PART B: STRUCTURED PROJECT CANDIDATE #: 1073741

Pre- and Post-Quantum Elliptic Curve Cryptography

March 2025

Mathematical Institute
University of Oxford

Contents

1 Introduction

The Information Age is one, if not the most rapid period of human innovation. At the center of it all is our modern communication. Science and culture now transcends geography and anything and everything that can be encoded as a signal can be sent wherever you want. Designing such technology to align with our values and wishes, however, is no easy task. In particular, secrecy poses a fair problem. There is no obvious analogy for "talking quietly" or "locking up" a message to be secure in the same way talking or physical/written communication intuitively lends. As such, information theory and cryptography have risen to prominence to ensure we can leverage our amazing new capabilities reassured with nothing to worry about. However, just as computers have provided us with so much, quantum computers seem to be the next big step for our progress. But the new power of such mystical machines also threatens some of the hard work we have done in cryptography over the last century.

The goal of this paper is to build from scratch an understanding of problems in cryptography, a relevant modern approach with elliptic curves and group theory, a potential concern with quantum computers, and potential solutions in the face of this quantum threat. We take a computational hands-on approach to the results to concretize most of the abstract algebra, for while it may explain *why* we should use something, it usually does not explain *how*. With much to cover, we try to maintain the core results and leave the rest to intuitive understandings and some of the rigor in the textbooks, with a special focus on the quantum theory and algorithms. When relevant, we appeal to the Appendix if proof strategies are isolated or unenlightening for later reference. Common and useful primitive algorithms discussed are also found in the Appendix.

1.1 Crediting and Sources

The way this is organized is that unless a section, result, or comment is cited, it is all of my own understanding and original process. Some sections like 3.2 draw heavily from only a couple specific references, and as such are credited at the beginning of the section as opposed to all throughout. While many results are typical and can be found in standard literature and are not necessarily my own derived work, all the motivations, intuitions, explanations, and proof sketches of such are my own (and such key words will be used to draw attention to these individual contributions).

2 Cryptography Preliminaries

Imagine two people, Alice and Bob, want to share a secret message m with each other over a public channel, like a radio frequency. Alice does not want to just send m freely, since someone might be listening in. So instead, she uses a function that takes a **private key** as a parameter to encrypt her message $E_k(m) = c$. This function naturally needs to comes with an inverse $D_k(c) = m$ to recover the message. Now she can send the encryption c to Bob without worry that someone might figure out what they are talking about. But now we are back at square one: how does Bob decrypt and recover the original message $m = D_k(c)$? Bob needs the key k, but Alice cannot send k over the channel since then a malicious eavesdropper could recover m. Alice could try encrypting k similar to m with $c_1 = E_{k'}(k)$, but then we have the same issue with needing to communicate the new private key k'. For Alice and Bob to be able to send secret messages to each other, *they already need to be able to send secrets to begin with.* They have a circular problem.

2.1 Diffie-Hellman and the Key Distribution Problem

In practice, this is where some additional step beyond the scope of the encryption method would have to take place. Alice and Bob could physically meet up, find a trusted third-party courier, or any other known, secure method to get the key to each other and importantly *only* each other. But once the private key is shared, they get the convenience of using insecure, public communication channels openly using their encryption methods.

In 1976, Whitfield Diffie and Martin Hellman [22] revolutionized this process, finding a method where Alice and Bob can together *create* a shared secret they can use for anything (in particular, creating a private key). What makes it especially incredible is that it only uses the public channel for them to make this secret.

Diffie-Hellman Key Exchange: Alice and Bob will together derive a shared secret piece of information, despite only communicating over public/insecure channels. We will specify the algorithm and analyze the security afterwards.

1. Alice and Bob agree on a value of n, and a generator g of a finite cyclic group $G \cong C_n$ of order n.
2. Alice chooses a random secret integer $1 < a < n$, and computes $A = g^a$. Similarly, Bob chooses a random secret integer $1 < b < n$, and computes $B = g^b$.
3. Alice sends the value of A to Bob, and Bob sends the value of B to Alice.
4. Alice computes $B^a = (g^b)^a$, while Bob computes $A^b = (g^a)^b$.
5. Alice and Bob have come to the shared value $(g^b)^a = g^{ab} = (g^a)^b$.

(A graphic of the above is in the Appendix) Together, they were able to come to a shared random element in $g^{ab} \in G$ without ever revealing it. The key lies in the associativity of exponents for group elements. Note the use of a cyclic group and its generator: if we use a non-generator of G, g^{ab} can no longer be any arbitrary element of G but rather only an element of the smaller subgroup $\langle g \rangle$, making a brute-force search of the shared secret easier. To be clear, here is what was and was not revealed in the public channel:

- Publicly known: G, g, n, g^a, g^b
- Privately known: a, b, g^{ab} (shared)

So Alice and Bob now have this shared element that they can use for whatever they require secrecy for, including their encryption keys. A common choice of cyclic group is using a subgroup of the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^\times$ for choice of prime p due to the efficiency of calculating modular exponents.

Note that this really is only practically useful for key distribution and generation. Alice and Bob independently think of a and b and cannot share these specific values with each other, so the shared value g^{ab} cannot be controlled by just Alice or just Bob to send a message. If Alice instead tried sending a message a as g^a, the fact we are working in the finite group C_n limits how many and what "messages" we can send. Moreover, we don't have a decryption protocol, so what makes it hard for an attacker to read a from g^a will be the same difficulty that Bob encounters when receiving g^a.

2.2 DLP and Security Assumptions

We are not quite in the clear yet. Crucially, we still need to show that if a malicious eavesdropper has all the public information $\{g, n, g^a, g^b\}$, they cannot also (feasibly) discover g^{ab}. This is the **Computational Diffie-Hellman Problem (CDH)**, and this is what we need to show that is hard to solve. If it is easy, then Alice and Bob still have the same problem of establishing secure secrets. We formalize this with a game against an attacker $\mathcal{A}$:

The CDH Experiment $\texttt{CDHExp}_{\mathcal{A},G}(n)$:

1. Pick a cyclic group G of order n with generator g. Pick two elements $h_1 = g^{k_1}$ and $h_2 = g^{k_2}$.
2. $\mathcal{A}$ is given G, n, g, h_1, h_2 and outputs an element h.
3. If $h = g^{k_1 k_2}$, we let $\texttt{CDHExp}_{\mathcal{A},G}(n) = 1$. Else $\texttt{CDHExp}_{\mathcal{A},G}(n) = 0$.

Definition 2.1 (Negligible function)**.** A function is *negligible* if for every positive polynomial $p(x)$, it is of order $O(1/p(x))$.

The idea is that if an attack only succeeds with probability some negligible function $f(n)$ where n is a security parameter of our encryption protocol (i.e. key length), then the success rate decreases faster than the increase in polynomial computing time of the attack; a not-tending-to-0 success probability would require a non-polynomial amount of attacks.

Definition 2.2 (CDH Hardness)**.** We say that the CDH problem is hard relative to G if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there is a negligible function $\texttt{negl}$ such that $\mathbb{P}(\texttt{CDHExp}_{\mathcal{A},G}(n) = 1) \leq \texttt{negl}(n)$.

Remark 2.3 (Time Complexity)**.** We simplify the nuance of time complexity for clarity. A polynomial time algorithm is one that takes an asymptotic polynomial number of steps $O(p(n))$ where n is some parameter. Here, n will often refer to a security parameter that ties to the strength of a protocol.

We consider polynomial time adversaries since those are efficient attacks; a non-polynomial time attack will quickly become infeasible. We do not actually have a definitive proof that CDH is hard, but we have strong heuristic evidence. As such, the CDH assumption states that there is a group for which CDH is hard. A related problem is the **discrete logarithm problem (DLP)**.

Definition 2.4 (Discrete logarithm)**.** If G is a finite group, b is an element of G, and $y = b^k$, then $k = \log_b y$ is the *discrete logarithm of y with respect to b.*

The DLP is to find k given b and y, and this is thought to be hard as well (we can formalize hardness in a similar experiment). Clearly, if DLP is easy, then CDH is also easy: just find a (or b) from $\{g, g^a\}$ and compute $(g^b)^a$. Because of the reduction of CDH to DLP, we often aim to work in groups where we believe DLP to be hard (formalized with a similar experiment as above; in fact, although they did not specify it formally, this was the original assumption Diffie and Hellman made when introducing their key-exchange [22]). The converse with solving CDH implies solving DLP is not known [32], but there are special cases where they are equivalent [37].

The CDH and DLP assumptions all assume existence of groups where solving their respective problem is difficult, but that assumption does not really matter if we cannot point to a specific group we can use (as the Diffie-Hellman Key Exchange uses finite cyclic groups, those will be our primary focus). Because solving DLP implies solving the former, much work has gone into DLP specifically to give at least a "bottom-line" difficulty for all these problems. We need to take care in this since there are clear cases where DLP is quite easy to solve.

Example 2.5. Consider the additive group $(\mathbb{Z}_{15}, +)$. Then DLP reduces to finding multiplicative inverses modulo 15. We can quickly solve $\log_4 7 = 13$ since $(4^{-1})(7) \equiv 13 \bmod 15$. In general, DLP can be solved efficiently for $(\mathbb{Z}_n, +)$ with the Euclidean algorithm in $O((\log n)^2)$ (see the Appendix for a sample implementation).

Before looking for a specific group G, we can already make some general observations about the group structure we want for DLP to be hard. For one, G seems like it should be at least of prime power order due to the Chinese Remainder Theorem (Section 9.5 [14]).

Theorem 2.6. *Let G be a finite cyclic group of order $|G| = p_1^{e_1} \cdots p_k^{e_k} = n$ with generator g, and p_i are prime. Set*

$$n_i = n/p_i^{e_i},\ g_i = g^{n_i},\ a_i = a^{n_i},\ x_i = log_{g_i} a_i.$$

If $x \in \mathbb{Z}_n$ is a solution to the simultaneous congruences $x \equiv x_i \bmod p_i^{e_i}$, then $x = \log_g a$.

Thus instead of needing to solving DLP in groups of general order n, we only have to solve them in groups with prime power orders and use the Chinese Remainder Theorem to stitch together a solution. In fact, we can further reduce solving DLP to solving DLP into just *prime order* subgroups.

Theorem 2.7. *Let G be a finite cyclic group of order $|G| = p^e = n$ with generator g and p prime, and let $a \in G$ be an arbitrary element. Then we can find $x = \log_g a$ with e sub-DL problems in groups of order p.*

The proofs are in the Appendix. They are not particularly enlightening, and rather only verify the general strategy that pervades much of algebra. Combining the two theorems above give the **Pohlig-Hellman algorithm** to find discrete logarithms. To complete the algorithm, we only now need a general DLP solving algorithm. It has been shown that for any non-group specific algorithm, we can do this at best in $O(\sqrt{n})$ group operations [42][53] (see Section 3.4). See the Appendix for one such $O(\sqrt{n})$ implementation with the Baby-Step Giant-Step algorithm.

Remark 2.8. Note $O(\sqrt{n})$ would seem to suggest we *have* found an efficient algorithm for solving DLP, contradicting the need to make any assumption that DLP is hard; $O(\sqrt{n})$ would suggest a polynomial time algorithm. This is where we require some nuance in how we understand time complexity. The point of time complexity is to understand how long an algorithm runs with respect to the *size* of the input we store in computer memory. Graph algorithms may measure size with number of vertices; sorting algorithms may measure size with number of elements to sort; here our input is *just a number*. Computers store numbers as bits, so we measure the size of a number by how many bits we need to represent it. Thus **we want our algorithms to not be polynomial in n, but polynomial in** $\log(n)$, in which case, our runtime is instead exponential $O(2^{.5\log(n)})$ (the 2 comes from binary length of a number). This is a useful metric as we often use number of bits as a security parameter, allowing us to parameterize complexity directly linked to our notions of security.

Remark 2.9. We now assemble the entire Pohlig-Hellman algorithm. Consider a prime p that divides $|G|$ with multiplicity e. In Theorem 2.5 we need $O(\log(|G|))$ group operations to compute the g^{n_p} and a^{n_p} (i.e. with a fast exponent algorithm; this one time cost is overall dominated). To determine the x_i as in Theorem 2.6, we solve e sub-DL problems in $O(\sqrt{p})$, and take $O(\log(|G|))$ for computing the powers in setting up the sub-DLPs for the p-adic expression. The Chinese Remainder Theorem has a minimal implementation cost. Together we see that the Pohlig-Hellman algorithm has time complexity $O\left(\sum_{\text{prime } p_i||G|}\left(e_i(\log(|G|)+\sqrt{p})\right)\right)$.

Example 2.10. If $|G| = 100,000,007$, which is of prime order, we would need roughly $\sqrt{100,000,007} \approx 10,000$ operations to solve DLP. On the other hand, if $|G| = 100,000,000 = 2^8 \cdot 5^8$, then it would only require roughly $(8+8)\cdot\log_2(100,000,000) + 8\sqrt{2} + 8\sqrt{5} \approx 455$ operations—a significant speedup!

One group we believe DLP to be difficult in is $\mathbb{Z}_p^\times$. It is cyclic with the existence of primitive roots (see *ASO: Number Theory*), and finding a generator is not terribly difficult (Section 11.1 [54]). Working with the multiplicative group is especially advantageous since the operation of integer multiplication is one we understand well and have implemented efficiently into computers, so we can calculate exponents modulo p very efficiently (in $O(\log n)$ with square-and-multiply algorithms). Yet, despite being a well-studied group, no other discrete-logarithm-solving algorithms have been found that are efficient. However, while there are no polynomial time algorithms that solve DLP in $\mathbb{Z}_p^\times$, there are subexponential ones that are faster than the general $O(\sqrt{n})$ algorithms mentioned. There are two glaring weaknesses for choosing $\mathbb{Z}_p^\times$: 1) $|\mathbb{Z}_p^\times| = p-1$ is not prime, so we have to work in a subgroup $G \leq \mathbb{Z}_p^\times$ hence requiring much larger choices of p to make the subgroup of suitable size, and 2) the properties of integers are more well-understood (like factorization) and attacks have been made exploiting these specific structures (i.e. *index-calculus attacks*; see the Appendix).

Remark 2.11 (Change of base for discrete logarithms)**.** I noticed the difficulty of DLP *only* relies on the choice of group G, and not the choice of generator as well. Let $b, \beta \in G$ be two generators in an n-order cyclic group, and $a \in G$ an element. Let $x = \log_b a$, $y = \log_\beta a$, and $z = \log_b \beta$. So $b^x = a = \beta^y = (b^z)^y$. Therefore $x = zy \bmod n$, and hence $\log_\beta a = (\log_b a)(\log_b \beta)^{-1} \bmod n$. So any algorithm that can efficiently solve DLP with respect to base b can also efficiently solve DLP with respect to any other base β.

Remark 2.12. While our definition of security is based on an asymptotic approach, we still need to eventually pick a concrete value of our security parameter n in practice; we still need to defend from the inefficient attacks at small scales as well. How do we pick this? Often, we pick n (i.e. key length in bits) such that the best-known attack takes at least 2^{128} operations [6]. For reference, the fastest supercomputer as of writing, El Capitan, can run at $\approx 1.74 \cdot 10^{18}$ floating-point operations/second [24]. El Capitan would need over 6 *trillion* years to get through all those operations (for reference, the Universe is estimated to be 13.8 billion years old [3]). (This ignores any parallelization, pre-processing, more specific types of computation, or any other optimized strategies, but gives a rough estimate of the difficulty induced by the demand of 2^{128}; it also requires at least 2^{128} possible keys to avoid brute force attacks). Requiring 2^n operations give the standard of n-bit *security level*, which we can use as a benchmark to compare the efficiency of different protocols which we will return to later [34]. A useful way to think of n-bit security is that it provides a benchmark against the perfect secrecy of the one-time pad; the best way to break the one-time pad is to brute force check all 2^n possible keys, so this gives a rough idea of the size needed to have an equally safe one-time pad. There are other ways to define security standards, but this is a simple reference for the efficiency of cryptographic

schemes for easy comparison.

2.3 Group Sizes and Parameter Selection

Just as before, even though DLP is thought to be hard, we still need to pick groups of an appropriate size so that computation is still infeasible. Given our best DLP solving algorithms run in $O(\sqrt{n})$ operations, we want the group we work in to be of an order about 256 bits long so that we can obtain the $\sqrt{2^{256}} = 2^{128}$ operation threshold from above. For the example above, that means we want our subgroup $G \leq \mathbb{Z}_p^\times$ to have order approximately 2^{256}. But this is a subgroup, so we have to defend from attacks against the *whole* group as well. The best known attack (specifically against $\mathbb{Z}_p^\times$) is a more refined index calculus-type method called the General Number Field Sieve that runs (heuristically) in $O(\exp[(64/9)^{1/3}(\ln n)^{1/3}(\ln \ln n)^{2/3}])$ [54] which suggests that p should be about 3072 bits long as a generous security barrier (including any algorithmic optimizations). In fact, it is suspected that using a 1024-bit prime is responsible for previous NSA leaks [2].

Table 1: Comparison between Classic Cryptographic Schemes*

n-Bit Security	RSA	Discrete Logarithm	
	Length of N	Order-q Subgroup of $\mathbb{Z}_p^\times$	Order p Additive Group $(\mathbb{Z}_p, +)$
112	2048	$p = 2048,\ q = 224$	2^{56}
128	3072	$p = 3072,\ q = 256$	2^{64}
192	7680	$p = 7680,\ q = 384$	2^{96}
256	15360	$p = 15360,\ q = 512$	2^{128}

*Cryptographic performance compared in terms of bit-lengths of their public keys.

Also, it is important to note that our choice of group is important. We have seen that solving DLP in some groups can be trivial, but it is also important to consider that some groups' operations are more difficult to implement. While it is true that all cyclic groups of the same order are isomorphic, the efficiency in implementing the group operation can vary greatly. The reason why it is common to use $\mathbb{Z}_p^\times$ is because we have spent the time to understand how to perform integer multiplication as computer operations efficiently. If we want to preserve this idea for DLP of easy to compute one way and hard to compute the other, the choice of cyclic group can greatly impact our performance. From Table 1, we can see the compared efficiencies of schemes with RSA and different choices of groups with DLP (the size of group directly impacts the amount of memory needed to store each element on a computer). If we are to consider another choice of group, ideally we would have smaller paramter sizes for equal or better security (and if same size parameters, then more efficient implementations). In our case, **elliptic curve cryptography (ECC)** and its choice of groups gives us what we are looking for.

3 Classic Elliptic Curve Cryptography

3.1 Mathematical Preliminaries

With a name that insinuates some amount of geometry, the math involved for ECC requires a bit more time to develop on its own. In particular, we need the machinery to work with elliptic curves over finite fields. We recall some facts about finite fields. The full details can be found in the Appendix.

Definition 3.1 (Field)**.** A set $\mathbb{K}$ equipped with two commutative operations $+$ and $\times$ is a *field* if $(\mathbb{K}, +)$ and $(\mathbb{K}\setminus\{0\}, \times)$ are groups with identity elements 0 and 1 respectively. Also, $\forall a, b, c \in \mathbb{K}\ \ a\times(b+c) = a\times b + a\times c$ ($\times$ distributes over $+$). For convenience, we often suppress $\times$ and write $a \times b$ as ab.

Example 3.2. $\mathbb{R}, \mathbb{Q}, \mathbb{C}, \mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ for prime p are all fields with the natural definitions of $+$ and $\times$.

Definition 3.3 (Characteristic)**.** The *field characteristic* $\mathrm{Char}(\mathbb{K})$ is the least integer n such that $\overbrace{1 + 1 + \cdots + 1}^{\text{n times}} = 0$. If there is no such integer, the field is said to have characteristic 0.

Proposition 3.4. *The characteristic of a field is either prime or 0.*

Proposition 3.5. *If* $\text{Char}(\mathbb{K}) = p$*, then* $(a+b)^p = a^p + b^p$ *for all* $a, b \in \mathbb{K}$*.*

Theorem 3.6. *There is a unique (up to isomorphism) finite field* $\mathbb{K}$ *of order* q *if and only if* $q = p^n$ *for some prime* p *and integer* n*. We denote this field* $\mathbb{F}_q$*.*

Proof Sketch. The forward direction can be proven by considering finite fields as finite dimensional $\mathbb{Z}_{\text{Char}(\mathbb{K})}$-vector spaces. The important idea with the converse is that we construct the field as the roots of $f(x) = x^{p^n} - x$ in the algebraic closure of $\overline{\mathbb{Z}_p}$. In particular, this can be seen as the defining feature of finite fields and their extensions: $\mathbb{F}_q \subseteq \{x^q = x : x \in \overline{\mathbb{Z}_p}\}$ by Lagrange's theorem, and $x^q - x = 0$ can only have at most q elements. □

Example 3.7 (Binary fields)**.** *Binary fields* of order 2^d are interesting since they allow for efficient computation and implementation. For example, if we represent elements $a_0 + a_1 x + a_2 x^2$ of the field $\mathbb{F}_8 \cong \mathbb{F}_2[x]/\langle x^3 + x + 1\rangle$ as the binary string $a_0 a_1 a_2$, addition just becomes the XOR operation since the field is characteristic 2 and polynomial addition is the same as coefficient addition of terms of the same degree. Similarly multiplication can be done via bit-shifts and XORs. These are worth using for hardware optimized for these operations, but extra care needs to be taken in ensuring the security of the protocol in which we use them (in our case, the elliptic curve [47]).

3.2 Elliptic Curves and the Group Law

Definition 3.8 (Elliptic curve)**.** Let $\mathbb{K}$ be a field, and $a, b, c \in \mathbb{K}$. An *elliptic curve over* $\mathbb{K}$, denoted $E/\mathbb{K}$, is an equation of one of the following forms based on its characteristic:

$$\begin{cases} \text{Char}(\mathbb{K}) = 2: & y^2 + cy = x^3 + ax + b \\ \text{Char}(\mathbb{K}) = 3: & y^2 = x^3 + ax^2 + bx + c \\ \text{Char}(\mathbb{K}) > 3: & y^2 = x^3 + ax + b \end{cases}$$

Let $E(\mathbb{K}) = \{(x, y) \in \mathbb{K}^2 : (x, y) \text{ satisfy } E\} \cup \{\infty\}$ where we include an element ∞ called the "point at infinity". In particular, we will be considering *non-singular* elliptic curves, i.e. if we consider the above as an implicit equation $F(x, y) = 0$, then there is no point $(x, y) \in E$ such that $\partial F/\partial x = \partial F/\partial y = 0$. For the last case, this is equivalent to $\Delta = 4a^3 + 27b^2 \neq 0$ (this is the *discriminant*).

Looking ahead, we are looking to construct a group out of an elliptic curve $E(\mathbb{K})$. So in particular, we want some way to take 2 points $P_1, P_2 \in E(\mathbb{K})$ and generate a third point $P_3 \in E(\mathbb{K})$.

Note that our elliptic curves are degree 3 equations in x. One of the ways to characterize cubic equations is that they are the equations that precisely have 3 roots in an algebraically closed field (i.e. $\mathbb{C}$). Put another way, they (sometimes) intersect the line $y = 0$ (in $\mathbb{R}^2$) in precisely 3 points (counting multiplicities). We can generalize this to any line: let $y = mx + b$, and substitute this into the elliptic curve equation, we get $(mx + b)^2 = x^3 + ax + b$. This is still a cubic and hence has 3 solutions. Since lines intersecting cubics relate these 3 points to each other, this suggests a potential way to maybe make a binary operation out of an elliptic curve E: define a line via P_1, P_2 and find the third intersection point P_3.

Definition 3.9 (Group law for elliptic curves over $\mathbb{R}$)**.** Let $E/\mathbb{R}$ be an elliptic curve, and $P, Q \in E(\mathbb{R})$. We define the inverse $-P$ and sum $P \oplus Q$.

- If $P = (x, y)$, then $-P = (x, -y)$. If $P = \infty$ is the point at infinity, then $-P = \infty$.

We define $P \oplus Q = -R$ where R is the intersection between the line $\ell = \overline{PQ}$ and E. That is, $P \oplus Q$ is the reflection of the third intersection point. We can define R as follows.

1. If P and Q have different x-coordinates, then the line $\ell = \overline{PQ}$ will intersect E in a distinct point R.
2. If $\ell = \overline{PQ}$ is tangent to E at P, then we let $R = P$ (and similarly if ℓ is tangent at Q).

3. If $P = Q$, then let R be the intersection point between E and the line tangent to E at P.
4. If $Q = -P$, then $R = \infty$.
5. $P \oplus \infty = \infty \oplus P = P$.

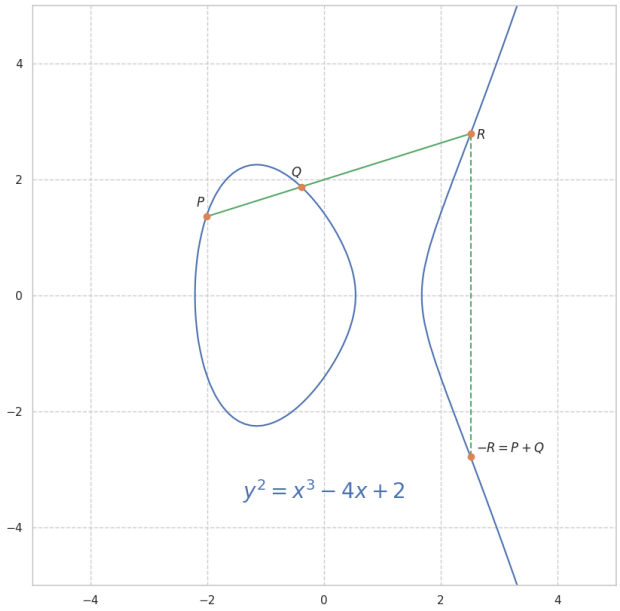


Figure 1: Example of the group addition on an elliptic curve over $\mathbb{R}$.

This definition makes it clear why now include the point at infinity ∞: vertical lines only intersect E at 2 points not 3 without ∞! ∞ can be thought of as a point that is at the end of every vertical line in our plane. We can also straightforwardly derive the coordinates of R algebraically. Let $P = (x_P, y_P), Q = (x_Q, y_Q), R = (x_R, y_R)$ be with $P \neq Q$. The line $\overline{PQ}$ is given by $y = \frac{y_Q - y_P}{x_Q - x_P}(x - x_P) + y_P$. We thus want to solve

$$\left(\frac{y_Q - y_P}{x_Q - x_P}(x - x_P) + y_P\right)^2 = x^3 + ax + b.$$

Note that for a monic cubic with 3 roots $(x-\alpha)(x-\beta)(x-\gamma) = x^3 - (\alpha+\beta+\gamma)x^2 + \cdots$, i.e. the coefficient of x^2 is the sum of the solutions of the cubic. We know that x_P, x_Q are solutions by construction of the line via $\overline{PQ}$, and hence we can find the third solution x_R. Finding y_R corresponds to reflecting the y-coordinate of the intersection:

$$x_R = \left(\frac{y_Q - y_P}{x_Q - x_P}\right)^2 - x_P - x_Q, \quad y_R = -\left(\frac{y_Q - y_P}{x_Q - x_P}(x_R - x_P) + y_P\right).$$

If $P = Q$, we replace the slope of the line with the tangent $\frac{dy}{dx}$ at P. Implicitly differentiating the elliptic curve nets $\frac{dy}{dx} = \frac{3x_P^2 + a}{2y_P}$. Rearranging similarly to above nets us a formula for $P \oplus P = 2P$:

$$x_R = \left(\frac{3x_P^2 + a}{2y_P}\right)^2 - 2x_P, \quad y_R = -\left(\frac{3x_P^2 + a}{2y_P}(x_R - x_P) + y_P\right).$$

(If working in characteristic 2 or 3, similar formulas can be derived with the same method). Note since we are assuming $P, Q \in E \subset \mathbb{R}^2$, the above clearly show $R = (x_R, y_R) \in \mathbb{R}^2$. By construction, it also satisfies the elliptic curve equation, and hence $R \in E$. Crucially, though, this works in *any* field. Since fields support addition and multiplication, and subtraction and division by the existence of inverses, if $P, Q \in \mathbb{K}^2$, our

above formulas show $R \in \mathbb{K}^2$ and satisfy the elliptic curve i.e. $R \in E$ (this last part is not as obvious; we did our derivation in $\mathbb{R}$, but one can just check the algebra to see $y_R^2 = x_R^3 + ax_R + b$). We can abstract away this geometric approach to the operation and treat it completely algebraically.

Theorem 3.10. *Let $E/\mathbb{K}$ be an elliptic curve. With the notion of addition above, $(E(\mathbb{K}), \oplus)$ forms an abelian group. That is, for all (not necessarily distinct) $P, Q, R \in E$:*

1. *(Commutativity) $P \oplus Q = Q \oplus P$*
2. *(Identity) $P \oplus \infty = \infty \oplus P = P$*
3. *(Inverses) $P \oplus (-P) = \infty$*
4. *(Associativity) $P \oplus (Q \oplus R) = (P \oplus Q) \oplus R$*

Proof. This is an interesting fact with a less interesting proof; having explicit formulas allows us to just compute and verify. (Commutativity): $\overline{PQ} = \overline{QP}$ are the same line and hence determine the same unique $P \oplus Q$. (Identity) By definition; also, ∞ to be the point above all vertical lines, $\overline{\infty P}$ intersects E at $-P$, and reflecting again gives P. (Inverses) The vertical line $\overline{P(-P)}$ intersects ∞, and $-\infty = \infty$. (Associativity) Pick 3 points and compute and verify (lots of cases). □

The only really non-obvious fact is associativity. For more elegant proofs outside of rote calculation, one can take a geometric (Lemma 2.11 [62], Section 3.1, Theorem 1.2 [30]) or a complex analytic approach (Section VI.1 [32]) to prove associativity in specific fields.

