Candidate Solutions for Quantum-Proof Cryptographic Protocols

Charles Bolton[*], Patrick Rademacher[†], Jesse Rapoport[‡]

May 25 2020

1 Abstract

We discuss three interesting domains in post-quantum cryptography including lattices, code theory and supersingular isogenies. These topics each are deep and active areas of research, so we provide a glimpse of some interesting aspects of each.

2 Introduction

The advent of quantum computers threatens the security of classical cryptography. Scientists have been developing new protocols for quantum key distribution (QKD), allowing for the safe distribution of public and private keys in the era of post-quantum cryptography (PQC). Some have already been tested, in Tokyo, a QKD network between Koganei, Otemachi, Hakusan, and Hongo is already being used in research. But QKD is not the only approach to maintaining security. After all, when quantum computers hit the shelves, the entire world most certainly will not throw out their Macbooks overnight. Furthermore, these new machines are unlikely to entirely (at least immediately) replace classical computers, even if they become widely available/affordable. And so other ideas are called for.

How can we, the humble citizenry, defend ourselves from the NSA when they fire up their quantum computers (if they haven't already)? A separate approach to PQC deals with analysis of and solutions to problems created by this new threat—solutions which can be implemented on a classical computer (This is

[*]Maseeh College of Engineering and Computer Science, Portland State University **boltch@pdx.edu**

[†]Maseeh College of Engineering and Computer Science, Portland State University **prad2@pdx.edu**

[‡]Maseeh College of Engineering and Computer Science, Portland State University **jhr@pdx.edu**

known as “transitional/hybrid” security). These are new algorithms and/or updates to existing algorithms which rely on, unsurprisingly, the hardness of certain mathematical problems. There exist a handful of domains where most of the ongoing research is taking place; these are, in brief: lattices, code theory, supersingular isogenies, cryptographic hashes, and multivariate solutions [3][15]. Here we present an overview and brief discussion of the first three only.

3 Lattice-Based Cryptography

While the security of the major public key schemes (Diffie-Hellman, ECC, RSA) are based on the hardness of algebraic problems in number and group theory, lattices as infinite linear-algebraic and geometric structures generate the hard-problems in lattice-based cryptosystems. A lattice $\mathcal{L} \subset R^n$ is an n dimensional mesh/grid of repeated, equally spaced (Euclidean), periodic and discrete points (generated by n basis vectors). For visualization, the image below shows basis vectors in a two dimensional lattice. Typically, we will have $n > 1000$.

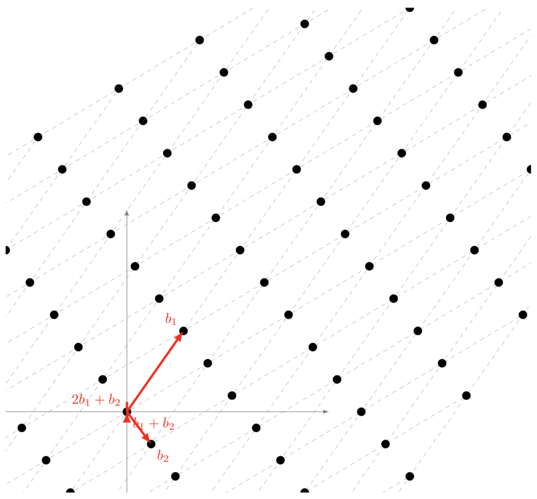

Figure 1: Depiction of two basis vectors in low-dimensional lattice [24]

3.1 Hard Lattice Problems

Unlike the other classical schemes which rely on integer factorization and discrete logarithms, Shor’s algorithm does not affect the strength of lattices [20]. Among the core hard problems that lattice systems rely on are the closest- and shortest-vector-problems (CVP and SVP), where the latter is the task of finding the set of basis vectors (among infinite possibilities) which are the shortest (excluding the zero vector). For sufficiently large n, the only known solutions to this problem have exponential running times, and therefore often approximation algorithms are substituted for exact solutions [19].

