/hour and solve the 317-year problem in 12 days for $6000. Alternatively, one can rent 1000 of those computers and solve the problem in 29 hours – for the same price of $6000. (This demonstrates why cloud computing is so attractive for these so-called *embarrassingly parallel* workloads.)

Today's massive cloud computing data centers provide only *linear* speedup for these hard problems: if 1000 computers will solve the problem in 29 hours, then 10 000 computers will solve the problem in 2.9 hours. And there's the rub: absent a more elegant algorithm, each additional electron in our hypothetical simulation increases the problem's difficulty *exponentially.* With 20 electron variables, the problem takes on the order of 10^{20} seconds or 3 168 808 781 402 years – 3168 billion years! – which is more time than anyone has.[16] Even with a million 96-core computers (a speedup of 96 million), our hypothetical computation would take 33 008 years, which is still too long. Classical computers are simply not well-suited to simulating probabilistic quantum physics.

Some people believe that quantum computers may be able to efficiently solve problems involving quantum modeling of chemical reactions. Even the "quantum simulators" discussed here, special-

[16]Current estimates are that the universe is somewhere between 15 and 20 billion years old.

https://doi.org/10.1017/9781108883719.009 Published online by Cambridge University Press

purpose machines constructed to solve a specific problem, should be dramatically faster than all of the world's computers working forever … provided that we can scale the quantum simulators to be large enough. As such, quantum chemistry simulation is likely to be the first application for quantum computers in which they are used for something other than doing research and writing papers about quantum computers.

Critics, meanwhile, argue that today's software packages (both commercial and open-source) are based on well-understood, validated approximations that have worked for decades, and that limitations of these systems might be solved merely with more conventional computing power. For example, a September 2020 article by Elfving et al. works real-world physical chemistry problems and concludes that a practical quantum computer that could solve these problems in hours, rather than years, would require millions of physical qubits. The authors' nuanced conclusion is that while quantum computing may one day produce systems that can make meaningful contributions to physical chemistry, a far more promising near-term solution would be to rewrite today's chemistry simulation packages to take advantage of graphical processing units.[17]

5.2 Quantum Factoring (Shor's Algorithm)

As we explained in Section 4.8, "Aftermath: The Quantum Computing Baby" (p. 164), Peter Shor's discovery of an algorithm that can rapidly break numbers down into their prime factors sparked the world's interest in quantum computing. In this section we will describe why Shor's algorithm was so important, how it became a driver of quantum computing, and why it is no longer a driver – at least, not in the public, commercial world. (See Section 3.5.6 (p. 116) for a discussion of what we mean by "rapidly.")

To understand why Shor's algorithm is such a big deal, we start with a discussion of public key cryptography. In Section 5.2.3 (p. 199) we discuss how a quantum computer makes factoring faster. We will then explore whether Shor's algorithm running on a quantum computer would truly be faster than anything that could ever run on a classical computer, or whether we just need better math.

[17]Elfving et al., "How Will Quantum Computers Provide an Industrially Relevant Computational Advantage in Quantum Chemistry?" (2020).

https://doi.org/10.1017/9781108883719.009 Published online by Cambridge University Press

The Monte Carlo Method

Modeling nuclear reactions was one of the first uses of electronic computers in the 1940s. Stanislaw Ulam at Los Alamos was trying to create a mathematical model for the movement of neutrons through material. He couldn't create an exact model, so he ran hundreds of individual mathematical experiments, each modeling the probabilistic interactions between a neutron and the material and finding a slightly different path. Ulam called this the *Monte Carlo method*, named after the casino where his uncle frequently gambled.[a]

Ulam shared his idea with fellow scientist John von Neumann, who directed the team at University of Pennsylvania to program the ENIAC to carry out the computations.

One requirement of algorithms like Monte Carlo is that the random numbers must be truly random. Generating such numbers requires physical randomness, something that the early computers didn't have. Instead, the systems of the day used algorithms to generate sequences of numbers that *appeared* random, but which were actually determined from the starting mathematical "seed." von Neumann later quipped: "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."[b]

It is necessary to use algorithms such as the *Monte Carlo method* when modeling quantum interactions, because it is not possible to solve the Schrödinger wave equation for even mildly complex systems.[c]

Ulam's success was evidenced by the fusion bomb test in November 1952 and decades of employment for physicists at weapons laboratories around the world. By the 1990s modeling had gotten so good that it was no longer necessary to even test the bombs, and the United States signed (but did not ratify) the Comprehensive Nuclear-Test-Ban Treaty.

[a]Metropolis, "The Beginning of The Monte Carlo Method" (1987).

[b]von Neumann, "Various Techniques Used in Connection with Random Digits" (1951).

[c]Random sampling can also be used to find approximate integrals to complex mathematical functions: instead of attempting to find an exact solution, the approach is to evaluate the function at a number of randomly chosen locations and interpolate. This is similar to statistical sampling, except that what's being sampled is a *mathematical universe*, rather than a universe of people or objects.

5.2.1 An Introduction to Cryptography

In modern usage, we use the word "cryptography" to describe the body of knowledge involved in creating and solving secret codes. Here the word "code" means a system for representing information, while "secret" implies that something about the code allows people who know the secret to decode its meaning, while people who do not know the secret cannot.

Secret Key Cryptography

One of the oldest known codes it the "Caesar cipher," which was reportedly used by Julius Caesar for messages to his generals. Messages are encrypted character-by-character by shifting each letter forward in the alphabet by three positions, so T becomes Q, H becomes E, E becomes B, the letter C wraps around to Z, and so on. To decrypt messages simply shift in the other direction. QEB ZXBPXO ZFMEBO FP KLQ SBOV PBZROB, that is, THE CAESAR CIPHER IS NOT VERY SECURE.

The Caesar cipher is called a *secret key algorithm* because the secrecy of the message depends upon the secrecy of the key, and the same key is used to encrypt and decrypt each message. It's not a very good secret key algorithm, because once you know the secret – shift by three – you can decrypt any encrypted message. We call this number three the *key* because it is the key to decrypting the message! You can think of the Caesar cipher as a lock that fits over the hasp used to secure a wooden box, and the number *three* as a key that opens the lock.

We can make the algorithm marginally more complicated by allowing the shift to be any number between 1 and 25: that creates 25 possible encryption keys, so an attacker needs to figure out which one is in play. It's still not very hard to crack the code.

There are lots of ways to make this simple substitution cipher stronger, that is, to make it harder for someone to decrypt or "crack" a message without knowing the secret piece of information used to encrypt the message in advance. This is directly analogous to making the lock on the box stronger. For example, instead of shifting every letter by the same amount, you can make the encrypted alphabet a random permutation of the decrypted alphabet. Now you have a word puzzle called a cryptogram. These can be easy or hard to solve depending on the length of the message, whether or not the message

https://doi.org/10.1017/9781108883719.009 Published online by Cambridge University Press

uses common words, and the number of times each letter is present in the message.

Humans solve these puzzles by looking for patterns in the encrypted message, called a *ciphertext*. We can eliminate such patterns by encrypting each letter with a different key. Now there are no patterns! This kind of encryption algorithm is sometimes called a Vernam cipher (named after its inventor, Gilbert Vernam) or more commonly a *one-time pad* (because spies of yore had encryption keys written on pads of paper, with instructions to use each key once and then destroy it). One-time pads are hard to use in practice, because the key needs to be both truly random and as long as the original message. We discuss them more in Section 7.4 (p. 276).

Public Key Cryptography

For all of human history until the 1970s, cryptography existed as a kind of mathematical deadbolt, in which each encrypted message was first locked and then later unlocked by the same key. There were thus four principal challenges in creating and deploying a working encryption system: 1) Assuring that the sender and the intended recipient of an encrypted message had the same key; 2) Assuring that no one else had a copy of the correct key; 3) Assuring that the correct key could not be guessed or otherwise discovered by chance; 4) Assuring that the message could not be decrypted without knowledge of the key. (See Figure 5.5.)

All of this changed in the 1970s with the discovery of public key cryptography, a term used to describe encryption systems in which a message is encrypted with one key and decrypted with a second.

Originally called *non-secret* encryption, it is now generally believed that public key cryptography was discovered in 1973 by James Ellis, Clifford Cocks, and Malcolm Williamson[18] at the Government Communications Headquarters (GCHQ), the United Kingdom's signals intelligence and information assurance agency (roughly the UK's equivalent of the US National Security Agency (NSA)). The UK intelligence agency reportedly shared the discovery with the NSA,[19] but neither sought to exploit the invention. The basic idea was then rediscovered at Stanford by Professor Whitfield Diffie and Professor Martin Hellman, whose paper "New Directions in Cryptography" in-

[18]Ellis, Cocks, and Williamson, "Public-Key Cryptography" (1975).
[19]Levy, "The Open Secret" (1999).

spired Ronald Rivest, Adi Shamir, and Leonard Adleman at MIT to create a working public key system.[20,21]

The basic concept of public key cryptography is a mathematical lock that is locked with one key and unlocked with a second. The key that locks (encrypts) is called the *public key*, while the key that unlocks (decrypts) is the *private key*. The two keys are mathematically linked and need to be made at the same time.[22]

A locked suggestion box is a good mental model for how public key cryptography works: to encrypt something, write it on a piece of paper and drop it into the locked box. Now the only way to get that message back is by unlocking the box and retrieving the message. In this example, the slot in the box represents the public key, and the key that unlocks the padlock represents the private key (Figure 5.3).

The great advantage of public key cryptography is that it dramatically simplifies the problem of key management. With public key cryptography, each person in an organization simply makes their own public/private keypair and then provides their public key to the organization's central registry, which then prints a phone book containing each employee's name and public key, then sends each employee their own copy. Now any employee can send an encrypted message to any other employee by simply looking up the intended recipient's

[20]Ronald L. Rivest, Adi Shamir, and Len Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" (1978).

