A Computational Introduction to Elliptic and Hyperelliptic Curve Cryptography

Nicholas Wilcox

March 2018

Abstract

At its core, cryptography relies on problems that are simple to construct but difficult to solve unless certain information (the "key") is known. Many of these problems come from number theory and group theory. One method of obtaining groups from which to build cryptosystems is to define algebraic curves over finite fields and then derive a group structure from the set of points on those curves. This thesis serves as an exposition of Elliptic Curve Cryptography (ECC), preceded by a discussion of some basic cryptographic concepts and followed by a glance into one generalization of ECC: cryptosystems based on hyperelliptic curves. Our primary source for this material is [10].

Contents

1 Cryptography

The basic model of cryptography is as follows: Adrian wishes to send a message M, the **plaintext**, to Beth. To do this, Adrian encrypts M with an encryption function E, obtaining $C = E(M)$. We call C the **ciphertext**. After Beth gets the ciphertext, she decrypts it with the decryption function D. For our purposes, we can characterize E and D as bijections defined on some message space, where D is the inverse of E, so that $D(E(M)) = M$ for all messages M. The security of the encryption relies on secret information k, the **key(s)**, that determines the encryption and decryption functions. So, if some eavesdropper were to obtain C, they could not feasibly recover M because it is hard to determine D without first determining k.

For instance, suppose we have an abelian, multiplicative group G, and our message is some element $M \in G$. Pick an element $k \in G$, and let E be defined by $E(x) = kx$. Then $C = kM$, and the decryption function D is given by $D(x) = k^{-1}x$. The problem for the eavesdropper is to determine k while only knowing kM and the structure of G.

1.1 The Discrete Logarithm Problem

Let $G = \langle g \rangle$ be a finite, cyclic group. For all $h \in G$, we know that there exists a solution k to the equation $g^k = h$. The discrete logarithm problem (DLP) is to solve for k, given g and h. [1]

▶ **Remark** If n is the order of g, then $g^{n+k} = g^n g^k = g^k = h$, so $n + k$ is a solution as well. We can characterize all of the solutions k to the equation $g^k = h$ as being congruent to the unique solution modulo n. We write $k = L(h)$ to express that all solutions are congruent to $k \pmod{n}$. In general, equations involving the discrete log L represent congruence relations modulo n. The group G and the base of the logarithm, g, are understood to be an arbitrary cyclic group and generator, or are made explicit in context.

As the name and definition suggest, the discrete logarithm L is analogous to the usual logarithm defined on the positive real numbers. Indeed, many of the same algebraic identities hold:

- $g^{L(h)} = h$,
- for all $h_1, h_2 \in G$, $L(h_1 h_2) = L(h_1) + L(h_2)$, and
- for all $m \in \mathbb{Z}$ and $h \in G$, $L(h^m) = mL(h)$.

Example 1.1 Modular arithmetic provides a good introductory example to the DLP. Consider the multiplicative group $\mathbb{F}_p^\times$ of nonzero integers modulo p, where p is prime. The group $\mathbb{F}_p^\times$ is cyclic with order $p - 1$. Therefore, if a is a generator for the group, then $a^k \equiv b \pmod{p}$ has a solution for all $b \in \mathbb{F}_p^\times$, and this solution is unique modulo $p - 1$.

Let $p = 7$, then 3 is a generator for $\mathbb{F}_7^\times$. The powers of 3 modulo 7 are

$$3 \to 2 \to 6 \to 4 \to 5 \to 1,$$

so $L(2) = 2$, $L(6) = 3$, and so on. ⊠

1.1.1 Index Calculus

The previous example illustrates a brute force method for obtaining a solution to the DLP in $\mathbb{F}_p^\times$. This is infeasible when p is very large. A quicker method, known as **index calculus**, is based on the idea that if $a^m = b$, then the discrete log of a can be used to solve the discrete log of b, because $L(b) = L(a^m) = mL(a)$. If we can compute the discrete logs of various primes, then those values can be used to compute the discrete log of numbers that factor into those primes.