3.1.1 Search Learning with Errors

A more general hard lattice problem is the Learning with Errors (LWE) problem [18]. For LWE, the goal is to find a 'secret' vector in the lattice $s \in Z_q^n$. An attacker is allowed to choose as many random n-dimensional a_i vectors ("samples") as they want, and in return they get back the result of the inner product of the sample and the secret

$b_i = \langle s, a_i \rangle + e_i \bmod q$

Where the e_i term is a "noise" term sampled from a (typically) Gaussian error distribution with standard deviation $\approx \sqrt{n}$. Since each b_i is a dot product and each a_i a vector, this can be expressed as

$b^T = s^T \mathbf{A} + e^T$

Where $\mathbf{A} = \begin{pmatrix} | & | & | & \cdots \\ a_1 & a_2 & a_3 & \cdots \\ | & | & | & \cdots \end{pmatrix}$ and and $b = \begin{pmatrix} | \\ b \\ | \end{pmatrix}$ is a vector of the b_i terms

3.1.2 Other Hard Problems

This specific version of the problem is called search-LWE. It is thought to be quantum-hard and is known to be at least as hard as the worst-case approximation problems for lattices. Such problems include C/SVP-variants BDD[1], uSVP[2], SIVP[3] and GapSVP[4][4][24]. In addition, if any solution for LWE is found then it will also be a quantum solution.

3.2 Quantum Reductions

While hard lattice problems are a well-studied area of research, what makes cryptosystems which use lattice problems strong is the fact that researchers have been able to prove many classical and quantum reductions. So the strength of lattice-based schemes isn't guaranteed, but merely relies on reductions to problems with well-studied bounds which no known quantum algorithm can solve any faster, with "known" being the operative word here. Indeed, many reductions have been proven between lattices problems like SIVP and LWE. One particularly interesting bound which illustrates the many strange and diverse connections quantum researchers have found shows that LWE and other lattice problems are not harder to solve than what is known as the Dihedral Coset Problem (DCP).

[1] The problem of finding the closest lattice vector guaranteed within some specified distance
[2] The problem of finding a unique shortest vector
[3] The problem of finding many linearly independent short lattice vectors
[4] The problem of whether there is a specifically "short" lattice vector

3.2.1 Diheral Group

A dihedral group is often described using polygons as examples. Given some polygon with N sides, the dihedral group is given by how many times that shape can be reflected and rotated while keeping the shape in place. Often a pentagon is used for demonstration. For a pentagon, $N = 5$, and this shape can be rotated 4 times before the fifth rotation returns the orientation back to the original state (this is the identity operation, akin to addition modulo 5) [11]. The four rotations and the original orientation form N of the group's elements. The other 5 (N) elements are gotten by reflecting this shape across its five axes (or, equivalently, reflecting and rotating the reflection 4 times). In total this group has $2N$ elements, 2 for each face, which is the etymological importance of "dihedral." Dihedral groups defy the well-known hidden subgroup problem, which is the reason that many classical cryptoschemes will be defeated by quantum computers[5].

3.3 Fourier Sampling

A process known as "Fourier sampling" can, in general determine a hidden subgroup H of a group G. This process of determining hidden subgroups is used in Simon's Problem and Shor's Algorithm, famously giving them remarkable speed-ups compared with their classical counterparts. Other applications include solving discrete logarithms and order-finding [16].

3.4 Hidden Subgroup Problem

The hidden subgroup problem considers a subgroup H and a function f. f is an invariant function with several rules. First f is constant on the elements of H. Secondly, f is constant on the cosets of H, which are produced by performing a translation of H. For example, in the case of a shape, suppose that the elements in H are pentagons. Each can be reflected, producing a coset of H, and if f is constant on these reflections (across the same axis), and all other possible reflections (across the other axes), then we say f is a function which hides H. There is one more important rule that says that f must also produce distinct results from distinct cosets, so in this example, each reflection of H's elements will be constant under f and will each have a distinct output.

While classical solutions to finding H given f can do no better than brute force search, Fourier sampling can determine H much more quickly using three quantum steps [14]. The first step prepares a superposition of all of the elements in group G along with an ancilla qubit $|0\rangle$

1. $\frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle \otimes |0\rangle$

The second step entangles the registers by passing the second register through f, giving us the following state

2. $\frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle \otimes |f(g)\rangle$

Finally, the third step disentangles the registers by taking a measurement of the second register, thus making the ancilla register garbage. What is left over in the first register is a superposition of one of the cosets of H

3. $\frac{1}{\sqrt{|H|}} \sum_{h \in H} |ch\rangle$

Although this method works for revealing certain hidden subgroups, there is no known way of performing Fourier sampling to solve the dihedral hidden subgroup problem or for other non-abelian groups [25]. Instead, there is a related problem mentioned above, the dihedral coset problem.