[21]The RSA crypto system was published first in Martin Gardner's column in *Scientific American* (Gardner, "Mathematical Games: A New Kind of Cipher That Would Take Millions of Years to Break" (1977)), in which the RSA-129 number that we will discuss on p. 261 was first published. In that article, the MIT professors famously offered US$100 to anyone who could factor the 129-digit number or otherwise decrypt the message that they had encrypted with it. The professors also offered a copy of their technical paper to anyone who sent a self-addressed stamped envelope to their offices at MIT. Rivest discusses this in his Turing award lecture (Ronald L. Rivest, "The Early Days of RSA: History and Lessons" (2011)), following Adleman's lecture (Leonard Adleman, "Pre-RSA Days: History and Lessons" (2011)), and followed by Shamir's (Adi Shamir, "Cryptography: State of The Science" (2011)).

[22]There is a more refined version of public key technology called *identity-based encryption* (IBE) that allows the keys to be made at separate times by a trusted third party. IBE was proposed by Adi Shamir in 1984 (Adi Shamir, "Identity-Based Cryptosystems and Signature Schemes" (1984)). Two working IBE systems were developed in 2001, one by Dan Boneh and Matthew K. Franklin (Boneh and Franklin, "Identity-Based Encryption From The Weil Pairing" (2001)), the other by Clifford Cocks of GCHQ fame (Cocks, "An Identity Based Encryption Scheme Based on Quadratic Residues" (2001)).

Figure 5.3. A locked suggestion box is a good metaphor for public key cryptography. To protect your message, just drop it through the slot. To retrieve your message, you must unlock the padlock and open the lid. Photograph by Hashir Milhan (CC BY 2.0) of a suggestion box in Sri Lanka.

key in the directory, using that key to encrypt a message, and then sending the message using the corporate email system. Nobody will be able to decrypt the message – not even the system administrators who run the corporate email system or the employee who printed the phone book.

Public key cryptography can also be used to create a kind of *digital signature*. In this case, the encrypting key is retained and the decrypting key is published. To sign a document, just encrypt it with your private key, then publish the result as a *signature*. Anyone who has access to your public key (from the directory) can decrypt your signature and get back to the original document. If you practiced good cryptographic hygiene and no one has obtained your private key, now called the *signing key*, then we now have good proof that you alone could have signed the document.

It is still possible for employees to send and receive messages within an organization without using public key cryptography, but the procedures are more involved. One possibility is for the central authority to create a different secret key for every pair of employees that needs to communicate, then to send each pair of employees all of the keys that they need in a sealed envelope. This approach has the feature that individuals can only exchange encrypted email with other individuals with whom they are authorized to exchange messages. Another feature is that the central key-making authority can in theory decrypt any message exchanged by a pair of employees if it retains that pair's key, although the authority can choose to destroy its copy if it wishes to allow the pair to communicate without the possibility of eavesdropping. This is the sort of system that military organizations traditionally set up, and it is presumably what GCHQ and the NSA were using in the 1970s, which is why they saw no need to develop the non-secret encryption that Cocks and Ellis had invented: GCHQ and NSA already had a system that was well-developed and deployed to meet their organizational requirements, and the benefits of digital signatures were not immediately obvious.

For the academics at Stanford and MIT, however, the discovery of public key cryptography opened the door on a new area of intellectual pursuit that combined the fields of number theory and computation. It was an academic green field, full of wonder, possibility, and low-hanging fruit. For example, in 1978, an MIT undergraduate named Loren Kohnfelder realized that digital signatures made it unnecessary for an organization to publish a directory of every employee's public key. Instead, the organization could have a single private/public keypair for the organization itself, and use the private key to sign each employee's public key. The employees could then distribute to each other their own public keys, signed by the organization's public key, to other employees as needed. As long as each employee had a copy of the organization's public key, they could verify each other's keys, and the organization would not need to send out a directory with every employee's public key. Today we call these signed public keys *digital certificates* and the central signing authority a *certificate authority*. With his 1978 undergraduate thesis, Kohnfelder had invented public key infrastructure (PKI).[23]

The following year, Ralph Merkle's PhD thesis[24] introduced the idea of cryptographic hash functions. A hash function is a mathematical function that takes an input of any size and produces an output of a fixed size. The basic concept was invented by IBM engineer Hans Peter Luhn in the 1950s.[25] Merkle's innovation was to have hash functions that produced an output that was both large

[23]Kohnfelder, "Towards a Practical Public-Key Cryptosystem" (1978).
[24]Merkle, *Secrecy, Authentication and Public Key Systems* (1979).
[25]Stevens, "Hans Peter Luhn and The Birth of The Hashing Algorithm" (2018).

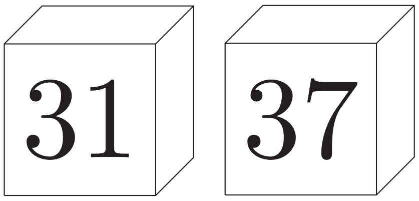
– more than a 100 bits – and unpredictable, so that it would be computationally infeasible to find an input that produced a specific hash. Given such a function, you don't need to sign an entire document, you just need to sign a hash of the document. Today we call such things *cryptographic hash functions* and there are many, the most prominent being the US Government's Secure Hash Algorithm version 3 (SHA-3).

In the end, the discovery catalyzed interest and innovation in cryptography. Academics and entrepreneurs were attracted to the field; they launched companies and ultimately set in motion the commercialization of the Internet, which was only possible because public key cryptography allowed consumers to send their credit card numbers securely over the Internet to buy things.

A Demonstration of RSA Public Key Cryptography

The most widely used public key encryption system today is RSA, named after its inventors Rivest, Shamir, and Adleman. The system is based on math that is beyond this book but it is easy to find if you have interest, and easy to understand if you understand basic number theory. For the purpose of this demonstration we will just assume that you have a set of magic dice that always roll prime numbers and a box that given these two prime numbers p and q outputs two sets of numbers: your public, encrypting key $\boxed{\text{e,n}}$ and your private, decrypting key $\boxed{\text{d,n}}$.

We roll the prime number dice and get two prime numbers:



We drop these into our key generator and get two keys:

public key	
e	7
n	1147

private key	
d	463
n	1147

To encrypt a plaintext message P (which is a number) to produce an encrypted message C (which is another number), we use this mathematical formula:

https://doi.org/10.1017/9781108883719.009 Published online by Cambridge University Press

$$C = P^e \pmod{n} \tag{4}$$

This means multiply the number P by itself e times and then take the integer remainder after dividing the resultant by n. For example, the number 53 (which represents the letter “S”) encrypts as 914:

$$C = 53^7 \pmod{1147} = 1\,174\,711\,139\,837 \pmod{1147} = 641 \tag{5}$$

To decrypt the number 914, we follow roughly the same procedure using the values for d and n:

$$P = C^d \pmod{n} = 641^{463} \pmod{1147} = 53 \tag{6}$$

We haven’t expanded 641^{463} above; the number is 1300 digits long. RSA implementations use a variety of mathematical tricks to avoid naively computing these numbers – for example, you can apply the modulo after *each* multiplication to prevent the intermediate number from getting too large – but it’s easy enough to do the math directly using the Python programming language if you want to check our work.

The RSA algorithm is secure as long as you can’t compute the number d knowing e and n (and provided that you follow some implementation guidance that was developed after the algorithm was first published[26]). It turns out that it’s easy to compute d, however, if you can factor n. Not a lot was known about the difficulty of factoring numbers in 1977, although the best factoring numbers took exponentially more time as the length of the number being factored increases. That’s still the case today. This may be something inherent in the nature of factoring, or it may reflect a limitation in our knowledge. After more than 40 years of intensely studying the question, mathematicians, computer scientists, and cryptographers still don’t know.

5.2.2 Forty Years of Public Key Cryptography

Despite the fact that humanity is still unsure about the fundamental hardness of factoring, we have learned a lot about cryptography over the past 40 years. Here we focus on three significant improvements: speed, algorithmic improvements, and key length.

[26] For an example of such guidance, see Housley, “Use of The RSAES-OAEP Key Transport Algorithm in Cryptographic Message Syntax (CMS)” (2003).

https://doi.org/10.1017/9781108883719.009 Published online by Cambridge University Press

Cryptographic Speed

The computers of the 1970s were too slow for public key cryptography to be practical: a single RSA encryption or decryption on a computer could take as long as 30 seconds. By the 1980s computers were fast enough that it took just a few seconds, and some companies developed and marketed *cryptographic co-processors* that could accelerate the math required to make RSA run fast as well as store the RSA private keys in tamper-proof hardware. By the 1990s general purpose microprocessors were fast enough that special purpose hardware was no longer needed, and these days most microprocessors include special instructions and dedicated silicon that can be used to accelerate both secret and public key cryptography.

As a result, cryptography has gone from being a technology that was only used occasionally, when it was absolutely needed, to a protection that is always enabled. For example, the early web used encryption just to send passwords and credit card numbers, sending everything else over the Internet in plaintext. These days encryption is the default, and web browsers warn when any page is downloaded without encryption.[27]

Algorithmic Improvements

Working together, cryptographers and security engineers have also made stunning improvements to cryptographic systems, making them both faster and more secure.

Although the underlying math of RSA is sound, cryptographers developed many subtle nuances to use it in practical applications. For example, if we simply encrypt letters one code at a time, as we did in the example above, an adversary has a straightforward method to attack the ciphertext. The adversary can encrypt all possible combinations of messages using the public key until a match emerges with the ciphertext. The attacker can do this because the attacker always has access to the target's public key – that's the core reason we are using public key cryptography. This approach of trying every possible combination is called a *brute-force attack* or a *key-search* attack. For this reason, whatever message that's encrypted is always combined with a random string of bits, called a pad. With a long pad it's

[27] Our understanding of Internet security has also expanded, so now we know that a single advertisement, image, or font downloaded without encryption over the Internet can be leveraged by an attacker to compromise your computer's interactions with a remote website.

Elliptic Curve Public Key Cryptography

In the 1980s, cryptographers Neal Koblitz and Victor S. Miller independently suggested that mathematical constructs called "elliptic curves over finite fields" might provide the sort of functionality operations required to build a working public key cryptography system.[a] They were right, and elliptic curve cryptography (ECC) was developed and standardized in the 1990s, culminating with the adoption of the Elliptic Curve Digital Signature Algorithm (ECDSA) in 1999. Following that, the US National Security Agency aggressively promoted ECC over RSA. Since relatively short ECC keys were just as secure as RSA keys that were much longer, ECC systems led to faster computations that required less power.