Example 1.2 Let $G = \mathbb{F}_{503}$, and note that 5 is a generator for $\mathbb{F}_{503}^\times$, so 5 will be the base of our discrete logarithm L. Also, let B be a set containing the first few primes, say $B = \{2, 3, 5, 7, 11\}$. We call B the **factor base**, and our first objective is to compute $L(p)$ for each $p \in B$. We start by computing powers

[1] We can consider the DLP for an arbitrary finite group G if one allows for the possibility that there is no solution. That is, once g and h are selected, a solution k exists if and only if $h \in \langle g \rangle$.

of 5 in G and noting those whose representatives (or whose additive inverses) factor in B. After several computations, we obtain the following congruence relations:

$$\begin{aligned}
5^1 &\equiv 5 \pmod{503} \\
5^6 &\equiv 2^5 \pmod{503} \\
5^{37} &\equiv -2 \cdot 7 \pmod{503} \\
5^{43} &\equiv 5 \cdot 11 \pmod{503} \\
5^{58} &\equiv 2^2 \cdot 3 \pmod{503}
\end{aligned}$$

The first relation seems trivial and redundant; however, it is not necessary that the generator–base should be congruent to one of the primes in our factor base, so we would still need to compute the discrete log of 5 regardless. By taking the discrete logarithm of these relations, we obtain the following (note that the implied modulus is 502):

$$\begin{aligned}
1 &= L(5) \\
6 &= 5L(2) \\
37 &= L(-1) + L(2) + L(7) \\
43 &= L(5) + L(11) \\
58 &= 2L(2) + L(3)
\end{aligned}$$

We can begin computing the discrete logs of the other primes. The fourth equation can be straightforwardly solved as $L(11) = 42$. Because 5 is unit modulo 502, we can solve the second equation $L(2) = 6 \cdot 5^{-1} = 202$. This value can be substituted into the last equation, so $L(3) = 58 - 2 \cdot 202 \equiv 156$. Finally, to compute $L(7)$ using the third equation, we need to know $L(-1)$. Since 5 is a generator for $\mathbb{F}_{503}$, it cannot be a quadratic residue. Therefore, by Euler's Criterion, $5^{(503-1)/2} = 5^{251} = -1$, so $L(-1) = 251$, and we can solve for $L(7) = 86$.

Now, suppose that we want to compute $L(255)$. We can use a similar approach as we did for the primes in our factor base. We will compute $5^m \cdot 255$ in $\mathbb{F}_{503}^\times$ for $m = 0, 1, \ldots$, and so on, until like before, we find a value that factors in B. Observe that $5^6 \cdot 255 \equiv 2^4 \cdot 7$. Taking the discrete logarithm obtains $6 + L(255) = 4L(2) + L(7)$, and since we know $L(2)$ and $L(7)$, we can compute $L(255) = 386$. ⊠

Index calculus is particular to discrete logarithms in multiplicative groups of finite fields. For arbitrary groups, including those we will encounter in elliptic curve cryptography, there is not a notion of prime factorization that we can exploit.

1.1.2 Baby Step–Giant Step

A more generally applicable method to solve the DLP is known as the "Baby Step–Giant Step" algorithm and is due to Daniel Shanks [7]. Let $G = \langle g \rangle$ be a cyclic group of order N, and suppose we want to find a solution k to the equation $g^k = h$. Instead of computing all N powers of g (which is the brute force approach), we compute two smaller sets of elements. One set is simply $\{g^i : i = 0, 1, \ldots, m-1\}$ for some integer $m \geq \sqrt{N}$. The other set, the "giant step", is $\{hg^{-jm} : j = 0, 1, \ldots, m-1\}$. The reason we negate the power of g and compose it with h is that we expect there to be a match between the two sets. Once we have some i and j such that $g^i = hg^{-jm}$, we have a solution for the DLP, because $g^{i+jm} = h$.