Remark 3.11. It might seem odd that we define $P \oplus Q$ to be the reflection of the intersection $-R$ as opposed to just R. However, in trying to form a group, ∞ is acting as our identity element. If we try to calculate $P \oplus \infty$, the third point of intersection would be $-P$. If we want to preserve $P \oplus \infty = P$, we need to reflect the intersection. This reflection is integral to our above group construction.

Remark 3.12 (Weierstrass normal form)**.** Our description of an elliptic curve is an equation written in *Weierstrass normal form*. If the crucial detail is we have these 3 intersection points between lines and the curve, Weierstrass equations seem to constitute a very narrow class of possible cubic equations we could consider. The reason we do not look at any arbitrary cubic is because any non-singular cubic can be rewritten into Weierstrass form [56] (see *Part B: Algebraic Curves*).

Remark 3.13 (Homogeneous Coordinates)**.** With the need to include a "point at infinity" ∞ that exists somewhat artificially in our group structure, it is natural to explore the topic in the context of projective planes and homogeneous coordinates. Beyond just unifying the theory a bit more, there are some nice implementation benefits that allow us to avoid calculating expensive divisions (see the Appendix).

We used $\mathbb{R}$ as the visuals make more sense, but it is not an obvious fact that we get our three intersections in any finite field (when also including the point at infinity). Our intuitive fact for algebraically-closed fields is a specific case of *Bézout's theorem* (see *Part B: Algebraic Curves*):

Theorem 3.14 (Bézout's theorem)**.** *If C and D are two algebraic curves in the projective plane $\mathbf{P_2}$ of degrees n, m with no common component, then they intersect (counting multiplicities) at nm points.*

Although we do not strictly need this theorem once we have the above formulas for addition, Bézout's theorem can be seen as another potential motivator to look at elliptic curves in the way we have.

Bringing it back to cryptography, we can now state the believed-to-be-hard problem of elliptic curves: DLP for elliptic curves (**ECDLP**). Given $P \in E(\mathbb{F}_q)$, then $nP = Q$, then n *is the discrete logarithm of Q with respect to P* (where nP denotes the n-repeated-addition of P to itself). Our goal now is to justify why this group in particular is preferable for DLP.

3.3 Group Structure and Hasse's Bound

We can now define groups over elliptic curves, but as part of constructing this group, clearly which field $\mathbb{K}$ we are working in impacts the structure of this group. The primary question is, as we have seen before,

how big is this group? For the sake of computation, we use finite groups, and hence will consider elliptic curves over a finite field $\mathbb{F}_q$. We'll denote the multiplicative group of the field by $\mathbb{F}_q^\times = \mathbb{F}_q \setminus \{0\}$.

First, note an elliptic curve $E(\mathbb{F}_q)$ can have at most $2q$ $\mathbb{F}_q$-points: for each x, there is at most 2 possible $y \in \mathbb{F}_q$ that satisfy E (each element has at most 2 square roots). However, since the map $x \mapsto x^2$ is 2-to-1 in $\mathbb{F}_q^\times$, only half the elements in $\mathbb{F}_q$ have well-defined square roots. So we might expect maybe at most half that many points to be in $E(\mathbb{F}_q)$. Including the point at infinity ∞, $2q/2 + 1 = q + 1$ would be an upper bound on the size of the elliptic-curve group. *Hasse's bound* says that this is heuristically the correct size.

Theorem 3.15 (Hasse's bound)**.** *For an elliptic curve $E/\mathbb{F}_q$, if $N = |E(\mathbb{F}_q)|$, then $|N - (q+1)| \leq 2\sqrt{q}$.*

To prove this, we need to consider **endomorphisms**. We will follow the computational approach in [62] (Section 2.9), taking inspired intuitions and adapted propositions and proofs from the more generalized abstractions as found in [56][55].

Definition 3.16 (Isogeny)**.** An *isogeny* between two elliptic curves $E_1/\mathbb{K}, E_2/\mathbb{K}$ is a group homomorphism $\alpha : E_1(\overline{\mathbb{K}}) \to E_2(\overline{\mathbb{K}})$ given by rational functions i.e. $\alpha(x, y) = (R_1(x, y), R_2(x, y))$ for rational R_i with coefficients in $\overline{\mathbb{K}}$ (where $\overline{\mathbb{K}}$ is the algebraic closure of $\mathbb{K}$). We also define that $\alpha(\infty) = \infty$. Two elliptic curves are *isogenous* if an isogeny exists between them.

Definition 3.17 (Endomorphism)**.** An *endomorphism* is an isogeny to and from the same elliptic curve.

In what follows, we will consider endomorphisms for convenience and clarity, but all definitions and propositions below further apply to isogenies too.

Remark 3.18. This fits the standard definition of endomorphisms with groups, only with the additional restriction that the coordinate-wise functions are also rational. The primary reason is to ensure our endomorphisms are well-behaved. Our objects, elliptic curves, are defined via polynomials, so it is natural to want to keep our operations confined to the same realm of objects; this follows the typical study in algebraic geometry. In a similar vein, note the use of the closure of the field $\overline{\mathbb{K}}$. We have to remember these elliptic curves are not just algebraic objects, but geometric ones as well. Working in the closure ensures we capture the total structure of such objects, and makes studying the original groups $E(\mathbb{K})$ just restrictions of these richer objects.

Remark 3.19. Isogenies inherit a lot of structure from the groups that allows us our definitions to be a lot looser than what we would imagine useful. For example, surjectivity is given for free (Theorem 2.22 [62]), and even being homomorphism is implied just being a non-constant rational map (III.4.8 [55]). A useful one we will refer to every now and then is a type of converse to the First Isomorphism Theorem for groups. By the First Isomorphism Theorem, we know for any isogeny $\ker \phi$ is a subgroup of the domain. For elliptic curves, the converse turns out to be true: for any finite subgroup $G \leq E(\overline{\mathbb{K}})$, there exists a unique elliptic curve E' and separable isogeny $\phi : E(\overline{\mathbb{K}}) \to E'(\overline{\mathbb{K}})$ such that $\ker \phi = G$ (III.4.12 [55]). We will write the unique codomain $E' = E/G$ to explicitly mark out a relation between E and G, but it is mostly syntactical since while the First Isomorphism Theorem justifies this notion of quotients, E' is still another elliptic curve equation that does not necessarily have the similar transfer of quotienting directly.

A possible motivation for looking at endomorphisms is given by this simple proposition in group theory.

Proposition 3.20. *Let $\varphi : G \to H$ be a group homomorphism, and $h \in \mathrm{Im}(\varphi)$. Then $|\varphi^{-1}(h)| = |\ker(\varphi)|$.*

Proof. Let $x \in \varphi^{-1}(h)$. By the First Isomorphism Theorem, $G/\ker(\varphi) \cong \mathrm{Im}(\varphi)$ via mapping cosets $g\ker(\varphi) \mapsto \varphi(g)$. Hence $\varphi^{-1}(h) = x\ker(\varphi)$ and thus $|\varphi^{-1}(h)| = |x\ker(\varphi)| = |\ker(\varphi)|$. □

If we can find a suitable homomorphism whose kernel is isomorphic to our group, we can estimate its size by finding the size of the preimage under the endomorphism. We will generalize this later. The following endomorphism, called the *Frobenius endomorphism* will be our primary focus.

Example 3.21. The map $[n] : P \mapsto nP$ where n is an integer is an endomorphism.

Example 3.22 (Frobenius endomorphism)**.** Let $(x, y) \in E(\mathbb{F}_q)$. Then $\Phi_q(x, y) = (x^q, y^q)$ is an endomorphism.

Proof. To check this, let $(x, y) \in E(\overline{\mathbb{F}_q})$, so $y^2 = x^3 + ax + b$. Since $q = p^n$, and $\text{Char}(\overline{\mathbb{F}_q}) = p$, we can use the fact that $(a + b)^q = a^q + b^q$. Raising the equation E to q on both sides gives: $(y^q)^2 = (x^q)^3 + a^q x^q + b^q$. Recall that $\mathbb{F}_q^\times$ is a group of order $q - 1$, so $a^q = a$ for all $a \in \mathbb{F}_q^\times$. Hence $(y^q)^2 = (x^q)^3 + ax^q + b$ and so $\Phi_q(x, y) \in E(\overline{\mathbb{F}_q})$.

For the homomorphism property, consider the sum of two arbitrary points $(x_1, y_1) \oplus (x_2, y_2) = (x_3, y_3)$. Using the formulas given above, we can find that $\Phi_q(x_3, y_3)$ is:

$$x_3^q = \left(\frac{y_2^q - y_1^q}{x_2^q - x_1^q}\right)^2 - x_1^q - x_2^q, \quad y_3^q = \frac{y_2^q - y_1^q}{x_2^q - x_1^q}(x_1^q - x_3^q) + y_1^q.$$

Comparing the formulas, we see then that $\Phi_q((x_1, y_1) \oplus (x_2, y_2)) = \Phi_q(x_1, y_1) \oplus \Phi_q(x_2, y_2)$. A similar check can be made on the formulas if adding the same point to itself; the key detail is that $(a + b)^q = a^q + b^q$ allowing us to distribute powers of q through our rational formulas. So Φ_q is an endomorphism. □

Remark 3.23. Note the Frobenius map Φ_q is not necessarily an endomorphism for all q. Notably, we relied on $|\mathbb{K}^\times| = q - 1$ that allows us to reduce the coefficients $a^q = a, b^q = b$ with group theory. More generally, if $\text{Char}(\mathbb{K}) = p$, then $\Phi_{p^k} : E(\overline{\mathbb{K}}) \to E^{(p^k)}(\overline{\mathbb{K}})$ is an isogeny where $E^{(p^k)}$ is the p^k-power Frobenius image of E (this is different from $\text{Im}(\Phi_{p^k})$; note we must use a power of p to preserve it being a homomorphism). Specifically, if $E : y^2 = x^3 + ax + b$ then $E^{(p^k)} : y^2 = x^3 + a^{p^k}x + b^{p^k}$. Using the same verification above of raising both sides of E to p^k, it is easy to see that if $(x, y) \in E(\overline{\mathbb{K}})$, then $\Phi_{p^k}(x, y) \in E^{(p^k)}(\overline{\mathbb{K}})$.

Proposition 3.24. *Let Φ_q be the Frobenius endomorphism, and let $(x, y) \in E(\overline{\mathbb{F}_q})$. Then $\Phi_q(x, y) \in E(\mathbb{F}_q) \iff \Phi_q(x, y) = (x, y)$.*

Proof. Since $x \in \overline{\mathbb{F}_q}$, $x \in \mathbb{F}_q \iff x^q = x$ (see Theorem 3.6; $\mathbb{F}_q$ is the q distinct solutions of $x^q = x$). Then along with our hypothesis $(x, y) \in E(\overline{\mathbb{F}_q})$, we get the straightforward equivalences:

$$(x, y) \in E(\mathbb{F}_q) \iff x, y \in \mathbb{F}_q \iff x^q = x, y^q = y \iff \Phi_q(x, y) = (x, y)$$ □

Proposition 3.25 (Endomorphism standard form)**.** *Every endomorphism can be written as $\alpha(x, y) = (r_1(x), r_2(x)y)$ for rational functions $r_1(x), r_2(x) \in \overline{\mathbb{K}}(x)$.*

The proof follows mostly considering the fact that all even powers of y can be eliminated in x with the curve equation $y^2 = x^3 + ax + b$, and the rest is considering what is necessary to preserve α as a homomorphism. The standard form makes working with endomorphisms much simpler. For one, we can now measure the "complexity" of endomorphisms in a similar way to how we would with polynomials. The degree of a polynomial tells us how many roots it has, and in particular tells us how many possible inputs can map to a given output. We can ask the same question for endomorphisms: for a given $Q = (x_Q, y_Q)$ how may points $P = (x_P, y_P)$ are there such that $\alpha(P) = Q$? Looking at the standard form, we want to solve $(r_1(x_P), r_2(x_P)y_P) = (x_Q, y_Q)$. If we write $r_1(x) = p(x)/q(x)$, then by looking at the x-coordinate, we want to solve $p(x_P) - x_Q q(x_P) = 0$. This is a polynomial in x_P of degree $d = \max(\deg p(x), \deg q(x))$, and hence we have up to d possible points such that $\alpha(P) = Q$. This gives us the following definition.

Definition 3.26 (Endomorphism degree)**.** Let $\alpha(x, y) = (r_1(x), r_2(x)y)$ be an endomorphism in standard form. Then write $r_1(x) = p(x)/q(x)$. The *degree of* α is $\deg(\alpha) = \max(\deg p(x), \deg q(x))$. If $\alpha \equiv 0$, then we say $\deg(\alpha) = 0$.

Example 3.27. The Frobenius endomorphism has degree $\deg(\Phi_q) = q$.

Example 3.28. Consider the endomorphism $[2](P) = 2P$. Since the formula for the x-coordinate is

$$x_{2P} = \left(\frac{3x_P^2 + a}{2y_P}\right)^2 - 2x_P = \frac{\left(3x_P^2 + a\right)^2 - 4x_P(x_P^3 + ax_P + b)}{4(x_P^3 + ax_P + b)}$$

we can see that $\deg([2]) = 4$. Inductively, we directly get $\deg([2^n]) = 2^{2n}$. If one spends the time substituting all the formulas for coordinates, one can find for all n that $\deg([n]) = n^2$

The motivation of this definition is that it makes degree act similarly to how it would act with polynomials. For example, it is multiplicative with respect to composition: $\deg(\alpha \circ \beta) = \deg(\alpha)\deg(\beta)$.

Definition 3.29 (Separability)**.** Let $\alpha(x, y) = (r_1(x), r_2(x)y)$ be a non-zero endomorphism in standard form. α is *separable* if $r_1'(x) \not\equiv 0$ (where we are using the formal derivative as opposed to limit definitions).

This too is adopted from a similar idea with polynomials. A polynomial $f(x) \in \mathbb{K}[x]$ is separable if it has distinct roots in the algebraic closure $\overline{\mathbb{K}}$ (this is not quite right; we consider a special field extension called its *splitting field*). If $f(x)$ has a repeated root α, it's easy to see that $\gcd(f(x), f'(x)) = (x-\alpha)g(x)$ for some $g(x)$ (easy to show converse too). So if f is irreducible and not separable, $\gcd(f, f') \neq 1$. But since $f(x)$ is irreducible, it must be that $f(x) \mid f'(x)$. Though, $\deg(f') < \deg(f)$, so it must be that $f' \equiv 0$. (Converses are easy to see to make these notions a proper equivalence and hence definition).

Example 3.30. The Frobenius endomorphism Φ_q is not separable, as $r_1'(x) = qx^{q-1} = 0$ since q is a power of $p = \mathrm{Char}(\overline{\mathbb{F}_q})$.

Remark 3.31 (Endomorphism decomposition)**.** [59] Using relatively straightforward algebra on the standard form, we can show all inseparable endomorphisms are of the form $\alpha(x, y) = (r_1(x^p), r_2(x^p)y^p)$ for rational functions r_1, r_2 in a field of characteristic p. Hence, we can decompose any endomorpshim $\alpha = \alpha_{\text{sep}} \circ \Phi_p^n$ for some separable endomorphism α_{sep} and Φ_p is the p-power Frobenius endomorphism by checking if α is inseparable, and "extracting" the inseparability by factoring out Φ_p repeatedly. We call $\deg_s(\alpha) = \deg(\alpha_{\text{sep}})$ the *separable degree of* α (hence note $\deg(\alpha) = p^n \deg_s(\alpha)$). If $\deg_s(\alpha) = 1$, we say α is *purely inseparable.*

The following is our most important tool in proving Hasse's theorem.

Theorem 3.32. *Let $\alpha \neq 0$ be an endomorphism of an elliptic curve E. Then $\deg(\alpha) \geq |\ker(\alpha)|$, with equality if and only if α is separable.*

We also need one more fact from field theory.

Proposition 3.33. *Any algebraically closed field $\overline{\mathbb{K}}$ is infinite.*

Proof. Say otherwise, and $\overline{\mathbb{K}} = \{\alpha_1, \cdots, \alpha_n\}$. Then consider $f(x) = 1 + \prod_{i=1}^n (x - \alpha_i)$ has no roots in $\overline{\mathbb{K}}$. □

Proof of Theorem 3.32. We first show the separable case. The idea is to find a point in $\mathrm{Im}(\alpha)$, show there are $\deg(\alpha)$ points in its preimage, and then use the above proposition to determine $|\ker(\alpha)|$. Write $\alpha(x, y) = (r_1(x), r_2(x)y)$ in standard form with $r_1(x) = p(x)/q(x)$. α is separable and so $r'(x) \not\equiv 0$, thus $p'(x)q(x) - p(x)q'(x) \not\equiv 0$. Let $S = \{x \in \overline{\mathbb{K}} : (p'q - pq')(x)(q(x)) = 0\}$ (note this is finite). Now pick a point $(a, b) \in E(\overline{\mathbb{K}})$ with the following conditions:

1. $a, b \neq 0, (a, b) \neq \infty$.
2. $\deg(p(x) - aq(x)) = \max(\deg p(x), \deg q(x)) = \deg(\alpha)$.
3. $a \notin r_1(S)$.
4. $(a, b) \in \alpha(E(\overline{\mathbb{K}}))$.

Conditions (1)–(3) ensure we pick a "normal" point, and (4) is so that we pick a point in the image. Note there are only finitely many points that individually contradict conditions (1)–(3). Also, since we are working in an algebraically closed field, for every x there is a corresponding y such that $(x, y) \in E(\overline{\mathbb{K}})$, and so $\alpha(E(\overline{\mathbb{K}}))$ is an infinite set. So there is a possible choice of (a, b) satisfying the above.

Let $(x_1, y_1) \in E(\overline{\mathbb{K}})$ such that $\alpha(x_1, y_1) = (a, b)$ i.e. $p(x_1)/q(x_1) = a$ and $r_2(x_1)y_1 = b$. Since x_1 determines y_1, we only count the possible x_1. The x_1 are the roots of $p(x_1) - aq(x_1) = 0$, and by assumption (2), this has $\deg(\alpha)$ solutions with multiplicity. We need to show these roots are distinct.

Let x_0 be a double root. Then $p(x_0) - aq(x_0) = p'(x_0) - aq'(x_0) = 0$. Hence, multiplying the equations $p(x_0) = aq(x_0)$ and $p'(x_0) = aq'(x_0)$, we have $ap'(x_0)q(x_0) = ap(x_0)q'(x_0)$. Since $a \neq 0$, this implies that x_0 is a root of $p'q - pq'$ i.e. $x_0 \in S$. But then $a = r_1(x_0) \in r_1(S)$, contradicting (3).

So $p(x) - aq(x)$ has $\deg(\alpha)$ distinct solutions, and by Proposition 3.20, $|\ker(\alpha)| = |\alpha^{-1}(a,b)| = \deg(\alpha)$.

For the inseparable case, the steps are the same, except we notice that $p' - aq'$ is identically 0, and hence $p - aq$ has a repeated root and so fewer than $|\deg \alpha|$ distinct solutions. □

Remark 3.34. The above proof is essentially the idea to prove isogenies are surjective: pick any point $(a,b) \in E(\overline{\mathbb{K}})$, and look at the roots of the corresponding polynomial $p(x) - aq(x)$, checking the cases when it is constant/non-constant.

I observe the following corollaries.

Corollary 3.35. *For all endomorphisms α, $\deg_s(\alpha) = |\ker(\alpha)|$.*

Proof. Note $|\ker(\alpha)| = |\ker(\alpha_{\text{sep}} \circ \Phi_p^n)|$. But $\ker(\Phi_p)$ and hence $\ker(\Phi_p^n)$ is trivial: $(X^p, Y^p; Z^p) = (0,1;0) \iff (X,Y;Z) = (0,1;0)$. Since all endomorphisms are surjective, Φ_p^n is also bijective. Hence $|\ker(\alpha)| = |\ker(\alpha_{\text{sep}} \circ \Phi_p^n)| = |\ker(\alpha_{\text{sep}})| = \deg(\alpha_{\text{sep}})$. □

Corollary 3.36. *All degree 1 isogenies are group isomorphisms.*

Proof. $1 = \deg(\alpha) \geq |\ker(\alpha)|$, hence the kernel is trivial and so α is injective. Since isogenies are also naturally surjective, all degree 1 isogenies must be group isomorphisms. □

Proposition 3.37 (Dual isogeny)**.** *Let $\alpha : E_1(\overline{\mathbb{K}}) \to E_2(\overline{\mathbb{K}})$ be a degree n isogeny. Then there exists a unique degree n isogeny $\hat{\alpha} : E_2(\overline{\mathbb{K}}) \to E_1(\overline{\mathbb{K}})$ such that $\alpha \circ \hat{\alpha} = [n]_{E_1}$ is the multiplication-by-n map on $E_1(\overline{\mathbb{K}})$, and $\hat{\alpha} \circ \alpha = [n]_{E_2}$ is the same map on $E_2(\overline{\mathbb{K}})$. We call $\hat{\alpha}$ the dual isogeny of α.*

Proof Idea. See Theorem 12.14 [62], III.6.1 [55]. The proof relies on constructing Galois extensions, and in turn deducing the existence of such a map. For a heuristic idea of what the dual is/why it should exist, we can turn to group theory. If an isogeny α is separable, then $|\ker \alpha| = \deg(\alpha) = d$, and hence is a d-to-1 map. The goal of the dual $\hat{\alpha}$ is to create a pseudo-inverse for α. But if the kernel is non-trivial (i.e. when $d > 1$), we cannot consider a unique inverse, but rather have a choice of elements in a coset i.e. if $\alpha(P) = Q$, then $\alpha^{-1}(Q) = P \ker \alpha$ (this is straight from the First Isomorphism Theorem). To create *some* map $E_2 \to E_1$, we might just consider the natural sum of elements in that coset: $\hat{\alpha}(Q) = \sum_{R \in P \ker \alpha} R = \sum_{k \in \ker \alpha} P + k = [|\ker \alpha|]P + \sum_{k \in \ker \alpha} k$. This gives a homomorphism, and since every element differs by this constant sum over the kernel, we can ignore it. Since we can choose the representative, this lends itself to the idea we can hopefully define that $(\hat{\alpha} \circ \alpha)(P) := [\deg \alpha]P$. In this way, the dual can be thought of as a type of pseudoinverse that loses information of $\ker \alpha$. Of course, this is a loose sketch given that this "sum over coset representatives" is non-unique for two representatives of the same coset. Many properties we would hope of the dual are true: $\widehat{\alpha_1 \circ \alpha_2} = \hat{\alpha}_1 \circ \hat{\alpha}_2$, $\widehat{\alpha_1 + \alpha_2} = \hat{\alpha}_1 + \hat{\alpha}_2$, $\hat{\hat{\alpha}} = \alpha$, etc. □

Proposition 3.38. *The degree map $\deg : \text{End}(E) \to \mathbb{Z}$ is a positive definite quadratic form. Further, the pairing $\langle \phi, \psi \rangle \mapsto \deg(\phi - \psi) - \deg(\phi) - \deg(\psi)$ is bilinear.*

Proof. (III.6.3 [55]) Since the degree of an endomorphism is defined in terms of the degree of polynomials, it is clear that $\deg(\phi) \geq 0$, $\deg(\phi) = 0 \iff \phi = 0$, and $\deg(\phi) = \deg(-\phi)$.

For the pairing, consider the ring injection $\mathbb{Z} \to \text{End}(E)$ via $n \mapsto [n]$. This is an injection since if $[n] = [0]$, $[n]$ is non-constant for $n > 1$ and $\deg([n]) \geq \ker([n])$, and hence the kernel cannot be all of $E(\overline{\mathbb{K}})$ (i.e. $[n]$ cannot be the 0 map). So it must be that $n = 0$ and the map is injective. Hence, and taking advantage of the dual isogeny,

$$[\langle \phi, \psi \rangle] = [\deg(\phi - \psi)] - [\deg(\phi)] - [\deg(\psi)] = \widehat{(\phi - \psi)} \circ (\phi - \psi) - \hat{\phi} \circ \phi - \hat{\psi} \circ \psi = -\hat{\phi} \circ \psi - \hat{\psi} \circ \phi$$

which is linear in both ϕ, ψ by linearity of the dual and composition. We conclude with the ring injection. □

Proposition 3.39. *Let ϕ, ψ be endomorphisms of E. Then $|\deg(\phi-\psi)-\deg(\phi)-\deg(\psi)| \leq 2\sqrt{\deg(\phi)\deg(\psi)}$.*

Proof. (V.1.2 [55]) Since $\deg(\phi-\psi)-\deg(\phi)-\deg(\psi)$ is a positive definite bilinear form, we can use a version of Cauchy-Schwarz. For $m, n \in \mathbb{Z}$, we have

$$mn\langle\phi,\psi\rangle = \langle m\phi, n\psi\rangle = \deg(m\phi - n\psi) - \deg(m\phi) - \deg(n\psi).$$

Note that $\deg(m\phi) = \deg([m])\deg(\phi) = m^2\deg(\phi)$, and that $\deg(\cdot) \geq 0$ by positive definiteness:

$$0 \leq \deg(m\phi - n\psi) = m^2\deg(\phi) + mn\langle\phi,\psi\rangle + n^2\deg(\psi).$$

If we let $m = -\langle\phi,\psi\rangle$ and $n = 2\deg(\phi)$, we get

$$\begin{aligned} 0 &\leq (-\langle\phi,\psi\rangle)^2\deg(\phi) - \langle\phi,\psi\rangle(2\deg(\phi))\langle\phi,\psi\rangle + (2\deg(\phi))^2\deg(\psi) \\ &= \deg(\phi)\left(4\deg(\phi)\deg(\psi) - \langle\phi,\psi\rangle^2\right) \end{aligned}$$

which is what we wanted. □

Proof of Hasse's bound. Let Φ_q be the Frobenius endomorphism $(x,y) \mapsto (x^q, y^q)$. Since $\mathbb{F}_q^\times$ is a group of order $q-1$, we have for all $P \in E(\mathbb{F}_q)$, $\Phi_q(P) = (x^q, y^q) = (x,y) = P$. That is, $\Phi_q(P) - P = 0$, or $(\Phi_q - 1)(P) = 0$ (with 1 being the identity operator). Therefore $E(\mathbb{F}_q) \subseteq \ker(\Phi_q - 1)$. $\Phi_q - 1$ is an endomorphism, so naturally we have $\ker(\Phi_q - 1) \subseteq E(\overline{\mathbb{F}_q})$. By Proposition 3.24, we can reduce this to $\ker(\Phi_q - 1) \subseteq E(\mathbb{F}_q)$ so $\ker(\Phi_q - 1) = E(\mathbb{F}_q)$. Also, if one computes $(x^q, y^q) \oplus (x, -y)$, they will find that $r'(x) \not\equiv 0$, and so $\Phi_q - 1$ is separable. Then using Theorem 3.32, we have that $N = |E(\mathbb{F}_q)| = |\ker(\Phi_q - 1)| = \deg(\Phi_q - 1)$. By Proposition 3.39:

$$|\deg(\Phi_q - 1) - \deg(\Phi_q) - \deg(1)| \leq 2\sqrt{\deg(\Phi_q)\deg(1)}.$$

Putting it all together, we obtain $|N - (q+1)| \leq 2\sqrt{q}$. □

A useful consequence of this theorem is that it means that it is not hard to find a point $P \in E(\mathbb{F}_q)$, since a decent proportion of elements we expect to be in E actually are in $E(\mathbb{F}_q)$. So if we want our group to be 128-bit secure, we want our group to be (ideally a prime) around 2^{256}, so Hasse's bound says we want to pick a field of order with around 2^{256}. This is the key detail that Hasse's bound demonstrates: we get groups approximately of the size of the field we pick, which implies our group generation is size efficient (i.e. don't need to work in a large field to only get a small group). Often we work in large prime fields $\mathbb{F}_p$ (to avoid complex, polynomial operations) or large binary fields $\mathbb{F}_{2^k}$ for optimized computations (but is less well-studied, and hence has varying levels of trust).

This is only half the the story though; we want the order of this group to be prime, or at the very least have large prime factors. For a while, this was the best we had for having a sense of the size of the group $E(\mathbb{F}_q)$. We could guess and check, or iteratively design curves (which is still fairly common [8]), but it was not until the 1990s we could exactly determine the size of these groups efficiently [51]. Even then, these were not particularly helpful given that we would need to factor these large order groups to find out if they are suitable (i.e. prime/has large prime factors to avoid the Pohlig-Hellman attack). It was not until only recently have we been able to design elliptic curves with precise orders in mind [13].

Not to mention, we still have not picked a generating point $P \in E(\mathbb{F}_q)$ in the case of non-prime groups. Finding those can be just as tedious, having to pick random points and testing its order (i.e. with Lagrange's theorem). Despite these issues, elliptic curves seem (and as has been shown) promising in providing a basis for cryptography. For even if all of these "issues" seem like problems, having the range of choices in parameters (i.e. the field and the curve), we can often design curves to lessen the seemingly big problem of having a suitable group. And if elliptic curves are really cryptographically viable, we can do the one-time computation of finding a field, curve, and generating point and call it a day.

3.4 The Nechaev-Shoup Theorem and Confidence in ECDLP

While it is true that we have not found any better ways to solve ECDLP than in $O(\sqrt{n})$, it does not give a solid reason for why we should use ECDLP for the foreseeable future. After all, $\mathbb{Z}_p^\times$ fell victim to index calculus, who is not to say there is not another efficient algorithm on the horizon? No one can say that there is not such an algorithm, but the inherent lack of structure in $E(\mathbb{F}_q)$ would suggest that perhaps it is protected from future attacks; there is no underlying structure like prime factorization to exploit. But that only suggests there are no elliptic-curve group specific attacks we have to worry about. Could there be better attacks that work for *any* group, like Baby-Step Giant-Step?