At first, the primary disadvantage of ECC was the need to license patents from Certicom, the Canadian company founded in 1985 to commercialize ECC technology. Whereas RSA was protected by a single US patent that expired in 2000,[b] Certicom aggressively patented many different aspects of both the ECC math and efficient ECC implementations.

More recently, security experts have raised some concerns regarding the technology – specifically that the number theory of elliptic curves is less well-studied than the number theory that underlies the RSA algorithm. In 2015, Neal Koblitz and Alfred Menezes noted that the NSA was moving away from elliptic curve cryptography.[c]

Like RSA, the math that underlies ECC is also vulnerable to quantum computers. And since the ECC keys are significantly shorter than RSA keys, quantum computers will be able to crack the ECC keys in use today long before they are able to crack today's RSA keys. Assuming that there are no fundamental scientific limits to scaling up the quantum computer, "it's just a matter of money," observed Koblitz and Menezes.

[a] Neal Koblitz, "Elliptic Curve Cryptosystems" (1987); Miller, "Use of Elliptic Curves in Cryptography" (1986).

[b] L. M. Adleman, R. L. Rivest, and A. Shamir, "Cryptographic Communications System and Method" (1983).

[c] N. Koblitz and Menezes, "A Riddle Wrapped in an Enigma" (2016).

impossible for the attacker to try every combination; padding also assures that the same message will always encrypt differently, which makes cryptanalysis harder. RSA without a pad is called *Textbook RSA*: it's good enough for textbooks, but it doesn't actually protect your message.

Engineers developed clever encryption protocols that limit the number of public key operations that need to be computed. This is done by combining public key cryptography with traditional secret key cryptography. For example, an hour of HD video (roughly 10 GB of data, with compression) can be encrypted with a single public key operation. This is done by first encrypting the video with a randomly generated secret key, and then encrypting the secret key with a public key algorithm. This approach is sometimes called a *hybrid system*; it is the approach that is used by both the Trusted Layer Security (TLS) protocol and the Secure Shell (SSH) protocols used to send information over the Internet.

5.2.3 Cracking Public Key with Shor's Algorithm

Here is one measure of public key technology's success: today the vast majority of information sent over the Internet is encrypted with TLS, the hybrid system described above that uses public key technology to exchange a session key, and then uses the session key to encrypt the information itself. If you are viewing web pages, you are probably using TLS.

TLS is sometimes called a *pluggable protocol*, meaning that it can be used with many different encryption algorithms – it's as simple as plugging-in a new algorithm implementation. When you type a web address into your browser, your browser opens a connection to the remote website and the remote website sends to your browser the website's public key certificate, which is used to establish the website's identity. The two computers then negotiate which set of algorithms to use based on which algorithmic plug-ins the web server and the web browser have in common. Today there are tools built into most web browsers to examine website certificates and the TLS connections, but these tools can be confusing because the same website can appear to provide different certificates at different times. This is typically because a single "website" might actually be a collection of several hundred computers, all configured with different certificates.

Because the public key certificate is sent over the Internet when a web page is downloaded, anyone who can eavesdrop upon and capture the Internet communications now has all of the information that they need to decrypt the communications, provided that they have sufficient computing power to derive the website's matching private key from its public key – that is, to "crack" the public key. In the case of RSA, this is the very factoring problem posed by decrypting the *Scientific American* message that was encrypted with RSA-129. In the case of elliptic curve algorithms, other mathematical approaches are used to crack the public key.

Before the invention of Shor's algorithm, the fastest factoring algorithms required exponentially more time to execute as the number of bits in the public key increased. Shor's algorithm uses an approach for factoring that has only *polynominal complexity*: longer keys still take longer to factor, just not exponentially longer. The catch is that Shor's algorithm requires a working quantum computer with enough stable qubits to run a quantum algorithm that helps to factor the number in question: with perfect qubits, factoring the numbers used in modern cryptographic system would require thousands of qubits. But if the qubits have even the smallest amount of noise, then it will be necessary to use quantum error correction, increasing the number of qubits needed to roughly a hundred million (see p. 206).[28] Of course, the first computer to use transistors was built in 1953 at Manchester University: it had just 92 point-contact transistors that had been constructed by hand. Today's Apple M1 microprocessor has 16 billion transistors, built with a feature size of just 5 nanometers.

Shor's algorithm contains a classical part and a quantum part. The classical part contains some of the same number theory that powers RSA encryption, which isn't terribly surprising since both are based on prime numbers, factoring, and Euler's Theorem. To use RSA, the *code-maker* randomly chooses two prime numbers, p and q. These numbers are multiplied to compute N and also used to create the public key and private key. With Shor's algorithm, the attacker just has the public key, which contains N. The attacker also has access to a quantum computer that can perform two quantum functions: the quantum Fourier transform and quantum modular exponentiation. With these functions, the attacker can factor N, learning p and q, and re-generate the code-maker's private-key.

[28]Mohseni et al., "Commercialize Quantum Technologies in Five Years" (2017).

With this private key, the attacker can decrypt any message that was encrypted with the code-maker's public key.

At a high level, one might consider Shor's algorithm as a carefully designed collection of dual-slit experiments, where the slits are arranged according to the public key, N, in such a way that the interference pattern displayed on the screen reveals information about the factors p and q. One might think of the quantum computer as taking an X-ray of the number N. If the bits of N are arranged in just the right way, if they are connected to just the right quantum circuit, and if the X-rays are sent from just the right directions, then the diffraction pattern (see Appendix B) will reveal properties of p and q.

Alas, explaining either the classical *or* the quantum aspects of Shor's algorithm requires more math and physics than we require for readers of this book, so we refer interested readers with sufficient skills to other publications, including the second version of Shor's 1997 paper[29] which can be downloaded from arXiv,[30] as well as the Wikipedia article on Shor's algorithm.[31]

If you had a quantum computer with sufficiently many stable qubits to run Shor's algorithm, and *if* you had recorded the complete encrypted communication between a web server and a web browser at anytime from the dawn of the commercial Internet through today, *then* decrypting that communication would be straightforward.

For example, consider an unscrupulous internet service provider (ISP) that wants to eavesdrop on a user's email. Before 2008, the ISP merely needed to capture the user's packets and reassemble them into web pages – a fairly trivial task.[32] But since 2008 Google has allowed users to access the server using encryption,[33] and in 2010 Google made encryption the default. Once the user started using encryption, the nosy ISP would be out of luck: the web pages

[29] Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer" (1997).

[30] arxiv.org/abs/quant-ph/9508027v2

[31] With some amusement, we note that in June 2021 the quantum algorithm section of the Wikipedia article contained this note: "This section **may be too technical for most readers to understand**. Please *help improve it* and *make it understandable to non-experts*, without removing the technical details." We encourage any of our readers with sufficient skill to accept this challenge.

[32] Ohm, "The Rise and Fall of Invasive ISP Surveillance" (2009b); Bellovin, "Wiretapping The Net" (2000).

[33] Rideout, "Making Security Easier" (2008).

would be encrypted using RSA cryptography. However, if the ISP had recorded these packets and later rented time on a sufficiently large quantum computer, all the ISP would need to do is to extract Gmail's public key certificate, factor N, apply the RSA key generation algorithm to compute the private key, use the private key to decrypt something that the *master secret* was used to encrypt the web pages, and then use the master secret to decrypt the individual pages. This is not hard to do – there exists software that readily performs all of the reassembly and decryption – provided that you have a copy of the server's private key.

If you had captured the packets and *didn't* have a quantum computer, there are still other ways to get that private key. You might be able to get it by hacking into Google's server and stealing it. Alternatively, you might be able to bribe someone at Google, or even obtain a court order against Google to force the company to produce its private key or use it to decrypt the captured transmission.

In 2011, Google made a change to its computers to remove the risk that a stolen private key could be used to compromise the privacy of its service users: Google implemented *forward secrecy* by default.[34] Also known as *perfect forward secrecy*, the term is applied to security protocols that use session keys that are not revealed even if long-term secrets used to create or protect those session keys are compromised. In the case of web protocol, forward secrecy is typically assured by using digital signatures to certify an ephemeral cryptographic key created using the Diffie–Hellman key agreement protocol, which is an interactive public key encryption algorithm that allows two parties to agree on a shared secret.[35]

Google's 2011 move to forward secrecy is a boon for privacy: it means that after the conclusion of communications between a user's web browser and the Gmail server, not even Google can use its own private key to decrypt communications that might have been covertly recorded. This is because Google's Gmail server destroys its copy of the ephemeral encryption key that was used to encrypt the session when the session concludes.

[34]Langley, "Protecting Data for The Long Term with Forward Secrecy" (2011).

[35]Diffie–Hellman is an *interactive* algorithm because performing the protocol requires the two parties to exchange information with each other and act upon the exchanged information. In this way it is different from RSA, which is a *non-interactive protocol*, because it is possible for one party to encrypt or decrypt information using RSA without the active participation of the other party.

It turns out that the forward secrecy algorithm used by Google, the Diffie–Hellman key agreement protocol, is also vulnerable to an attacker that has a quantum computer. This is because the security of the Diffie–Hellman algorithm depends on the difficulty of computing something known as a *discrete logarithm*, and the quantum part of Shor's algorithm can do that as well. So those packets recorded by the ISP in our scenario are still vulnerable to some future attacker with a large-enough quantum computer.