▶ **Remark** Our choice of m is what ensures a match between the two sets. We know that some solution k exists such that $0 \leq k < N \leq m^2$. The correct values for i and j are the remainder and quotient of k divided by m.

The Baby Step–Giant Step algorithm takes up $O(\sqrt{N})$ space as it runs, since we need to hold the baby step's elements in memory as we check for matches during the giant step part of the algorithm. The implementation of the baby step's storage affects the time complexity of checking for a match. Suppose that we stored those points in an ordinary array. Then, for each of the $\sqrt{N}$ giant step elements, we would need to check for a match against each of the $\sqrt{N}$ elements in the array. Using this method, the algorithm's running time would be $O(N)$.

Alternatively, we can use a hash table to store the elements from the baby step. A **hash table** is a set of key-value pairs, such that the keys are processed using a **hashing function** h that determines where they and their associated values are stored internally. When implemented properly, hash tables allow for a constant lookup time on average, since hashing the keys significantly narrows down the search. Hash tables generalize arrays in the sense that for an array, a value's key is its index in the array. We provide a small example to illustrate how a basic hash table can be implemented. A more rigorous and sophisticated treatment of the subject is given in [2, Chapter 11].

Suppose that we want to store pairs of integers and strings, such as (5, “dog”) and (1008, “cat”), where the integers are the keys and the strings are their associated values. Even though the number of possible keys is very large, we don't expect every integer to be used as a key, so we use a hashing function h to map the universe of keys (here it is $\mathbb{Z}$) to a smaller, finite set of **hash values**. When the keys are integers, a natural choice for h is to compute the remainder of a key k when divided by some previously selected prime p.

Since the set of keys is strictly larger than the set of hash values, there will exist two keys k_1 and k_2 such that $h(k_1) = h(k_2)$. This is called a **collision**, and one method to deal with collisions is known as **chaining**. Essentially, every hash value h_i has its own list (or chain) of key-value pairs, such that all pairs (k, v) where $h(k) = h_i$ are stored in h_i's list. Here is a depiction of what our hash table might look like after several items are added, supposing that $p = 11$:

0	1	2	3	4	5	6	7	8	9	10
(132, “asp”)	(45, “fish”)				(5, “dog”)		(1008, “cat”)		(75, “bear”)	
(99, “bird”)					(126, “bee”)					
					(27, “pony”)					

If we wish to lookup the value associated with the key 27, then we could compute $h(27) = 5$, and then iterate through 5's chain until we find the item whose key is 27 and return its value, “pony”. Alternatively, if we want to lookup the value for 36, we would inspect the chain with hash value $h(36) = 3$, observe that the chain is empty, and return some value indicating that 36 has no associated value in the hash table. No matter how many items we added to the other chains, these operations would take the same amount of time. When chaining is utilized, it is common for hash tables to adjust the hash function and reconstruct the table whenever the ratio of stored values to the number of chains, called the **loadfactor**, becomes too large. In our scenario, we might choose a new prime $p' > 2p$, so that the loadfactor is cut in half.

For the Baby Step–Giant Step algorithm, we use the group elements g^i as the keys, and the integers i as their respective values. Because checking a giant step element for a match can be done in constant time on average when using a hash table, this implementation of the Baby Step–Giant Step algorithm runs in $O(\sqrt{N})$ time.

1.2 Cryptosystems based on the DLP

As previously stated, the primary task of cryptography is to enable Adrian to send Beth a message without letting any eavesdroppers, whom we collectively personify as “Eve”, read it. There are two major classes of cryptography. If both the encryption and decryption functions are derived from the same key k, then we say that a **symmetric cryptosystem** is being used. A fundamental dilemma for symmetric cryptosystems is how to distribute the key to trusted parties like Adrian and Beth, while preventing Eve from obtaining it as well.