The answer turns out to be *no*. The Nechaev-Shoup Theorem states that any non-group specific discrete logarithm solving algorithm must run in *at least* $O(\sqrt{n})$ [42][53]. The idea is that with only $g, h = g^x$ given, we can learn about the group by *only* iteratively constructing new group elements. But the new elements will only be of the form $g^\alpha h^\beta = g^{\alpha+\beta x}$ for some integers α, β. So if a choice of α_1, β_1 creates an element we have seen before, say for α_2, β_2, then we have learned something about the secret discrete logarithm x via the relation $\alpha_1 + \beta_1 x = \alpha_2 + \beta_2 x$. This does not explicitly determine x, but it is new information regarding relations between x. If we use m operations constructing possibly m total expressions, we get a total of $O(m^2)$ possible pairs of expressions that might give information about x. The full formalization and a proof sketch can be found in the Appendix.

Theorem 3.40 (Nechaev-Shoup)**.** *Let G be a group of prime order p and $\mathcal{A}$ be a probabilistic algorithm. If $\mathcal{A}$ makes m mathematical operations via a black box oracle, then the probability that $\mathcal{A}$ solves the DLP is*

$$\mathbb{P}\left(\mathcal{A}(p, g, g^x) = x\right) < \frac{(m+2)^2}{2p} + \frac{1}{p}$$

Corollary 3.41. *Any generic group DLP solving algorithm runs in $\Omega(\sqrt{p})$.*

Proof. We have an upper bound on the success probability of any such algorithm $\mathcal{A}$. If we want $\mathcal{A}$ to succeed with probability at least, say, $1/2$ (or any number bounded away from 0), we need at least $\frac{1}{2} < \frac{(m+2)^2}{2p} + \frac{1}{p}$. Rearranging, we get $\sqrt{p-2} - 2 < m$. Since m denotes the number of operations/requests we made, if we want bounded-away-from-0 probability of success, we need at least $\Omega(\sqrt{p})$ operations. □

So we do not have much to worry about at the moment with respect to the discrete logarithm problem; in some sense, it is *provably* hard that its hard to solve for all groups at once, and instills some confidence in elliptic curve groups. With their complicated formulas for addition alongside the difficulty in pinning down the size of the groups, understanding the structure of elliptic curves is far from easy. This "black box" structure that obscures our understanding allow us, for the time being, to treat them as a generic group and hope that DLP remains hard within them.

3.5 Security and Implementation Analysis

Despite ECDLP being a very strong candidate for cryptographic applications, they are not without flaw. Just as we reduced DLP to need to be in prime order, or more efficiently attack DLP in $\mathbb{Z}_p^\times$ exploiting its structure, there are similar considerations that if not accounted for can leave us with similar vulnerabilities. Here are a couple of considerations.

3.5.1 Anomalous Curves

We have been heralding prime order groups and the use of prime order fields in elliptic curves as optimal, but if it happens to line up such that $|E(\mathbb{F}_p)| = p$ (with p prime), then turns out our protocols break. These are called *anomalous curves*, and were found to be vulnerable in 1999 [57]. As is the idea in much of algebra, we can simplify DLP by considering a lift to a bigger field with more tools at our disposal. By considering our elliptic curve over $\mathbb{Q}_p$, the p-adic numbers, we gain access to a formal logarithm that directly solves DLP with simple arithmetic and division. This only works for anomalous curves since the orders of the group and field being equal makes it such that multiplication-by-p behaves nice enough to give the isomorphisms necessary to use this easy logarithm formula [57][35].

3.5.2 Supersingular Curves

Say we are working with an elliptic curve over a field $\mathbb{K}$ of characteristic p. Consider the multiplication-by-p map $[p]$ on our specific elliptic curve. We say a curve $E/\mathbb{K}$ is *supersingular* if $\ker([p]) = \{\infty\}$ is trivial (note: this is a separate notion of the singularity of a curve). There are equivalent notions of supersingularity [55], and this additional structure the curve inherits from this property lend itself to new vulnerabilities [38] via the Menezes-Okamoto-Vanstone attack. The MOV attack ultimately works by transferring the problem of ECDLP in $E(\mathbb{F}_q)$ to DLP in an extension of the ground field $\mathbb{F}_{q^k}$ (we have seen via index calculus that DLP is easy for prime p in $\mathbb{F}_p^\times \cong \mathbb{Z}_p^\times$; similar fast index calculus methods exist for any finite field [31]). The specific isomorphism that translates the problem is given by a pairing function from $\ker([p]) \times \ker([p]) \to \overline{\mathbb{F}_q}$, and the supersingularity of E makes this field extension $\mathbb{F}_{q^k}$ relatively small (i.e. $k \leq 6$), making the DLP in the finite field still tractable (relative to the size of the original field $\mathbb{F}_q$).

3.5.3 Invalid Point Attacks

This is less of a concern of specific curves, but more so implementation of these protocols. Recall the doubling formula $2P = (x_P, y_P) \oplus (x_P, y_P)$ on an elliptic curve $y^2 = x^3 + ax + b$ we found earlier:

$$x_{2P} = \left(\frac{3x_P^2 + a}{2y_P}\right)^2 - 2x_P, \quad y_{2P} = -\left(\frac{3x_P^2 + a}{2y_P}(x_R - x_P) + y_P\right)$$

Note how this only depends on the choice of point P and a. Similarly, the general point addition formula does not depend on either a, b given we know the two points we are adding. So the general formula for nP only depends on a and P. So for *any* value b', the value nP is calculated the same on the elliptic curve $y^2 = x^3 + ax + b'$. This is where the danger lies. If an attacker has access to the result nP during a protocol, they can engineer a curve E' with a base point P' that specifically has small order (say $kP = \infty$), and falsely send P' to compare the result nP' to deduce information of $n \bmod k$. Hence it is important to somehow validate that the point received to calculate nP is on the desired curve and not a malicious attempt to gain information.

3.5.4 Comparison to Other Classic Cryptographic Schemes

What makes elliptic-curve cryptography so appealing is that no one has been able to break the elliptic-curve discrete logarithm problem any more efficiently than $O(\sqrt{n})$. This is unlike DLP in $\mathbb{Z}_p^\times$ and integer factorization for RSA, where index calculus and number field sieves have greatly reduced their efficacy. However, security is not the only important detail. Different operations and algorithms offer different implementations and hence different speeds.

Table 2: Comparison between Classic Cryptographic Schemes*

n-Bit Security	RSA	Discrete Logarithm	
	Length of N	Order-q Subgroup of $\mathbb{Z}_p^\times$	Elliptic-Curve Group Order q
112	2048	$p = 2048,\ q = 224$	224
128	3072	$p = 3072,\ q = 256$	256
192	7680	$p = 7680,\ q = 384$	384
256	15360	$p = 15360,\ q = 512$	512

*Cryptographic performance compared in terms of bit-lengths of their public keys.

For ECC, say we are working in an n-bit order field $\mathbb{F}_q$ (i.e. $n = \log q$). For an n-bit random integer k, we want to know the time complexity of computing the public key kP. Calculating kP in an elliptic curve requires $O(n)$ "double-and-add"-ings that involve a number of field operations given by our addition formulas (see Appendix). Our formulas are dominated by the cost of multiplication and division, which is $O(n^2)$. So the total cost of computing the public point kP is dominated by $O(n^3)$ operations.

For RSA, say we are working with an n-bit private key $N = pq$. We need to find two primes p, q of of $n/2$ bits. Using something like the Miller-Rabin Test (see *ASO: Number Theory*) has complexity of $O((n/2)^4)$.

Computing the product $N = pq$ and $\varphi(N) = (p-1)(q-1)$ is also bounded by $O((n/2)^2)$. The public exponent e is usually fixed, and computing $d \equiv e^{-1} \bmod N$ can be done with the extended Euclidean algorithm in $O(n^2)$. Hence overall is dominated by $O(n^4)$ in the prime generation.

So not only does RSA require larger keys than ECC for equivalent security, it is still slower to generate public keys even assuming equal size keys/security parameters. ECC gets a bit of an edge allowing design of the base point P alongside the elliptic curve in the public parameters (with random point generation, we would expect it to be about exponential as the probability of picking a random point is $|E(\mathbb{F}_q)|/|(\mathbb{F}_q)^2| \approx 1/q = 1/2^n$). In practice, ECC's speedups in both key generation and encryption/decryption (i.e. with signatures) become more and more apparent at bigger security levels [36].

There are practical differences, however. For one, elliptic curves and their groups are much more geometric than algebraic. Unlike RSA that just operates with numbers that can naturally encode letters, words, and messages, encoding text with arbitrary points on a curve seems far less intuitive. As such, elliptic-curve algorithms have found themselves in use for more primitive applications, like key exchanges and digital signature algorithms as shown above. But nevertheless, they remain a core part of cryptography due to their relative efficiency at achieving n-bit security for less memory.

4 Introduction to Quantum Computing

So far, our discussion has been centered around elliptic curves and algorithms to use them to protect and attack secrets. However, our analysis has centered around classical computers. Look back at Theorem 3.45, and our analysis that a general algorithm requires at least $O(\sqrt{n})$ operations to solve the discrete logarithm problem. The key to the proof is that for m operations, we get at most m^2 possible comparisons to get information; we have to take m queries to the oracle to update a running list $\mathcal{L}_{\text{elem}}$, as we *sequentially* compute elements. This is the natural interpretation for computers, as that is also how *we* perform tasks: one step at a time. Quantum computers challenge this. The following is an inspired interpretation of [48] [1] [9].

4.1 What is Classical Computing?

Classical computers arose as a means to formalize *computation*—the mathematical symbols and processes according to well-defined rules to solve problems and determine solutions. The big question, though, is how can we represent these algorithmic (and often numerical) operations? This notion of computability and decidability of mathematical problems culminated in the 1930s to the idea of a Turing machine: a hypothetical machine that can carry out any computation given enough time. The critical detail in the Turing machine, that later influenced the design of electronic computers, is that we store information as binary digits (or better known as *bits*): cells where data is stored can either be a 1 or a 0. These well-defined, clear-cut binary states naturally force computers to complete tasks sequentially to update the memory states.

4.2 Key Differences of Quantum and Classical Computing

With the advent and major paradigm shift of quantum mechanics in the early 1900s, understanding the new, non-intuitive phenomena was at the forefront of this new field. Thought experiments that have now pervaded pop culture, like Shcrödinger's cat—where a cat can be both dead *and* alive at the same time—were significant hurdles of the time. When computers continued to be developed and become more powerful, modelling classical physics became a matter of precision and available memory. Quantum mechanics was less friendly to current technology, requiring exponentially higher overhead to simulate. So in the 1980s, it was proposed that perhaps we need to leverage quantum phenomena to model quantum phenomena. These motivations in tandem with previously proposed ideas led to the first ideas of the modern quantum computer.

The most fundamental difference between classical and quantum computers is how information is stored. As mentioned, **standard bits** can occupy one of two definite **states**, either 0 or 1. Quantum bits, or **qubits**, likewise can *measure* to two states $|0\rangle$ or $|1\rangle$ (like how the cat is either dead or alive once the box is opened). But before measurement, qubits can be in **superposition**: an intermediate state that is neither quite 0 nor

quite 1. A way to think about it this as we will see is that instead of working with just 2 states as with classical computers, quantum computers operate in a 2-dimensional vector space of states with a basis of our typical 2 states; instead of just flipping a coin heads or tails, we now allow ourselves infinitely many edge flips as well. We will see that allowing this range of in-between states, and specifically allowing for complex parameters follows from a generalization of probability. These superpositions is where lies the potential to surpass previous computing limits. We outline these motivations below.

4.2.1 Brief Aside on Qubits (Mathematical Approach)

Quantum computing can be derived from the physics approach in light of all of the modelling and machinery that was developed in response to the apparent paradoxes of the science. However, for our purposes, we can derive a notion of quantum computation and qubits purely from a generalization of probability.

Consider a probabilistic classical bit, that is 0 with probability α and 1 with probability β. What can we say about these parameters is that $\alpha, \beta \geq 0$ and $\alpha + \beta = 1$. Writing these as a probability vector $\vec{p} = (\alpha\ \beta)$, the latter assumption can be written with the L^1-norm $\|\vec{p}\|_1 = 1$. These are two of the axioms of probability. We use these since they are intuitive. Probabilities represent a proportion of occurrences, or confidence, and treating these as linear gives consistent behavior and simple math. Operators that map probability vectors to probability vectors are *stochastic matrices* with rows that sum to 1.

But these do not limit us theoretically: what if instead we used the Euclidean norm and required that $\|\vec{p}\|_2 = \alpha^2 + \beta^2 = 1$? We still want the parameters $(\alpha\ \beta)$ to still correspond to some probability of the bit outputting 0 or outputting 1, so we simply just let it do so with probability α^2 and β^2 respectively. Unitary matrices take the place of stochastic matrices to map these normalized vectors (as they preserve norms; in fact, this is the *only* other choice of norm; any other norm does not have a consistent set of transformations; pg. 118 [1]). This choice is the true difference with quantum computing: not only do we allow ourselves to these "in between" states, but the probabilities are governed by this Euclidean norm and thus we can have two distinct qubits that have the same output probability distribution.

This 2-norm bit $(\alpha\ \beta)$ is our *qubit*. Instead of writing $(\alpha\ \beta)$, we write $\alpha\left|0\right\rangle + \beta\left|1\right\rangle$ (think of $\left|0\right\rangle, \left|1\right\rangle$ as basis vectors with this notation). When $\alpha, \beta > 0$ then we say the qubit is in *superposition*. Immediately we start to see some of the strangeness of qubits appear. Consider the unitary matrix

$$H = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}.$$

This is a "randomizing operator", called the **Hadamard transformation/gate**. Consider applying this to the deterministic qubit $\left|0\right\rangle$ that outputs 0 with probability 1.

$$H\left|0\right\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \frac{1}{\sqrt{2}}\left|0\right\rangle + \frac{1}{\sqrt{2}}\left|1\right\rangle.$$

H makes our qubit go from deterministic to in superposition, half the time now outputting $\left|0\right\rangle$ and the other half $\left|1\right\rangle$. If we apply H again:

$$H^2\left|0\right\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \left|1\right\rangle$$

Randomizing a random qubit outputted a deterministic qubit! For a qubit $\alpha\left|0\right\rangle + \beta\left|1\right\rangle$, while our probabilities are in terms of squared terms α^2, β^2, our parameters are not and can hence cancel each other out across these transformations. This physically manifests in *deconstructive interference*; in the context of waves, adding the *amplitudes* of two waves pointwise will have peaks and valleys cancel out. If we think of H as inducing a coin flip, applying H once can make our qubit either $\left|0\right\rangle$ or $\left|1\right\rangle$. In either of those states, applying H again causes another coin flip. So we have two paths to end up at $\left|0\right\rangle$ after two flips: $\left|0\right\rangle \to \left|0\right\rangle \to \left|0\right\rangle$ and $\left|0\right\rangle \to \left|1\right\rangle \to \left|0\right\rangle$. But the latter path has negative probability amplitude and cancels out with the former path with positive amplitude. So at the end of either path, the only positive probability outcome is $\left|1\right\rangle$.

Our qubit is not quite fully correct. For example, consider for the operator $A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$. Ideally, we could break this up into intermediate steps, i.e. there should be a B such that, say, $B^2 = A$. We want a type of continuity among operators. To make this possible, we let our parameters be complex (since it is algebraically closed). This is easy to amend our 2-norm condition with the complex variant: we require $|\alpha|^2 + |\beta|^2 = 1$. Complex parameters also ensure a consistent uniformity in states: if we pick a random uniform (specifically with respect to the Haar measure) qubit $\alpha |0\rangle + \beta |1\rangle$, the corresponding probability vector $(|\alpha|^2 \ \ |\beta|^2)$ will also be uniformly distributed.

Definition 4.1 (Qubit)**.** A *qubit* is a system described by parameters $\alpha, \beta \in \mathbb{C}$ such that $|\alpha|^2 + |\beta|^2 = 1$ that outputs 0 or 1 with probability $|\alpha|^2$ or $|\beta|^2$ respectively. A set of qubits are stored in memory in groups with *registers* (similar to bits being grouped in bytes).

Mathematically, a qubit is a vector $|v\rangle = (\alpha \ \beta) = \alpha |0\rangle + \beta |1\rangle$ in a 2-dimensional complex inner product space with orthonormal basis $\{|0\rangle, |1\rangle\}$. If $\alpha, \beta \neq 0$, we say v is in *superposition*. Upon *measuring* a qubit v, it returns $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$.

We can combine qubits exactly as one would imagine with classical bits. In a classical system, if we know one bit has probability p_1 to output 0 and another bit has (independent) probability p_2 to output 0, then we would expect to see the string 00 with probability $p_1 p_2$. Qubits work similarly. Suppose we have a two-qubit system, with the first qubit having state $\alpha_1 |0\rangle + \beta_1 |1\rangle$, and the second one being $\alpha_2 |0\rangle + \beta_2 |1\rangle$, then the two-qubit system is the tensor product

$$(\alpha_1 |0\rangle + \beta_1 |1\rangle) \otimes (\alpha_2 |0\rangle + \beta_2 |1\rangle) = \alpha_1\alpha_2 |00\rangle + \alpha_1\beta_2 |01\rangle + \beta_1\alpha_2 |10\rangle + \beta_1\beta_2 |11\rangle .$$

This tensor product represents having a 2-qubit state with one qubit concatenated with another, similar to our 2-bit states in the classical sense with one bit next to another. We will suppress the $\otimes$ often and just write $|\phi\rangle |\psi\rangle = |\phi\rangle \otimes |\psi\rangle$. Also, for simplicity, we will combine the deterministic qubits together i.e. $|0\rangle |1\rangle = |01\rangle$. Not all two-(or more) qubit systems can be written as a tensor product. The *Bell/EPR state*

$$\frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |11\rangle$$

is one such state. These are *entangled* qubits (the second qubit is inherently correlated to the first).

With all this spelled out, though, how are qubits *fundamentally* different from a probabilistic classical bit that we assign probabilities $|\alpha|^2, |\beta|^2$ to its outcomes? The key lies in superpositions. They may seem to describe a probabilistic classic (set of) bit(s), but since parameters are complex amplitudes, we can leverage interference as we have seen above to try and "encourage" higher amplitudes and thus higher probabilities of "desirable" outcomes while destructively interfering and cancelling the less desirable outcomes. This ability to reinforce and discourage the set of outcomes all at once is what makes quantum computing better than classical algorithms in some cases.

Remark 4.2. For completeness sake, all of this can also be rephrased in terms of a complex Hilbert space equipped with the complex dot product. Our qubits are then elements of norm 1, with output states $\{|0\rangle, |1\rangle\}$ (and other combinations if needed with an orthonormal basis of $\{|0\rangle, |1\rangle, \cdots, |N-1\rangle\}$).

4.2.2 So What? Quantum Parallelism is What

So, what is the point of all this? Say we have a function $f : \{0,1\} \to \{0,1\}$. We will consider the map on two qubits given by $U_f : |x\rangle |y\rangle \to |x\rangle |y \oplus f(x)\rangle$ where $\oplus$ is XOR. This preserves the inner product between basis states $\{|0\rangle |0\rangle, |0\rangle |1\rangle, |1\rangle |0\rangle, |1\rangle |1\rangle\}$, so is unitary. What makes quantum computing interesting is how these maps act on *non-basis states*. Now, suppose we do the following set of operations starting on the quantum state $|0\rangle |0\rangle$:

1. Apply the Hadamard map to the first qubit to create a uniform superposition: $H |0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

2. Apply unitary U_f to the new combined state, noting U_f is linear:

$$U_f(H|0\rangle \otimes |0\rangle) = U_f\left(\frac{1}{\sqrt{2}}\Big(|0\rangle|0\rangle + |1\rangle|0\rangle\Big)\right) = \frac{1}{\sqrt{2}}\Big(|0\rangle|f(0)\rangle + |1\rangle|f(1)\rangle\Big)$$

Look what has happened. We applied U_f, and hence f, only once, yet we were able to put our two-qubit system into a state that stores information about both $f(0)$ *and* $f(1)$. It is as if we calculated two values of f with only one use of f. This is the illustrious *quantum parallelism* that makes quantum computing so promising for algorithmic speedups. However, it should be noted that we are still working with probabilistic superpositions. If we try to extract this data and measure the state, we get the collapsed state of either just $|0\rangle|f(0)\rangle$ or $|1\rangle|f(1)\rangle$, a single value of f which any classical computer could have done. What we aim to do later is exploit constructive and destructive interference to make this extraction useful.

Remark 4.3 (Partial measurement)**.** It is worth noting we allow ourselves to *partial measurement* of qubits. In the above case, we measured the *whole system* and every qubit to get either $|0\rangle|f(0)\rangle$ or $|1\rangle|f(1)\rangle$. Just like in classical computing, we may just want to look at 1 qubit in memory instead of everything. We do this since sometimes as partial measurement will only determine the qubits we measure, while possibly leaving the others in superposition, allowing us to work with the rest as quantum states. Consider the following 2-qubit state:

$$\frac{1}{\sqrt{4}}|00\rangle + \frac{1}{\sqrt{4}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle = \frac{1}{\sqrt{2}}|0\rangle \otimes \left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right) + \frac{1}{\sqrt{2}}|1\rangle|0\rangle$$

If we measure the first qubit, and get $|1\rangle$, we know the second qubit must be $|0\rangle$. If we instead got $|0\rangle$ from the first qubit, then we know the second qubit can still be either $|0\rangle$ or $|1\rangle$ with equal probability, so is still in superposition. This is the key difference between partial and total measurement.

4.3 A Note On Quantum Time Complexity and Security

A key detail we need to address is how time complexity works in the new computing setting. Notably, since quantum computing is inherently probabilistic, we need some way to formalize a computing class that takes into account non-deterministic outcomes for deterministic problems. The efficient class of algorithms in this setting are the **bounded-error quantum polynomial time (BQP)** algorithms [43][1]. This is similar to the classical computing class of **bounded-error probabilistic polynomial (BPP) time**, which clearly contains the class of polynomial time algorithms we were previously considering. Also, because of the nature of superpositions, we will see that we can compute a lot in parallel on a quantum computer. Thus we need a new metric for what an "operation" is in the quantum setting. These operations are typically defined via *elementary quantum gates*, which as we have discussed are represented as unitary matrices. There are universal sets of gates in which any other gate/unitary operation can be approximated (Section 4.5 [43]; in the same vain from the `NOT` and `AND` classic logic gates, we can build any other logic gate). There are also some special gates that are exceptionally expensive that we take extra care of [44]. For the purposes of this survey, we will largely ignore the finer details in our complexity analysis, for we mostly want to show how quantum algorithms can achieve polynomial time in the cases where classical ones have not yet.

We also need to adapt security thresholds, since quantum operations are not equivalent to the classical ones. The notion of n-bit security was built off an idea of essentially relating the best attacks to an equivalent brute force attack of the simpler one-time pad cipher. Unlike classical computers, however, quantum computing hardware is extremely variable with too many factors that affect viability of any given algorithm's performance. This poses a major hurdle if we wish to standardize the security offered by quantum algorithms in terms of the number of quantum operations or the quantum circuit size since implementation rarely will mimic the theory. Not to mention, many attacks may use hybrid approaches that bridge classical and quantum computers, further making such a security metric hard. NIST has set some "levels" of security standards in terms of equal difficulty as breaking typical symmetric ciphers in both the classical and quantum settings [39][41]. Roughly, n-bit quantum security translates to requiring *both* n-bit classical and $(n/2)$-qubit security (as Grover's quantum search algorithm offers a quadratic speedup for breaking symmetric ciphers).

4.4 Is This Viable?

We have described qubits and their unitary transformations as a generalization of probability, but the theory is not helpful beyond a curiosity if we cannot implement it in some way. Although this was an abstract motivation, these notions were originally formulated in the context of the first quantum experiments; superposition, unitary evolution, entanglement, etc. all have a physical origin. We implement classical bits via transistors and their corresponding bit as the occupying voltage. We can implement qubits in the same vein as their phenomena was discovered. We can take a photon (a la double slit experiment) and say it is in state $|0\rangle$ if it is vertically polarized or $|1\rangle$ if it is horizontally polarized. Using polarized filters (like those found in sunglasses), we can put photons into superposition by making them aligned along another axis. There are other implementations, and the viability will vary, but all are relatively difficult due to the nature of measuring qubits accidentally, and forcing superpositions to collapse unexpectedly.

5 Quantum Computing Threat

5.1 Shor's Algorithm

The main quantum algorithm of concern is **Shor's algorithm**. This is an algorithm that can solve not just the discrete-logarithm problem, but also factoring and other important problems in cryptography (see Section 5.5).

As an example, let us explore how Shor's algorithm attempts to solve factoring (this is much simpler to re-derive Shor's algorithm than with DLP as our working example as we will see later; also integers being the objects will concretize the abstract ones in Section 5.5 when we generalize). The idea is that we can calculate $\gcd(x, N) = d$ quickly, so if we can find a suitable x that makes $d > 1$, we will have found a non-trivial factor of N; $\gcd(x, N)$ acts as a filter for factors of N. Assuming we have a reasonably "hard" number N to factor (i.e. is not even or prime power), we pick a random integer $2 \leq n < N$. We can quickly calculate $\gcd(a, N) = d$. If $d > 1$, then $\{d, N/d\}$ are factors of N. If $d = 1$, then a is in the group $\mathbb{Z}_N^\times$ and hence has multiplicative order $a^{o(a)} = 1 \bmod N$ i.e. $N \mid a^{o(a)} - 1$. If $o(a)$ is even, then we can use the difference of squares $N \mid (a^{o(a)/2} - 1)(a^{o(a)/2} + 1)$. Since the order is $o(a)$, $N \nmid a^{o(a)/2} - 1$. If $d = \gcd(N, a^{o(a)/2} - 1) > 1$, we have found new non-trivial factors $\{d, N/d\}$ of N. If $d = 1$, then $\gcd(N, a^{o(a)/2} - 1) = 1$ and $\gcd(N, a^{o(a)/2} + 1) = N$ so we learn no information about the factors of N and must try again with a new a.

The important step here is determining $o(a)$. Doing this classically is not known to be easy—it requires either factoring the group size $\varphi(n)$, or just trying every possible value. The trick that quantum algorithms take advantage of is considering the function $f(x) = a^x \bmod N$. $f(x)$ is periodic with period $o(a)$. If we can find this period, we have found $o(a)$.

5.1.1 Why Do We Need Quantum At All?

Building on the last comment, it is worth pointing out that there is a natural suggestion: repeated trials of Shor's algorithm involves picking random a and calculating $\gcd(a, N) = d$. The algorithm continues as described if $d = 1$, but instantly terminates if $d > 1$ giving us a nontrivial factor. So why not just skip the quantum algorithm, and only calculate this gcd, sampling random a every time? Well consider the hardest possible numbers to factor: products of two primes $N = pq$. Our random $\gcd(a, N)$ only succeeds if a is not coprime to N. The numbers less than N that are not coprime to N are precisely the multiples of p and q less than N (since they are prime) i.e. $\{1p, 2p, \cdots, (q-1)p\}$ and $\{1q, 2q, \cdots, (p-1)q\}$ (also note these are disjoint; any number that is a both multiple of p and of q must be a multiple of pq). So the probability of picking a viable a is $\frac{(p-1)+(q-1)}{N-1} \approx 1/p + 1/q$. In RSA, we want $N \approx 2^{2048}$ at least, so we would have $p, q \approx 2^{1024}$. So the probability of picking a viable a is around $1/2^{256} \approx 1/10^{77}$. For reference for how small this is, the probability of picking any particular *atom* in the entire universe is about $1/10^{80}$.

This explains why trying $\gcd(a, N)$ directly is bad, but not necessarily why $\gcd(a^{o(a)/2} - 1, N)$ is good. This follows from two observations. First, if $x^2 \equiv 1 \bmod N$ but $x \not\equiv \pm 1 \bmod N$, then we have $N \mid x^2 - 1 = (x-1)(x+1)$ and $N \nmid x \pm 1$. Hence the factors of N are split across $x \pm 1$ with neither containing N entirely

and so $1 < \gcd(x \pm 1, N) < N$, each giving a non-trivial factor of N. Second, suppose we are trying to factor $N = p_1^{e_1} \cdots p_k^{e_k}$. Pick uniform random $a \in \mathbb{Z}_N^\times$ i.e. a coprime to N. Then $\mathbb{P}(o(a)$ is even and $a^{o(a)/2} \not\equiv -1 \bmod N) \geq 1 - 1/2^k$ (see Section 5.3.2 of [43]). This can be shown considering the Chinese Remainder Theorem. Combining these two theorems shows that considering $o(a)$ gives a high chance of finding non-trivial factors of N.

5.2 The Quantum Fourier Transform

The problem of finding periods inspires the use of something along the lines of a *Fourier transform*. The **discrete Fourier transform (DFT)** of a vector of complex numbers $\mathbf{X} = \{x_0, x_1, \cdots, x_{N-1}\}$ is another vector $\mathbf{Y} = \{y_0, y_1, \cdots, y_{N-1}\}$ of equal length with components

$$y_k = \sum_{n=0}^{N-1} x_n \cdot e^{2\pi i k \frac{n}{N}}$$

The idea of the DFT is to see given a period of N/k (or a frequency of k/N), how many times the data aligns with itself (with the $k = 0$ term encoding the mean of the vector). Each y_k tells us the amplitude of the component of phase N/k. The more the data aligns with itself every N/k elements, the greater the magnitude of y_k. This naïve implementation requires N additions per coordinate, so we can compute the DFT in $O(N^2)$. In particular, if $N = 2^n$ (i.e. if we are working with coordinates of n-bits across all n-bit strings), DFT runs in $O(2^{2n})$.