3.5 Dihedral Coset Problem

Given a quantum black box, the DCP is formulated as finding the value of d, where the input to the black box function is some random x and the output is $(|x\rangle|0\rangle + |x + d\rangle|1\rangle)$. This quantum state is actually an encoding of a dihedral group, where the first qubit encodes the number of reflections and the second the number of rotations. A solution to this problem also gives a solution to the dihedral hidden subgroup problem. Another reduction to DCP from a generalized problem called the 2-point problem can be shown. Like LWE, the worst-case lattice problems such as SVP can be reduced to the 2-point problem. This gives us a chain of reductions

lattice problems $\leq$ 2-point-problem $\leq$ DCP $\leq$ DHSP

3.6 Research Continues

The reason this is important is because the DHSP has no known polynomial time or fast quantum algorithm. Solutions to this problem using a quantum computer are no faster than they would otherwise be using a classical computer [14]. The chain of reductions shows that these lattice problems can be used safely for constructing strong cryptosytems because they link the problems used in key exchange systems based on lattice problems to hard problems which don't have any known fast quantum solutions. However, it should be noted that ongoing research in quantum algorithms may discover solutions to either the dihedral coset problem or the dihedral hidden subgroup problem which could jeopardize the strength of these cryptopsystems. As it is often remarked, a chain is only as strong as its weakest link, and lattices are linked, via a complex web of reduction, to many problems whose quantum resilience is suspended in belief.

4 Code Theory

Code theory is another exciting avenue for post-quantum cryptography. Code theory may sound like it was designed for sending secure messages, but in reality it was not; it was originally designed for ensuring that errors occurring in transmitted data could be reliably found and reverted [2]. Code theory originally was simply about this: how to handle errors when transmitting data. This is still an application of code theory in use today, e.g. data from the internet using code theory via CRC checksums [10]. In fact, it's also used in this way for quantum computation [22]. However, in addition to it being used for naturally-occurring-errors in quantum computation, it's also used for ensuring security between intended parties.

So, what makes code theory special? That is, it would appear that, yes, it is difficult to correct for some implicit errors, but isn't it also difficult to factor a very large number, like in RSA cryptography, so why would we use this indirect method for security? Yes, on classical computers, this is a valid argument. However, on quantum computers, these are two very different types of problems. Specifically, while code-theory has been shown to be NP-Complete [3], RSA encryption actually is not NP-complete [1]. So, on quantum computers, the former would appear to be intractable, while the latter is actually rather easy. This is significant.

4.1 Introducing the McEliece Cryptosystem

A very old, popular and still-promising implementation of code theory is the McEliece Cryptosystem [17]. This system can be viewed as simply a combination of 5 pieces of public information, and 3 pieces of private information [13]. The public information is as follows:

1. The scrambled $k \times n$ generator matrix G'
2. The k-length plain text
3. The n-length vector
4. The error correction capacity t
5. The exponent m

And the private information:

1. The random invertible $k \times k$ matrix S_{Mc}
2. The $n \times n$ permutation matrix P_{Mc}
3. An efficient decoding algorithm for G

Let's break down what all of this means.

4.2 Understanding the McEliece Cryptosystem

To start out, let's define a *generator matrix.*

4.2.1 Generator Matrices

Essentially, a generator matrix is a matrix that mathematically converts a plain-text message into a codeword, which is an alternative representation of the message with added redundancy [13].

For example, say Alice wants to send a 4-bit message to Bob, $m = \begin{pmatrix} A & B & C & D \end{pmatrix}$. However, she is worried that one of her bits may be accidentally flipped during transmission. As a solution, she can convert her message m using a generator matrix G into a codeword c that will include redundancy that can be used to identify and fix that error. (In general, this can correct up to t errors, which will be defined later).

Note: a $k \times n$ generator matrix will convert a k-length message m into an n-length codeword c.

In this case, let the generator matrix $G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$

What will happen when Alice's message is multiplied by her generator matrix? Specifically, $m \cdot G = c$, so what will her codeword c be?