Otherwise, if Adrian and Beth need not have previously established a shared secret key, then we say they are using an **asymmetric cryptosystem**. Asymmetric cryptosystems are also called **public key cryptosystems** because a popular technique in asymmetric cryptography is to let both Adrian and Beth generate their own pair of public and private keys. In this case, a message encrypted by one's public key can only be decrypted using their private key, and vice versa. To send a message to Beth, Adrian uses Beth's public key to encrypt it, and since Beth is the only person with her private key, only she can decrypt and read the message.

While public key cryptosystems dodge the issue of key distribution, their encryption algorithms are typically slower than those that use symmetric keys. A common practice is to use asymmetric cryptography to generate a shared secret key for a symmetric cryptosystem that will be used for the bulk of the communication

```
**Given**  : A group $G$ with order $N$.
**Input**  : $g, h \in G$, where $g$ is a generator for $G$.
**Output**: An integer $k$ such that $g^k = h$.
let $m = \lceil \sqrt{N} \rceil$,  $temp = e$ // where $e$ is the identity element of $G$
let $L$ be a hash table
for $i = 0, 1, \ldots, m-1$ do // baby step
    $L[temp] = i$
    $temp \to g \cdot temp$ // "$\cdot$" is the group operation
end
let $inc = g^{-m}$
$temp \to h$ // $temp$ is the current $hg^{-jm}$
for $j = 0, 1, \ldots, m-1$ do // giant step
    if $L[temp]$ exists then // match found
        let $i = L[temp]$
        return $k = i + jm$
    $temp \to temp \cdot inc.$
end
// If the program never halts in the for loop, then no solution $k$ exists.
```

Algorithm 1: Baby Step–Giant Step algorithm for the Discrete Logarithm Problem.

between Adrian and Beth. One such method is called Diffie–Hellman key exchange, which we will now describe.

In Diffie–Hellman key exchange, Adrian and Beth publicly agree on a group G and an element $g \in G$. Therefore, Eve also knows G and g. Adrian and Beth each choose their own secret integers a and b. Adrian computes g^a and sends it to Beth, while Beth computes g^b and sends it to Adrian. Finally, they each compute $(g^a)^b = (g^b)^a = g^{ab}$, from which a shared secret key can be derived.

Eve, on the other hand, does not know g^{ab}. She only knows g, g^a, and g^b. If she could solve the discrete logarithm problem in G, then Eve could deduce the values of a and b, and thus compute the key that Adrian and Beth use. So, we say that Diffie–Hellman key exchange effectively relies on the difficulty of the DLP.

Another asymmetric technique is the Three-pass protocol, which assumes that the encryption and decryption functions are commutative with respect to composition. This is because Adrian and Beth take turns encrypting and decrypting data between each of the three passes.

For instance, let G be a publicly agreed upon group with order N, and let $M \in G$ be the message Adrian wants to send to Beth. Adrian selects a secret integer m_A that is relatively prime to N. After he sends $M_1 = M^{m_A}$ to Beth, she picks her own secret integer m_B, which is also relatively prime to N. She then computes $M_2 = M_1^{m_B}$ and sends it to Adrian.[2]

Before the third and final pass, Adrian computes the multiplicative inverse of m_A modulo N and then sends $M_3 = M_2^{m_A^{-1}}$ to Beth. Beth computes the inverse of her own m_B and finally computes $M_3^{m_B^{-1}}$, which is the original message M. This is because the final value is

$$M_3^{m_B^{-1}} = M^{m_A m_A^{-1} m_B m_B^{-1}},$$

and because both $m_A m_A^{-1}$ and $m_B m_B^{-1}$ are each congruent to 1 modulo N, the value Beth ultimately obtains is $M^{dN+1} = M$ by Lagrange's theorem.