5.2.4 Evaluating The Quantum Computer Threat to Public Key Cryptography

Factoring is clearly a problem that quantum computers will be able to solve faster than classical computers if they become sufficiently large. Will quantum computers ever actually be large enough to pose a threat to public key cryptography? *We don't know the answer to this question today.*

In 2001, a 7-qubit bespoke quantum computer constructed by Isaac Chuang's group at IBM Alamaden Research Center successfully factored the number 15 into its factors 3 and 5.[36] The number 15 is represented in binary by four bits: `1111`. The number 15 is also, not coincidentally, the smallest number that is not prime, not even, and not a perfect square. So realistically, it's the smallest number that the IBM team could have meaningfully factored.[37]

The quantum "computer" that IBM used doesn't look anything like our modern conception of a computer: it was a tube containing a chemical that IBM had synthesized especially for the experiment, a chemical called a "perfluorobutadienyl iron complex with the inner two carbons," and with chemical formula (Figure 5.4):

$$F_2C{=}C(Fe(C_5H_5)(CO)(CO))CF{=}CF_2$$

The quantum circuit was played through the tube as a series of radio frequency pulses, and the qubits were measured using nuclear magnetic resonance (NMR), a procedure in which a material is placed in a strong magnetic field and probed with radio waves at

[36] Vandersypen et al., "Experimental Realization of Shor's Quantum Factoring Algorithm Using Nuclear Magnetic Resonance" (2001).

[37] Even numbers are easy to factor: just divide them by two. Numbers that are perfect squares are also easy to factor: just take their square root, which can be quickly computed using Newton's method. The number 15 is the smallest non-even number that is the product of two different primes: three and five.

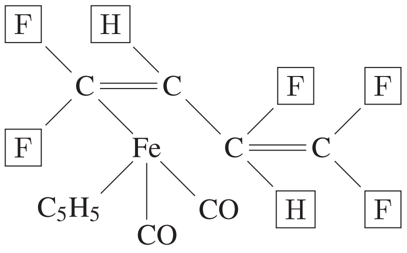


Figure 5.4. The perfluorobutadienyl iron complex with the inner two carbons that IBM scientists synthesized in 2001 for the purpose of factoring the number 15. The seven qubits are represented by the five fluorine (F) and two hydrogen (H) atoms shown surrounded by a $\boxed{\text{box}}$.Vandersypen et al., "Experimental Realization of Shor's Quantum Factoring Algorithm Using Nuclear Magnetic Resonance" (2001).

different frequencies. We discuss NMR-based quantum computers in Section 4.8.2 (p. 168).[38]

Since IBM's demonstration, other researchers have factored other numbers on quantum computers. None of these approaches have managed to factor a number out of reach of a conventional computer. Most of the numbers factored can be factored with pen and paper. For example, in 2012 a team led by Nanyang Xu at the University of Science and Technology of China, Hefei, successfully factored the number 143 using "a liquid-crystal NMR quantum processor with dipole–dipole couplings."[39] The factors were 11 and 13, of course. What's exciting is that the researchers used a different factoring approach called *adiabatic quantum computation* (AQC), using only four qubits. In 2014, Nikesh Dattani at Kyoto University and Nathaniel

[38]It may seem implausible that a tube containing a solution of a specially synthesized compound inside a scientific instrument is actually *computing*, at least in the way that we typically think of the term. But the IBM experiment demonstrated that the computational media responded in a way that was consistent with factoring the number 15, producing the numbers 3 and 5.

It turns out that computing is more fundamental than electronics, and there are many different media that can be used for computation. For example, in the 1970s Danny Hillis created a computer from Tinkertoy rods and wheels that could play Tic-Tac-Toe. "It could have been built by any six-year old with 500 boxes of tinker toys and a PDP-10," Hillis wrote at the time (D. Hillis and Silverman, "Original Tinkertoy Computer" (1978)). Another improbable computing medium is the seemingly haphazard but highly structured collection of lipids, proteins, nucleic acids, small amine molecules, amino acids, and neuropeptides that make up the human neurological system.

[39]Xu et al., "Quantum Factorization of 143 on a Dipolar-Coupling Nuclear Magnetic Resonance System" (2012).

Bryans at University of Calgary posted a follow-up article to the arXiv open-access archive purportedly showing that the published results of the Chinese researchers could also be used to factor the numbers 3599, 11 663, and 56 153.[40][41] The work on AQC factoring is exciting because it suggests that research in quantum computing may eventually lead researchers to make fundamental discoveries about factoring or even the nature of computation, with results that could then be applied to both quantum and classical computers. Although there have been no such discoveries to date, the field of quantum factoring is still quite young compared with other branches of number theory.

As of January 2019, the current record for factoring published in the peer-reviewed literature is held by Chinese scientists, who factored the 7-digit (20-bit) number 1 005 973 using 89 qubits on a D-Wave quantum annealing machine. The team noted that by using a factoring algorithm based on quadratic unconstrained binary optimization (QUBO), the team was able to constrain the factoring problem to the type of qubits that D-Wave provides. "Factoring 1 005 973 using Shor's algorithm would require about 41 universal qubits, which current universal quantum computers cannot reach with acceptable accuracy," the authors noted wryly.[42] This development was exciting because it demonstrated a new use for the D-Wave annealer, discussed further in Chapter 6, which is limited to certain kinds of applications. The scientists reasoned that because D-Wave scaled its annealer from just 128 bits to 2000 in just seven years, perhaps a machine capable of factoring the kinds of numbers used to secure today's commercial Internet might soon be constructed.

We disagree: such a capacity would require a D-Wave computer with significantly more qubits than seems likely for the foreseeable future. (As of June 2021, D-Wave's largest system, the Advantage, has just 5000 qubits.[43]) To crack the RSA systems that are used

[40]Dattani and Bryans, "Quantum Factorization of 56153 with Only 4 Qubits" (2014).

[41]The Dattani/Bryans work was covered by the news site Phys.org (Zyga, "New Largest Number Factored on a Quantum Device Is 56,153" (2014)), but the work did not appear in the peer-reviewed literature.

[42]Peng et al., "Factoring Larger Integers with Fewer Qubits via Quantum Annealing with Optimized Parameters" (2019).

[43]D-Wave Systems Inc., "D-Wave Announces General Availability of First Quantum Computer Built for Business" (2020).

to protect today's commercial Internet would require the ability to factor 2048- or 4096-bit numbers.[44]

Even with this work on factoring – perhaps because of it – there is still wide agreement in the scientific community that a practical application of quantum computing to factoring is far off. It is unclear whether the winning system will be a universal quantum computer with stable qubits that can also factor, or a special purpose device designed to perform factoring quickly. The advantage of the first machine is generality. The advantage of the second is that it could likely be developed years before a general-purpose quantum computer, and it could probably be developed for less money, and possibly in secret.

Yet another threat could come through a quantum-classical approach where the factoring problem is solved in parts with a quantum computer, and a classical computer is used to combine and process these parts to come to a full solution.[45] The partial analysis approach might afford today's small quantum computers a role in cryptanalysis.

Google scientists have projected that factoring a conventional RSA public key in use on the commercial Internet today "would take 100 million qubits, even if individual quantum operations failed just once in every 10 000 operations."[46] A National Academies group assessed in 2019 that "to create a quantum computer that can run Shor's algorithm to find the private key in a 1024-bit RSA encrypted message requires building a machine that is more than five orders of magnitude larger and has error rates that are about two orders of magnitude better than current machines, as well as developing the software development environment to support this machine." The authors of the report stated that it is "highly unexpected" that a quantum computer that can break a 2048-bit RSA key will be built before 2030.[47]

[44]For comparison, as of February 28, 2020, the largest RSA challenge number to be publicly factored is RSA-250, a 250-digit, 829-bit number (Boudot et al., "Factorization of RSA-250" (2020)). The total amount of computer time required to perform the computation "was roughly 2700 core-years, using Intel Xenon Gold 6t130 CPUs as a reference (2.1Ghz)," the authors reported (Goodin, "New Crypto-Cracking Record Reached, with Less Help Than Usual From Moore's Law" (2019)).

[45]Ekerå and Håstad, "Quantum Algorithms for Computing Short Discrete Logarithms and Factoring RSA Integers" (2017).

[46]Mohseni et al., "Commercialize Quantum Technologies in Five Years" (2017).

[47]Grumbling and Horowitz, *Quantum Computing: Progress and Prospects* (2019).

DNA-Based Computing and Storage

DNA (deoxyribonucleic acid) is the polymerized molecule inside cells that carries inheritance information used to synthesize proteins. It has been called "the building block of life."

Before the event of quantum computers, some researchers thought DNA's ability to encode and to reproduce information might also make DNA a useful substrate for computing. One proponent was Leonard Adleman (the "A" of RSA), who is frequently credited with inventing the field.

Adleman encoded a small graph into a DNA molecule and then used biomolecular reagents "to solve an instance of the directed Hamiltonian path problem."[a] This was highly significant, as the Hamiltonian Path problem is NP-complete. If DNA computing could solve it efficiently, and if the system can be scaled up, DNA can be used to solve any other NP problem. In particular, a DNA computer would be able to factor efficiently.[b]

Work on DNA computing has continued, with researchers developing a variety of DNA-based algorithms.[c] A recent review of "DNA-based Cryptanalysis"[d] found that the field remains promising. But it has been eclipsed by quantum computing.

There have been significant breakthroughs in using DNA to encode information directly. In June 2019, a Boston-based startup called Catalog announced that it had encoded all 16 GB of Wikipedia into a set of DNA strands the size of a pencil eraser.[e] DNA is also stable over long periods of time; DNA is now routinely recovered from humans that lived thousands of years ago. Since DNA is the basis of life, the ability to transcribe DNA is likely to be re-invented by any future biologically based civilization on Earth, should our current technological society fail. DNA thus makes an excellent backup medium not just for organizations, but also for the intellectual heritage of our civilization.

[a] Leonard Adleman, "Molecular Computation of Solutions to Combinatorial Problems" (1994).

[b] Factoring is not NP-complete, but it is contained within NP.

[c] W. Chang, Guo, and M. S. Ho, "Fast Parallel Molecular Algorithms for DNA-Based Computation: Factoring Integers" (2005).

[d] Sadkhan and Yaseen, "DNA-Based Cryptanalysis: Challenges, and Future Trends" (2019).

[e] Shankland, "Startup Packs All 16GB of Wikipedia Onto DNA Strands to Demonstrate New Storage Tech" (2019); Catalog Technologies, Inc., "Catalog" (n.d.).