This can be determined by looking at the generator matrix. Essentially, each column here corresponds to a resulting element in the codeword vector c. The first column in G corresponds to the first element in c, the second column to the second element, and so on. In each column there are four rows, one for each element in the plain-text message m. To determine what an element in c will be, simply look at the values in that same column, and sum those rows' values from the message m. For example, the 5th column is $\begin{pmatrix} 0 & 0 & 1 & 1 \end{pmatrix}^T$. What this indicates is that the 3rd and 4th elements of m will be summed to compute the 5th element in c. So, then, the 5th element in c will be $C + D$. Doing this calculation for each of the 8 columns yields the following codeword: $c = \begin{pmatrix} A & B & C & D & C+D & A+B & A+C & B+D \end{pmatrix}$

And, indeed, this is the same result as multiplying $m \cdot G$.

Okay, so now we know how to use a generator matrix to convert a message into a codeword. However, why would Alice actually want to do such a thing?

As mentioned earlier, the reason for this is to provide *redundancy.* Although redundancy is normally considered a bad thing, in the case of error-correcting codes, it is quite good. It provides the same function that it provides in natural languages, which is to ensure that a missed or misinterpreted small piece of information will not corrupt an entire message. In this case, it actually allows the entire message to stay entirely intact, even with a flipped bit. Without redundancy, this would not be possible; Bob would have no way of knowing which of the sent bits were original and which were flipped. However, with redundancy, he can rest assured that he is receiving Alice's intended message correctly [13].

So, now Bob has received a codeword from Alice that will allegedly defend against potential data loss from errors. However, how does Bob actually use that codeword to retrieve Alice's message?

In this case, it may look like the answer is simple, just look at the first four elements in the codeword. Although it is not usually that simple in a real scenario, the bigger problem here is that there could be an error in that data. So how do we use the redundant bits (the last 4) to ensure that there are indeed no errors?

For this, we use a *parity check matrix.*

4.2.2 Parity Check Matrices

A parity check matrix is essentially the opposite of a generator matrix. Here, instead of converting a message into a codeword, a parity check matrix will convert a codeword into a message, with a high degree of certainty that it is correct, regardless of the presence of an error.

In this case, the parity check matrix will be $H = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$

Does this look familiar? In fact, it is very similar to the generator matrix. The right of of H is the identity matrix, just like the left half of G. Furthermore, the left half of H is actually simply the transpose of the right half of G. This works out pretty nicely.

Does this property always hold? No, it does not. It's only even possible to hold if $n = 2k$, i.e. that the codeword length is exactly twice that of the message length. Even if that's the case, it does not always hold however. When it does hold, we call it *systematic form.* This is a very convenient set-up, so it is a common format of a parity check matrix [13].

So, how does one actually use a parity check matrix? Just multiply the codeword by the parity check matrix, and then get the message out, right? Not quite.

Although, we do indeed compute that operation firstly. In this case, although we use the transpose of the codeword:

$$H \cdot c^T = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \\ C+D \\ A+B \\ A+C \\ B+D \end{pmatrix} = \begin{pmatrix} C+D+(C+D) \\ A+B+(A+B) \\ A+C+(A+C) \\ B+D+(B+D) \end{pmatrix}$$

So, for our result, it looks a little meaningless. For example, for the first element, the result is essentially $C + D$ twice. What information could one gather from that?

Well, it actually is pretty easy to deduce what this computes to. Because addition here is an exclusive-or operation, $X + X = 0$ for all X. Therefore, in this case, $C + D + (C + D) = 0$ as well. Then, using that logic for all 4 elements, that yields a final result of $\begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}^T$.

4.2.3 Syndromes

So now we have this vector of all 0's. That should be the result every time, regardless of the values of A, B, C, and D, right? What use is that?

That actually is true. If there are no errors, then for all values of A, B, C, and D, this resulting vector (called a *syndrome*) will yield all 0's.

Where this becomes useful is if there is an error. For example, let's say the value of the 4th bit in the codeword was accidentally flipped during transmission. Although now that element in the codeword vector is incorrect, there is now a way to discover this is the case. In this case, D will be the opposite of its true self, so if it was a 1 before, it will be a 0 now, and vice versa. Substituting D for not D yields:

$$H \cdot c^T = \begin{pmatrix} C + \neg D + (C + D) \\ A + B + (A + B) \\ A + C + (A + C) \\ B + \neg D + (B + D) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

As can be seen here, when D is flipped, then the value of the resulting syndrome also changes. This can be deduced by knowing that $C + \neg D \neq (C + D)$, so $C + \neg D + (C + D) = 1$. This will always lead to the presented syndrome, then.