The **quantum Fourier transform (QFT)** is exactly the same, only rewritten to fit the vector notation convention; the DFT is written in terms of coordinates, while the QFT is written in terms of basis vectors. On an orthonormal basis of $\{|0\rangle, |1\rangle, \cdots, |N-1\rangle\}$, the QFT acts on the basis vectors as

$$|n\rangle \xrightarrow{\texttt{QFT}} \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i k \frac{n}{N}} |k\rangle$$

The $\frac{1}{\sqrt{N}}$ term is just a normalizing term so the output is also a quantum state vector. Hence by linearity, the QFT acts on a general quantum state via

$$\sum_{k=0}^{N-1} \alpha_k |k\rangle \xrightarrow{\texttt{QFT}} \sum_{k=0}^{N-1} \beta_k |k\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \left(\sum_{n=0}^{N-1} \alpha_n \cdot e^{2\pi i k \frac{n}{N}} \right) |k\rangle$$

In the QFT, the coefficient of the basis vector $|k\rangle$ encodes the DFT of the data for cycles of period N/k.

Proposition 5.1. *The QFT is a unitary operation.*

Proof. Note by the above formula, we can write the QFT as a matrix $U = (u_{ij})$. Let $\omega = e^{2\pi i/N}$ be the primitive N-th root of unity, so $u_{ij} = \frac{1}{\sqrt{N}} \omega^{(i-1)(j-1)}$. Consider the product $UU^* = (a_{j_1 j_2})$. Then

$$a_{j_1 j_2} = \sum_{k=0}^{N-1} \frac{1}{\sqrt{N}} \omega^{j_1 \cdot k} \overline{\frac{1}{\sqrt{N}} \omega^{j_2 \cdot k}} = \frac{1}{N} \sum_{k=0}^{N-1} \left(e^{2\pi i/N} \right)^{j_1 \cdot k} \left(e^{-2\pi i/N} \right)^{j_2 \cdot k} = \frac{1}{N} \sum_{k=0}^{N-1} e^{2\pi i k (j_1 - j_2)/N} = \frac{1}{N} \sum_{k=0}^{N-1} \omega^{k(j_1 - j_2)}$$

When $j_1 = j_2$, we see $a_{j_1 j_2} = \frac{1}{N} N = 1$. When $j_1 \neq j_2$, the sum on the right is a geometric series.

$$\frac{1}{N} \sum_{k=0}^{N-1} \omega^{k(j_1 - j_2)} = \frac{1}{N} \cdot \frac{1 - (\omega^{j_1 - j_2})^N}{1 - \omega^{j_1 - j_2}} = \frac{1}{N} \cdot \frac{1 - (\omega^N)^{j_1 - j_2}}{1 - \omega^{j_1 - j_2}} = 0$$

The last equality follows since $\omega^N = 1$ is an N-th root of unity. Hence $UU^* = I$, and similarly $U^*U = I$. $\square$

So we are free to apply the QFT on our state vectors without worry. When we are working with n qubits, we often consider the 2^n dimensional space with orthonormal basis $\{|x\rangle : x \in \{0,1\}^n\} = \{|0\rangle, \cdots, |2^n - 1\rangle\}$.

Also, note with some algebra, we can rewrite the QFT in the following product form (Equation 5.4 [43]). Let $\texttt{bin}(k) = b_1 b_2 \cdots b_n$ be written in n bits. Let $0.a_1 a_2 \cdots a_n = a_1/2 + a_2/4 + \cdots + a_n/2^n$ represent the fractional part of a binary number. Then

$$\texttt{QFT}(|k\rangle) = \texttt{QFT}(|b_1 \cdots b_n\rangle) = \frac{1}{2^{n/2}}\Big(\big(|0\rangle + e^{2\pi i 0.b_n} |1\rangle\big) \big(|0\rangle + e^{2\pi i 0.b_{n-1}b_n} |1\rangle\big) \cdots \big(|0\rangle + e^{2\pi i 0.b_1 \cdots b_{n-1}b_n} |1\rangle\big) \Big)$$

Interpreted classically, this means for each of our 2^n coordinates we need to do n multiplications. So the DFT can run in $O(n2^n) = O(N \log N)$ on a classic computer—a significant improvement over $O(N^2)$ (while not necessarily in this form, this has equivalent runtime to the classical *Fast Fourier Transform*, or FFT).

Although we have described the DFT and QFT as equivalent, the major benefit of the QFT is being able to transform superpositions all at once. The DFT requires us to compute each coordinate individually, but by the nature of qubits and superpositions, we do not have to track each coordinate individually. The physical system of a qubit inhabits a superposition that naturally tracks all the possible basis states and hence DFT outputs at once; the superposition tracks the current output vector. This is seen most by our product form: we create n superpositions (1 for each qubit. This is easy; i.e. with Hadamard matrix), and then for the first qubit we need 1 phase-correcting operations to get the factor of $0.b_n$. We need two for the second qubit to get $0.b_{n-1}b_n$, etc. So we need a total of $n(n+1)/2$ phase corrections, and that is it! We need $O(n^2) = O((\log N)^2)$ operations: a significant improvement over both the normal DFT's $O(N^2)$ and the FFT's $O(N \log N)$ (although we only described it for basis states, this time complexity remains for any arbitrary quantum state by linearity and quantum parallelism). These can be realized via a sequence of unitary operations to decompose the QFT that we cannot do in a classical setting (i.e. with typical logic gates/operations).

The downside is in that we cannot extract all the data of the QFT. Upon measurement of our superposition, we only get a qubit collapse with some probability. While the QFT does detect periodicities like the typical DFT, we need to design algorithms around the QFT to make the specific outputs we get useful.

5.3 Phase Estimation

We are not quite in a position to use the QFT for our order finding problem. While we can technically find periods over vectors, if we wanted to find the period and apply the QFT to the vector $(a^0 \bmod N, a^1 \bmod N, \cdots, a^k \bmod N)$, we have to first calculate all the powers of $a^x \bmod N$ to begin with (they would be our coefficients, not our kets to apply the QFT to that vector). We would already know the period! And if we use try to use quantum parallelism, we have the other issue of extracting the period data since measurement only gives us only one and not all of the end states.

Phase estimation is the complementary algorithm that bypasses this issue. Recall unitary operators have eigenvalues of modulus 1, i.e. eigenvectors $|v\rangle$ have corresponding eigenvalues of the form $\lambda = e^{2\pi i \varphi}$. So in a basis of eigenvectors, learning about how U acts on a vector u is equivalent to learning the corresponding values of φ. The idea of phase estimation is actually quite simple. Consider the following:

$$\frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle\, U^j\, |v\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle\, e^{2\pi i j\varphi}\, |v\rangle = \texttt{QFT}(|N\varphi\rangle) \otimes |v\rangle$$

Since $|v\rangle$ is an eigenvector of U, the sum on the left becomes something that looks quite similar to our an evaluation of the QFT on $|N\varphi\rangle$. So we can estimate φ by finding a way to compute these powers of U in this way, and then take the inverse QFT. Phase estimation occurs in stages.

Set-Up: Let U be unitary and $|v\rangle$ an eigenvector of U with corresponding eigenvalue $e^{2\pi i\varphi}$. We want to estimate $\varphi \in [0, 1)$.

Initialiation: We start with two registers $|00\cdots0\rangle\, |v\rangle$. The first register contains k qubits that are initially in state 0. The second register is initialized with the eigenvector whose phase we want to estimate $|v\rangle$ (stored with however many qubits we need to specify v). Since our algorithm will involve collapsing our qubits to get an estimate, there will be some probability of success/failure. k is the parameter that dictates how successful we will be both in terms of probability, and how precise our estimate will be.

Transformation: We apply the Hadamard matrix to each qubit in the first register:

$$|00\cdots0\rangle\,|v\rangle \xrightarrow{H} \left(\bigotimes_{j=1}^{k}\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)\right)|v\rangle = \frac{1}{\sqrt{2^k}}\sum_{j=0}^{2^k-1}|\texttt{bin}(j)\rangle\,|v\rangle$$

where `bin` just converts the integer i to its k-bit binary representation (we can write it without the binary representation, but here it makes the cost of storage clearer). Applying the Hadamard map to our k qubits in state $|0\rangle$ creates the superposition where all possible k-bit outcomes are equally likely (note: this uniform superposition could have been alternatively achieved by the QFT on $|0\rangle$).

Next, we apply the following unitary map that acts on the states as follows:

$$U_{\texttt{control}}(|\texttt{bin}(k)\rangle\,|v\rangle) = |\texttt{bin}(k)\rangle \otimes (U^k\,|v\rangle) = |\texttt{bin}(k)\rangle\,U^k\,|v\rangle$$

with the final RHS as a convenient abbreviation. This is unitary since its matrix is block diagonal, and both the identity and powers of U are unitary (see the Appendix for it fully written out). The idea is that the first register $|j\rangle$ tells us how times to apply U to the second register $|v\rangle$ (and the inclusion of the I blocks ensures we do not disturb the first register's qubits). This is known as a controlled-U operation, where the state of some qubits (the control qubits) determines how many times we apply U to the second register (this is often implemented via controlled-U^{2^j} sub-transformations which can be easy to implement; note this also means we need $O(k^3)$ quantum operations to compute the superposition of the sum of U^j below by using fast exponent algorithms, and only needing the binary representation to expand each power). Applying it to our superposition and noting that $U\,|v\rangle = e^{2\pi i\varphi}\,|v\rangle$ as it is an eigenvector, we get:

$$U_{\texttt{control}}\left(\frac{1}{\sqrt{2^k}}\sum_{j=0}^{2^k-1}|\texttt{bin}(j)\rangle\,|v\rangle\right) = \frac{1}{2^{k/2}}\sum_{j=0}^{2^k-1}|\texttt{bin}(j)\rangle\,U^j\,|v\rangle = \frac{1}{2^{k/2}}\sum_{j=0}^{2^k-1}|\texttt{bin}(j)\rangle\,e^{2\pi ij\varphi}\,|v\rangle$$

Estimation: Having applied U repeatedly, the phase φ has accumulated across the different qubit-states in the superposition in the first register. Take a look at the first register again:

$$\frac{1}{2^{k/2}}\sum_{j=0}^{2^k-1}|\texttt{bin}(j)\rangle\,e^{2\pi ij\varphi}$$

This looks almost exactly like applying the QFT to the basis vector $|\varphi\rangle$! So we apply the *inverse QFT* to our first register

$$\frac{1}{2^{k/2}}\sum_{j=0}^{2^k-1}|\texttt{bin}(j)\rangle\,e^{2\pi ij\varphi} \xrightarrow{\texttt{QFT}^{-1}} |\hat{\varphi}\rangle$$

Looking at the formula for the QFT, the phase accumulated in the QFT for a corresponding basis vector $|n\rangle$ is n/N. φ might not exactly be a rational number of this form, so $\hat{\varphi} \approx 2^k\varphi$ is an approximation of φ. So measuring this final qubit, we are likely to be returned $\hat{\varphi}$ with high probability and have an estimate of the phase φ.

This makes it clear the roles of our two registers: the second register $|v\rangle$ is only there so evaluating powers of U corresponds to scalar multiplication of the eigenvalue with the phase we want to estimate, and k controls how precise of an approximation of φ we are likely to get.

5.3.1 Phase Estimation Accuracy

If φ can be written with k bits, then we only need k qubits in the first register to get φ exactly. If we cannot do this (i.e. not enough physical qubits to implement, φ is irrational, etc.), how good can phase estimation be? It can be shown with some careful algebra on the inverse QFT that to estimate φ to k bits with probability $1-\epsilon$, we need at least $k + \lceil \log_2\left(\frac{1}{2\epsilon}+2\right)\rceil$ bits in the first register (see Section 5.2.1 [43]). As we will see later for our purposes, if we write the eigenstate $|v\rangle$ in our basis (i.e. as a superposition), we often need roughly k qubits to also represent the second register. So our total number of qubits is $\approx 2k$. Also note by the Spectral Theorem, computing powers of U is easy since we always diagonalize if needed.

5.4 Quantum Order-Finding

Back to Shor's algorithm, we now describe a quantum order-finding algorithm. Determining $o(a) = x$ in $\mathbb{Z}_N^\times$ is equivalent to using phase estimation on the unitary operator

$$U|j\rangle = \begin{cases} |aj \bmod N\rangle & \text{if } j < N \\ |j\rangle & \text{if } j \geq N \end{cases}$$

with $0 \leq j < 2^\ell$ where $\ell = \lceil \log_2 N \rceil$ is the number of bits needed to represent N in binary (this is unitary since $\gcd(a, N) = 1$ so can easily show preserves inner product). Note since $U^x = I$, that suggests that for all $0 \leq k < x$ the following are eigenstates of U:

$$|v_k\rangle = \frac{1}{\sqrt{x}} \sum_{j=0}^{x-1} e^{-2\pi i j k/x} |a^j \bmod N\rangle .$$

These are eigenstates since

$$U|v_k\rangle = \frac{1}{\sqrt{x}} \sum_{j=0}^{x-1} e^{-2\pi i j k/x} U|a^j \bmod N\rangle = \sum_{j=0}^{x-1} e^{-2\pi i j k/x} |a^{j+1} \bmod N\rangle = e^{2\pi i k/x} |v_k\rangle$$

and $o(a) = x$. Phase estimation allows us to recover k/x with high precision. The only issue is how do we give an eigenstate of $|v_k\rangle$ without already knowing x? The solution is instead of doing a single phase estimation, we do many in parallel via superposition since (let $\omega = e^{-2\pi i/x}$)

$$\frac{1}{\sqrt{x}} \sum_{k=0}^{x-1} |v_k\rangle = \frac{1}{\sqrt{x}} \sum_{k=0}^{x-1} \left(\frac{1}{\sqrt{x}} \sum_{j=0}^{x-1} e^{-2\pi i j k/x} |a^j \bmod N\rangle \right) = \frac{1}{x} \sum_{k=0}^{x-1} \sum_{j=0}^{x-1} \omega^{jk} |a^j \bmod N\rangle = |1\rangle$$

and $\sum_{k=0}^{x-1} \omega^{jk} = 0$ for $j \neq 0$. $|1\rangle$ is much easier to prepare in a physical system. So we let the second register of phase estimation be $|1\rangle$ (written in ℓ bits). Let $\hat{\varphi}_k$ be the associated phase for $|v_k\rangle$, phase estimation returns something that looks like

$$|0\rangle^{\otimes L} |1\rangle = |0\rangle^{\otimes L} \left(\frac{1}{\sqrt{x}} \sum_{k=0}^{x-1} |v_k\rangle \right) \xrightarrow{\text{phase estimation}} \frac{1}{\sqrt{x}} \sum_{k=0}^{x-1} |\hat{\varphi}_k\rangle |v_k\rangle$$

Measuring the first register gives a $1/x$ chance to return any of the $|\hat{\varphi}_k\rangle$, each of which returns $2^L(k/x)$ with high probability (and thus dividing by 2^L gives us k/x). (Note we often use $L = 2\ell$ bits in the first register to obtain high accuracy phase estimates)

The estimation $\hat{\varphi}_k/(2^L) \approx k/x$ may not exactly return a rational number, or it may be rational but lose factors and make x hard to recover (if $x = 4$ and $k = 2$, having $1/2$ does not give us useful information about x alone). Hence we use typical methods from number theory to approximate $\hat{\varphi}_k/(2^L)$ it as rationals, and take *repeated trials* of the order-finding method above to corroborate the data and recover x. Though, Shor's original analysis shows the probability of success after just one trial is bounded below by $4/\pi^2 \approx 0.405$ [52] [17]. With some additional classical post-processing, the success of a single run can be further improved [23]. Heuristically this makes some sense when considering this is the same order of probability of picking two coprime integers (see the Appendix).

Shor's Algorithm for Order-Finding: Given a such that $\gcd(a, N) = 1$ and above unitary U, find minimal x such that $a^x \equiv 1 \bmod N$.

1. Initialize two registers, the first one with L bits: $|0\rangle^{\otimes L} |1\rangle$.
2. Apply the Hadamard map or QFT to the first register to create a uniform superposition:

$$|0\rangle^{\otimes L} |1\rangle \mapsto \frac{1}{\sqrt{2^L}} \sum_{j=0}^{2^L-1} |j\rangle |1\rangle = \frac{1}{\sqrt{2^L}} \sum_{j=0}^{2^L-1} |j\rangle \left(\frac{1}{\sqrt{x}} \sum_{k=0}^{x-1} |v_k\rangle \right)$$

3. Apply U to the second register in accordance to the first register via $U_{\texttt{control}}$:

$$U_{\texttt{control}}\left(\frac{1}{\sqrt{x2^L}}\sum_{j=0}^{2^L-1}\sum_{k=0}^{x-1}|j\rangle\,|v_k\rangle\right)=\frac{1}{\sqrt{x2^L}}\sum_{j=0}^{2^L-1}\sum_{k=0}^{x-1}|j\rangle\,U^j\,|v_k\rangle=\frac{1}{\sqrt{x2^L}}\sum_{j=0}^{2^L-1}\sum_{k=0}^{x-1}e^{2\pi ijk/x}\,|j\rangle\,|v_k\rangle$$

4. Rearrange, and apply the inverse QFT to the first register to extract the phase:

$$\frac{1}{\sqrt{x}}\sum_{k=0}^{x-1}\left(\frac{1}{\sqrt{2^L}}\sum_{j=0}^{2^L-1}e^{2\pi ijk/x}\,|j\rangle\right)|v_k\rangle\xrightarrow{\texttt{QFT}^{-1}}\frac{1}{\sqrt{x}}\sum_{k=0}^{x-1}|2^L(k/x)\rangle\,|v_k\rangle$$

5. Measure the first register to obtain the approximation of the phase $2^L(k/x)$, and divide by 2^L to obtain the phase k/x.
6. Repeat as needed to determine x.

5.5 Quantum DLP and the Hidden Subgroup Problem

We spent the time exploring Shor's algorithm for factoring and hence know how to break RSA, but our primary concern has been elliptic curves and the discrete logarithm problem. If we can find a similar "period-finding problem" for DLP, we can easily adapt our algorithm.

Say we are trying to solve $g^x = h$ in the finite group G of order $|G| = p$. Instead of considering $f(x) = a^x \bmod N$, for DLP we might consider the function $f(a) = g^a h^{-1}$. But g is a generator, so has period p which we already know; this does not help us in anyway. So we consider the more general function $f : \mathbb{Z}_p \times \mathbb{Z}_p \to G$ defined by $f(a,b) = g^a h^{-b} = g^{a-bx}$. Since g is a generator, $f(a,b) = f(a',b')$ if and only if $a - bx \equiv a' - b'x \bmod p$ i.e. $a - a' \equiv x(b-b') \bmod p$. So f has a "periodic" structure that is not 1-dimensional like with factoring above, but rather has a 2D period embedded in a lattice. Our equivalence above shows $f(a,b) = f(a',b')$ whenever $(a',b') = (a,b) + \alpha(x,1) + \beta(p,0)$ (with the β term given by the order of the group is a trivial period). In other words, the lattice of periods is spanned by $\mathbb{Z}$-linear combinations of $\langle (x,1),(p,0)\rangle$. This is what Shor's DLP algorithm would try to find: this nontrivial basis of the lattice.

In particular, factorization and the discrete logarithm problem's "periodicities" are two realized instances of the *hidden subgroup problem*. Instead of specifying the DLP implementation directly (see Section 5.4.2 [43] for one possibility), we now describe the generic Shor's algorithm for finding hidden subgroups.

Definition 5.2 (Hidden subgroup)**.** Let G be a group, $H \leq G$ a subgroup, S a set, and $f : G \to S$ a function. We say f *hides the subgroup* H if f is constant on the cosets G/H i.e. $f(g_1) = f(g_2) \iff g_1 H = g_2 H$.

In essence, we lose the specific structure of each coset, reducing them to just their representatives; the action of H on G becomes indistinguishable to f. A more concrete way to think about this hiding is in how f maps H: $f(h) = e$ for all $h \in H$; H and all the structure with it gets reduced to the trivial element. We see this particular behavior with the First Isomorphism Theorem: given a homomorphism between two groups $\phi : G_1 \to G_2$, we have $G_1/\ker\phi \cong \operatorname{im}\phi$. The idea is that ϕ naturally hides $\ker\phi$, so we can induce an injection by quotienting out $\ker\phi$ and making these elements effectively the same. But in the process, we lose the information about $\ker\phi$.

Hidden Subgroup Problem (HSP): Given a group G, and a function $f : G \to S$ that hides $H \leq G$, find a set of generators for H.

Example 5.3 (Factorization as HSP)**.** Above, given an integer a such that $\gcd(a,N) = 1$ we reduced factorization to finding the order $n = o(a)$. We can reformulate this as an HSP: let $G = \mathbb{Z}$ and $f : \mathbb{Z} \to \mathbb{Z}_N$ via $f(x) = a^x \bmod N$. Since $f(x+n) = f(x)$, this is constant on the cosets of $H = n\mathbb{Z}$ in $\mathbb{Z}$. Hence the goal is to find n since $\langle n\rangle = n\mathbb{Z}$.

Example 5.4 (DLP as HSP)**.** Let g be a generator of a group G with order p, and we want to find the minimal x such that $g^x = h$. We consider the function $f : \mathbb{Z}_p \times \mathbb{Z}_p \to G$ via $f(a,b) = g^a h^{-b} = g^{a-bx}$. As discussed, this has period $(x, 1)$:

$$f((a,b)+(x,1)) = g^{(a+x)}h^{-(b+1)} = g^a g^x h^{-b} h^{-1} = g^a h h^{-b} h^{-1} = g^a h^{-b} = f(a,b)$$

So we let $H = \langle (x,1) \rangle = \{(kx, k) : 0 \leq k < p\}$. Hence the goal of this HSP is to recover the generator $(x, 1)$.

5.5.1 QFT on Finite Abelian Groups

If we are going to use the QFT in a general circumstance, we need to clarify what that means in a non-obvious/-integer context. Fortunately, there is an obvious analog to the cyclic group $\mathbb{Z}_N$. Let $g \in \mathbb{Z}_N$, then the QFT of g is

$$\texttt{QFT}(|g\rangle) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i g \frac{k}{N}} |k\rangle$$

By the Fundamental Theorem for Abelian Groups, every finite abelian group can be identified as the direct product of cyclic groups $\mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k}$, so in general we can represent a group element as k registers each of length $\lceil \log N_i \rceil$, apply the QFT to these k different registers individually, and then tensor them together. So whenever I write $|\chi(g)\rangle = \texttt{QFT}(|g\rangle)$ as shorthand for QFT, it is meant to refer to the above formula in $\mathbb{Z}_N$ (or its relevant direct product) applied to the element g is isomorphic to.

More generally, we define the Fourier transform of a function on a finite abelian group to map a function $f : G \to \mathbb{C}$ to another function $\hat{f} : \hat{G} \to \mathbb{C}$ where $\hat{G} = \text{Hom}(G, \mathbb{C}^\times)$ is the set of homomorphisms from G to $\mathbb{C}^\times$. The elements $\chi \in \hat{G}$ are called *characters of* G and together form the *dual group of* G. The formula is given by

$$\hat{f}(\chi) = \frac{1}{\sqrt{|G|}} \sum_{g \in G} f(g)\chi(g)$$

(Usually, we use the conjugate $\overline{\chi}$ instead of χ, but it just changes which is used in the inverse; they produce identical results.) If $G = \mathbb{Z}_N$ is cyclic, then $G \cong \hat{G}$ via the map $a \mapsto \chi_a$ where $\chi_a(x) = e^{2\pi i a x/N}$, we may write $\hat{f}(g) := \hat{f}(\chi_g)$ as shorthand. If we let $f(g) = g$, we end up with the formula above, and the general QFT/DFT we stated earlier. In particular, the relationship looks like

$$\texttt{DFT}(\big[f(0), f(1), f(2), \cdots, f(N)\big]) = \big[\hat{f}(0), \hat{f}(1), \hat{f}(2), \cdots, \hat{f}(N)\big]$$

Because of this isomorphism in cyclic groups, we will often just write this in terms of group elements represented by integers, and combine them via direct products for non-cyclic abelian groups via the Fundamental Theorem for Abelian Groups.

Remark 5.5. There is an even more general Fourier transform on any finite group involving representations. The Fourier transform of a function $f : G \to \mathbb{C}$ at a representation $\rho : G \to \text{GL}(\mathbb{C})$ is $\hat{f}(\rho) = \sum_{g \in G} f(g)\rho(g)$. In the case of cyclic group $C_N \cong \mathbb{Z}_N$, the (1-dimensional, so scalar instead of matrix) representations we consider are our characters $\chi_a(x) = e^{2\pi i a x/N}$. Conceptually, our normal DFT/QFT is asking, "How much of frequency x is in our function?" In the group case, we are asking, "How much of representation ρ is in our function?"

For a subgroup $H \leq G$, we define $H^\perp = \{\chi \in \hat{G} : \chi(h) = 1 \ \forall h \in H\} \leq \hat{G}$. Again, because of the isomorphism in cyclic groups, we may equivalently write $H^\perp = \{\ell \in G : \chi_\ell(h) = 1 \ \forall h \in H\} \leq G$. We introduce these to note a simplification of the Fourier basis for hidden subgroups. In particular,

$$f(g) = \sqrt{\frac{|H|}{|G|}} \sum_{\ell \in H^\perp} \overline{\chi_\ell(g)} \hat{f}(\ell),$$

where my full derivation can be found in the Appendix. (It will be useful to note $|H^\perp| = |G|/|H|$. Consider the homomorphism $\phi : \hat{G} \to \hat{H}$ via $\chi \mapsto \chi|_H$. Then $\ker \phi = H^\perp$, so by First Isomorphism Theorem, $\hat{G}/H^\perp \cong \hat{H}$. Finish with $\hat{G} \cong G$ and $\hat{H} \cong H$.)

Generic Shor's Algorithm for Abelian HSP [9]: Given an input of abelian group G, and a function f that hides a subgroup H, output H.

1. Initialize two registers $|0\rangle|0\rangle$, where the first register's basis states are elements in G, and the second's are in S (the set f hides H in).
2. Apply the QFT to the first register to create a uniform superposition over the group elements (note $\chi_0(G) = \{1\}$, and the function in the Fourier transform description is the identity in this case):
$$|0\rangle|0\rangle \mapsto \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle|0\rangle$$
3. Apply f to the second register in accordance to the first register with a unitary operator U_f:
$$\frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle|0\rangle \mapsto \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle|f(g)\rangle$$
4. By the above, write $f(g) = \sqrt{\frac{|H|}{|G|}} \sum_{\ell \in H^\perp} \overline{\chi_\ell(g)} \hat{f}(\ell)$:
$$\frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle|f(g)\rangle = \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle \left(\sqrt{\frac{|H|}{|G|}} \sum_{\ell \in H^\perp} \overline{\chi_\ell(g)} |\hat{f}(\ell)\rangle \right)$$
5. Apply the inverse QFT to the first register.
$$\sqrt{\frac{|H|}{|G|}} \sum_{\ell \in H^\perp} \left(\frac{1}{\sqrt{|G|}} \sum_{g \in G} \overline{\chi_\ell(g)} |g\rangle \right) |\hat{f}(\ell)\rangle \xrightarrow{\texttt{QFT}^{-1}} \sqrt{\frac{|H|}{|G|}} \sum_{\ell \in H^\perp} |\ell\rangle |\hat{f}(\ell)\rangle$$
6. Measure the first register to obtain a random $\ell \in H^\perp$, which gives information on H.
7. Repeat steps 1–6 until H can be determined via the linear relations of $H^\perp$.

Note, that although we need to know H to be able to do the rewriting in step 4, the rewriting is just an equality; we do not actually change the second register, but rather put it in a form that is illustrative why the inverse QFT should return something useful. A bigger problem is step 3: how do we get $|f(g)\rangle$ as the second register in step 3? The idea remains the same as we did in Shor's algorithm for factoring: represent f as an efficient-to-calculate unitary operator (for factoring, that was repeated modular exponentiation). In fact, rewriting f in terms of $\hat{f}$ is a similar reason why we started with $|1\rangle$ in the factoring case (it was just more clear there where phase estimation's eigenvalues came from in that case; in fact, similarly the Fourier basis states are eigenvectors of the cyclic group operator). It is not quite the same, but the motivation is the same: introduce terms to get a Fourier-like expression. The trick lies in the fact that because f hides H, its Fourier transform collapses onto the cosets G/H. To see a perhaps more clear iteration of this "coset collapse", see another derivation of Shor's algorithm for HSPs in the Appendix.

Remark 5.6. This above general formulation highlights why we especially needed phase estimation and to specify the size of the first register to approximate the order when solving factorization from before: the group we were hiding information in was $\mathbb{Z}$, which is infinite. So we naturally needed to truncate our algorithm somewhere to stop searching; we only have a finite number of qubits to create superpositions.

5.6 Time Complexity Analysis

We need at least $\log(|G|)$ bits in the first register to represent every group element uniquely. As discussed, then the QFT runs in $O(\log(|G|)^2)$ operations. We then need one call of the unitary f, which we assume we can compute in some polynomial time $p(\log|G|)$ (since if we could not, our hopes of solving HSP are doomed anyway). Finally, note that the size of a minimal generating set of H is at most $\log(|H|) \leq \log(|G|)$.