5.2.5 Post-Quantum Cryptography

Fully realized, large-scale, and sufficiently error-free, quantum computers will mean that public key encryption systems based on the RSA, Diffie–Hellman, and Elliptic Curve systems are no longer secure. But this will not mean the end of public-key cryptography.

Since the discovery of public key cryptography in the 1970s, dozens of public key encryption algorithms have been devised. Of these, many do not depend on the difficulty of factoring or computing a discrete logarithm, and as such these algorithms would not be crushed by Shor's algorithm and a suitably large quantum computer. In fact there are so many choices and they are all so significantly different that it is not immediately clear which is the best.

To help the world choose, in 2016 NIST embarked on the Post-Quantum Cryptography (PQC) Standardization effort. At the time, NIST stated that the competition for a PQC asymmetric algorithm would likely be more complex than its successful competitions to pick the Advanced Encryption Standard (AES) and the Secure Hash Algorithm 3 (SHA-3). "One reason is that the requirements for public-key encryption and digital signatures are more complicated. Another reason is that the current scientific understanding of the power of quantum computers is far from comprehensive. Finally, some of the candidate post-quantum cryptosystems may have completely different design attributes and mathematical foundations, so that a direct comparison of candidates would be difficult or impossible."[48]

NIST started with a field of 82 algorithm candidates, which was reduced to 26 algorithms in early 2019. In July 2020 NIST announced the "Round 3 candidates" for the competition, with four public-key and key-establishment algorithms under consideration as "finalists:" Classic McEliece,[49] CRYSTALS-KYBER,[50] NTRU,[51] and SABER.[52] Another three algorithms are under consideration for digital signature algorithms: CRYSTALS-DILITHIUM,[53] FALCON,[54] and Rainbow.[55] Each algorithm is being presented in a web-based

[48]National Institute of Standards and Technology, "Post-Quantum Cryptography" (2017).

[49]classic.mceliece.org

[50]pq-crystals.org

[51]ntru.org

[52]www.esat.kuleuven.be/cosic/pqcrypto/saber/

[53]pq-crystals.org

[54]falcon-sign.info

[55]www.pqcrainbow.org

seminar open to the public, with the previous presentations and videos archived on the NIST website. It is unclear when the process will be finished, but it is likely that the scientific community will have standardized a new family of asymmetric algorithms long before the availability of quantum computers with sufficient power to crack the algorithms in use today.

In the meantime, all of the algorithms that NIST is evaluating are published, several with accompanying intellectual property statements stating that the authors do not hold patents on the algorithms, have not filed for patents, and have no intention to file for patents. This means that the algorithms are available for experimentation now! And indeed, July 2016, Google announced that it had deployed its experimental CECPQ1 key agreement protocol in "Chrome Canary," the experimental, nightly build version of its popular Chrome web browser.

"Quantum computers exist today but, for the moment, they are small and experimental, containing only a handful of quantum bits," Google's software engineer wrote in the company's Security Blog.[56] "However, a hypothetical, future quantum computer would be able to retrospectively decrypt any internet communication that was recorded today, and many types of information need to remain confidential for decades. Thus even the possibility of a future quantum computer is something that we should be thinking about today."

Google's protocol uses the conventional and PQC algorithms in parallel, so that both must be successfully attacked together, during the same session, in order for the contents of a protected session to be compromised.

One of the reasons that Google decided to experiment with PQC is that the PQC data structures are significantly larger and slower to compute than the data structures used today. Thus, it makes sense to experiment with this technology now, on a limited scale.

In 2019 Google and the webhosting company Cloudflare continued the experiment, jointly deploying an improved algorithm called CECPQ2. "With Cloudflare's highly distributed network of access points and Google's Chrome browser, both companies are in a very good position to perform this experiment."[57]

[56]Braithwaite, "Experimenting with Post-Quantum Cryptography" (2016).
[57]Kwiatkowski, "Towards Post-Quantum Cryptography in TLS" (2019).

If you are interested in learning more about the PQC algorithms, Kwiatkowski's illustrated blog post does a great job explaining them, although it would be useful to have first taken a course in college-level algebra.

5.3 Quantum Search (Grover's Algorithm)

Two years after Shor showed that a large enough quantum computer would be able to factor the numbers used to secure the Internet, Lov Grover (also at Bell Labs) made a startling discovery: a properly constructed quantum computer could speed up all sorts of computations that have a certain mathematical property. The speedup was not as significant as Shor's: instead of turning a problem that is computationally intractable into one that can be solved in just a few hours, Grover's algorithm gives a square-root speedup: if solving a problem takes on order of N steps without Grover, typically abbreviated $O(N)$, it now takes on the order of the square root of N steps – that is, $O(\sqrt{N})$. On the other hand, whereas Shor's algorithm can only be applied to the relatively obscure domain of number theory, Grover's algorithm can be broadly applied to a wide range of practical problems. Grover's algorithm is the second major quantum computing algorithm.

Later in this section we will discuss how Grover's algorithm can be used to crack a version of one of the world's most popular encryption algorithms. We'll show why this was such a big deal at the time, and then discuss why it's not really a big deal any more. After that, we'll discuss other applications for Grover's algorithm. To get started, though, we need to further explore the world of cryptography and code cracking.

5.3.1 Symmetric Ciphers: DES and AES

In 1977 the US Government adopted a standard algorithm for encrypting data that it unceremoniously named the Data Encryption Standard. Before the adoption of the DES, the few companies that sold data security equipment to the public generally made up their own encryption algorithms and asserted that they were secure. This created a difficult commercial environment, because most customers (including most government customers) were not equipped to evaluate the vendors' claims. The DES solved this problem: after it was adopted, vendors could simply follow Federal Information Processing Standard 46: no longer did they need to claim that the algorithm they had cooked up in their labs was mathematically secure. This is the function of standards, and with the DES the standardization process worked beautifully. Both inside and outside the US government, the algorithm was rapidly adopted and deployed.

The adoption of the DES was not without controversy, however. In choosing the DES, the National Bureau of Standards did not use an existing military encryption algorithm. Instead, NBS (the precursor to today's National Institute of Standards and Technology) invited submissions from industry and academia. The first submission round was unsuccessful. For the second round, IBM submitted an algorithm it had developed called Lucifer, based on a novel construction created by the German-born mathematician Horst Feistel (1915–1990).[58]

Ideally, symmetric block cipher algorithms like DES and Lucifer have the property that the only way to decrypt an encrypted message is by knowing (or guessing) the correct key. Clearly, one way to attack such a cipher is to try all possible keys – the brute-force approach. In practice there are other kinds of attacks; such attacks make it possible to correctly guess the decryption key without explicitly trying all of them.

The original Lucifer algorithm had a 128-bit key length (see the sidebar "Key Length" on page 213), but after analysis by the National Security Agency, the algorithm's internals were changed somewhat and the key shortened to 56 bits. (It was widely assumed at the time that the US Government had intentionally weakened Lucifer because US intelligence agencies didn't want an encryption algorithm adopted as a national standard that was too difficult to be cracked. In fact, we now know that the final DES algorithm with its 56-bit keys was *stronger* than the 128-bit algorithm: unlike Lucifer, DES was resistant to a cryptanalysis technique called "differential cryptanalysis" that was not widely known in the 1970s and would not be

[58]Feistel's family fled Germany in 1934. He enrolled at MIT in Physics and graduated in 1937, then proceeded to earn a master's degree at Harvard. At the outbreak of World War II Feistel immediately came under suspicion because of his German citizenship, but his talents were well recognized by others in the US government: Feistel was granted US citizenship on January 31, 1944, and awarded a top secret security clearance the following day. He worked at the US Air Force Cambridge Research Center, MIT's Lincoln Laboratory, and MITRE, before moving to IBM.

discovered by academic cryptographers until the 1990s.[59])

When DES was adopted in 1977 it was not feasible for an attacker to try all $2^{56} = 72\,057\,594\,037\,927\,936$ possible keys to crack a message, but this proved to be possible by the 1990s. To make DES stronger, some organizations adopted a variant called *triple-DES* in which DES was used three times over, each time with a different key, to encrypt a message. This produced an effective key size of 168 bits, but it was also three times slower than a single encryption. There were also lingering doubts as to whether or not the DES had vulnerabilities that had been intentionally hidden by its creators which might make even triple-DES suspect.

In the late 1990s, NIST ran a second public competition to select a new national encryption standard. This time the vetting process was public as well. After two years, NIST adopted the Advanced Encryption Standard (AES), a symmetric block encryption algorithm developed in the 1990s that is better than DES in every possible way.

AES has three primary modes of operation: AES-128, AES-192, and AES-256, with 128-bit, 192-bit, and 256-bit keys respectively. In practice, only AES-128 and AES-256 are widely used: AES-128 is the fastest, for applications that require the fastest possible algorithm, and AES-256 for the applications where speed is not the most important factor. Because the strength of the algorithm doubles with each additional bit, AES-256 is at least 2^{128} times stronger than the 128-bit version.

In fact, the number 2^{128} is so impossibly large that it is not possible to crack a message encrypted with AES-128 using brute-force search on a classical computer: there is simply not enough time. For example, if you had five billion computers that could each try 90 billion AES-128 keys per second, it would take 24 billion years – roughly the age of the Universe – to try all possible AES-128 keys. Without a functioning quantum computer running Grover's algorithm, the only way that an AES-128 message will be cracked will be if a significant underlying mathematical vulnerability is found in the AES algorithm itself. Today such a discovery does not seem likely.

However, it may be possible to crack such messages using Grover's algorithm running on a sufficiently large quantum computer. We dis-

[59] Coppersmith, "The Data Encryption Standard (DES) and Its Strength against Attacks" (1994).

Key Length

The most visible change in cryptography over the past 40 years is the way that cryptographic keys have steadily increased.

Key length is traditionally expressed in bits. A key length of two means that there are four possible secret keys: `00`, `01`, `10`, and `11`. With a key length of three, there are eight possible secret keys: `000`, `001`, `010`, `011`, `100`, `101`, `110`, and `111`. With 4 bits there are 16 possible keys, and with 8 bits there are 256. Concisely, if there are n bits, there are 2^n possible secret keys – the number of keys grows *exponentially* as the number of bits increases. With a strong secret key algorithm, it is necessary to try every possible key in order to crack the message: there are no algorithmic short-cuts.