But how does this syndrome actually convey that the 4th bit was flipped? It appears to be a sort of random response.

Actually, it is very easy to convert this strange-looking syndrome into a valuable piece of information, i.e. which bit was flipped. To do so, simply look at the parity check matrix, and look for the column that is equal to this syndrome. Once a match is found, that column is known to belong to the erroneous bit.

In this case, $\begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix}^T$ is indeed the same as the 4th column in H, so we know that the 4th bit was flipped [13]. This is indeed what happened, so the method works.

4.3 The Public and Private in the McEliece Cryptosystem

Now that all of that has been covered, let's go back to the components of a McEliece Cryptosystem. This can all be pretty easily explained with this basic foundation just built.

4.3.1 The Public

Following are the 5 public components of the McEliece Cryptosystem, each with an explanation now:

1. The scrambled $k \times n$ generator matrix.

   This is the generator matrix just looked at, with a slight twist: all of the rows are randomly re-arranged. [23]

2. The k-length plain text

   This is the message m that Alice wrote for Bob.

3. The n-length vector

   This is the codeword c that Bob received.

4. The error correction capacity t

   This is the maximum number of errors that a cryptosystem can recover. This is correlated to the sizes of n and k. In this case, $t = 1$, but in other cases it can get quite large. [23]

5. The exponent m

   This is related to Goppa Codes [13]. As was just demonstrated, there is a lot of information in this field, so explaining Goppa codes effectively would probably double this paper's length. Therefore, that is outside the scope of this paper.

4.3.2 The Private

Following are the 3 private components of the McEliece Cryptosystem, each with an explanation now:

1. The random invertible $k \times k$ matrix S_{Mc}

   This is one of two matrices that obscure the contents of the generator matrix. This one is responsible for purer scrambling [13].

2. The $n \times n$ permutation matrix P_{Mc}

   This is the the second of two matrices that obscure the contents of the generator. Instead of scrambling, this simply permutes.

3. An efficient decoding algorithm for G

   This is a pre-determined method to quickly decode the received ciphertext. An attacker should be unable to discover this method. [13]

4.4 The McEliece Cryptosystem in Practice

There you have it. Although this is a gross oversimplification of how the McEliece Cryptosystem works, it does give a basic idea of the foundational concepts, so the reader should be able to take those concepts and now go and learn the more technical details if so desired.

The McEliece Cryptosystem is one of many options for post-quantum code-based cryptography, but it is surprisingly robust. That is, it was originally conceived in 1978, but it is still considered viable for post-quantum cryptography. Although there have been successfully exploited vulnerabilities in the initial proposal of the cryptosystem, fixes have been proven every single time that solve those problems, so it is still considered secure. For example, the European Union commissioned a recommendation on long-term cryptographic security, and the McEliece Cryptosytem was their #1 recommendation for public-key encryption! However, it is worthwhile to note that it's quite different from the style just shown. In addition to being significantly more complex in implementation, also the key sizes would be $n = 6960$, $k = 5413$ and $t = 119$ [7], versus our example of $n = 8$, $k = 4$ and $t = 1$! However, at the end of the day, they are the same concepts in essence, the former is just implemented in a much more sophisticated and effective way.

5 Supersingular Isogenies

The notion of designing modern cryptosytems which leverage the strength of supersingular isogenies represents one of the newest ideas suggested to preserve post-quantum security. It nonetheless shows promise. Supersingular isogenies are based on the beautiful mathematics which also buttress elliptic curve cryptography, but they extend the construction of ECC to include much more complex ideas and instruments; here we present an overview, as the details are much too involved.