This implementation of the Three-pass protocol also relies on the difficulty of the DLP for security. Eve knows M_1, M_2, and M_3, which are M^{m_A}, $M^{m_A m_B}$, and M^{m_B}, respectively. If she could solve the DLP,

[2] Adrian and Beth must also make sure not to select integers m_A and m_B that are multiplicative inverses of each other modulo N. Otherwise, $M_2 = M$ will be broadcast to Eve.

then Eve could determine m_A and m_B, and then compute either of their multiplicative inverses modulo N in order to compute M.

2 Elliptic Curves

Now that we know how to do cryptography with groups, we move on to study elliptic curves as a means of obtaining those groups.

Given a field F, with algebraic closure $\overline{F}$, and two constants $a, b \in F$, we may define an **elliptic curve** E as the set of points $(x, y) \in \overline{F}^2$ such that $y^2 = x^3 + ax + b$. Equations of the form $y^2 = x^3 + Ax + B$ are called **Weierstrass equations**. Observe that since y only appears with even degree in the equation, $(x, y) \in E$ if and only if $(x, -y) \in E$ as well.

If we have constants $a_1, \ldots, a_6 \in F$, then we can consider the set of points that satisfy the **generalized Weierstrass equation**:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

When the characteristic of F is neither 2 nor 3, we can make various substitutions and obtain a regular Weierstrass equation $y_0^2 = x_0^3 + a'x_0 + b'$. We will only consider this case. Furthermore, for the sake of simplicity, we assume that the polynomial in x on the right hand side of Weierstrass equation has distinct roots so that the curve is non-singular.

▸ **Remark** The study of elliptic curves relies on considering points whose coordinates lie in an algebraically closed field, hence the definition given above. However, the cryptographic application of elliptic curves comes from restricting the set of points on a curve to those whose coordinates lie in a finite field. In general, if $E : y^2 = x^3 + ax + b$ is an elliptic curve and K is a field that contains a and b, then we say that E is **defined over** K, and we let $E(K)$ denote the set of points on E whose coordinates are in K.

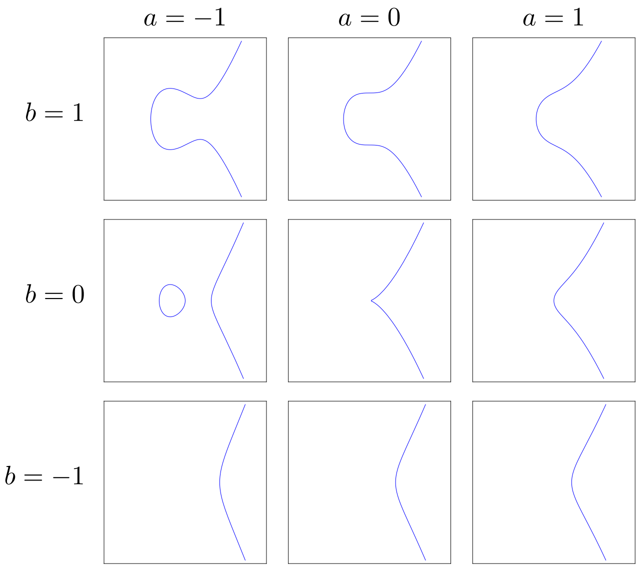

Figure 2.1: A series of elliptic curves $E(\mathbb{R}) : y^2 = x^3 + ax + b$, where a and b vary over $\{-1, 0, 1\}$. Note that the curve in the center, given by $y^2 = x^3$, is singular and so is not truly an elliptic curve under our definition.

2.0.1 Some Preliminary Algebraic Geometry

One may work with elliptic curves whose points are in the Cartesian product of a field with itself, such as $\mathbb{R}^2$, provided that some mathematics is swept under the rug through formal symbolism. However, a proper understanding of elliptic curves requires us to consider points in the **projective plane**.