Whereas adversaries will attack a message encrypted with a secret-key algorithm by trying to decrypt the message, attacks against public-key algorithms typically involve attacking the public key itself. In the case of RSA, such attacks involve factoring the product of the two prime numbers p and q. Such factoring is harder with longer public keys. As a result, engineers have used longer and longer public keys as computers have gotten better at factoring.

In the early days of the commercial Internet, web browsers supported an intentionally weak 512-bit RSA algorithm and a stronger 1024-bit algorithm. The idea was that the weakened algorithm was to be used outside the US and for non-commercial applications, and the 1024-bit version was to be used within the US for commercial applications. Today there are no significant export restrictions on cryptographic software and 2048-bit RSA (617 decimal digits) is widely used, although 4096-bit RSA (1234 decimal digits) systems are increasingly being deployed. For comparison, the original RSA-129 number is 426 bits (129 decimal digits), and the number 1147 used in the example on page 195 is 11 bits (4 decimal digits).

cuss this below in Section 5.3.3 (p. 218).

5.3.2 Brute-Force Key Search Attacks

As we mentioned above, messages encrypted with symmetric encryption algorithms can be forcibly decrypted, or "cracked," by trying all possible keys in sequence. In Table 5.1 we show how this works in practice. We have an 8-character message that has been encrypted with a key that was specially created for this text. The first few attempts fail, but eventually we find one that succeeds. In an actual brute force search, the computer stops when it finds a decryption succeeds, but in the table we keep going until we've tried all 72 quadrillion possibilities.

There are two technical challenges to conducting a key search attack: the time it takes to try all possible keys, and the difficulty of

Figure 5.5. A safe with a combination lock on its door is a good metaphor for secret key cryptography and symmetric ciphers. To protect your message, just enter the combination lock on the panel, open the safe, put in your message, and close the door. To retrieve your message, enter the same combination on the panel, open the door, and retrieve your message. Photograph by Dave L. Jones (EEVBlog), Wikimedia Commons Account Binarysequence (CC BY-SA 4.0).

Table 5.1. Decrypting a message encrypted with the Data Encryption Standard by trying all possible keys. Each DES key is 56 bits long; there are roughly 72 quadrillion keys. Characters that are not printable are displayed with a bullet (•). Notice that when the correct key is found, all of the decrypted characters are printable. In this case the key was found roughly halfway through because it starts with the bit sequence 1000. The same approach can be used with AES, except that there are 2^{128} = 340 282 366 920 938 463 463 374 607 431 768 211 456 possible keys in its weakest implementation.

Trial	Binary Key (56-bits)	Decrypted Output	Text
0	0000 … 0000	BE 47 A1 7A 2E 81 0E 8C	¾G¡z.•••
1	0000 … 0001	62 59 0B B1 CB 67 8F 3A	bY•±Ëg•:
2	0000 … 0010	B3 9B 0D 12 1F C5 A9 7C	³••••Å©|
3	0000 … 0011	84 19 9D C6 B0 F5 AD 75	•••Æ°õ•u
4	0000 … 0100	D4 E6 90 8D 8F 77 EA 07	Ôæ•••wê•
	…		
38 326 038 678 974 151	1000 … 0111	42 65 72 6B 65 6C 65 79	Berkeley
	…		
72 057 594 037 927 935	1111 … 1111	FB 90 3D D5 99 A3 27 3D	û•=Õ•£'=

recognizing a correct decryption.[60] The time is determined by how many keys per second your code-cracking machine can attempt, and how many code-cracking machines you happen to have. For example, at Bletchley Park during World War II, the Bombe (see p. 80), designed to crack the three-rotor version of the Germans' Enigma code, could cycle through all 17 576 possible rotor combinations in 20 minutes. With two of these machines, the British could try half the combinations on one machine and half on the other, and crack a message in 10 minutes. Or they could attack two messages with the two machines, and use the full 20 minutes to crack each. Of course, 20 minutes to crack a message was the *worst case*; on average a message would be cracked after half of the rotor positions had been tried. It was also necessary to detect when the correct rotor position was found. The Germans made this easier by their tendency to begin their encrypted messages with the same sequence of characters.

[60] Many treatises on cryptography and code-breaking ignore the challenge of detecting when text is correctly decrypted. In practice, this challenge is readily overcome, provided that the attacker knows something about the format of the decrypted messages. This is called a *known plaintext attack*. In some cases the attacker can arrange for a message of its choosing to be encrypted by the system under attack; this is called a *chosen plaintext attack*.

When the US Data Encryption Standard was adopted by the National Bureau of Standards (NBS) in 1977, Hellman wrote a letter to NBS arguing that the reduction of the DES keysize from 64 bits to 56 bits suggested that it was done "to intentionally reduce the cost of exhaustive key search by a factor of 256."[61] In a follow-up article, Diffie and Hellman hypothesized that it should be possible to create a special-purpose DES-cracking microchip that could try a million keys each second. With a million such chips, it would be possible to try all 2^{56} keys in a day. They estimated the cost of constructing such a machine at \$20 million in 1977 dollars; assuming a five-year life of the machine and a daily operating cost of \$10 000, the average cost of cracking a DES-encrypted message in 1977 would be just \$5000, including the cost of developing the machine.[62] With expected improvements in microelectronics, the Stanford professors estimated that the cost of their hypothetical DES-cracking machine would be just \$200 000 by 1987. In fact, it actually took 20 years. In 1998 the Electronic Frontier Foundation (EFF) announced that it had spent \$250 000 and constructed the fabled DES Cracker. The EFF machine tried 90 billion 56-bit DES keys every second, and cracked its first challenge message after only 56 hours of work.[63] The project is widely credited with putting the last nail into the coffin of weak symmetric encryption schemes.

When cracking symmetric encryption systems with a brute force attack, each additional bit of key length doubles the difficulty of the attack, because each additional bit doubles the number of keys that need to be searched. With 4 bits, there are 16 keys to search; with 8 bits there are 256, and so on. For a while, the US Government's proposed replacement for DES was the so-called "Clipper" chip, which supported an 80-bit key, making it 2^{24} or roughly 16 million times harder to crack – except that each Clipper chip was gimmicked so that the government *didn't need* to perform such an attack to decrypt a message encrypted with Clipper. That's because the Clipper implemented the government's "Escrowed Encryption Standard" (FIPS-185), which meant that every Clipper had its own secret decryption key that could be used to decrypt any message that

[61] Blanchette, *Burdens of Proof: Cryptographic Culture and Evidence Law in The Age of Electronic Documents* (2012).

[62] Diffie and Hellman, "Special Feature Exhaustive Cryptanalysis of The NBS Data Encryption Standard" (1977).

[63] Electronic Frontier Foundation, *Cracking DES* (1998).

the chip encrypted, and the government kept copies of these keys so that messages could be decrypted for legal process or in the event of a national security emergency. To prevent companies from creating software-only Clipper chips that didn't implement key escrow, the government declared that the encryption algorithm used by the chip had to be kept secret in the interest of national security.

As might be expected, Clipper chip was a commercial failure.

When the National Institute of Standards and Technology initiated its efforts to create a replacement algorithm for the Data Encryption Standard in the late 1990s, it committed itself to an open, unclassified project. NIST invited submissions for the new algorithm, held two academic conferences to discuss the submissions, and ultimately adopted an algorithm invented outside the United States by a pair of Belgian cryptographers, Vincent Rijmen and Joan Daemen. The algorithm, originally named Rijndael, is faster than DES and supports key sizes of 128, 192, and 256 bits. It was adopted by the US government as the Advanced Encryption Standard in 2001.

For many years after it was adopted, AES-128 was the preferred use of AES because it ran significantly faster than the more secure AES-256. That extra security is in fact the reason that AES-256 was slower. The design of AES is based on a function that is repeated a certain number of "rounds" for every block of data that the algorithm encrypts. AES-128 has 10 rounds, AES-256 has 14.[64] Today those differences are less significant than they were in 2001, as computers are faster and many microprocessors now contain hardware support to make AES run faster still. In most modern computers, encrypting with AES-128 is essentially free. For example, the Apple iPhone contains a chip that automatically encrypts data with AES when it is written from the CPU out to the phone's flash memory, and automatically decrypts the data when it is read back in.

However, absent quantum computing, the differences between AES-128 and AES-256 are inconsequential for most users. That's because 2^{128} is a really big number: in a world without quantum computers, a message encrypted with a 128-bit key will *never* be cracked using a brute-force, key search attack.

[64]AES-256 may in fact be more than 2^{128} times stronger than AES-128, as AES-256 has 14 internal "rounds" of computation, while AES-128 has only 10. If there is an algorithmic weakness in the underlying AES algorithm, that weakness should be easier to exploit if there are fewer rounds.

5.3.3 Cracking AES-128 with Grover's Algorithm

Grover's algorithm makes it possible to use a quantum computer to guess the right key with fewer steps than it would take to try all possible keys. To understand why AES-128 is vulnerable to a quantum computer running Grover's algorithm but AES-256 is not, it is necessary to understand more about how Grover's algorithm works in practice.

Although Grover's discovery is frequently described as an algorithm for speeding up "database search," this gives a misleading impression as to what the algorithm actually does. The "database" is not the kind of database that most people are familiar with: it doesn't actually store data. Instead, the database is a database of guesses and whether or not each guess is correct.

In Table 5.2, we have recast the problem of cracking an encrypted message into a database search problem that could then be searched using Grover's algorithm. To perform a brute force search for the correct key, just start at the top and examine each row until the database value is a `1`. In this example, a little more than half of the rows need to be examined. If you have a computer that can examine 90 billion rows a second – on par with the speed of the EFF DES Cracker – then you will find the answer in roughly five days.