Like ECC, SIs are used in key-exchange protocols which rely on the strength of the discrete logarithm problem, a problem that was famously deemed to be insufficiently hard following the analysis of Shor's algorithm. Protocols such as 2011's supersingular isogeny Diffie–Hellman key exchange (SIDH) (De Feo, et. al) save the wounded ECC/DLP/DHKE from quantum oblivion with some rather ingenious updates which require an advanced vocabulary in group theory. Elliptic curves are algebraic groups over some finite field modulo a prime number p that contain sets of points; the points are the group elements which are closed under an operator usually referred to as point addition or point doubling. The definition of an elliptic curve is a Weierstrauss function of the form

$y^2 \equiv x^3 + ax + b \bmod p$

[12] gives us functions which are symmetric across the x-axis and are often

depicted pedagogically as the turnip-shaped curves below.

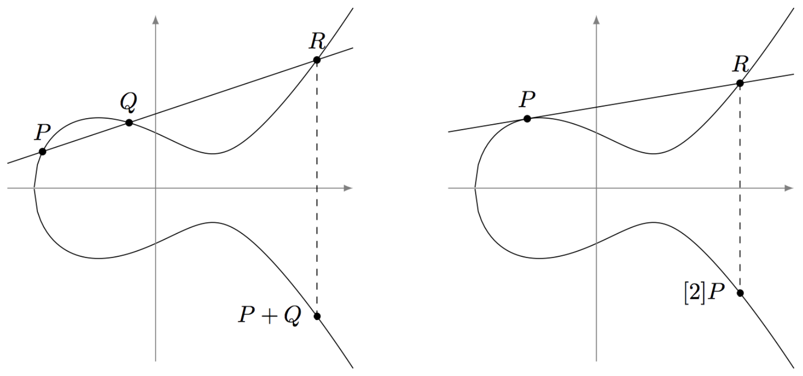

Figure 2: Two elliptic curves showing the point addition (left) and point doubling group operations (right) [8]

The addition and doubling operations are equivalent in both but we will discuss doubling. The operation begins at point P and, differentiating the curve at that point, drawing the tangent, and flipping the resulting point of intersection across the horizontal axis, we arrive at the result of one application of the group operation, what is variously notated as $[2]P$, $P + P$ or P^2. In classic ECDH, Alice and Bob share the same curve E, prime p and starting point P, and each alone performs a and b operations on P before trading. Each alone then "raises (adds)" the result to their exponent(sum) (both jump across the curve ab times) arriving at P^{ab} (or abP if you like). This is cryptographically equivalent to DHKE but allows for much smaller key lengths and implementation on smaller devices.

SI takes this concept to another dimension, essentially through the same concepts and properties of DHKE but expanding their limitations to a whole new world that still exists within its domain. Rather than restricting the key exchange process to one elliptic curve, the main idea behind SI is to use an entire field or graph comprised of several elliptic curves that share a complicated yet reliable mathematical relationship with one another. In particular, these elliptic curves have to be supersingular, where their connections are made up of both isomorphic and isogenous links to one another. This is where its name stems from, so now let us delve into and unpack what these terms represent in a manner that is hopefully understandable on a general or surface level.

As mentioned earlier, there is the Weierstrass form of the equation to define an elliptic curve. However, it is important to note that this equation can be formed in many ways, as the most traditional form previously mentioned often works in the form of particular coefficients being equal to 1 or 0. The Weierstrass equation can be expressed in this full form for an elliptic curve E [21]:

$$E : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$$

From here, we can further expand all these coefficients into further constants:

$$\begin{aligned}
&b_2 = a_1^2 + 4a_z\\
&b_4 = a_1a_3 + 2a_4,\\
&b_6 = a_3^2 + 4a_6,\\
&b_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2,\\
&c_4 = b_2^2 - 24b_4,\\
&c_6 = -b_2^3 + 36b_2b_4 - 216b_6
\end{aligned}$$

Putting the equation into this form and then substituting in these particular coefficients allow us to find what is called the determinant of the curve, which is defined in two different forms:

$$\Delta = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6$$

$$\Delta = (c_4^3 - c_6^2)/1728$$

With the determinant handy, this allows us to find what is called the j-invariant of the curve, a crucial identity in SI:

$$j(E) = c_4^3/\Delta$$

The j-invariant is such a critical component because it gives confirmation of two elliptic curves having an isomorphic relationship with one another, though as you can see thus far, the math necessary to determine the j-invariant is far from trivial. To make matters more complex, these elliptic curves are also defined over a field of a prime number that is greater than 3 and is usually quite large.