Proposition 5.7. *Let $H = \langle h_1, \cdots, h_n \rangle \leq G$, and $g \notin H$. Then $|\langle H, g \rangle| \geq 2|H|$.*

Proof. Since $g \notin H$, we have H and gH are distinct cosets, both contained in $\langle H, g \rangle$. Hence $|\langle H, g \rangle| \geq |H \cup gH| = 2|H|$. □

A corollary of the above is that we can find a generating set of a group G with no more than $\log(|G|)$ elements. As it turns out, we do not expect to need many more iterations than this [46] to obtain a generating set. Since $|H^\perp| = |G|/|H|$, we need roughly $\log(|G|/|H|) \leq \log(|G|)$ iterations to reconstruct $H^\perp$ (and so H) with high probability. Hence, the overall complexity is $O(\log(|G|)(2\log^2(|G|) + p(\log|G|)))$ which is often dominated overall by $O((\log|G|)^3)$ when evaluating f is cheap (else it is still polynomial in $\log(|G|)$ which all we care about). This is a significant speed-up to not just factoring, but discrete logarithms, and many other thought-to-be-difficult-to-solve problems.

5.7 Impact on Classical ECC and Immediate Timeline

At the moment, we do not have to be all that concerned about the use of classic elliptic curve cryptography. Recall to provide 128-bit security, we need to use an elliptic curve over a field of prime order around 256 bits. Directly from our previous analyses, for our phase estimation-type procedures to be remotely accurate, we would need at least $\log|\mathbb{Z}_p \times \mathbb{Z}_p| = 2\log|G|$ qubits in the first register, and $\log|G|$ qubits in the second register (since our f hides H in G for DLP). So we would need at least ≈ 768 qubits just to formulate the problem on a quantum computer—this does not even include the machinery necessary to carry out the quantum arithmetic, or unitary operations. Estimates have ranged based on implementations that to solve ECDLP specifically with a Shor-like algorithm would require between roughly $6n$ [49] and $9n$ [50] qubits (where $n = \lceil \log p \rceil$ of the field $\mathbb{F}_p$ used i.e. n is the number of bits needed to represent the order of the field and hence elliptic curve group). So for 128-bit security with a 256-bit order field, we would need roughly between 1500 to 2300 qubits (with the latter being a stronger estimate for they simulated the implementation instead of just writing the theory). In fact, the specific implementation matters a lot. Recent estimates suggest only around $2n$ [7] to $3n$ [27] qubits are needed to factor an n-bit number (but likely more for actual efficient, error-free implementation). Even more concerning, some HSPs have yet to be implemented efficiently even on quantum computers, such as the graph isomorphism or shortest vector problems [9][43].

Moreover, these estimates are idealized *logical qubits* that are error-free, treated with the theoretical perfection and precision that physical implementations just do not offer [49]. Due to the probabilistic nature of qubits (and just their volatile, fragile nature), we need some number of physical qubits to error-correct one another to produce the nice behavior our theory desires. Correcting for errors (caused by the environment or manipulation) is not obvious either, since while in classical computing we only have bit errors (i.e. a 0 gets flipped to a 1), qubits can have infinitely many, continuous phase errors. Even if qubits look close together in phase, the small difference can add up over time resulting in large output errors. Handling these is tricky, especially since we cannot measure/read qubits without disturbing them and having them collapse [21], and it is in fact impossible to duplicate quantum data to create the redundancy we might expect to need in these correcting codes (i.e. the *no-cloning theorem*) [28][1][43]. In fact, these errors are such a problem, attempts to factor the number $N = 35$ with one implementation of Shor's algorithm in 2019 succeeded only 14% of the time due to the cumulative error build up [4].

There are some short, effective correcting codes, like the Steane code, that can mimic a logical qubit with 7 physical bits in a manner similar to the classical Hamming code [21][43], but for truly effective, large scale computations (i.e. can account for errors in the error correction systems; satisfy the *threshold theorem* [43]), modern codes estimate the need from 1000-10000 high quality physical qubits/hardware (i.e. maintain coherence for extended period of time; introduce a qubit error rate is less than 1%, etc.) [25][27]. So conservatively, we would expect *at least* $2300 \cdot 1000 \approx 2.3 \cdot 10^6$ physical qubits to break ECDLP in a field of order 2^{256}. As of now, the most powerful quantum computer in terms of sheer number of qubits is IBM's Condor processor that has 1121 physical qubits, with many commercially available and in use processors being no more than a couple hundred qubits [26]. However, scaling these quantum processors is at the forefront of quantum computing research, and Microsoft just announced their Majorana 1 chip in February 2025 that promises to scale up to 1 million qubits [12].

At the moment, many expect to have a significant chance of quantum computing powerful enough for cryptographic applications by the early 2030s [33][58][16]. For ECC in particular, though, quantum computing is especially concerning since we already use small parameters in the classical computing setting. Despite factoring and DLP/ECDLP have significantly different time complexities to break on classical computers, they have approximately the same time complexity in the quantum setting, so the smaller parameters for equal security levels for ECDLP in the classical setting lend themselves to smaller qubit requirements in the quantum setting. As such many expect ECC to be broken by quantum computers before RSA/factoring [50] [27]. But for the time being, until we have significant optimizations or developments of quantum processors, classical computers seem to be the best option at this scale [61][29]. **We are safe for now.**

6 Post-Quantum Elliptic-Curve Cryptography

Despite quantum computing being a relatively far out threat, we cannot wait until the field realizes its capabilities as actual processors to design safeguards against them. With Shor's algorithm in place even in concept, the task at hand now is to find a cryptographic scheme that is also resistant to quantum attacks.

6.1 Isogeny Graphs

Quantum computing's biggest upsell has been in its use of superpositions and quantum parallelism, in a way "trivializing" computing the group operations, and finding the relevant generators of our subgroup and leveraging the abelian structure of the group. This suggests a sort of structural shift in our cryptographic protocols: we look for problems that are algorithmically hard not because of the group structure, but because of the algorithm itself; if we wanted to focus on a particular group structure, we would need to at least supplement it in some way to fundamentally shift away from the original group. Classical ECC is fundamentally flawed in that the way it is set up does not directly allow for direct application to post-quantum methods, but we can still adapt the tools we have found along the way to create new schemes. One such solution is *isogeny based cryptography*, where we focus not on the relationships between points on an elliptic curve, but rather relationships between elliptic curves themselves; we look at how many groups interact instead of sticking to just one group.

6.1.1 Additional Mathematical Background

For the sake of brevity, clarity, and applicability to cryptographic methods, the math above was taken as concretely as possible; I made an active attempt to bridge the tangible equations as used in [62] and the more isolated theory as found in [56][55]. We have been able to largely avoid the abstractions that textbooks dedicate hundreds of pages to the completeness of the study of elliptic curves, all in the interest to stick to our specific goals. It is probably not surprising that to create the methods that can evade the quantum threat, we have to elevate our sophistication a bit. As such, by nature of our previous discussions and the need for new tools, we will mention and source out results as we need, and briefly touch on intuitions if available. Of course, we have already done this a bit with dual isogenies, and the elliptic curves specific converse to the First Isomorphism Theorem. In this case, we just need to do it a bit more.

Before we begin, we first simplify our discussion of elliptic curves. One way we can do this is try and find equivalence classes of elliptic curves to group them together. One equivalence class we could define is two elliptic curves $E_1/\mathbb{K}, E_2/\mathbb{K}$ are equivalent if their groups are isomorphic $E_1(\overline{\mathbb{K}}) \cong E_2(\overline{\mathbb{K}})$ (and following from before, this isomorphism is given by rational functions; when such a group isomorphism exists, we say the two curves $E_1/\mathbb{K}, E_2/\mathbb{K}$ are isomorphic). This is a perfectly correct characterization to use, and follows the group structure we want.

To have an isomorphism, we critically need it to preserve the group law. The group law is defined via the geometry of some lines, so in particular, we need lines to map to lines, hence it is linear. Secondly, we mentioned how every elliptic curve equation can be put into Weierstrass form after a change of variables, so we will skip that step and just ensure that our map preserves the Weierstrass form. Consider an arbitrary

linear map $(x, y) \mapsto (sx + ty + u, px + qy + r)$. Applying this to a general Weierstrass equation:

$$y^2 = x^3 + ax + b \longrightarrow (px + qy + r)^2 = (sx + ty + u)^3 + a(sx + ty + u) + b$$
$$(px + qy + r)^2 = (sx + u)^3 + a(sx + u) + b$$
$$p^2x^2 + 2pqxy + 2prx + q^2y^2 + 2qry + r^2 = s^3x^3 + 3s^2ux^2 + 3su^2x + u^3 + asx + au + b$$

- We do not want cubic terms outside of x^3, so $t = 0$.
- We need to eliminate the xy term: $2pqxy = 0$. $q \neq 0$, since that would make our transformation only a function of x and hence not injective (since if $(x, y) \in E$, then so is $(x, -y) \in E$) and hence not bijective. So $p = 0$.
- We need to eliminate the y term: $2qry = 0$, and since $q \neq 0$, we have $r = 0$.
- We need to eliminate the x^2 term: $3s^2ux^2 = 0$. $s \neq 0$ (else not invertible), so $u = 0$.

$$q^2y^2 = s^3x^3 + asx + b$$

- We need the lead coefficients $q^2 = s^3$ so we can normalize them to be monic. So we need $q = \mu^{1/2}, s = \mu^{1/3}$ for some non-zero $\mu \in \overline{\mathbb{K}}$. In particular let $\mu = \lambda^6$, so $q = \lambda^3, s = \lambda^2$.

Hence all isomorphisms are of the form $(x, y) \mapsto (\lambda^2 x, \lambda^3 y)$, mapping the curve $y^2 = x^3 + ax + b$ to $(y')^2 = (x')^3 + a'(x') + b'$ where $a' = \lambda^4 a, b' = \lambda^6 b$. This can easily be verified this does in fact give an isomorphism, though our assumption preserve that; our map essentially scales the Weierstrass equation by λ^6. For a more rigorous proof, see [62], [55]. We took an algebraic property concerning points on the equation, and deduced a geometric equivalence in terms of the overall curves themselves. Treating isomorphism classes as curves related by this change of variables, we can construct a concrete algebraic invariant that categorizes these classes. In particular, for any isomorphism class, for every elliptic curve E in the class, the quantity

$$j(E) = 1728\frac{4a^3}{4a^3 + 27b^2}$$

remains invariant (we take $j(E) \in \mathbb{K}$ reducing 1728 mod Char($\mathbb{K}$)). We hence call this the j-invariant of E.

Remark 6.1. These may seem arbitrary or random, but the theory of elliptic curves is wide and varied. The results we have derived thus far try to contain themselves as computational tools, since those are the most useful for cryptographers. But the basis for the theory lies in an intersection of complex analysis and abstract algebra with far too much additional background outside the scope of this essay. In particular, the j-invariant is borrowed from the complex analytic view with modular forms and are from where the seemingly arbitrary coefficients arise.

Remark 6.2 (Twists)**.** As before when we defined isogenies, we are working in the closure $\overline{\mathbb{K}}$, and as such we may find scalar for our change of variables such that $\lambda \in \overline{\mathbb{K}} \setminus \mathbb{K}$. This would leave $(x, y) \mapsto (\lambda^2 x, \lambda^3 y)$ mapping $\mathbb{K}$-rational points to *non*-$\mathbb{K}$-rational points that are not on $E_2(\mathbb{K})$! Curves that are isomorphic over $\overline{\mathbb{K}}$ but not $\mathbb{K}$ are called *twists* of one another. The degree of the twist is the least integer n it takes for $\lambda^n \in \mathbb{K}$ i.e. a quadratic twist would be one where $\lambda \notin \mathbb{K}$ but $\lambda^2 \in \mathbb{K}$. Twists create interesting uses and attacks in the classical setting, but for the time being, will only impact our implementations in the quantum setting.

Remark 6.3. Note that for all $j \in \mathbb{K} \setminus \{0, 1728\}$, we have that

$$E_j : y^2 = x^3 + \frac{3j}{1728 - j}x + \frac{2j}{1728 - j}$$

has j-invariant $j(E_j) = j$. For the other two cases, we can pick $E_0 : y^2 = x^3 + 1$ and $E_{1728} : y^2 = x^3 + x$. In particular, this is an elliptic curve defined over $\mathbb{K}$. So for any elliptic curve $E/\mathbb{F}$ with $j(E) = j$, it is isomorphic to E_j and hence can move between the E and E_j via a change of variables. This might seem sketchy given we have defined E over a field $\mathbb{F}$ and $j \in \mathbb{K}$ and it is not necessarily that $\mathbb{K} \cong \mathbb{F}$. However, our notions of isomorphism asks to check over the closure of the base fields, so we are still in the clear with this change of variables. Existence of such a twist via E_j will be important when trying to standardize implementing curves.

We complete this characterization of the isomorphism classes with the converse of the j-invariant.

Proposition 6.4. *Let E_1, E_2 be elliptic curves. Then E_1 is isomorphic to E_2 if and only if $j(E_1) = j(E_2)$.*

Proof. [62] The forward direction we discussed and is easy by inspection of the change of variables. The converse is easy: assume equality, check the cases for when $j(E) = 0$ and $j(E) \neq 0$, and since we are working in the closure $\overline{\mathbb{K}}$, we can pick a suitable value λ. □

Considering curves over isomorphism, we identify classes with this convenient j-invariant quantity. Going back to our motivation, we want to "connect" these isomorphic curves. Isogenies naturally do that, as note that the relation of being isogenous is an equivalence relation on the isomorphism classes (isomorphisms are degree 1 isogenies as given by our change of variables, and so composing them with any isogeny of degree ℓ is also an isogeny of degree ℓ). We hence define the following:

Definition 6.5 (Isogeny graph)**.** An *isogeny graph* is a (multi)graph with our nodes as j-invariants and edges connecting isogenous curves.

Remark 6.6. Since dual isogenies guarantee the existence of same degree isogeny in the reverse direction, we often draw these undirected.

Remark 6.7. On first glance, this graph should be trivial—we said isogenous curves form an equivalence relation and hence the graph should be complete. We will often restrict specifically to isogenies of degree ℓ, instead of being isogenous in general.

6.1.2 Supersingular Elliptic Curves

We turn our focus to supersingular elliptic curves. Recall the definition of supersingular elliptic curves. We give a more general characterization based on $E[p] := \ker([p])$.

Proposition 6.8. *Let $\mathbb{K}$ be a field of characteristic p, and $E/\mathbb{K}$ be an elliptic curve. Then either $E[p] \cong \mathbb{Z}/p\mathbb{Z}$ or $E[p] = \{\infty\}$.*

Proof. [55] Let Φ_p be the p-power Frobenius endomorphism. We use a combination of results involving the dual isogeny and the relation $|\ker(\Phi_p)| = \deg_s(\Phi_p)$:

$$|E[p]| = |\ker([p])| = \deg_s([p]) = \deg_s(\hat{\Phi}_p \circ \Phi_p) = \deg_s(\hat{\Phi}_p)$$

noting that Φ_p is inseparable. Recalling that $\deg(\hat{\Phi}_p) = \deg(\Phi_p) = p$, we have two options to consider. If $\deg_s(\hat{\Phi}_p) = 1$, then $|E[p]| = 1$, giving us the second conclusion. If $\deg_s(\hat{\Phi}_p) = p$, then $|E[p]| = p$, and so it is cyclic and $E[p] \cong \mathbb{Z}/p\mathbb{Z}$. □

Definition 6.9 (Supersingular vs. ordinary ellitpic curve)**.** Over a field $\mathbb{K}$ of characteristic p, an elliptic curve $E/\mathbb{K}$ is *supersingular* if $E[p] = \{\infty\}$ is trivial, and say it is *ordinary* otherwise i.e. $E[p] = \mathbb{Z}/p\mathbb{Z}$.

This is a fundamental divide among elliptic curves, and the distinction arises frequently.

Proposition 6.10. *Let E_1, E_2 be two isogenous elliptic curves over a field $\mathbb{K}$ of characteristic p. Then E_1 is supersingular if and only if E_2 is.*

Proof. Let $\phi : E_1(\overline{\mathbb{K}}) \to E_2(\overline{\mathbb{K}})$ be our isogeny. Let $[p_1]$, $[p_2]$ denote the multiplication-by-p endomorphisms on E_1, E_2 respectively. Then note $[p_2] \circ \phi = \phi + \phi + \cdots + \phi = \phi \circ [p_1]$. Comparing (separable) degrees:

$$\begin{aligned}
\deg_s([p_2] \circ \phi) &= \deg_s(\phi \circ [p_1]) \\
\deg_s([p_2]) \deg_s(\phi) &= \deg_s(\phi) \deg_s([p_1]) \\
\deg_s([p_1]) &= \deg_s([p_2]) \\
|\ker([p_1])| &= |\ker([p_2])| \\
|E[p_1]| &= |E[p_2]|
\end{aligned}$$

So E_1 is supersingular if and only $|E[p_1]| = 1$ if and only if $|E[p_2]| = 1$ if and only if E_2 is supersingular. $\square$

In particular, the above tells us we can never have an isogeny between supersingular and ordinary curves, and so our isogeny graphs will be similarly divided only containing one type of curve. Ordinary graphs and supersingular graphs have distinctly different structures, so for the time being, we restrict attention to supersingular curves.

Theorem 6.11. *Let $E/\mathbb{K}$ be a supersingular elliptic curve over a field with $\mathrm{Char}(\mathbb{K}) = p$. Then $j(E) \in \mathbb{F}_{p^2}$.*

Proof. Since E is supersingular, $1 = |E[p]| = |\ker([p])| = \deg_s([p]) = \deg_s(\hat{\Phi}_p \circ \Phi_p) = \deg_s(\hat{\Phi}_p)$, so $\hat{\Phi}_p$ is purely inseparable. So decomposing $\hat{\Phi}_p : E^{(p)}(\overline{\mathbb{K}}) \to E(\overline{\mathbb{K}})$ into its separable and inseparable parts, $\hat{\Phi}_p = \psi \circ \Phi'_p$ where $\Phi'_p : E^{(p)}(\overline{\mathbb{K}}) \to E^{(p^2)}(\overline{\mathbb{K}})$ is the Frobenius map on $E^{(p)}$ and $\psi : E^{(p^2)}(\overline{\mathbb{K}}) \to E(\overline{\mathbb{K}})$ is a degree 1 separable isogeny (i.e. an isomorphism; note to match degrees, we need only one composition of Φ'_p). Since ψ is an isomorphism, $j(E^{(p^2)}) = j(E)$. Since $j(E) \in \mathbb{K}$,

$$j(E) = j\left(E^{(p^2)}\right) = 1728\frac{4(a^{p^2})^3}{4(a^{p^2})^3 + 27(b^{p^2})^2} = \left(1728\frac{4a^3}{4a^3 + 27b^2}\right)^{p^2} = j(E)^{p^2}$$

Hence $j(E) \in \mathbb{F}_{p^2}$ (again by considering the definition/construction of finite fields $\mathbb{F}_{p^k}$ from earlier). $\square$

So there is a type of uniformity among our supersingular curves; regardless of the field we are working in, we can reduce all of our calculations of our isogeny graphs to $\mathbb{F}_{p^2}$. Since we only care about curves up to isomorphism, we can choose the twist representative $E_j/\mathbb{F}_{p^2}$ of our isomorphism class denoted by its relevant j-invariant. Also, this tells us there are only finitely many distinct j-invariants and thus finitely many supersingular elliptic curves up to isomorphism. This importantly bounds the size of isogeny graphs for supersingular curves. In fact, we can make this much more precise.

Theorem 6.12. *Let $\mathbb{K}$ be a field with $\mathrm{Char}(\mathbb{K}) = p \geq 5$. Then there are*

$$\left\lfloor \frac{p}{12} \right\rfloor + \begin{cases} 0 & \text{if } p \equiv 1 \bmod 12 \\ 1 & \text{if } p \equiv 5, 7 \bmod 12 \\ 2 & \text{if } p \equiv 11 \bmod 12 \end{cases}$$

distinct supersingular elliptic curves over $\mathbb{K}$ up to isomorphism.

Proof. See Corollary 4.40 [62] or V.4.1(c) [55] for the complete proof. Let $E/\mathbb{K} : y^2 = f(x)$ where $f(x)$ is a cubic in Weierstrass form with distinct roots. The idea of the proof is to create a criterion of supersingularity of E by considering the coefficient of x^{p-1} in $f(x)^{(p-1)/2}$, and then hence an equivalent formulation in terms of the roots of a polynomial H_p representing this coefficient of x^{p-1}. Thus counting the number of supersingular curves reduces to counting roots of H_p and adjusting for potential special ordinary cases. $\square$

So the size of our graph grows linearly with respect to the characteristic. We now make our first observation of the edges of the isogeny graph.

Theorem 6.13. *Let $E/\mathbb{K}$ be an elliptic curve over a finite field $\mathbb{K}$ with $\mathrm{Char}(\mathbb{K}) = p$. Let $\ell \neq p$ be a prime, then there are $\ell + 1$ distinct isogenies of degree ℓ with domain E over $\overline{\mathbb{K}}$.*

Proof Sketch. We sketch the idea without introducing too much more technology. Since $\ell \neq p$, and in particular $p \nmid \ell$, all degree ℓ isogenies are separable (by comparing the degrees in their isogeny decomposition). We use the fact that for any finite subgroup $G \leq E(\overline{\mathbb{K}})$, we can compute a unique separable isogeny $\phi : E \to E/G$ with $\ker \phi = G$. Separability implies $|G| = |\ker \phi| = \deg \phi$, so to count all isogenies of degree ℓ, we count the number of subgroups G of order ℓ. But since ℓ is prime every point in such a subgroup would have order ℓ, so $G \leq E[\ell]$, and can restrict to counting subgroups of $E[\ell]$.

It is easy to see for similar reasons above that $[\ell]$ is separable since $\deg([l]) = \ell^2$. So $|E[\ell]| = \deg([\ell]) = \ell^2$, and by analyzing the structure of this finite abelian subgroup, see that $E[\ell] \cong \mathbb{Z}/\ell\mathbb{Z} \times \mathbb{Z}/\ell\mathbb{Z}$ since every non-zero element in $E[\ell]$ has order ℓ. Every element in $E[\ell]$ is of order ℓ, and each cyclic subgroup contains $\ell - 1$ non-zero points. Hence there are $|E[\ell] \setminus \{0\}|/(\ell - 1) = (\ell^2 - 1)/(\ell - 1) = \ell + 1$ subgroups of order ℓ, and thus that many isogenies as well. □

Remark 6.14 (Vélu's Formulas)**.** It is worth mentioning that the above result of determining isogenies from subgroups is not left to be existential. Vélu's formulas [55][19] allow us to actually compute the unique isogeny. It is a classical algorithm that runs linearly in the degree of the isogeny being computed, but optimizations to have it run asymptotically in the square root have been found too [11].

Recall Hasse's bound: $||E(\mathbb{F}_q)| - (q+1)| \leq \sqrt{2q}$. This is not the complete formulation of the bound. In particular, Hasse's bound, as seen in our proof of it, is intimately connected to the Frobenius map Φ_q. The full statement of Hasse's bound is $|E(\mathbb{F}_q)| = q + 1 - t$ where $|t| \leq \sqrt{2q}$ is the *trace of* Φ_q. The trace and (and corresponding determinant) of Φ_q come from viewing Φ_q as essentially a 2×2 matrix with the *Tate module*, and computing the trace and determinant as you would with any matrix. Subsequently, Φ_q comes with a corresponding characteristic polynomial: $\chi_{\Phi_q}(x) = x^2 - tx + q$ (Theorem 2.3.1 [55]). More generally, if we let $\alpha, \beta \in \mathbb{C}$ be the roots of $\chi_{\Phi_q}(x)$, it can be shown that $|E(\mathbb{F}_{q^n})| = q^n + 1 - (\alpha^n + \beta^n)$. For us, the important cases with supersingularity gives us that $p \mid t = \alpha + \beta$. For $q = p^2$, we have that $|t| \leq 2\sqrt{p^2} = 2p$ i.e. $t = \pm 2p, \pm p, 0$ depending on the specific prime p. See [38][63] for further discussion.

Proposition 6.15. *If $E/\mathbb{F}_{p^2}$ is a supersingular elliptic curve with p chosen such that $t = \pm 2p$ with t described above, then $E(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(p \mp 1)\mathbb{Z} \times \mathbb{Z}/(p \mp 1)\mathbb{Z}$.*

Proof Sketch. Let Φ_q denote the q-power Frobenius map. If $t = \pm 2p$, then by the characteristic polynomial and the Cayley-Hamilton theorem, $\Phi_{p^2}^2 \pm 2p + p^2 = (\Phi_{p^2} \pm p)^2 = 0$. So in particular, $\Phi_{p^2} = [\mp p]$ as endomorphisms. But note, $\Phi_{p^2} = [1]$ also as an endomorphism for $E(\mathbb{F}_{p^2})$. So in particular, $[1 \mp p] = 0$ as maps for these supersingular curves i.e. $E(\mathbb{F}_{p^2}) \subseteq E[1 \pm p]$. We get the other containment with our proposition from before: for a point $P \in E(\overline{\mathbb{F}_{p^2}})$, $P \in E(\mathbb{F}_{p^2}) \iff \Phi_{p^2}(P) = P$. So using the exact same equivalences, $0 = [1 \mp p](P) = P \mp [p]P = P - \Phi_{p^2}(P)$. So $E[1 \mp p] \subseteq E(\mathbb{F}_{p^2})$. □

Knowing the above is a key step in understanding the isogeny-based protocol we will consider later, for it greatly simplifies the group structure. The final observation we make is that the isogeny graph is connected. We make use of the following:

Theorem 6.16 (Sato-Tate)**.** *[5][19] Two elliptic curves $E_1/\mathbb{K}, E_2/\mathbb{K}$ are isogenous over $\mathbb{K}$ if and only if $|E_1(\mathbb{K})| = |E_2(\mathbb{K})|$ (note, this is over the base field, not the closure).*

Remark 6.17. To be isogenous over $\mathbb{K}$ means that our rational functions $r_1(x), r_2(x)$ defining our isogeny takes coefficients in $\mathbb{K}$ as opposed to $\overline{\mathbb{K}}$ like our definition stipulates.

Proposition 6.18. *All supersingular elliptic curves over $\mathbb{F}_{p^2}$ are isogenous.*

Proof. [64] We divide the cases into the possible values of t and calculate the corresponding roots α, β of $\chi_{\Phi_{p^2}}(x) = x^2 - tx + p^2$.

- $t = \pm 2p \longrightarrow x^2 \mp 2px + p^2 = (x \mp p)^2 = 0$ so $\alpha = \beta = \pm p$.
- $t = \pm p \longrightarrow x^2 \mp px + p^2 = 0$ so $\alpha, \beta = \frac{\pm p \pm \sqrt{p^2 - 4p^2}}{2} = p \cdot \frac{\pm 1 \pm i\sqrt{3}}{2} = \pm p \cdot e^{i\pi/3}$
- $t = 0 \longrightarrow x^2 + p^2 = 0$ so $\alpha = -\beta = ip$.

In any case, $\alpha^{12} = \beta^{12} = p^{12}$. So considering our elliptic curve groups over $\mathbb{F}_{(p^2)^{12}} = \mathbb{F}_{p^{24}}$, we have $|E(\mathbb{F}_{p^{24}})| = p^{24} + 1 - (\alpha^{12} + \beta^{12}) = p^{24} + 1 - 2p^{12}$. This quantity is independent of E, and since every elliptic curve over $\mathbb{F}_{p^2} \subseteq \mathbb{F}_{p^{24}}$, by the Sato-Tate theorem, every supseringular curve over $\mathbb{F}_{p^2}$ are isogenous. □

So every supersingular elliptic curve is isomorphic to one in $\mathbb{F}_{p^2}$, and are isogenous over $\overline{\mathbb{F}_{p^2}} \supseteq \mathbb{F}_{p^{24}}$. So the general isogeny graph is not only complete (and so connected), but also includes *all* supersingular graphs over a given field. More importantly for us, we have the ℓ-isogeny graph is connected.

Proposition 6.19. *The graph of supersingular elliptic curves over $\mathbb{F}_{p^2}$ with vertices as j-invariants and edges as ℓ-isogenies is connected.*

Unfortunately, the proof of this theorem far exceeds anything we have discussed so far, requiring explicit graph theory. There is no easy way to decompose the general existence of an isogeny ϕ from our above result into a series of compositions $\psi_1 \circ \cdots \circ \psi_n$ of ℓ-isogenies since the degrees may not match i.e. $\deg \phi \neq \ell^n$. The actual proof relies on analyzing the spectrum of the adjacency matrix of the ℓ-isogeny graph (i.e. the matrix whose entries are either 1 or 0 to indicate if two vertices share an edge), using $(\ell+1)$-regularity and symmetry of the adjacency matrix (our graph is undirected due to dual isogenies) to make conclusions using the eigenvalues and the Perron-Frobenius theorem [20].

We mention one last property of these torsion subgroups. Since $E[\ell] \cong \mathbb{Z}/\ell\mathbb{Z} \times \mathbb{Z}/\ell\mathbb{Z}$, $E[\ell]$ has a 2-dimensional basis as a $\mathbb{Z}$-module. Let $P \in E[\ell]$ be a point of order ℓ, and then pick an element $Q \in E[\ell] \setminus \langle P \rangle$ also of order ℓ. The set $\{[a]P + [b]Q : 0 \leq a, b < l\}$ clearly has ℓ^2 choices of a, b and is easy to check that these are distinct. A similar argument can be extended to powers of primes: $E[\ell^e] \cong \mathbb{Z}/\ell^e\mathbb{Z} \times \mathbb{Z}/\ell^e\mathbb{Z}$ and similarly also generated by two elements (see [59] for the full proof, but it is essentially the same observation as above).

6.2 SIDH/SIKE Protocols

With the above in mind, we can describe an upgraded version of the Diffie-Hellman Key Exchange from before, referred to as **Supersingular Isogeny Diffie-Hellman (SIDH)**, or **Supersingular Isogeny Key Encapsulation (SIKE)** [19][20][18]. It will be easiest describing the protocol first and connecting the rationale afterwards.