A key search attack is possible because 2^{56} is not such a fantastically large number after all – that's the point that Hellman made in his letter to the NBS when he urged that 56 bits was just too small. If NBS had gone with a 64-bit key length, then an average search time of 20 hours would become 1280 days. That's better, but it's still not good enough for government work, which requires that national security secrets be declassified after 50 years[65] unless they contain names of confidential intelligence sources, contain information on weapons of mass destruction technology, would "reveal information that would impair US cryptologic systems or activities," or meet a few other specified requirements.[66] Clearly for US government use, an encryption algorithm that might be crackable at any point in the foreseeable future due to the likely advance of computer technology is not acceptable.

[65] For an explanation of the origin of this phrase and its corruption, see Lerman, *Good Enough for Government Work: The Public Reputation Crisis in America (And What We Can Do to Fix It)* (2019).

[66] Obama, "Executive Order 13526: Classified National Security Information" (2009).

https://doi.org/10.1017/9781108883719.009 Published online by Cambridge University Press

Table 5.2. To use Grover's algorithm to crack an encryption key, Table 5.1 is recast as a database search problem, where one row has the value of `1` stored and all of the other rows have the value of `0`. In this example the keys are 56-bit DES keys. If this table instead used 128-bit AES keys, the last row would be number 340 282 366 920 938 463 463 374 607 431 768 211 455 ($2^{128} - 1$).

Row	Row number in binary	Database Value
0	`0000` ... `0000`	0
1	`0000` ... `0001`	0
2	`0000` ... `0010`	0
3	`0000` ... `0011`	0
4	`0000` ... `0100`	0
	...	
38 326 038 678 974 151	`1000` ... `0111`	1
	...	
72 057 594 037 927 935	`1111` ... `1111`	0

As we have stated above, AES-128 doesn't have this problem, because 2^{128} is fantastically larger than 2^{56} – unless the attacker has a functioning quantum computer that's large enough to compute AES-128.

Cracking AES-128 with Grover's algorithm is surprisingly straightforward. First, it is necessary to construct an implementation of AES-128 on a quantum computer with at least 129 qubits, such that when the first 128 qubits have the correct decryption key, the 129th qubit has the value of `1`. Additional qubits are required to implement various details of Grover's algorithm and to properly implement AES-128 (we won't go into the details here).

AES-128 has 10 rounds, which means there is an inner algorithm that is repeated in a loop 10 times. Quantum computers don't have this kind of loop, so it is necessary to *unroll* the rounds, meaning that the circuits for the inner AES function need to be repeated 10 times. Additional circuitry is required to detect when the correct decryption key has been found.

It's relatively straightforward to imagine how the AES-128 circuit might be run on the kinds of superconducting quantum computers being developed by IBM and Google. On these computers, the qubits are "artificial atoms" made up of superconducting circuits operating at close to absolute zero, while the quantum gates and circuits are implemented by precisely timed and aimed pulses of radio waves. The speed of the quantum computation is determined by how quickly the quantum computer can cycle through a specific combination of radio waves that it sends into the artificial atoms. When the computation is finished, the qubits are measured with other radio wave pulses.

To run Grover's algorithm, each of the unknown bits (here, the 128-bit AES key) starts off as a superposition of `0` and `1`. The algorithm is then cycled $\sqrt{2^N}$ times, where N is the number of unknown bits. At the end of these cycles, the unknown bits are measured, and they are overwhelmingly likely to have the answer to the problem. Superposition must be maintained for the entire time: if it is lost, the computation is ruined.

It turns out that $\sqrt{2^N} = 2^{N \div 2}$. So when cracking AES-128, only 2^{64} iterations are required, rather than 2^{128}. Because 2^{64} is not a fantastically large number, the mere existence of Grover's algorithm and the possible future existence of large-enough quantum computers was enough for cryptography experts to recommend discontinuing the use of AES-128 when these results became generally understood. However, AES-256 is still fine, because even with Grover's algorithm reducing the security parameter from 2^{256} to 2^{128}, that's okay because 2^{128} is a fantastically large number. All of this was clear from the theory, without the need to create an actual working quantum implementation of AES to actually try out Grover's algorithm.

In 2016, quantum computing theoreticians in Germany and the US carried out the hard work of actually building "working" quantum circuits of AES-128, AES-192, and AES-256 – at least, in theory. They found that implementing cracking a single AES-128 encryption key with Grover's algorithm requires *at most* 2953 qubits and on order of 2^{86} gates. For AES-256 the estimate was 6681 qubits and 2^{151} gates.

"One of our main findings is that the number of logical qubits required to implement a Grover attack on AES is relatively low, namely between around 3000 and 7000 logical qubits. However, due to the large circuit depth of unrolling the entire Grover iteration, it seems challenging to implement this algorithm on an actual physical quantum computer, even if the gates are not error corrected," the authors write. The authors conclude "It seems prudent to move away from 128-bit keys when expecting the availability of at least a moderate size quantum computer."

The word "prudent" requires additional explanation, as even a work factor of 2^{86} is likely to be beyond the limits of any human technology for the foreseeable future. For example, a quantum computer that could sequence quantum gates every *femtosecond* (that is, 10^{15} times per second) would still require 2451 *years* to crack a single AES-128 key using the implementation described in the 2016 publication. And a femtosecond clock would be a big deal – it would be 250 times faster than the clock speed of today's 4 GHz microprocessors. Chemical reactions take place at the femtosecond scale; the time is so short that light only travels 300 nanometers.

Of course, given a cluster of 1024 quantum computers, each running with a femtosecond clock, each one attempting to crack AES-128 with a different 10-bit prefix, an AES-128 message could be cracked in less than a year. So if mass-produced femtosecond quantum computers with a thousand qubits that can compute a single calculation error-free for a year is a risk that you consider relevant, then you should not be using AES-128 to protect your data!

But remember – the 2016 article describes an *upper bound*: it might be possible to create AES-cracking quantum computing circuits that require fewer gates. In fact, two 2019 efforts[67] lowered the upper bound on the work factor to crack AES-128 to 2^{81} and 2^{79} respectively by developing better quantum gate implementations for the AES oracle (the quantum code that determines when the correct key has been guessed). It has long been the case that hand-tuning algorithms to squeeze out the last few cycles of performance has been something of a parlor game among computer scientists.[68] So instead of looking for upper bounds, it might be more productive to look for theoretical lower bounds.

The absolute lowest bound for a circuit that could crack AES using Grover's algorithm would be a circuit that executed a single gate over a large number of qubits: such a perfect implementation would require a minimum of 2^{64} cycles to crack AES-128, and 2^{128} to crack AES-256. We (the authors) do not think that such a circuit is possible. However, this "perfect" quantum AES implementation would be able to crack AES-128 in 5.12 hours using our fictional

[67] Jaques et al., "Implementing Grover Oracles for Quantum Key Search on AES and LowMC" (2019); Langenberg, Pham, and Steinwandt, "Reducing The Cost of Implementing AES As a Quantum Circuit" (2019).

[68] For example, in 2010, a group of researchers at the Naval Postgraduate School that included one of us published a high-speed implementation of AES for the Sony PlayStation. See Dinolt et al., *Parallelizing SHA-256, SHA-1 MD5 and AES on The CellBroadbandEngine* (2010).

quantum computer with the femtosecond clock; even this perfect implementation would require 10 782 897 *billion years* to crack a single AES-256 encryption.

To push the absurd hypothetical even more, there's no fundamental reason why we should limit our fictional quantum computer to a femtosecond clock. What if we had a smaller, more compact quantum computer that could fit in a nanosphere – perhaps two thousand packed atoms in a blob just 10 nm across. The maximum cycle time of this computer would be roughly $\frac{1}{30}$ of a femtosecond, the time it takes light to move from one side of the sphere to the other. With this computer and the (fictional) perfect Grover AES circuit, you could crack AES-128 in just 10 minutes, but it would still take 360 billion years to crack AES-256. Here parallelism finally begins to help: with a billion of these computers, you could crack an AES-256 encryption in at most 3.6 years. Of course, if you have the kind of technology that can make and control a billion of these computers, there are probably far more productive things you would be able to do than to go after AES-256 keys from the 2020s.

So to summarize, although it's conceivable that AES-128 might one day fall to a futuristic quantum computer, there is no conceivable technology that could crack an AES-256 encryption using exhaustive key search. What's more, AES-128 is sufficiently close to the boundary of what a quantum computer might be able to crack over the next 20 or 30 years that it is indeed "prudent" to stop using AES-128 in favor of AES-256. In part, this is because the cost increase of using AES-256 instead of AES-128 is quite minor: on a 2018 Apple "Mac Mini" computer, encrypting a 7 GiB file took 7.1 s with AES-128 running in "cipher block chaining" mode; with AES-256 it took 9.1 s. For the vast majority of applications this 28 percent increase in encryption time is simply not significant.

But remember – all of the analysis above assumes that AES-256 is a perfect symmetric encryption algorithm. There might be underlying vulnerabilities that make it possible to crack AES-256 encrypted messages with significantly less work than a full brute-force attack. To date, no such attacks have been published that offer speedup greater than Grover's algorithm,[69] but there's always tomorrow. Certainly, if computer scientists discover that P=NP, then

[69] There is one classical attack against AES-256 that lowers the work factor from 2^{256} to $2^{254.4}$; Grover's quantum algorithm lowers the work factor to 2^{128}.

attacking AES-256 could become the stuff of high school science fairs soon thereafter.

5.3.4 Grover's Algorithm Today

The impact of the square-root speedup offered by Grover's algorithm has been systematically misrepresented in the popular press over the past two decades. Recall that although Grover's algorithm speeds up *search*, it is not the kind of search that we do with Google looking for a web page or using an accounting system when we are looking for a specific transaction. Those kinds of searches involve the computer scanning through a database and looking for a matching record, as we discuss in Section 3.5.1 (p. 102). Although Grover's algorithm *could* be applied to such a search, it would require storing the entire database in some kind of quantum storage – a system that has only been well-specified in works of science fiction – playing the entire database through the quantum circuit, a process that would eliminate any speedup provided by Grover's algorithm in the first place.