With this said, every elliptic curve over some field has a unique j-invariant. Because an isomorphic relationship is a bijection, this means any curve with the same j-invariant can be mapped back and forth, meaning if we are at one, there is a guaranteed way to get back to the other. Here is a picture to demonstrate the set of j-invariants over the field where $p = 431^2$ [6]:

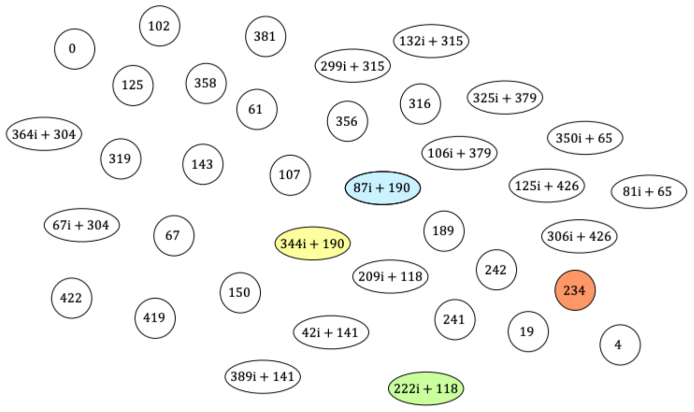

Figure 3: A simplified version of the j-invariants within the field $p = 431^2$ [6]

Similar to the isomorphic relationship, supersingular curves are guaranteed to have what's called an isogeny with another curve that exists in another field. These do not have the same j-invariants in the same field, like an isomorphism, but do share a j-invariant across different fields, such that:

Let $E = \{(j(E), j(E')) : \text{there is an isogeny } \phi : E \longrightarrow E' \text{ with } \deg(\phi) \in S\}$ [9].

These isogenies, E and E' share what's called a kernel, which is represented as G, meaning they share a subgroup. So what's the connection and importance between elliptic curves that are isomorphic to one another and those that are isogenous to each other as well? The two go hand in hand with creating a complex mapping. In the picture you saw of all the elliptic curves that share the same j-invariant, each one has two pathways or edges; these pathways are isogenies. In other words, we can use the isogenies of two elliptic curves to get to the isomorphisms of two others. However, isogenies are not bijective, so we simply cannot go back and forth between two isomorphisms using the same pathway, but there does exist some way to reach the destination through this series of paths [6].

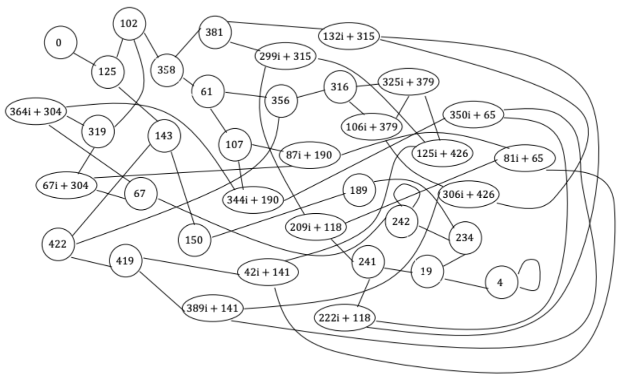

Figure 4: An isogeny mapping of j-invariants in the field $p = 431^2$[6]

This in turn, gives us the same setup for ECDH, which uses one elliptic curve, but now with multiple elliptic curves. Instead of Alice and Bob beginning on the same P on an elliptic curve, they have a starting point E, which is to say they start on the same elliptic curve in general. Alice then chooses a kernel $A \subset E(F_{p^2})$ and sends E/A to Bob. Bob does the same process but chooses a kernel $B \subset E(F_{p^2})$ and send E/B to Alice. The differences between these kernels, as stated, denotes that Alice and Bob are choosing two separate isogenies in respect to E. Then, the beauty lies within the fact that they are able to make an exchange through the following shared secret:

$$E/ < A, B >= (E/A)\phi_A(B) = (E/B)/\phi_B(A)$$

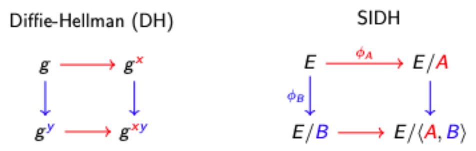
Diffie-Hellman (DH)
$g \longrightarrow g^x$
$g^y \longrightarrow g^{xy}$
SIDH
$E \xrightarrow{\phi_A} E/A$
ϕ_B
$E/B \longrightarrow E/\langle A, B\rangle$