Supersingular Isogeny Diffie-Hellman:

1. **Public Parameters:** Alice and Bob begin by picking a public elliptic curve E and prime $p = p_a^{e_a} p_b^{e_b} - 1$, where p_i are prime. In particular, this prime is chosen such that $E(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(p+1)\mathbb{Z} \times \mathbb{Z}/(p+1)\mathbb{Z}$. Alice picks two public generating points, P_a, Q_a that generate $\langle P_a, Q_a \rangle = E[p_a^{e_a}] \cong \mathbb{Z}/p_a^{e_a}\mathbb{Z} \times \mathbb{Z}/p_a^{e_a}\mathbb{Z}$ (we get this structure since $p_a \nmid p$). Bob does the same for his prime: P_a, Q_a that generate $\langle P_b, Q_b \rangle = E[p_b^{e_b}]$.
2. **Secret Parameters**: Alice and Bob pick secret integers m_a, n_a, m_b, n_b and compute the corresponding secret elements $A = [m_a]P_a + [n_a]Q_a$ and $B = [m_b]P_b + [n_b]Q_b$. Using Vélu's formulas or otherwise, Alice and Bob compute the separable isogenies corresponding to their subgroups generated by A, B i.e. $\alpha : E(\overline{\mathbb{F}_{p^2}}) \to E/\langle A \rangle(\overline{\mathbb{F}_{p^2}})$ and $\beta : E(\overline{\mathbb{F}_{p^2}}) \to E/\langle B \rangle(\overline{\mathbb{F}_{p^2}})$. By our above comments, the $\deg(\alpha) = \ker(\alpha) = |\langle A \rangle| = p_a^{k_a}$ for some $k_a \leq e_a$ (by Lagrange's theorem), and similarly $\deg(\beta) = p_b^{k_b}$ for $k_b \leq e_b$. Hence these isognies correspond to walks of k_a, k_b steps in the p_a- and p_b-isogeny graphs.
3. **Public Exchange**: Alice sends $\{E/\langle A \rangle, \alpha(P_b), \alpha(Q_b)\}$ to Bob, and vice versa with $\{E/\langle B \rangle, \beta(P_a), \beta(Q_a)\}$.
4. **Shared Secret**: Alice computes with her secret parameters $[m_a]\beta(P_a) + [n_a]\beta(Q_a) = \beta(A)$, and creates the new isogeny $\alpha' : E/\langle B \rangle \to (E/\langle B \rangle)/\langle \beta(A) \rangle$. Consider $\alpha' \circ \beta : E \to (E/\langle B \rangle)/\langle \beta(A) \rangle$. Then

$$\begin{aligned}\ker(\alpha' \circ \beta) = \{P \in E : \beta(P) \in \ker(\alpha') = \langle \beta(A) \rangle\} &= \{P \in E : \exists k \text{ s.t. } P - [k]A \in \ker(\beta) = \langle B \rangle\} \\ &= \{P \in E : \exists k, \ell \text{ s.t. } P = [k]A + [\ell]B\} \\ &= \langle A, B \rangle\end{aligned}$$

   The isognies and their codomains of elliptic curves are uniquely determined (up to isomorphism) by their kernels, so $(E/\langle B \rangle)/\langle \beta(A) \rangle \cong E/\langle A, B \rangle$. Similarly, with what Alice shares, Bob computes $\beta' : E/\langle A \rangle \to (E/\langle A \rangle)/\langle \alpha(B) \rangle = E/\langle A, B \rangle$ with similar reasoning. Hence, both Alice and Bob reach the same elliptic curve isomorphism class i.e. j-invariant.

While we do not have direct commutativity—since our isogenies have different codomains—that the original Diffie-Hellman Key Exchange used $(g^a)^b = g^{ab} = (g^b)^a$, we leveraged the connectedness of the isogeny

graphs to still reach a shared secret piece of information, using secret isogenies to encode the information necessary for either party to essentially redo the first calculation the other person did.

Just as the Diffie-Hellman problem can be reduced to the more general DLP, we pose a similar general problem that if solved efficiently would also solve SIDH/SIKE:

Isogeny Computation Problem: Given two elliptic curves E, E' over a finite field that are isogenous of degree d, find an isogeny $\phi : E \to E'$ with $\deg(\phi) = d$.

If one could find an isogeny $\phi : E \to E/\langle A \rangle$ of degree $p_a^{k_a}$, one could then compute $\ker(\phi) = \langle A' \rangle = \langle A \rangle$ and find a generator A' (same subgroup, possibly different generator). Writing A' in terms of the public basis P_a, Q_a, they then could continue on to recover the entire shared secret $j(E/\langle A, B \rangle)$. But this problem on the whole is considered hard, for the formulation relies on instead of the algebraic hardness of a problem, on a structural issue of these highly-connected graphs; traversing and searching such a graph has no known solution*.

Some considerations:

- While we engineer our curve E and prime p to have a nice group structure, we can often pick twists of E that might optimize some of the group computations.
- We picked a prime of the form $p = p_a^{e_a} p_b^{e_b} - 1$ such that $E(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(p+1)\mathbb{Z} \times \mathbb{Z}/(p+1)\mathbb{Z}$. We do this so that $E[p_a^{e_a}], E[p_b^{e_b}] \leq E(\mathbb{F}_{p^2})$ are subgroups over the base field. While its true that these torsion points do form subgroups, they do so in the closure $\overline{\mathbb{F}_q}$ as that is how we defined the domains of isogenies and hence their kernels lie there. This choice of prime allows the subgroups and hence generators that Alice and Bob use to be in the simpler base field. This is not completely strict: we can also take a prime $p = p_a^{e_a} p_b^{e_b} + 1$ with $E(\mathbb{F}_{p^2}) \cong \mathbb{Z}/(p-1)\mathbb{Z} \times \mathbb{Z}/(p-1)\mathbb{Z}$ to get the similar nice structure we want. We can even include a small cofactor h and be fine with $p = h \cdot p_a^{e_a} p_b^{e_b} \pm 1$.
- The possible secrets lie in $A \in E[p_a^{e_a}], B \in E[p_b^{e_b}]$. To make sure no side is easier than the other to break, we try to keep the subgroups roughly the same size and hence $p_a^{e_a} \approx p_b^{e_b}$ (see more below).
- Finding isogenies of particular degrees correspond to taking walks of prescribed lengths through the isogeny graph. To make finding these walks as difficult as possible, we maximize the length and hence degree of the hidden isognies. So we pick A, B such that $|\langle A \rangle| = p_a^{e_a}$ and $|\langle B \rangle| = p_b^{e_b}$. This requires at least one coefficient coprime to the prime p_a (or p_b), so an easy way to do this is just fix $m_a = m_b = 1$.
- For convenience, we often take $p_a = 2, p_b = 3$. Since Alice and Bob still have to compute some isogenies themselves, like with Vélu's formulas, we keep the primes small so that the individual steps in the isogeny walk/calculations are fast.

How big of an exponent should we use? Naively, a meet-in-the-middle attack with Vélu's formulas would expect us to find, say, Alice's isogeny, in $O(p_a^{e_a/2})$: list all walks on the p_a-isogeny graph of length $\lfloor e_a/2 \rfloor$ starting from E, and then list all walks of length $\lceil e_a/2 \rceil$ starting from $E/\langle A \rangle$. There is a slightly more optimized attack based on a quantum search algorithm that runs in $O(p_a^{e_a/3})$ [60][20]. Since $p_a^{e_a} \approx p_b^{e_b}$, we have $p_a^{e_a} \approx \sqrt{p}$. Letting $n = \log p$, from these attacks we obtain $\approx (n/4)$-bit classical security, and $\approx (n/6)$-qubit quantum security.

Remark 6.20. While we only discussed the essential properties of ℓ-isogeny graphs, we did not give a full characterization of their properties. While $(\ell+1)$-regularity and connectedness alone give the impression of a "complex" and intertwined graph, it is completely formalized in the *expander* and *Ramanujan* properties of these graphs. Expander graphs are nice in that the furthest distance between any two nodes is not only $O(\log |V|)$ in the number of vertices, and moreover, a random walk on the graph in these number of steps converges to a uniform distribution on the vertices. Ramanujan graphs are optimally expander, in the sense that the time for convergence to uniformity is low. [19][18]. This ensures that for sufficiently long walks (by picking large enough exponents), there is no structural weakness within the graph itself that can be exploited for deducing the isogeny or not.

Remark 6.21. While not explicitly described here, Supersingular Isogeny Key Encapsulation (SIKE) is very similar to the SIDH protocol. There are attacks that leverage the reusing of the same secret parameters. Borrowing ideas from authentication and integrity protocols, SIKE takes the same SIDH protocol but makes the process asymmetric. Instead of a simultaneous exchange of curves and torsion points, only Alice will send her public key first. To prevent Bob from sending malicious, construed public parameters, he will instead make his private key a random hash of Alice's information with a random integer, and proceeds as normal. After computing the shared secret j, he sends over a hashed/obscured version of j. Alice can now compute j and verify and reverse engineer all the hashed values to check that Bob's information is as it should be, and that he did not try to tamper with the keys. The problem of computing isogenies still underlies the difficulty of breaking SIKE. See [18] for a worked example and description of the specific attack this resolves.

6.3 Unfortunate Recent Developments

There is an * that needs to be addressed. SIDH/SIKE provides a nice example of what quantum-resistant algorithms can look like, and in practice it was a strong contender for future implementations. *Was* is the key word. In July 2022, SIDH was not only broken, it was broken *with a classical attack* [15]. This attack was so efficient, that it broke SIKE with a 751 bit prime (a prime big enough to satisfy NIST's highest post-quantum security standard) in just about 3 hours. While it is true that solving SIDH reduces to solving ICP, the converse is not clear (just like with DLP and classical computational Diffie-Hellman problem). This attack strongly relies on the additional information given in the exchange $\{\alpha(P_b), \alpha(Q_b)\}$ beyond just the curve $E/\langle A\rangle$. The strategy falls far outside the scope of this essay, involving strategies of pairing and considering maps between surfaces instead of just curves, making much heavier use of the high-level algebra to justify computation strategies.

7 Other Post-Quantum Solutions and Further Directions

Despite the recent deprecation of SIDH/SIDH as specific isogeny-based cryptographic schemes, isogeny-based cryptography stil remains an open possibility. We have highlighted a possible such problem to base some post-quantum solutions, but there are other varieties that seem promising. We focused on isogeny-based cryptography for it built on some of the math behind elliptic curves we have been discussing. Here are a few highlights of other thought-to-be-hard problems in post-quantum settings (see Chapters 3 and 5 of [9]).

7.1 Lattice-Based Cryptography

One promising alternative is *lattice-based cryptography*. A lattice can be thought of as the integer points on a grid; instead of looking continuous combinations of vectors, we only look at a discrete ones.

Definition 7.1 (Lattice)**.** Let $\mathbf{X} = \{v_1, \cdots, v_n\}$ be a basis for $\mathbb{R}^n$. The lattice generated by $\mathbf{X}$ is $\langle v_1, \cdots, v_n\rangle_{\mathbb{Z}} = \{\sum_{i=1}^{n} a_i v_i : a_i \in \mathbb{Z}\}$.

The most basic problem that most other protocols build on is the **shortest vector problem (SVP)**.

Shortest Vector Problem: Given a basis $\mathbf{X}$, find the shortest vector in the lattice generated by $\mathbf{X}$.

"Shortest" can be defined via any norm, but is easiest thought with the Euclidean norm. All known algorithms (classical or quantum) for solving SVP suffer from a performance and accuracy trade-off: they either produce accurate approximations and run in non-polynomial time, or are fast and approximate the solution up to a non-polynomial factor.

7.2 Hash-Based Cryptography

Hash functions are a type of function that have nice cryptographic properties that make them essentially one-way functions. The two most important properties of hash functions are that they are hard to invert (i.e. given $y = H(x)$, it is hard to find x) and that they are *collision resistant* (i.e. it is hard to find x, x'

such that $H(x) = H(x')$). These both can be formalized as types of difficult problems, and form the basis for *hash-based cryptography.*

7.3 Efficiency and Security Comparisons

Algorithm	**Public key** (bytes)	**Ciphertext** (bytes)	**Key gen.** (ms)	**Encryption*** (ms)	**Decryption*** (ms)
curve25519 (ECC)	32	32	0.0222	0.0221	0.0221
RSA-2048	256	256	17.7245	0.0025	0.4173
SIKEp434 (I)	330	346	0.9781	1.5984	1.7095
Kyber512 (L)	800	768	0.0041	0.0058	0.0047
FrodoKEM-640 (L)	9616	9736	0.1536	0.2285	0.2287
SPHINCS-256 (H)	1056	41000	0.5011	8.9399	0.2618

Table 3: Performance comparison at NIST Level 1 security i.e. $\approx$ 128-bit classical security.

The above table describes the performance and speed of classical ECC and RSA methods, SIKE (isogeny based), Kyber512 and FrodoKEM (lattice based), as well as the SPHINCS signature (hash based) cryptographic schemes. All benchmarks were performed on an Intel Core i7-13700H (Raptor Cove microarchitecture, b06a2) with P-cores running at 4800MHz, using the SUPERCOP benchmarking suite (version 20231107) [10] using the original algorithms, i.e. not specially optimized for specific arithmetic/operations (and it tends to be in agreement with performance patterns from [45]). We denote (I), (L), and (H) to denote isogeny, lattice, and hash based protocols. Also, it is important to note that this is a bit of a forced comparison as we are comparing key exchange, key encapsulation, and signature methods to one another, so their purposes are not exactly the same. As such, the **Ciphertext** column denotes the general encrypted output (i.e. the shared secret, the signed message, etc.), and the **Encryption** and **Decryption** column is how long it takes to generate and read these outputs (for SPHINCS, the assumed size of messages to be signed are 59 bytes); they represent the time for the sender and receiver to obtain what they need (in ECC's key exchange, both have the same role in computing the shared secret). **Key generation** says how long it takes to create the public key used in these protocols. The table highlights roughly the expected overhead one would imagine of isogeny-based cryptography: the overhead is incredibly high by nature of the calculations involved. Deciding on a protocol to be used is primarily governed by whatever is most useful: are we generating many keys and need speeds to be fast, or do we have limited storage and bandwidth where key size is the primary factor.

8 Conclusion

This essay took a path from the ground up with the original key exchange problem to schemes that have yet to be physically implemented. We discussed the Diffie-Hellman Key Exchange, faults in it and the discrete logarithm problem, and how DLP alone influences the choice of groups. With the foundations set, we sketched the ideas of Hasse's bound that is the ultimate justification for elliptic curve cryptography and its efficacy. Then we took a detour into quantum computing models to rediscover Shor's algorithm and how superposition gives us access to new types of algorithms that gives speedups that no classical computer could dream of. Finally, we took our computational approach and diverted a bit more into the algebra of isogenies to describe a promising toy model of how post-quantum solutions could look. The histories of elliptic curves, cryptography, and quantum computing are deep and varied, with their intersections being brief but fruitful, and while giving a full explanation of them all would span thousands of pages, I hope this essay has given some intuitions and guiding principles behind all the topics and how they unify in a way that can make us optimistic of a safer and more protected future internet.

9 References

[1] Scott Aaronson. *Quantum Computing since Democritus*. QC174.17.M35A27. Paperback. Cambridge, UK: Cambridge University Press, 2013. ISBN: 978-0-521-19956-8. DOI: 10.1017/CBO9780511979309.

[2] David Adrian et al. "Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. Denver, Colorado, USA: ACM, 2015, pp. 5–17. DOI: 10.1145/2810103.2813707.

[3] N. Aghanim et al. "Planck 2018 results - VI. Cosmological parameters". In: *Astronomy and Astrophysics* 641 (Sept. 2020), pp. 644–654. DOI: 10.1051/0004-6361/201833910.

[4] Mirko Amico, Zain H. Saleem, and Muir Kumph. "An Experimental Study of Shor's Factoring Algorithm on IBM Q". In: *Physical Review A* 100 (2019). arXiv:1903.00768v3, p. 012305. DOI: 10.1103/PhysRevA.100.012305. arXiv: 1903.00768 [quant-ph].

[5] Liljana Babinkostova et al. "On Isomorphic K-Rational Groups of Isogenous Elliptic Curves Over Finite Fields". In: *arXiv preprint arXiv:2011.08471* (2020). arXiv: 2011.08471 [math.NT].

[6] Elaine Barker. *Recommendation for Key Management: Part 1 – General*. Special Publication 800-57 Part 1, Revision 5. National Institute of Standards and Technology, May 2020. DOI: 10.6028/NIST.SP.800-57pt1r5. URL: https://doi.org/10.6028/NIST.SP.800-57pt1r5.

[7] Stéphane Beauregard. "Circuit for Shor's algorithm using 2n+3 qubits". In: *Quantum Information and Computation* 3.2 (2003), pp. 175–185. DOI: 10.48550/arXiv.quant-ph/0205095.

[8] Daniel J. Bernstein. "Curve25519: new Diffie-Hellman speed records". In: *Security and Cryptography for Networks* (2006). Document ID: 4230efdfa673480fc079449d90f322c0, pp. 207–228. DOI: 10.1007/11745853_14.

[9] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, eds. *Post-Quantum Cryptography*. Mathematics Subject Classification (2000): 94A60. Printed on acid-free paper. Berlin Heidelberg: Springer-Verlag, 2009. ISBN: 978-3-540-88701-0. URL: springer.com.

[10] Daniel J. Bernstein and Tanja Lange. *eBACS: ECRYPT Benchmarking of Cryptographic Systems*. Diffie-Hellman benchmarks: https://bench.cr.yp.to/results-dh/amd64-raptor.html; Key-encapsulation mechanisms benchmarks: https://bench.cr.yp.to/results-kem/amd64-raptor.html; Digital signature benchmarks: https://bench.cr.yp.to/results-sign/amd64-raptor.html. 2023. URL: https://bench.cr.yp.to/ (visited on 03/13/2025).

[11] Daniel J. Bernstein et al. "Faster computation of isogenies of large prime degree". In: *Fourteenth Algorithmic Number Theory Symposium (ANTS XIV)*. Vol. 4. The Open Book Series. Mathematical Sciences Publishers, 2020, pp. 39–55. DOI: 10.2140/obs.2020.4.39. URL: https://doi.org/10.2140/obs.2020.4.39.

[12] Catherine Bolgar. *Microsoft's Majorana-1 chip carves new path for quantum computing*. https://news.microsoft.com/source/features/innovation/microsofts-majorana-1-chip-carves-new-path-for-quantum-computing/. Accessed: 2025-02-20. Microsoft, Feb. 2025.

[13] Reinier Broker and Peter Stevenhagen. *Constructing elliptic curves of prime order*. 2007. arXiv: 0712.2022 [math.NT]. URL: https://arxiv.org/abs/0712.2022.

[14] Johannes A. Buchmann. *Introduction to Cryptography*. Springer, 2000. ISBN: 0387950346.

[15] Wouter Castryck and Thomas Decru. "An efficient key recovery attack on SIDH". In: *IACR Cryptology ePrint Archive* 2022.975 (2022). Published in Theoretical Computer Science, 410(50):5285–5297. URL: https://eprint.iacr.org/2022/975.

[16] Lily Chen et al. *Report on Post-Quantum Cryptography*. NIST Internal Report NISTIR 8105. Available free of charge from: http://dx.doi.org/10.6028/NIST.IR.8105. Gaithersburg, Maryland: National Institute of Standards and Technology, Apr. 2016, p. 15. DOI: 10.6028/NIST.IR.8105.

[17] R. Cleve et al. "Quantum Algorithms Revisited". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 454.1969 (1998), pp. 339–354. DOI: 10.1098/rspa.1998.0164.

[18] Craig Costello. "Supersingular isogeny key exchange for beginners". In: *International Conference on Selected Areas in Cryptography* (2019), pp. 1–31. DOI: 10.1007/978-3-030-10970-7_1.

[19] Luca De Feo. "Mathematics of Isogeny Based Cryptography". In: *arXiv preprint arXiv:1711.04062* (2017). URL: https://arxiv.org/abs/1711.04062.

[20] Luca De Feo, David Jao, and Jérôme Plût. "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies". In: *Journal of Mathematical Cryptology* 8.3 (2014), pp. 209–247.

[21] Simon J. Devitt, Kae Nemoto, and William J. Munro. "Quantum Error Correction for Beginners". In: *arXiv preprint arXiv:0905.2794* (2009). arXiv:0905.2794v4 [quant-ph]. DOI: 10.1088/0034-4885/76/7/076001. URL: https://arxiv.org/abs/0905.2794.

[22] Whitfield Diffie and Martin Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* IT-22 (Nov. 1976), pp. 644–654.

[23] Martin Ekerå. "On the Success Probability of Quantum Order Finding". In: *ACM Transactions on Quantum Computing* 5.2 (May 2024). DOI: 10.1145/3655026.

[24] Hewlett Packard Enterprise. *Hewlett Packard Enterprise delivers world's fastest direct liquid-cooled exascale supercomputer, "El Capitan", for Lawrence Livermore National Laboratory*. 2024. URL: https://www.hpe.com/us/en/newsroom/press-release/2024/11/hewlett-packard-enterprise-delivers-worlds-fastest-direct-liquid-cooled-exascale-supercomputer-el-capitan-for-lawrence-livermore-national-laboratory.html (visited on 01/05/2024).

[25] Austin G. Fowler et al. "Surface codes: Towards practical large-scale quantum computation". In: *Phys. Rev. A* 86 (3 Sept. 2012), p. 032324. DOI: 10.1103/PhysRevA.86.032324. URL: https://doi.org/10.1103/PhysRevA.86.032324.

[26] Jay Gambetta. *Quantum Roadmap 2033*. https://www.ibm.com/quantum/blog/quantum-roadmap-2033. Accessed: 2025-02-20. IBM Quantum, Dec. 2023.

[27] Craig Gidney and Martin Ekerå. "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits". In: *Quantum* 5 (2021), p. 433. DOI: 10.22331/q-2021-04-15-433. arXiv: 1905.09749 [quant-ph].

[28] Daniel Gottesman. "An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation". In: *arXiv preprint arXiv:0904.2557* (2009). Includes expanded version of quant-ph/0004072 plus 30 pages of new material, p. 46. DOI: 10.48550/arXiv.0904.2557. arXiv: 0904.2557 [quant-ph].

[29] Torsten Hoefler, Thomas Häner, and Matthias Troyer. "Disentangling Hype from Practicality: On Realistically Achieving Quantum Advantage". In: *Communications of the ACM* 66.5 (May 2023), pp. 82–87. DOI: 10.1145/3571725.

[30] Dale Husemöller. *Elliptic Curves*. 2nd ed. Vol. 111. Graduate Texts in Mathematics. New York: Springer-Verlag, 2004. ISBN: 978-0-387-95490-5. DOI: 10.1007/978-0-387-21577-8.

[31] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC, 2014. ISBN: 9781466570276.

[32] Neal Koblitz. *A Course in Number Theory and Cryptography*. Springer, 1994. ISBN: 9780387942933.

[33] Thalia Laing and Tommy Charles. *Anticipating the Quantum Threat to Cryptography*. https://threatresearch.ext.hp.com/anticipating-the-quantum-threat-to-cryptography/. Accessed: 2025-02-20. HP Security Lab, Feb. 2024.

[34] Arjen K. Lenstra. "Key Lengths: Contribution to The Handbook of Information Security". In: (2010).

[35] Franck Leprévost et al. "Generating Anomalous Elliptic Curves". In: *Elsevier Science* (Nov. 2004). Preprint submitted to Elsevier Science.

[36] Kerry Maletsky. *RSA vs. ECC Comparison for Embedded Systems*. White Paper DS00003442A. Originally published by Atmel as document 8951A in July 2015. Microchip Technology Inc., Apr. 2020. URL: https://www.microchip.com/.

[37] Ueli M. Mauer. "Towards the Equivalence of Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms". In: *Advances in Cryptology* 839 (1994), pp. 271–281.

[38] Alfred J. Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. "Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field". In: *IEEE Transactions on Information Theory* 39.5 (1993), pp. 1639–1646.

[39] Dustin Moody. *Let's Get Ready to Rumble - The NIST PQC Competition*. Presentation. NIST Post-Quantum Cryptography Standardization Process. National Institute of Standards and Technology (NIST), 2018.

[40] National Institute of Standards and Technology. *Recommended Elliptic Curves for Federal Government Use*. Technical Report. Contains specifications for elliptic curves over prime and binary fields, including curve parameters, implementation details for modular arithmetic, and basis conversion methods. National Institute of Standards and Technology, July 1999.

[41] National Institute of Standards and Technology. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. NIST Call for Proposals. Federal Information Processing Standards (FIPS). National Institute of Standards and Technology, 2016. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization.

[42] V. I. Nechaev. "Complexity of a determinate algorithm for the discrete logarithm". In: *Mathematical Notes* 55 (Feb. 1994), pp. 165–172. DOI: 10.1007/BF02113297.

[43] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. 10th Anniversary Edition. Cambridge, UK: Cambridge University Press, 2010. ISBN: 978-1-107-00217-3.

[44] Francisco Orts, Gloria Ortega, and Ester M Garzón. "Studying the cost of n-qubit Toffoli gates". In: *International Conference on Computational Science*. Springer. 2022, pp. 108–121. DOI: 10.1007/978-3-031-08760-8_10.

[45] Christian Paquin, Douglas Stebila, and Goutam Tamvada. "Benchmarking Post-Quantum Cryptography in TLS". In: *Cryptology ePrint Archive* (Feb. 2020). Available at: https://github.com/xvzcf/pq-tls-benchmark.

[46] Carl Pomerance. "The Expected Number of Random Elements to Generate a Finite Abelian Group". In: *Periodica Mathematica Hungarica* 43.1-2 (2001), pp. 191–198.

[47] Thomas Pornin. *Efficient and Complete Formulas for Binary Curves*. Cryptology ePrint Archive, Paper 2022/748. Oct. 2022. URL: https://eprint.iacr.org/2022/748.

[48] John Preskill. *Physics 219/Computer Science 219: Quantum Computation*. http://theory.caltech.edu/~preskill/ph219/. First prepared 1997-98, with various chapters updated through 2020. 2024.

[49] John Proos and Christof Zalka. "Shor's discrete logarithm quantum algorithm for elliptic curves". In: *arXiv preprint quant-ph/0301141* (Jan. 2004). Version 2.

[50] Martin Roetteler et al. "Quantum Resource Estimates for Computing Elliptic Curve Discrete Logarithms". In: *arXiv preprint arXiv:1706.06752* 3 (Oct. 2017).

[51] René Schoof. "Counting points on elliptic curves over finite fields". In: *Journal de Théorie des Nombres de Bordeaux* 7.1 (1995), pp. 219–254.

[52] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172.

[53] Victor Shoup. "Lower Bounds for Discrete Logarithms and Related Problems". In: *Advances in Cryptology — EUROCRYPT '97* 1233 (July 2001), pp. 256–166. DOI: 10.1007/3-540-69053-0_18.

[54] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2009. ISBN: 9780521516440.

[55] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. 2nd ed. Vol. 106. Graduate Texts in Mathematics. New York: Springer, 2009. ISBN: 978-0387094939. DOI: 10.1007/978-0-387-09494-6.

[56] Joseph H. Silverman and John Tate. *Rational Points on Elliptic Curves*. Undergraduate Texts in Mathematics. New York: Springer-Verlag, 1992. ISBN: 0-387-97825-9.

[57] N. P. Smart. "The Discrete Logarithm Problem on Elliptic Curves of Trace One". In: *Journal of Cryptology* 12.3 (1999). Received 9 December 1997 and revised 11 March 1998, pp. 193–196. DOI: 10.1007/s001459900052.

[58] Stephanie Susnjara and Ian Smalley. *What is quantum-safe cryptography?* https://www.ibm.com/think/topics/quantum-safe-cryptography. Accessed: 2025-02-20. IBM Think, Sept. 2024.

[59] Andrew Sutherland. *Isogeny Kernels and Division Polynomials*. Lecture Notes for 18.783 Elliptic Curves. Lecture #5, Fall 2023. Sept. 2023.

[60] Seiichiro Tani. "Claw Finding Algorithms Using Quantum Walk". In: *Theoretical Computer Science* 410.50 (2009). Originally submitted on 20 Aug 2007; revised 3 Mar 2008, pp. 5285–5297. DOI: 10.1016/j.tcs.2009.08.030. arXiv: 0708.2584 [quant-ph]. URL: https://arxiv.org/abs/0708.2584.

[61] Joseph Tindall et al. "Efficient Tensor Network Simulation of IBM's Eagle Kicked Ising Experiment". In: *PRX Quantum* 5.010308 (Jan. 2024). DOI: 10.1103/PRXQuantum.5.010308.

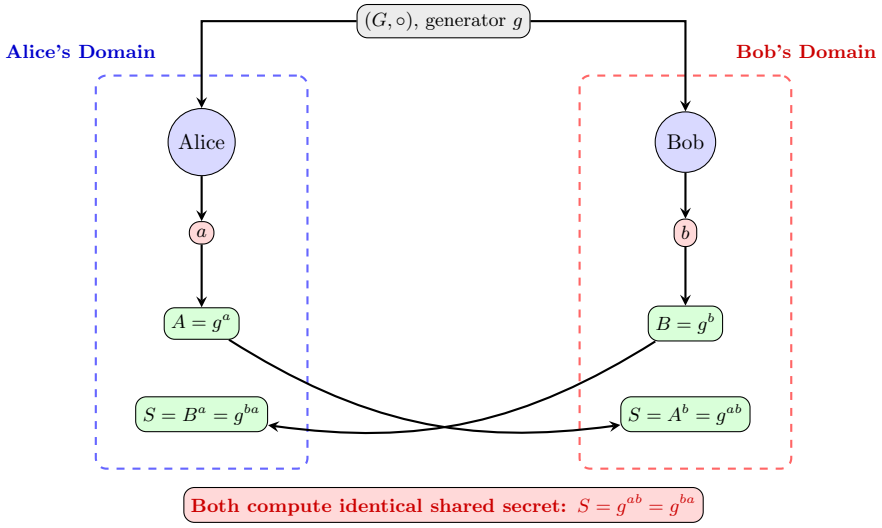
[62] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. 2nd ed. Discrete Mathematics and its Applications. Boca Raton: Chapman and Hall/CRC, 2008. ISBN: 978-1420071474. DOI: 10.1201/9781420071474.