To date, scientists have accomplished only limited demonstrations of Grover's algorithm. Beit, a quantum software company with a lab in Kraków, Poland, released two unpublished papers in 2020 reporting state-of-the-science accomplishments in applications of Grover's search. A September 2020 paper from the group demonstrated a Grover implementation in IBM hardware, where the team performed an unstructured search among a list with just 16 elements. The goal of such a search is to identify one element in the list successfully, but the system was able to do so on average only 18–24 percent of the time.[70] A subsequent study employed Honeywell's 6-qubit Model H0 ion trap, which is commercially available. In June 2020, Honeywell hailed the device as the world's most powerful quantum computer, claiming that it has a quantum volume of 64.[71] The Beit team, using Honeywell's API, tested Grover's search in 4, 5, and 6-qubit implementations. Respectively, the team could select the right result 66

[70]Gwinner et al., "Benchmarking 16-Element Quantum Search Algorithms on IBM Quantum Processors" (2020).

[71]*Quantum volume* (QV) is a metric that IBM created that measures the square of the number of quantum circuits that a quantum computer can implement. According to IBM, QV combines "many aspects of device performance," including "gate errors, measurement errors, the quality of the circuit compiler, and spectator errors" (Jurcevic et al., "Demonstration of Quantum Volume 64 on a Superconducting Quantum Computing System" (2020)).

percent of the time with a 4-qubit circuit (selecting from a list with 16 elements), 25 percent of the time with a 5-qubit circuit (using a list with 32 elements), and just 6 percent of the time using all 6 qubits in a circuit (using a list with 64 elements).[72]

Some articles in the popular press incorrectly describe quantum computers as machines that use superposition to simultaneously consider all possible answers and select the one that is correct. Such machines do exist in the computer science literature, but they are called "nondeterministic Turing machines" (see Section 3.5.3, p. 107). And while such machines do exist *in theory*, they do not exist *in practice*: the conservation of mass and energy makes them impossible to build in this universe.[73]

Quantum computers use superposition to simultaneously consider a multitude of solutions, which *does* allow them to compute the answers to *some kinds of problems* faster than computers that are not based on superposition and entanglement. But they don't do this by coming up with the single, best answer to those problems. Instead, modern quantum computers are like a carefully designed collection of dual-split experiments (see Section B.1.3, p. 490): they have a distribution of possible answers – like an interference pattern on the screen – with the more probable answers coming up more often and the less probable answers coming up less often. The trick to programming the machines is to set up the computer so that the answers that are correct are significantly more probable and that incorrect answers are significantly less probable. This is done, ultimately, with constructive and destructive interference at the quantum level, in the machine's Schrödinger wave equation.

Another source of confusion might be that quantum computers can solve particular kinds of problems in polynomial time that are thought to be harder than the complexity class known as *P* (polynomial). The key example here is factoring. Because *NP* (nondetermin-

[72]Hlembotskyi et al., "Efficient Unstructured Search Implementation on Current Ion-Trap Quantum Processors" (2020).

[73]Such machines are not even possible if you subscribe to the many-worlds interpretation of quantum physics: it may be that a computer facing an NP-hard problem with a quantum-mechanical random number generator splits the universe 2^N times and that in one of those universes a computer immediately finds the correct answer. The problem is that in all of the *other* $2^N - 1$ universes the computers all discover that their answer is incorrect, and there is no inter-universe network to allow the computer that guessed correctly to inform its clones of the correct choice.

The Limits of Quantum Computation

"The manipulation and transmission of information is today carried out by physical machines (computers, routers, scanners, etc.), in which the embodiment and transformations of this information can be described using the language of classical mechanics," wrote David P. DiVincenzo, then a theoretical physicist at the IBM T.J. Watson Research Center, in 2000.[a] "But the final physical theory of the world is not Newtonian mechanics, and there is no reason to suppose that machines following the laws of quantum mechanics should have the same computational power as classical machines; indeed, since Newtonian mechanics emerges as a special limit of quantum mechanics, quantum machines can only have greater computational power than classical ones."

"So, how much is gained by computing with quantum physics over computing with classical physics? We do not seem to be near to a final answer to this question, which is natural since even the ultimate computing power of classical machines remains unknown."

For example, DiVincenzo wrote, we know that quantum computing does not speed up some problems at all, while some are sped up "moderately" (in the example of Grover's algorithm), and others are "apparently sped up exponentially" (Shor's algorithm).

DiVincenzo notes that, on purely theoretical grounds, quantum computing also could result in a "quadratic reduction" in the amount of data required to be transmitted across a link between two parties to complete certain mathematical protocols. But such a reduction requires the data is transmitted as quantum states – over a quantum network – rather than as classical states. "The list of these tasks that have been considered in the light of quantum capabilities, and for which some advantage has been found in using quantum tools, is fairly long and diverse: it includes secret key distribution, multiparty function evaluation as in appointment scheduling, secret sharing, and game playing."

[a]DiVincenzo, "The Physical Implementation of Quantum Computation" (2000).

istic polynomial) is the class that most people think is harder than *P*, and *NP* is the class solved by nondeterministic Turing machines, some people jump to the conclusion that quantum computers can solve NP-hard problems.

There are several problems with this line of thinking. First, just because mathematicians haven't found an algorithm that can factor in polynomial time doesn't mean that such an algorithm doesn't exist: it wasn't until 2002 that mathematicians had an algorithm for primality testing that ran in polynomial time. So factoring might be in *P*, and we just haven't found the algorithm yet. Or, more likely, factoring might be harder than *P* and still not in *NP*. Or, it might be that $P = NP$, which would mean factoring in both *P and NP*, because they would be the same. As we discussed in Section 3.5.6 (p. 116), computer scientists use the complexity class called *BQP* to describe the class of decision problems solvable by a quantum computer in polynomial time. Just as we don't know if *P* is equal to *NP*, we don't know if *BQP* is the same as or different from *P* or *NP*. This can be written as:

$$P \overset{?}{=} BQP \overset{?}{=} NP \tag{7}$$

For further discussion of this topic, we recommend Aaronson's article "The Limits of Quantum."[74]

Similar to the situation with the NP-hard and NP-complete problems, there is no proof that quantum computers would *definitely* be faster at solving these problems than classical computers. Such a mathematical proof would put theoreticians well on their way to solving the whole $P \neq NP$ conjecture, so it is either right around the corner or it is a long way off. It is simply the case that scientists have discovered efficient algorithms for solving these problems on quantum computers, and no such corresponding algorithms have been discovered for classical computers.

5.4 Conclusion

Whereas the electromechanical and early electronic computers of the 1940s were transformative, allowing the United Kingdom to crack the German Enigma code and the United States to create the hydrogen bomb, the main use of quantum computers today in 2021 is by researchers who are developing better quantum computers, better quantum algorithms, and students who are learning about quantum

[74] Aaronson, "The Limits of Quantum" (2008).

The Quantum Algorithm Zoo

Stephen Jordan, a physicist at Microsoft Research who works on quantum computing, maintains a database of quantum algorithms – the Quantum Algorithm Zoo. Jordan categorizes today's quantum algorithms into four types:[a]

1. **Algebraic and number theoretic algorithms,** which use properties of quantum computers to solve number theory problems. An example is Shor's algorithm for factoring.

2. **Oracular algorithms,** which depend upon an *oracle* that can provide an answer to a question. An example is Grover's algorithm for speeding up search.

3. **Approximation and simulation algorithms**, such as would be used to simulate the process of nitrogen fixation as discussed in Section 5.1.1, "Nitrogen Fixation, without Simulation" (p. 181).

4. **Optimization, numerics, and machine learning algorithms,** which could be used for improving systems based on so-called neural networks, including speech, vision, and machine translation.

[a]You can find the list of algorithms at Jordan's website, http://quantumalgorithmzoo.org/, which is based on his May 2008 MIT PhD thesis (S. P. Jordan, *Quantum Computation beyond The Circuit Model* (2008)).

computers. The main output of today's quantum computers is not military intelligence and might, but papers published in prestigious journals.

Nevertheless, it would be a mistake to dismiss this research as quantum navel gazing. Unlike the limits that have impacted Silicon Valley's efforts to make increasingly faster electronic computers, we may be a far way off from hitting any fundamental limit or law of nature that will prevent researchers from making larger and faster quantum computers – provided that governments and industry continue to invest the necessary capital.[75]

[75]If it turns out that we can never make machines that work at large scale, then it

This may be why some governments continue to pour money into quantum computing. Although promoters speak about the benefits in terms of simulation and optimization, they are surely also driven by that darker goal of being able to crack today's encryption schemes used to secure the vast majority of information transmitted over the Internet and through the air. And because information transmitted in secret today might be useful if decrypted many decades from today, *the mere possibility* that powerful, reliable quantum computers might exist several decades in the future is a powerful influencer today.

Today's quantum computers are not nearly powerful enough to break the world's cryptography algorithms (or do anything else), but each year they improve, as quantum computing engineers become more adept at precisely controlling fundamental quantum processes. For this reason alone, our society should seek to rapidly transition from today's quantum-vulnerable encryption algorithms like RSA and AES-128 to the next generation of post-quantum encryption algorithms. If our understanding of quantum mechanics is correct, it is only a matter of time until the machines are sufficiently powerful.

We are still at the beginning of quantum computing, and very basic questions of technology and architecture still have to be worked out. The next chapter canvasses the research groups that are wrestling with different physical substrates for representing quantum information, different ways of organizing those physics packages into computing platforms, and different languages that programmers can use to express quantum algorithms. Much research in quantum computing is so preliminary and theoretical that an idea can have a major impact years before it's been reduced to practice and demonstrated. What's concerning is that the field hasn't had a mind-blowing discovery since the breakthroughs of Shor and Grover in the mid-1990s.

is likely that there is something fundamentally wrong about our understanding of quantum physics. Many advocates say that this alone is worth the study of quantum computers. And while some funding agencies might disagree, the amount of money spent on quantum computing to date appears to be significantly less than the \$10–\$20 billion that the US high energy physics community proposed spending on the Superconducting Super Collider in the 1990s, or even the \$4.75 billion that Europe spent on the Large Hadron Collider between 1994 and 2014.