Figure 5: A side-by-side comparison of Elliptic Curve Diffie Hellman Key Exchange (ECDH) and Supersingular Isogeny Diffie Hellman Key Exchange (SIDH) [12]

According to Jao [12], one of the most important figures within SI, he says the hardness of finding A when given E/A and E is essentially impossible, even for a quantum computer, where the search space is of size $deg(\phi) \approx 2^{300}$. The complexity yet simplicity behind such a system shows strong promises for the future, though there is still much research to be done for developing algorithms using them in a safe and efficient manner.

References

[1] Scott Aaronson. The limits of quantum computers. In *International Computer Science Symposium in Russia*, pages 4–4. Springer, 2007.

[2] Suanne Au, Christina Eubanks-Turner, and Jennifer Everson. The mceliece cryptosystem. *Unpublished manuscript*, 5, 2003.

[3] Xenia Bogomolec and Jochen Gerhard. Post-quantum secure cryptographic algorithms. *arXiv preprint arXiv:1809.00371*, 2018.

[4] Zvika Brakerski, Elena Kirshanova, Damien Stehlé, and Weiqiang Wen. Learning with errors and extrapolated dihedral cosets. In *IACR International Workshop on Public Key Cryptography*, pages 702–727. Springer, 2018.

[5] Curtis Bright. From the shortest vector problem to the dihedral hidden subgroup problem. 2011.

[6] Craig Costello. Supersingular isogeny key exchange for beginners. 2019.

[7] Augot Daniel, B Lejla, et al. Initial recommendations of long-term secure post-quantum systems. *(PDF). PQCRYPTO*, 2015.

[8] Luca De Feo. Mathematics of isogeny based cryptography. *arXiv preprint arXiv:1711.04062*, 2017.

[9] Steven Galbreith. Isogeny cryptography: strengths, weaknesses and challenges. 2018.

[10] Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential coding theory. 2012.

[11] Michael Harrison. Dihedral group (abstract algebra). https://www.youtube.com/watch?v=rPh7EQPSaO4, 2014. Accessed 6/11/20.

[12] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. 2019.

[13] Michiel Marcus. White paper on mceliece with binary goppa codes. 2019.

[14] Cristopher Moore, Alexander Russell, and Leonard J Schulman. The symmetric group defies strong fourier sampling. *SIAM Journal on Computing*, 37(6):1842–1864, 2008.

[15] Engineering National Academies of Sciences, Medicine, et al. *Quantum computing: progress and prospects.* National Academies Press, 2019.

[16] Ryan O'Donnell. The hidden subgroup problem: Lecture 17 of quantum computation at cmu. https://www.youtube.com/watch?v=x8i6qs8uAog, 2018. Accessed 6/11/20.

[17] Raphael Overbeck and Nicolas Sendrier. Code-based cryptography. In *Post-quantum cryptography*, pages 95–145. Springer, 2009.

[18] Chris Peikart. Lattice-based cryptography. https://www.youtube.com/watch?v=FVFw_qb1ZkY, 2016. Accessed 6/11/20.

[19] Chris Peikart. Lattice cryptography: A new unbreakable code. https://www.youtube.com/watch?v=2IyotuA8eJc, 2019. Accessed 6/11/20.

[20] Oded Regev. The learning with errors problem. *Invited survey in CCC*, 7:30, 2010.

[21] Nigel Paul Smart et al. *Cryptography: an introduction*, volume 3. McGraw-Hill New York, 2003.

[22] Andrew M Steane. A tutorial on quantum error correction. In *Proceedings-International School of Physics Enrico Fermi*, volume 162, page 1. IOS Press; Ohmsha; 1999, 2007.

[23] Martin Tomlinson, Cen Jung Tjhai, Marcel A Ambroze, Mohammed Ahmed, and Mubarak Jibril. *Error-Correction Coding and Decoding*. 2017.

[24] Bill Tourloupis. Example: Drawing lattice points and vectors. http://www.texample.net/tikz/examples/lattice-points/, 2012. Accessed 6/11/20.

[25] Frédéric Wang. The hidden subgroup problem. *arXiv preprint arXiv:1008.0010*, 2010.