[63] William C. Waterhouse. "Abelian varieties over finite fields". In: *Annales scientifiques de l'École Normale Supérieure*. 4th ser. 2.4 (1969), pp. 521–560. DOI: 10.24033/asens.1183. URL: http://www.numdam.org/item?id=ASENS_1969_4_2_4_521_0.

[64] Watson. *Answer to "Why all supersingular elliptic curves over $\overline{\mathbb{F}}_p$ are isogenous?"*. Licensed under CC BY-SA 3.0: https://creativecommons.org/licenses/by-sa/3.0/. Apr. 2023. URL: https://mathoverflow.net/questions/444846/why-all-supersingular-elliptic-curves-over-bar-mathbbf-p-are-isogenous (visited on 03/13/2025).

10 Appendix

10.1 Diffie-Hellman Visualization



10.2 Understanding the Pohlig-Hellman Algorithm

We give full proofs of Theorems 2.6 and 2.7 to reduce DLP for any group G to prime order subgroups of G.

Theorem 10.1. *Let G be a finite cyclic group of order $|G| = p_1^{e_1} \cdots p_k^{e_k} = n$ with generator g, and p_i are prime. Set*

$$n_i = n/p_i^{e_i},\ g_i = g^{n_i},\ a_i = a^{n_i},\ x_i = log_{g_i} a_i.$$

If $x \in \mathbb{Z}_n$ is a solution to the simultaneous congruences $x \equiv x_i \bmod p_i^{e_i}$, then $x = \log_g a$.

Proof. First note our defined variables and relations are well-defined. Since $o(g) = n$, then $o(g_i) = p_i^{e_i}$. Also, since g is a generator, for all $a \in G$ there exists l such that $g^l = a$. Hence $g_i^l = (g^{n_i})^l = (g^l)^{n_i} = a^{n_i} = a_i$ and so $a_i \in \langle g_i \rangle$ and hence $\log_{g_i} a_i = x_i$ exists and is well-defined.

Since $x \equiv x_i \bmod p_i^{e_i}$, we have for all $i = 1, \ldots, k$ that $(g^{-x}a)^{n_i} = (g^{n_i})^{-x} a_i = g_i^{-x_i} a_i = (a_i)^{-1} a_i = 1$. Hence $o(g^{-x}a) \mid n_i$ for all i and therefore $o(g^{-x}a) \mid \gcd(n_1, n_2, \cdots, n_k) = 1$. Hence $o(g^{-x}a) = 1$, i.e. $g^{-x}a = 1$ and $a = g^x$ as claimed. □

Thus instead of needing to solve DLP in groups of general order n, we only have to solve them in groups with prime power orders and use the Chinese Remainder Theorem to stitch together a solution. In fact, we can further reduce solving DLP to solving DLP into just *prime order* subgroups.

Theorem 10.2. *Let G be a finite cyclic group of order $|G| = p^e = n$ with generator g and p prime, and let $a \in G$ be an arbitrary element. Then we can find $x = \log_g a$ with e sub-DL problems in groups of order p.*

Proof. We want to solve $g^x = a$. Since $x \in \mathbb{Z}_{p^e}$, $x < p^e$. Now write x in its p-adic expansion: $x = x_0 + x_1 p + \cdots + x_{e-1} p^{e-1}$ with $0 \le x_i < p$ for $0 \le i < e - 1$. If we can determine every x_i, we will also have determined x. Raising our DLP to p^{e-1} on both sides, we have $g^{p^{e-1}x} = a^{p^{e-1}}$. Substituting the p-adic expression for x and noting $o(g) = p^e$ as it is a generator, we have

$$g^{p^{e-1}x} = g^{p^{e-1}\left(x_0 + x_1 p + \cdots + x_{e-1}p^{e-1}\right)} = g^{p^{e-1}x_0 + p^e\left(x_1 + \cdots + x_{e-1}p^{e-2}\right)} = g^{p^{e-1}x_0} g^{p^e\left(x_1 + \cdots + x_{e-1}p^{e-2}\right)} = g^{p^{e-1}x_0}.$$

Hence $\left(g^{p^{e-1}}\right)^{x_0} = a^{p^{e-1}}$, and since $o\left(g^{p^{e-1}}\right) = p$, x_0 is a discrete logarithm in a subgroup of order p. We now inductively determine the remaining coefficients x_i in the expansion. Suppose we have determined $x_0, \cdots, x_{k-1}$. Then

$$g^{x_0 + x_1 p + \cdots + x_{e-1}p^{e-1}} = a \quad \Longrightarrow \quad g^{x_k p^k + x_{k+1}p^{k+1} + \cdots + x_{e-1}p^{e-1}} = a g^{-\left(x_0 + x_1 p + \cdots + x_{k-1}p^{k-1}\right)} := a_k.$$

Call the RHS term a_k, and raise both sides to p^{e-k-1}. As above, terms simplify to $\left(g^{p^{e-1}}\right)^{x_k} = a_k^{p^{e-k-1}}$. This is a discrete logarithm in a group of order p. Hence, with e discrete logarithms in a group of order p, we can determine $x = \log_g a$ in $\mathbb{Z}_{p^e}$. □

The two theorems above give the **Pohlig-Hellman algorithm** to find discrete logarithms. To complete the algorithm, we only now need a general DLP solving algorithm, like the Baby-Step Giant-Step algorithm that runs in $O(\sqrt{n})$.

10.3 Some Notes on DLP Algorithms

10.3.1 Baby-Step Giant-Step: A General DLP Algorithm

For an arbitrary finite abelian group of order n, we need at least $O(\sqrt{n})$ group operations. While the proofs of the bounds are abstract, we in fact have algorithms that meet this bound. Below is one such algorithm that is commonly implemented with the Pohlig-Hellman algorithm when storage is not an issue.

The naive method to calculate discrete logarithms is to just check every possible value of x to see if $g^x = a$. But because of the division algorithm, we need only $\sqrt{n}$ elements of G to calculate all other elements. Let $m = \lceil \sqrt{n} \rceil$ and write $x = qm + r$ where $0 \le q < m$ and $0 \le r < m$. So we have $g^{qm+r} = a$, which implies that $(g^m)^q = ag^{-r}$. This suggests a two-loop search to find x: one loop involving "baby steps" of size g (iterating over r) and another loop of "giant steps" of size g^m (iterating over q).

Let $B = \{(ag^{-r}, r) : 0 \le r < m\}$ be our baby-steps. If the pair $(1, r) \in B$, then we have $ag^{-r} = 1$ i.e. $a = g^r$ and we're done. If we don't find such a pair, we store B, and compute the giant-steps $G = \{(g^m)^q : 0 \le q < m\}$. We now check if there is a value of q such that $(g^m)^q = ag^{-r}$ for some r in B. Rearranging this last equality gives $g^{qm+r} = g^x = a$ as desired. Since prime order groups have been the main focus, we implement that below. Otherwise the main issue is calculating g^{-1}, which can be done with the extended Euclidean algorithm from below (by identifying $C_n \cong \mathbb{Z}_n$).

Baby-Step Giant-Step Algorithm (Shanks 1969): Let G be a finite cyclic group of primer order $|G| = p$, $g \in G$ a generator, and $a \in G$ an element. We want to find x such that $g^x = a$.

```
def baby_step_giant_step(g,a,p):                # Solves g^x = a mod p
    m = int(ceil(sqrt(p - 1)))
    baby_steps = {}                             # Storing the baby steps as pair (ag^-r, r)
    g_inv = pow(g, p-2, p)                      # Compute g^-1 using Fermat's little theorem
    gs = a                                      # Initialize baby steps ag^-r

    for r in range(m):                          # First loop: baby steps
        if gs == 1:                             # Early exit if solution found
            return r
        baby_steps[gs] = r                      # Add to baby step dictionary
        gs = (gs * g_inv) % p                   # Incrememnt baby step by 1

    delta = pow(g, m, p)                        # g^m is giant step value
    giant = 1                                   # Initialize giant steps g^m

    for q in range(m):                          # Second loop: giant steps
        if giant in baby_steps:                 # Check if (g^m)^q = ag^-r for some r
            return q * m + baby_steps[giant]
        giant = (giant * delta) % p             # Increment giant step by 1

    return False                                # False if nothing found
```

The major cost of this algorithm is storing `baby_steps` and efficiently looking up elements from it, but otherwise, this is an optimal algorithm to calculate discrete logarithms in any group, meeting the bound of Nechaev-Shoup. If we do not care about generic algorithms and are okay working in specific groups, there are more specialized algorithms that leverage the structure of the specific group.

Remark 10.3. Some might wonder why we use $m = \lceil \sqrt{n} \rceil$ as our loop values. For a given loop value m, we will need to do m passes on the baby step loop, and n/m passes on the giant step loop. If we want to minimize the total number of passes $m + n/m$, it is easy to see that this value is minimized when $m = \sqrt{n}$.

10.3.2 Using $\mathbb{Z}_p^\times$, and the Weakness of Factorization

While $\mathbb{Z}_p^\times$ is a natural group to consider for how well-understood it is in both structure and operations, a glaring issue in the light of DLP, however, is that $|\mathbb{Z}_p^\times| = p - 1$ is not prime, so we typically work in a subgroup instead of the whole group. Here is one method to generate such a subgroup without having to worry about factoring $|\mathbb{Z}_p^\times|$ and using a generator from above [31].

Theorem 10.4. *Let $p = rq + 1$ with p, q prime. Then $G = \{h^r \bmod p : h \in \mathbb{Z}_p^\times\}$ is a subgroup of order q.*

Proof. The idea is that we quotient out all the cycles of length r, leaving us with a group of order q. Let $\phi : \mathbb{Z}_p^\times \to G$ be the homomorphism $\phi(h) = h^r \bmod p$. By the First Isomorphism Theorem, we have $\mathbb{Z}_p^\times / \ker\phi \cong \operatorname{Im}\phi = G$. We now show $|\ker\phi| = r$.

Let g be a generator for $\mathbb{Z}_p^\times$, and suppose $g^i \in \ker\phi$ i.e. $(g^i)^r = 1$ and so $p - 1 \mid ir$. Since $p - 1 = rq$, we have $q \mid i$. Therefore, the possible values of i that satisfy $g^i \in \ker\phi$ are $\{0, q, 2q, \cdots, (r-1)q\}$. Hence, $|\ker\phi| = r$, and $|G| = |\mathbb{Z}_p^\times|/|\ker\phi| = (p-1)/r = q$. □

Finding this group is easy: just exponentiate every element in $\mathbb{Z}_p^\times$ by r and remove duplicates. Also since it is prime order, every element is a generator and so we can use every element in our use of the Key Exchange as we please.

10.3.3 Index Calculus Attack

We have discussed that general algorithms can solve DLP in at best $O(\sqrt{n})$ and shown such an algorithm, but that does not mean that we cannot do better if we tailor an algorithm to a specific group. $\mathbb{Z}_p^\times$ is one such group in which DLP can be solved in subexponential time with *index calculus methods* [14][31][54]. It is an algorithm that is adapted from ones to factor integers. To solve DLP, the idea is that instead of solving a single DLP in $O(\sqrt{n})$, we can solve *many* DLPs simultaneously via a system of linear equations, and stitch together a solution. For this section, suppose that we are trying to solve $g^x \equiv a \bmod p$ with generator g.

Let B be an integer bound, and let $F(B) = \{q \leq B : q \text{ primes}\}$ be our *factor base*. We say an integer is B-smooth if its prime factors are in $F(B)$. We now solve DLP in two steps.

Step 1: Fix a bound B, and let $k = |F(B)|$ be the number of primes in our factor base. Then find distinct $x_1, \cdots, x_k$ such that $g_i \equiv g^{x_i} \bmod p$ are all B-smooth (i.e. just randomly pick x_i until enough are found). Since these are B-smooth, we can factor them in $F(B)$:

$$\begin{aligned} g^{x_1} \equiv g_1 &\equiv \prod_{q \in F(B)} q^{e_{1,q}} \bmod p \\ &\vdots \\ g^{x_k} \equiv g_k &\equiv \prod_{q \in F(B)} q^{e_{k,q}} \bmod p \end{aligned}$$

with nonnegative exponents $e_{i,q}$. Taking discrete logarithms:

$$\begin{aligned} x_1 &\equiv \sum_{q \in F(B)} e_{1,q} \log_g q \bmod (p-1) \\ &\vdots \\ x_k &\equiv \sum_{q \in F(B)} e_{k,q} \log_g q \bmod (p-1) \end{aligned}$$

where $\log_g q$ is unknown. But, this is a system of k linear equations in k unknowns $\{\log_g q\}_{q \in F(B)}$, so we can solve for $\log_g q$ for all $q \in F(B)$ efficiently (i.e. with linear algebra).

Step 2: Then, find an exponent $y \in \mathbb{Z}_p^\times$ such that $ag^y \bmod p$ is B-smooth (i.e. by sampling random y). Similarly, since it is B-smooth, we can factor $ag^y \equiv \prod_{q \in F(B)} q^{e(q)} \bmod p$ (for new nonnegative exponents $e(q)$). Taking the discrete logarithm:

$$\log_g a + y \equiv \sum_{q \in F(B)} e(q) \log_g q \bmod (p-1)$$

which allows us to solve for $\log_g a$ since since we determined $\log_g q$ for all $q \in F(B)$ in the previous step.

10.3.4 Analysis for Index Calculus

This works for $\mathbb{Z}_p^\times$ and not other groups in general is because of the need for a notion of "prime" elements to do the factorizations. In terms of complexity, the primary bottleneck in index calculus methods is picking a bound B, finding B-smooth elements, and factoring g^{x_i} at each step. For larger B, the more likely our numbers chosen will be B-smooth, but the larger the system of equations we would have to solve. In particular, the complexity of this algorithm can be shown to be $O(\exp(\sqrt{\ln(p)\ln(\ln p)}))$ (with particular constants able to be determined with more complex analysis). The main cost is in generating the desired linear relations and factoring, but notice that this step only involves the public knowledge involved in DLP: p (by knowledge of the group $\mathbb{Z}_p^\times$) and g; the knowledge of the particular DLP to solve for element a is only used in Step 2. So we can use Step 1 as pre-processing step, and compute lots of discrete logarithms in a group quickly with a one-time heavy cost.

10.3.5 Homogeneous Coordinates for ECDLP

With the need to include a "point at infinity" ∞ that exists somewhat artificially in our group structure, it is natural to explore the topic in the context of projective planes and homogeneous coordinates. Instead of considering points (x, y), we consider $(X, Y; Z)$. We then consider the affine point $(x, y) = (x, y; 1)$, or more generally, to be the equivalence class of homogeneous points of the form $(xZ, yZ; Z)$ as for non-zero Z this uniquely identifies $(x, y; 1)$. Substituting $x = X/Z$ and $y = Y/Z$, we see that our elliptic curve equation becomes $Y^2Z = X^3 + aXZ^2 + bZ^3$. If we let $Z = 0$, then we are left with $X^3 = 0$ i.e. $X = 0$. Thus we have a singular equivalence class of $(0, 1; 0)$ that does not "nicely" map to one of our affine points. This point we call $\infty = (0, 1; 0)$ is our point at infinity, formalizing this idea of ∞ being above all the vertical lines simultaneously in affine space as it literally is the intersection of the y-axis and a line at infinity.

Besides unifying the geometry a bit more, there are real practical implications. Notably, having access to a class of points to represent a single coordinate (x, y) allows us to rationalize denominators within the field, avoiding the need to calculate inverses (which can be expensive!). Recall our point addition formulas $R = P \oplus Q$:

$$R = \left(\left(\frac{y_Q - y_P}{x_Q - x_P} \right)^2 - x_P - x_Q, -\left(\frac{y_Q - y_P}{x_Q - x_P}(x_R - x_P) + y_P \right) \right).$$

Converting these into homogeneous coordinates $P = (x_P, y_P) = (X_P/Z_P, Y_P/Z_P; 1), Q = (x_Q, y_Q) = (X_Q/Z_Q, Y_Q/Z_Q; 1)$:

$$\begin{aligned} R &= \left(\left(\frac{Y_Q/Z_Q - Y_P/Z_P}{X_Q/Z_Q - X_P/Z_P} \right)^2 - X_P/Z_P - X_Q/Z_Q, -\left(\frac{Y_Q/Z_Q - Y_P/Z_P}{X_Q/Z_Q - X_P/Z_P}(x_R - X_P/Z_P) + Y_P/Z_P \right); 1 \right) \\ &= \left(\left(\frac{Y_QZ_P - Y_PZ_Q}{X_QZ_P - X_PZ_Q} \right)^2 - X_P/Z_P - X_Q/Z_Q, -\left(\frac{Y_QZ_P - Y_PZ_Q}{X_QZ_P - X_PZ_Q}(x_R - X_P/Z_P) + Y_P/Z_P \right); 1 \right). \end{aligned}$$

Since we can pick any value of Z_R to scale our homogeneous coordinates by, we can rationalize X_R and Y_R to remove all instances of division. If we let $Z_R = Z_1Z_2(X_QZ_P - X_PZ_Q)^3$, we can clear all denominators; every term is just solely in terms of addition and multiplication. Hence instead of needing to calculate lots of multiplicative inverses during computation, we can do everything in homogeneous coordinates without division, and spend one division operation on converting back to affine coordinates $R = (x_R, y_R)$.

10.4 Understanding The Nechaev-Shoup Theorem

The Nechaev-Shoup Theorem states that any non-group specific discrete logarithm solving algorithm must run in *at least* $O(\sqrt{n})$ [42][53]. The Theorem works on the principle of a *generic groups*, which formalizes this abstraction away from the specifics of a given group. When working with a generic group, 1) we encode elements with random bit strings, 2) we can compute the group operation via a "black box" oracle ■ that takes in 2 elements and outputs another (mathematical operation), and 3) we can check if any two elements we have are equal (logical operation). An algorithm in this scenario only starts as input (the bit-encodings of) the identity 1, the generator g, and the target element $h = g^x$. The point of encoding our elements is that we cannot use any meta-knowledge we may have regarding the elements in the group (like in $\mathbb{Z}_p^\times$). So for convenience, I will just write the group elements or outputs from our oracle ■ as they are, but just know that what is intended is the bit encoding and that we essentially lack all knowledge of the element except its name (this includes powers; even if I write g and g^x, in the model, we do not outright know g^x is a power of g unless we explicitly compute it and compare them for equality).

The idea is that with only g, h known at the beginning, we can iteratively construct new group elements (via the oracle). But the new elements will only be of the form $g^\alpha h^\beta = g^{\alpha+\beta x}$ for some integers α, β. So if a choice of α_1, β_1 creates an element we have seen before, say for α_2, β_2, then we have learned something about the secret discrete logarithm x via the relation $\alpha_1 + \beta_1 x = \alpha_2 + \beta_2 x$. This does not explicitly determine x, but it is new information regarding relations between x.

Theorem 10.5 (Nechaev-Shoup)**.** *Let G be a group of prime order p and $\mathcal{A}$ be a probabilistic algorithm. If $\mathcal{A}$ makes m mathematical operations via the oracle, then the probability that $\mathcal{A}$ solves the DLP is*

$$\mathbb{P}\left(\mathcal{A}(p,g,g^x)=x\right) < \frac{(m+2)^2}{2p} + \frac{1}{p}$$

Sketch of Shoup's Proof. The idea is as above: the only way $\mathcal{A}$ learns information about the group elements is by computing the product of elements to find relations between them, and these relation will take place as the exponent expressions above. So we will create two lists: $\mathcal{L}_{\text{elem}} \subseteq G$ keeps track of our (encoded) group elements we have computed thus far, and $\mathcal{L}_{\text{exp}} \subseteq \mathbb{Z}[t]$ that keeps track of the linear expressions of the exponents $F_i(t) = \alpha_i + \beta_i t$ that represent each element $g_i \in \mathcal{L}_{\text{elem}}$. Remember, all of our group combinations are of the form $g^\alpha h^\beta = g^{\alpha+\beta x}$, so we can identify elements via their exponent $\alpha + \beta x$, so for the element $g_i \in \mathcal{L}_{\text{elem}}$, we have a corresponding expression $F_i(t) = \alpha_i + \beta_i t \in \mathcal{L}_{\text{exp}}$. So initially, $\mathcal{L}_{\text{elem}} = \{g, g^x\}$ and $\mathcal{L}_{\text{exp}} = \{1, t\}$.

Now we model the most general approach any such algorithm $\mathcal{A}$ can take solving DLP. All such $\mathcal{A}$ will ask the oracle ■ to create new elements via products of previous known elements in $\mathcal{L}_{\text{elem}}$. To $\mathcal{A}$, since there is no knowledge on the relationships between group elements (i.e. random bit-encoded), the oracle seems to output random elements. Note, since all of our exponents are of the form $\alpha_i + \beta_i t$, we see that

$$■(g_i, g_j) = ■(g^{\alpha_i+\beta_i t}, g^{\alpha_j+\beta_j t}) = g^{\alpha_i+\beta_i t} g^{\alpha_j+\beta_j t} = g^{(\alpha_i+\alpha_j)+(\beta_i+\beta_j)t}$$

This suggests we follow a procedure of computation with the oracle via our expression list $\mathcal{L}_{\text{elem}}$: if $\mathcal{A}$ requests $■(g_i, g_j)$, we look up the expression $F_i(t) + F_j(t) = (\alpha_i + \alpha_j) + (\beta_i + \beta_j)t = \alpha_{n+1} + \beta_{n+1} t$ in $\mathcal{L}_{\text{exp}}$. If there is such an expression $\alpha_{n+1} + \beta_{n+1} t = \alpha_k + \beta_k t \in \mathcal{L}_{\text{exp}}$ (i.e. we already found this expression), we let ■ return the corresponding $g_k \in \mathcal{L}_{\text{elem}}$. If there is not such an expression, we let the oracle return a random new element g_{n+1} to $\mathcal{L}_{\text{elem}}$ not already in it (like the random product element ■), and add $F_{n+1} = F_i(t) + F_j(t)$ to $\mathcal{L}_{\text{exp}}$.

We let $\mathcal{A}$ do its m requests to ■, and generate up to m new elements for $\mathcal{L}_{\text{elem}}$ and $\mathcal{L}_{\text{exp}}$. The key detail, now, is that these conversations between $\mathcal{A}$ and ■ represent the most general approach an algorithm could take to solve DLP. And since there is no knowledge of the individual group elements (they are encoded), the output of $\mathcal{A}(p, g, g^x)$ would appear to be independent of the input x (and hence independent of g^x). Suppose the output is $\mathcal{A}(p, g, g^x) = y$, and *after* pick a random value of the exponent/discrete logarithm x^* we wish to compute. There are two ways $\mathcal{A}$ could be successful.

1. $y = x^*$ i.e. $\mathcal{A}$ happened to get lucky.
2. $F_i(x^*) = F_j(x^*)$ for $i \neq j$ i.e. $\mathcal{A}$ has found a non-trivial relation regarding x^* and gained information.

The probability that $x^* = y$ after y is outputted is $1/p$ since we specified $o(g) = p$ and so there are p choices for the exponent. Similarly, note $F_i(t)$ is a linear polynomial for all i, and so $F_i(t) - F_j(t)$ is also linear and hence has 1 root. The probability that x^* is this root is also $1/p$, but we add the factor $(m+2)(m+1)/2$ to account for all possible pairs of equations $i \neq j$ (with an upper limit of at most m possible equations in $\mathcal{L}_{\text{elem}}$). These success conditions are far more generous than the requirement of actually solving DLP. Hence the success rate that $\mathcal{A}$ solves DLP is bounded:

$$\mathbb{P}\left(\mathcal{A}(p,g,g^x)=x\right) < \frac{1}{p} \cdot \frac{(m+2)(m+1)}{2} + \frac{1}{p} < \frac{(m+2)^2}{2p} + \frac{1}{p} \qquad \square$$

Remark 10.6. Often, the collision between $F_i(x^*) = F_j(x^*)$ is enough to solve the DLP entirely. The collision implies $\alpha_i + \beta_i x^* = \alpha_j + \beta_j x^*$ i.e. $x^* = (\alpha_j - \alpha_i)/(\beta_i - \beta_j)$ with division taking place in the field $\mathbb{Z}_p$. So long as $\beta_i \neq \beta_j$, these collisions are really detrimental. We said collisions only give non-trivial information to make it clear what the point of these collisions are for (and to ignore the cases when $\beta_i = \beta_j$), but it is worth highlighting just *how* non-trivial this information is.

Corollary 10.7. *Any generic group DLP solving algorithm runs in at least $\Omega(\sqrt{p})$, and hence at best $O(\sqrt{p})$.*

Proof. We have an upper bound on the success probability of any such algorithm $\mathcal{A}$. If we want $\mathcal{A}$ to succeed with probability at least, say, $1/2$ (or any number bounded away from 0), we see with simple algebra that

$$\begin{aligned} \frac{1}{2} &< \frac{(m+2)^2}{2p} + \frac{1}{p} \\ p &< (m+2)^2 + 2 \\ \sqrt{p-2} - 2 &< m \end{aligned}$$

m denotes the number of operations/requests we made to the oracle ■. Hence if we want bounded-away-from-0 probability of success, we need at least $\Omega(\sqrt{p})$ operations. □

10.5 Proof of Endomorphism Standard Form

Proposition 10.8 (Endomorphism standard form)**.** *Every endomorphism can be written as* $\alpha(x,y) = (r_1(x), r_2(x)y)$ *for rational functions* $r_1(x), r_2(x) \in \overline{\mathbb{K}}[x]$.

Proof. Let $R(x,y)$ be a rational function. Since $y^2 = x^3 + ax + b$, for all $(x,y) \in E(\overline{\mathbb{K}})$, we can replace every even power of y with a polynomial in x. With this substitution, we will have at most one power of y in the numerator and denominator:

$$R(x,y) = \frac{p_1(x) + p_2(x)y}{p_3(x) + p_4(x)y}$$

with $p_i(x)$ polynomials. Rationalizing the denominator via $p_3(x) - p_4(x)y$ and again replacing $y^2 = x^3 + ax + b$:

$$R(x,y) = \frac{q_1(x) + q_2(x)y}{q_3(x)} \tag{1}$$

for some polynomials $q_i(x)$.

Now let $\alpha(x,y) = (R_1(x,y), R_2(x,y))$. Since α is a homomorphism and $(x,-y) = -(x,y)$ in the group $E(\overline{\mathbb{K}})$, we have $\alpha(x,-y) = \alpha(-(x,y)) = -\alpha(x,y) = (R_1(x,y), -R_2(x,y))$. Therefore, $R_1(x,-y) = R_1(x,y)$ and so if we write R_1 in the form of (1), $q_2(x) \equiv 0$ in this case. Similarly $R_2(x,-y) = -R_2(x,y)$ forcing $q_1(x) \equiv 0$. Thus, $\alpha(x,y) = (r_1(x), r_2(x)y)$ for rational polynomials r_1, r_2. □

10.6 Some Useful Algorithms

10.6.1 Extended Euclidean Algorithm

The Euclidean algorithm has been known for centuries to calculate the greatest common divisor of two integers.

Consider we want to find $\gcd(m,n)$. We can write $m = a_1 n + r_1$ for $0 \le r_1 < n$. Now note that if a number d divides m, n, then by this equation d must also divide $r_1 = m - a_1 n$. Likewise, if d divides divides n, r_1 it must also divide a. Hence it must be that $\gcd(m,n) = \gcd(n, r_1)$. We can repeat the division algorithm again and write $n = a_2 r_1 + r_2$ for $0 \le r_2 < r_1$. With similar reasoning, $\gcd(n, r_1) = \gcd(r_1, r_2)$. We continue to iterate this:

$$\begin{aligned} m &= a_1 n + r_1 & 0 &\le r_1 < n \\ n &= a_2 r_1 + r_2 & 0 &\le r_2 < r_1 \\ r_1 &= a_3 r_2 + r_3 & 0 &\le r_3 < r_2 \\ &\vdots & &\vdots \\ r_{n-3} &= a_{n-1} r_{n-2} + r_{n-1} & 0 &\le r_{n-1} < r_{n-2} \\ r_{n-2} &= a_n r_{n-1} \end{aligned}$$

Since $r_1 > r_2 > \cdots > r_{n-1}$ form a sequence of strictly decreasing and non-negative integers, this process must eventually terminate with all $r_m = 0$ after a certain point m *in a finite number of steps*. This termination justifies this as an algorithm, eventually producing a result. By induction and extending the reasoning from above, we have

$$\gcd(m,n) = \gcd(n,r_1) = \gcd(r_1,r_2) = \gcd(r_2,r_3) = \cdots = \gcd(r_{m-2},r_{m-1}) = \gcd(r_{m-1},0) = r_{m-1}$$

This is easy to implement, especially when we have access to modulo operators.

```
def euclidean_gcd(m, n):
    if n == 0: return m
    else: return gcd(n, m % n)
```

An important corollary for us in particular is Bézout's Lemma:

Corollary 10.9 (Bézout's Lemma)**.** *For all $m,n \in \mathbb{Z}$, there exists $x,y \in \mathbb{Z}$ such that $mx+ny = \gcd(m,n)$.*

Proof. We take the same sequence of integers r_i from before, and work our way up from the bottom, noting that we can write r_i in terms of r_{i-1}, r_{i-2}:

$$\gcd(m,n) = r_{m-1} = r_{m-3} - a_{m-1}r_{m-2} = (r_{m-5} - a_{m-3}r_{m-4}) - a_{m-1}(r_{m-4} - a_{m-2}r_{m-3}) = \cdots$$

While tedious, eventually we will only get terms of two initial values $r_{-1} = m, r_0 = n$. □

Now we can calculate inverses modulo n easily. Say $\gcd(m,n) = 1$ (else m would not have a multiplicative inverse). By Bézout's Lemma, there are $x,y \in \mathbb{Z}$ such that $mx+ny = 1$. Reducing this modulo n, we have $mx \equiv 1 \bmod n$, hence $x \bmod n$ is our multiplicative inverse of m. We sometimes call this process the *extended Euclidean algorithm*. We will use a similar recursive call like in the normal Euclidean algorithm. Recall that $g = \gcd(m,n) = \gcd(n, m\%n)$. Suppose we have found integers s,t such that $sn + t(m\%n) = g$. Note $m\%n = m - \lfloor m/n \rfloor \cdot n$. Hence

$$g = sn + t(m\%n) = sn + t(m - \lfloor m/n \rfloor \cdot n) = (s - t\lfloor m/n \rfloor)n + tm$$

Since we want $mx+ny = 1$, we compare coefficients and add to our recursion $x = t, y = s - t\lfloor m/n \rfloor$.

```
def extended_gcd(m, n):                    # Given m,n return (gcd(m,n),x,y) with mx+ny=gcd(m,n)
    if n == 0:
        return (m, 1, 0)                   # If n=0, then gcd(m,n)=m and 1*m + 0*n = gcd(m,n)
    else:
        gcd, s, t = extended_gcd(n, m % n) # Reduce with Euclidean algorithm
        return (gcd, t, s - t * (m//n))

def mod_inverse(m, n):                     # Find m^-1 mod n
    gcd, x, y = extended_gcd(m, n)         # mx + ny = gcd(m,n)
    if gcd != 1:
        return None                        # Inverse does not exist
    else:
        return x % n                       # mx = 1 mod n
```

10.6.2 Fast Exponent and Double-and-Add

Our discussion frequently references being able to efficiently calculate group operations in $\mathbb{Z}_N^\times$ and $E(\mathbb{F}_q)$. This is an integral component of using these groups in DLP, and as such what justifies being able to also break them with Shor's algorithm. Here we give a common algorithm that computes $g^n \bmod p$ and nP in $O(\log n)$ group operations. The idea is to write $n = b_0 b_1 b_2 \cdots b_k$ in binary, and use repeated squaring/point doublings to "hone in" on n faster than just repeated additions. The main property to calculate $g^n = g^{b_0 2^0 + b_1 2^1 + \cdots + b_k 2^k}$ used is that we can easily calculate squares iteratively: note $(g^{2^k})^2 = g^{2 \cdot 2^k} = g^{2^{k+1}}$.

```
def fast_expt(g, n):          # Given g in group G, calculate g^n
    if n == 0: return 1       # g^0 = 1
    output = 1                # Tracks output: initialize at 1, and add powers of 2 of g
    while n > 0:              # Reading from least significant to most significant bit
      if (n % 2 == 1):        # Check if n odd i.e. current bit is 1,
        g = g * output        # and add factor of g^(2^k) to y if so
        n = n - 1
      g = g * g               # Squaring/doubling of g to move onto next power of g^(2^k)
      n = n / 2
    return output
```

Here we let g be our group element, n the number of repeated operations, and $*$ denote the group operation (i.e. multiplication for $\mathbb{Z}_p^\times$ or $\oplus$ addition for $E(\mathbb{F}_q)$). There are other similar ways of implementing this i.e. one can instead read from the most significant to least significant bit and do the y-additions before the squarings, but the big picture remains the same: by looking at the binary expansion of n, we only need $\lceil \log_2 n \rceil$ squarings with at most $\lceil \log_2 n \rceil$ additions/adjustments to y to store the calculation. This is as opposed to needing n operations just computing $g * g * \cdots * g$ directly.

10.7 Existence of Finite Fields

We first outline some basic facts about fields (adapted from *Part A: Rings and Modules* and *Part B: Galois Theory*).

Definition 10.10 (Field)**.** A set $\mathbb{K}$ equipped with two commutative operations $+$ and $\times$ is a *field* if $(\mathbb{K}, +)$ and $(\mathbb{K} \setminus \{0\}, \times)$ are groups with identity elements 0 and 1 respectively. Also, $\forall a, b, c \in \mathbb{K}$ $a \times (b + c) = a \times b + a \times c$ ($\times$ distributes over $+$). For convenience, we often suppress $\times$ and write $a \times b$ as ab.

Example 10.11. $\mathbb{R}, \mathbb{Q}, \mathbb{C}, \mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ for prime p are all fields with the natural definitions of $+$ and $\times$.

Definition 10.12 (Characteristic)**.** The *field characteristic* $\mathrm{Char}(\mathbb{K})$ is the least integer n such that $\overbrace{1 + 1 + \cdots + 1}^{\text{n times}} = 0$. If there is no such integer, the field is said to have characteristic 0.

Remark 10.13. It is natural to want to shorthand $n \cdot 1 := \overbrace{1 + 1 + \cdots + 1}^{\text{n times}}$. We may do this, but we must differentiate between two notations of multiplication: $\times$ between field elements, and $\cdot$ between an integer and a field element. Context will determine what type of multiplication we are referring to.

Proposition 10.14. *Let $a, b \in \mathbb{K}$ in a field. If $ab = 0$, then $a = 0$ or $b = 0$ (i.e. fields have no zero divisors).*

Proof. Let $a, b \in \mathbb{K}$. WLOG, let $a \neq 0$ and say $ab = 0$. Multiply both sides by a^{-1}: $a^{-1}(ab) = a^{-1}(0) \Rightarrow b = 0$. □

Proposition 10.15. *The characteristic of a field is either prime or 0.*

Proof. Say $\mathrm{Char}(\mathbb{K}) = ab$ for positive integers a, b. Then $0 = \overbrace{1 + \cdots + 1}^{\text{ab times}} = (\overbrace{1 + \cdots + 1}^{\text{a times}})(\overbrace{1 + \cdots + 1}^{\text{b times}})$. Fields have no zero divisors, so $(\overbrace{1 + \cdots + 1}^{\text{a times}}) = 0$ or $(\overbrace{1 + \cdots + 1}^{\text{b times}}) = 0$, contradicting the minimality of $\mathrm{Char}(\mathbb{K}) = ab$. □

Proposition 10.16. *If $\mathrm{Char}(\mathbb{K}) = p$, then $(a + b)^p = a^p + b^p$ for all $a, b \in \mathbb{K}$.*

Proof. $(a + b)^p = \sum_{n=0}^{p} \binom{p}{n} a^n b^{p-n}$, and $p \mid \binom{p}{n}$ for $0 < n < p$ and hence those terms reduce to 0 in $\mathbb{K}$. □

Corollary 10.17. *If* $\text{Char}(\mathbb{K}) = p$*, then for all* k*,* $(a+b)^{p^k} = a^{p^k} + b^{p^k}$*.*

Proof. We can show this inductively. We showed it is true for $k = 1$, and say it holds for $n-1$. Then $(a+b)^{p^n} = ((a+b)^{p^{n-1}})^p = (a^{p^{n-1}} + b^{p^{n-1}})^p = a^{p^n} + b^{p^n}$. □

Theorem 10.18. *There is a unique (up to isomorphism) finite field* $\mathbb{K}$ *of order* q *if and only if* $q = p^n$ *for some prime* p *and integer* n*.*

Proof. ($\Rightarrow$) $\text{Char}(\mathbb{K}) = p$ for some prime, since if $\text{Char}(\mathbb{K}) = 0$ then $\mathbb{K}$ would be infinite (by generating new elements repeatedly adding 1 to itself). Then note by considering all the elements generated by 1 under $+$, $\mathbb{K}$ contains a (isomorphic) copy of $\mathbb{Z}_p \subseteq \mathbb{K}$. The field axioms allow us to view $\mathbb{K}$ as a $\mathbb{Z}_p$-vector space (with elements in $\mathbb{K}$ as vectors, and addition and multiplication being taken from the field), and since $\mathbb{K}$ is finite, this vector space is certainly finite dimensional, say $\dim_{\mathbb{Z}_p}(\mathbb{K}) = n$. Thus the number of elements in $\mathbb{K}$ can be counted as the number of vectors in this space. There are p choices per coordinate, and n coordinates identify an element in this vector space. Hence $|\mathbb{K}| = p^n$. □

The converse requires a bit more work. Let $\mathbb{K}[x]$ be the univariate polynomials in $\mathbb{K}$.

Definition 10.19 (Irreducibility)**.** Let $f(x) \in \mathbb{K}[x]$. $f(x)$ is *irreducible* if f cannot be factored into non-constant polynomials i.e. cannot write as $f(x) = g(x)h(x)$ for $g(x), h(x) \in \mathbb{K}[x]$ where $\deg(g), \deg(h) \geq 1$.

Example 10.20. $x^2 + 1$ is irreducible over $\mathbb{R}$, but factors as $(x+i)(x-i)$ over $\mathbb{C}$.

Proposition 10.21. *Let* $\mathbb{K}$ *be a field, and consider the set* $\mathbb{K}[x]/\langle f(x)\rangle = \{g(x) \bmod f(x) : g(x) \in \mathbb{K}[x]\}$*. If* $f(x)$ *is irreducible, then* $\mathbb{K}[x]/\langle f(x)\rangle$ *is a field. Further, if* $\deg(f) = d$*, then* $\mathbb{K}[x]/\langle f(x)\rangle$ *is a* d*-dimensional* $\mathbb{K}$*-vector space.*

Proof. Addition and multiplication are taken as their normal polynomial variants, and addition can be seen to make a group. The only non-trivial fact is showing the existence of multiplicative inverses. Let $g(x) \in \mathbb{K}[x]$. Since $\gcd(g(x), f(x)) = 1$ (since f is irreducible), by Bezoút's lemma for polynomials (over a field), there are polynomials $a(x), b(x) \in \mathbb{K}[x]$ such that $a(x)f(x) + b(x)g(x) = 1$. So $b(x)g(x) = 1 \bmod f(x)$ and hence b is the multiplicative inverse of g.

The second part follows from the division algorithm: let $h(x) \in \mathbb{K}[x]$ and write it as $h(x) = q(x)f(x) + r(x)$ where $r(x) \equiv 0$ or $\deg(r) < \deg(f) = d$. Therefore $h(x) = r(x) \bmod f(x)$. Hence every polynomial has a representative of degree less than d i.e. is in the span of $\{1 \bmod f(x), x \bmod f(x), \cdots, x^{d-1} \bmod f(x)\}$. □

Definition 10.22 (Generating a field)**.** Let be a subfield $\mathbb{K} \subseteq \mathbb{F}$ and consider a subset $S \subseteq \mathbb{F}$. Then $\mathbb{K}(S)$ is the smallest subfield containing $\mathbb{K}$ and S, and is the *field generated by* $\mathbb{K}$ *and* S*.*

Definition 10.23 (Splitting field)**.** Let $\mathbb{K}$ be a field, and $f(x) \in \mathbb{K}[x]$ be an irreducible polynomial. A field $\mathbb{F} \supseteq \mathbb{K}$ is a *splitting field* of f if f decomposes entirely into linear factors, and $\mathbb{F}$ is the field generated by $\mathbb{K}$ and the roots of f.

Proposition 10.24. *Let* $f(x) \in \mathbb{K}[x]$*. There exists a splitting field of* f*.*

Proof. Factor $f(x) = f_1(x)f_2(x)\cdots f_n(x)$ into its irreducible factors. If all the f_i are linear, then $\mathbb{K}$ is a splitting field and we're done. So say f_1 is not linear. Since f_1 is irreducible, we can consider the field $F_1 = \mathbb{K}[x]/\langle f_1(x)\rangle$. Here $f_1(x)$ has the root $\alpha_1 := x \bmod f_1(x)$, since $f_1(x \bmod f_1(x)) = f_1(x) \bmod f_1(x) = 0$. So we can pull an additional (maybe more than 1) linear factor $f_1(x) = (x - \alpha_1)g(x)$. If all irreducible factors are linear under K_1, we're done. Else, we pick another irreducible factor under K_1 (say g) and continue to quotient $K_2 = K_1/\langle g(x)\rangle$. At each step we reduce the degree of (at least) one irreducible factor, and hence this process will eventually terminate. □

Definition 10.25 (Formal derivative)**.** Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0 \in \mathbb{K}[x]$. The *formal derivative* of f is $D(f) = na_n x^{n-1} + (n-1)a_{n-1}x^{n-2} + \cdots + a_1$. Many of the same properties from calculus remains like linearity and the product rule.

Proof of ($\Leftarrow$) *of Theorem 10.18.* We show existence of such a field and defer uniqueness to *Part B: Galois Theory*. Consider $f(x) = x^{p^n} - x \in \mathbb{Z}_p[x]$, and the splitting field $\mathbb{S}$ of $f(x)$ (which exists by Proposition 10.24). Let $\mathbb{K} = \{x \in \mathbb{S} : f(x) = 0\}$. We will show $|\mathbb{K}| = p^n$ and $\mathbb{K}$ is a subfield. Say f has a repeated root α in $\mathbb{S}$. Then $D(f)(\alpha) = 0$, but this is not possible as $D(f) = p^n x^{p^n - 1} - 1 = -1$ since the characteristic of $\mathbb{S}$ is p (since it contains $\mathbb{Z}_p$; hence $p^n = 0$). So f has p^n distinct roots and hence $|\mathbb{K}| = p^n$.

Note $1 \in \mathbb{K}$, $\mathbb{K}$ is closed under multiplication and inversion (for non-zero solutions). So to show $\mathbb{K}$ is a field, we just need to show it is an additive group. It's clearly closed under additive inverses. Remember that $\text{Char}(\mathbb{S}) = p$ and so $(a+b)^p = a^p + b^p$ for all $a, b \in \mathbb{S}$. Repeatedly raising this expression to p nets us that $(a+b)^{p^n} = a^{p^n} + b^{p^n}$. Hence if $a, b \in \mathbb{K}$, then $a + b \in \mathbb{K}$ since $f(a+b) = (a+b)^{p^n} - (a+b) = a^{p^n} - a + b^{p^n} - b = f(a) + f(b) = 0$. Hence $\mathbb{K}$ is a field of order p^n. □

As noted, by the uniqueness up to isomorphism of finite fields, we denote the finite field of order q by $\mathbb{F}_q$ (note $\mathbb{F}_p \cong \mathbb{Z}_p$ for prime p). This theorem will be especially useful as it allows us to explicitly control how big of a finite field we want to work in: we can pick any prime power, and there is a field of that size we can use.

Remark 10.26. The above shows existence of fields of any prime power order, but not necessarily how to construct them. If we want a field of order p^d, we often try to work in $\mathbb{F}_p[x]/\langle f(x)\rangle$ where f is irreducible and $\deg(f) = d$ (this generates a $\mathbb{F}_p$-vector space of dimension d by Proposition 10.21). Constructing irreducible polynomials is not straightforward, but we have tests to check if polynomials are irreducible. So we might just give a guess and just test for irreducibility. For example, when constructing fields of order 2^d via $\mathbb{F}_2[x]/\langle f(x)\rangle$, a common choice is $f(x) = x^d + x^k + 1$ with lowest k possible [40].

Example 10.27 (Binary fields)**.** The polynomial $f(x) = x^3 + x + 1$ is irreducible in $\mathbb{F}_2$ (since it is a cubic and has no roots), so we can construct the field $\mathbb{F}_8 \cong \mathbb{F}_2[x]/\langle x^3 + x + 1\rangle$ of order 2^3. Building elements from the basis $\{1, x, x^2\}$, we see that the field has elements $\{0, 1, x, x+1, x^2, x^2+x, x^2+1, x^2+x+1\}$ (recalling we are also working in characteristic 2, so we reduce coefficients modulo 2).

Remark 10.28 (Simple field extensions)**.** The above method of quotienting out by an irreducible polynomial gives a general method of identifying fields that is quite useful in field theory. In particular, say we have a field $\mathbb{F}$ and an element α in some ambient field $\mathbb{K} \supseteq \mathbb{F}$. Now consider the field homomorphism $\phi : \mathbb{F}[x] \to \mathbb{F}[\alpha]$ given by the evaluation map $f(x) \mapsto f(\alpha)$ (that is we're mapping polynomials in x to polynomial expressions in α). If we take the minimal polynomial $m_{\alpha,\mathbb{F}}(x)$ of α over $\mathbb{F}$, then $f(\alpha) = 0 \iff m_{\alpha,\mathbb{F}}(x) \mid f(x) \iff f(x) \in \langle m_{\alpha,\mathbb{F}}(x)\rangle$. Hence $\ker\phi = \langle m_{\alpha,\mathbb{F}}(x)\rangle$. Clearly $\text{Im}\ \phi = \mathbb{F}[a]$, so by the First Isomorphism Theorem (for Rings), we have $\mathbb{F}[x]/\langle m_{\alpha,\mathbb{F}}(x)\rangle \cong \mathbb{F}[a]$ given by $x \bmod m_{\alpha,\mathbb{F}}(x) \mapsto \alpha$. However, since the LHS is actually a field since $m_{\alpha,\mathbb{F}}(x)$ is irreducible as we discussed in Proposition 10.21, it is actually the case that $\mathbb{F}[\alpha] = \mathbb{F}(\alpha)$ and is the field that is generated by $\mathbb{F}$ and α (for a more sophisticated view, $m_{\alpha,\mathbb{F}}(x)$ being irreducible means $\langle m_{\alpha,\mathbb{F}}(x)\rangle$ is a maximal ideal and hence the quotient ring is a field; also hence for any algebraic element its polynomial ring is actually a field). The idea is that we're adjoining the root α of $m_{\alpha,\mathbb{F}}(x)$ as the coset $x \bmod m_{\alpha,\mathbb{F}}(x)$ (see Proposition 10.24 and creating splitting fields).

Now why does this matter? Well, as we said before, if $\deg(m_{\alpha,\mathbb{F}}(x)) = d$, then $\mathbb{F}[x]/\langle m_{\alpha,\mathbb{F}}(x)\rangle$ is a d-dimensional $\mathbb{F}$-vector space with basis $\{1, x, \cdots, x^{d-1}\}$. By our isomorphism, this means that $\mathbb{F}(\alpha)$ is also a d-dimensional $\mathbb{F}$-vector space with basis given by $\{1, \alpha, \cdots, \alpha^{d-1}\}$ (the idea is that if any higher power of α appears in an expression, we can reduce it with the relation of $m_{\alpha,\mathbb{F}}(x)$, and that's what our isomorphism confirms). In field theory, we denote the dimension of $\mathbb{K}$ as an $\mathbb{F}$-vector space $\dim_{\mathbb{F}} \mathbb{K} = [\mathbb{K} : \mathbb{F}]$ and call it the *degree of the field extension* $\mathbb{K}/\mathbb{F}$. So in particular we have shown that $[\mathbb{F}(\alpha) : \mathbb{F}] = \deg(m_{\alpha,\mathbb{F}}(x))$. This is a key relation that forms the fundamental ideas for much of field and Galois theory. We have already seen how useful this interchange of fields and vector spaces can be with identifying finite fields and prime power orders.

Example 10.29. $\mathbb{R}[x]/\langle x^2 + 1\rangle \cong \mathbb{R}(i)$ is a 2-dimensional $\mathbb{R}$-vector space we know as $\mathbb{C}$.

10.8 Some Comments on Shor's Algorithm

10.8.1 The Controlled-U Gate in Quantum Phase Estimation

It might be unclear what exactly our $U_{\texttt{control}}$ actually looks like. Here is what the full matrix looks like.

$$U_{\texttt{control}} = \begin{array}{c|c|c|c|c|c|c|c|c} & |\texttt{bin}(0)\rangle|v\rangle & |\texttt{bin}(1)\rangle|v\rangle & |\texttt{bin}(2)\rangle|v\rangle & |\texttt{bin}(3)\rangle|v\rangle & \cdots & |\texttt{bin}(j)\rangle|v\rangle & \cdots & |\texttt{bin}(2^k-1)\rangle|v\rangle \\ \hline |\texttt{bin}(0)\rangle|v\rangle & I & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ \hline |\texttt{bin}(1)\rangle|v\rangle & 0 & \begin{matrix} I & 0 \\ 0 & U \end{matrix} & 0 & 0 & \cdots & 0 & \cdots & 0 \\ \hline |\texttt{bin}(2)\rangle|v\rangle & 0 & 0 & \begin{matrix} I & 0 \\ 0 & U^2 \end{matrix} & 0 & \cdots & 0 & \cdots & 0 \\ \hline |\texttt{bin}(3)\rangle|v\rangle & 0 & 0 & 0 & \begin{matrix} I & 0 \\ 0 & U^3 \end{matrix} & \cdots & 0 & \cdots & 0 \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \hline |\texttt{bin}(j)\rangle|v\rangle & 0 & 0 & 0 & 0 & \cdots & \begin{matrix} I & 0 \\ 0 & U^j \end{matrix} & \cdots & 0 \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \hline |\texttt{bin}(2^k-1)\rangle|v\rangle & 0 & 0 & 0 & 0 & \cdots & 0 & \cdots & \begin{matrix} I & 0 \\ 0 & U^{2^k-1} \end{matrix} \end{array}$$

As stated, this is not what the matrix would look like in practice, however. In particular, we would break this up into a series of "controlled-U^{2^j}" gates that are applied according to the state of $|\texttt{bin}(k)\rangle$. This would only require k total small matrices as opposed to this big one that has all 2^k powers with superposition taking care of the rest.

10.8.2 A Comment on the Success of Shor's Algorithm

The success probability of Shor's algorithm makes some heuristic sense to be of the order at least $O(1/\pi^2)$ at least for the reason that is around the probability of two numbers being coprime: if we "randomly" pick any two integers, the probability they are both divisible by a prime p is $1/p^2$, so the probability at least one of them is not divisible by p is $1 - 1/p^2$. Extending this to all the primes,

$$\prod_{p \text{ primes}} \left(1 - \frac{1}{p^2}\right) = \left(\prod_{p \text{ primes}} \frac{1}{1-p^{-2}}\right)^{-1} = \left(\prod_{p \text{ primes}} \sum_{k=0}^{\infty} p^{-2k}\right)^{-1} = \left(\sum_{n=0}^{\infty} \frac{1}{n^2}\right)^{-1} = \frac{6}{\pi^2}$$

The notion of "random integers" and extending this probability over all primes can be formalized, but this gives a nice heuristic argument for the probability of coprimality and hence the probability of recovering a useful qubit state to find our period.

10.8.3 Deriving the Fourier Basis for Hidden Subgroups

To save time in the general algorithm, we describe a special case of the Fourier transform for hidden subgroups. Since we are working in cyclic groups, we will use additive notation for the groups.

Let G be a finite abelian group and f a function that hides the subgroup $H \leq G$. Let $\hat{f}(\chi_\ell) = \frac{1}{\sqrt{|G|}} \sum_{g \in G} f(g)\chi_\ell(g)$ be the Fourier transform of f given above, so we can write $f(g) = \frac{1}{\sqrt{|G|}} \sum_{\ell \in G} \hat{f}(\ell)\overline{\chi_\ell(g)}$ (note that is the actual inverse: since we are working in a finite group, say of order n, then $\chi(g)^n = \chi(g^n) = \chi(1) = 1$ hence $\chi(g)$ for all g is a root of unity. In particular, $\overline{\chi(g)} = 1/\chi(g)$). Now, rewrite $\hat{f}(\ell)$ in terms of cosets:

$$\hat{f}(\ell) = \frac{1}{\sqrt{|G|}} \sum_{g \in G} f(g)\chi_\ell(g) = \frac{1}{\sqrt{|G|}} \sum_{g \in G/H} \sum_{h \in H} f(g+h)\chi_\ell(g+h)$$

f is constant on cosets so $f(g+h) = f(g)$, and χ_ℓ is a homomorphism, so:

$$\hat{f}(\ell) = \frac{1}{\sqrt{|G|}} \sum_{g \in G/H} \sum_{h \in H} f(g+h)\chi_\ell(g+h) = \frac{1}{\sqrt{|G|}} \sum_{g \in G/H} \chi_\ell(g) f(g) \sum_{h \in H} \chi_\ell(h)$$

We can simplify this: note if $\ell \in H^\perp$, then $\chi_\ell(h) = 1$ so $\sum_{h \in H} \chi_\ell(h) = |H|$. If $\ell \notin H^\perp$, then $\exists h_0 \in H$ such that $\chi_\ell(h) \neq 1$. But note, since $h \mapsto h + h_0$ is surjective,

$$\sum_{h \in H} \chi_\ell(h) = \sum_{h \in H} \chi_\ell(h_0 + h) = \chi_\ell(h_0) \sum_{h \in H} \chi_\ell(h)$$

But $\chi_\ell(h_0) \neq 1$, so it must be that $\sum_{h \in H} \chi_\ell(h) = 0$ (this is analogous to conditions on evaluating $\sum_{\ell=0}^{N-1} e^{2\pi i \ell / N}$). So based on these cases, and renormalizing (so later we have well-defined quantum states):

$$\hat{f}(\ell) = \frac{1}{\sqrt{|G|}} \sum_{g \in G/H} \chi_\ell(g) f(g) \sum_{h \in H} \chi_\ell(h) = \begin{cases} \sqrt{\frac{|H|}{|G|}} \sum_{g \in G/H} \chi_\ell(g) f(g) & \text{if } \ell \in H^\perp \\ 0 & \text{if } \ell \notin H^\perp \end{cases}$$

(This is the correct normalization. Consider the homomorphism $\phi : \hat{G} \to \hat{H}$ via $\chi \mapsto \chi|_H$. Then $\ker \phi = H^\perp$, so by First Isomorphism Theorem, $\hat{G}/H^\perp \cong \hat{H}$. Finish with $\hat{G} \cong G$ and $\hat{H} \cong H$.) Hence we can write the inverse as

$$f(g) = \sqrt{\frac{|H|}{|G|}} \sum_{\ell \in H^\perp} \overline{\chi_\ell(g)} \hat{f}(\ell)$$

10.8.4 Another Approach to the Generic HSP Algorithm

Below is the "measurement approach" to solving the generic HSP. Our original approach took the simplification by realizing the Fourier basis above, but this one instead measures an additional time to highlight where exactly we are extracting information from the hidden subgroup. Here, instead of just measuring the first measure, we also measure the second register to project onto the entangled coordinates of the coset.

1. Initialize two registers $|0\rangle |0\rangle$.
2. Apply the QFT to the first register to create a uniform superposition over the group elements:

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

3. Apply f to the second register in accordance to the first register:

$$\frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle |0\rangle \mapsto \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle |f(g)\rangle$$

4. Measure the second register, fixing the output to some $|f(g_0)\rangle$:

$$\frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle |f(g)\rangle \mapsto \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle |f(g_0)\rangle \mapsto \frac{1}{\sqrt{|H|}} \sum_{h \in H} |g_0 + h\rangle |f(g_0)\rangle$$

   Measuring $|f(g)\rangle \mapsto |f(g_0)\rangle$ gives us some information on the potential state of the first register $|g\rangle$ (this is a result of the fact we have entangled the first and second registers by applying f). In particular, if we see $|f(g_0)\rangle$, we know that the first register must be an element from $g_0 + H$ since f is constant on cosets. We do not know what element this is, but it can be uniformly any of them, so we end up with the reduced coset superposition on the right. (At this point, we no longer need the second register for it has given us the coset information we need.)

5. Apply the QFT to the first register. By linearity and properties of characters:

$$\begin{aligned}
\frac{1}{\sqrt{|H|}} \sum_{h \in H} |g_0 + h\rangle &\mapsto \frac{1}{\sqrt{|H|}} \sum_{h \in H} \frac{1}{\sqrt{|G|}} \sum_{g \in G} \chi_{g_0+h}(g) \, |g\rangle \\
&= \frac{1}{\sqrt{|H||G|}} \sum_{g \in G} \chi_{g_0}(g) \sum_{h \in H} \chi_h(g) \, |g\rangle \\
&= \frac{1}{\sqrt{|H||G|}} \sum_{g \in G} \chi_g(g_0) \left(\sum_{h \in H} \chi_g(h) \right) |g\rangle \\
&= \frac{|H|}{\sqrt{|H||G|}} \sum_{g \in H^\perp} \chi_g(g_0) \, |g\rangle \\
&= \sqrt{\frac{|H|}{|G|}} \sum_{g \in H^\perp} \chi_g(g_0) \, |g\rangle
\end{aligned}$$

6. Measure the first register to obtain a random $g \in H^\perp$, which gives information on H.
7. Repeat steps 1–6 until H can be determined via the linear relations of $H^\perp$.

The last few steps are quite similar to our original approach with the rewritten Fourier basis, and by the end that is precisely what we obtained. We see some familiar and new properties of characters being used. The two new properties we used were $\chi_{g_0+h}(g) = \chi_{g_0}(g)\chi_h(g)$ and $\chi_h(g) = \chi_g(h)$. These are both best seen by looking at the explicit exponential representation for the characters when treating G as its isomorphic (product of) cyclic group(s). That is how we explained the isomorphism $G \cong \hat{G}$ to begin with, and makes the symmetry clear. We also used the characterization from before in evaluating $\sum_{h \in H} \chi_g(h)$.