For a given field F, let (x, y, z) be a point in F^3 such that $(x, y, z) \neq (0, 0, 0)$. Then we can define the equivalence class $[x : y : z]$ whose elements are the nonzero scalar multiples of (x, y, z). We define the projective plane $\mathbf{P}^2_F$ to be the set of these equivalence classes. Symbolically,

$$\begin{aligned}[x : y : z] &= \{(\lambda x, \lambda y, \lambda z) : \lambda \neq 0\}, \text{ and} \\ \mathbf{P}^2_F &= \{[x : y : z] : (x, y, z) \neq (0, 0, 0)\}.\end{aligned}$$

If $z \neq 0$, then $({}^{x}\!/\!{}_{z}, {}^{y}\!/\!{}_{z}, 1) \in [x : y : z]$, and furthermore, $[x : y : 1] \in \mathbf{P}^2_F$ for all $x, y \in F$. In other words, the equivalences classes in $\mathbf{P}^2_F$ whose members have nonzero z-components form a copy of F^2, what we call the **affine plane**. Therefore,

$$\mathbf{P}^2_F = F^2 \cup \{[x : y : 0] : (x, y) \neq (0, 0)\}.$$

This other set is similar to $\mathbf{P}^2_F$, except that the z-dimension has been omitted. As the nonzero equivalence classes from F^3 form $\mathbf{P}^2_F$, the nonzero equivalence classes from two coordinates form $\mathbf{P}^1_F$. In general, we have $\mathbf{P}^n_F = F^n \cup \mathbf{P}^{n-1}_F$, where $\mathbf{P}^{n-1}_F$ is called the **hyperplane at infinity**. So, $\mathbf{P}^2_F$ is composed of the affine plane and a **line at infinity**, $\{[x : 1 : 0] : x \in F\}$. This line at infinity also has a **point at infinity**, $[1 : 0 : 0]$.

If we have an algebraic curve defined over the affine plane, such as the solutions (x, y) to a Weierstrass equation, we can extend this curve to the projective plane using a method called **homogenization**, which can also be generalized to higher dimensions, but we limit our discussion to the plane. Homogenization introduces the z-component by multiplying each term in the equation by a nonnegative power of z so that every term has the same degree. The homogenized form of $y^2 = x^3 + ax + b$ is $y^2z = x^3 + axz^2 + bz^3$. Observe that if $z = 1$, then we are left with our original equation, so every point (x, y) that satisfied the original equation also satisfies the homogenized one because we identify (x, y) with $[x, y, 1]$.

Otherwise, if $z = 0$, then we obtain $0 = x^3$, so $x = 0$ as well. The set of solutions in F^3 is $\{(0, y, 0) : y \in F\}$. In the projective plane, this reduces to a single point on the line at infinity, $[0 : 1 : 0]$.

▶ **Remark** What we have shown is that when we move elliptic curves from the affine plane to the projective plane, we add a single point to the curve, $[0 : 1 : 0]$. We denote this point with the ∞ symbol, and it will be the identity element for the group structure we will define on the points on an elliptic curve. Furthermore, since this is the only non-affine point, we do not need to adjust our notation. Every other point on an elliptic curve can be represented by a point on the affine plane, i.e. by using two coordinates. From now on, it will be assumed that if E is an elliptic curve defined over F, then the points on E lie in $\mathbf{P}^2_{\overline{F}}$, and the points on $E(F)$ lie in $\mathbf{P}^2_F$.

2.1 The Group Law

The group operation defined for elliptic curves is known as the Chord and Tangent Method, which we will denote additively. Let E be an elliptic curve. For two points $P, Q \in E$, we compute $P + Q$ by extending the line through P and Q (the "chord") to a third point of intersection, R, with the curve. We construct the vertical line through R (the "tangent", more or less) which also intersects E at R', which is the reflection of R about the x-axis. Then, $P + Q = R'$. This process is depicted in Figure 2.2.

▶ **Remark** That there will always be a third point of intersection R between E and the line through P and Q is not obvious. Bézout's Theorem tells us that a line and an elliptic curve, which is of degree 3, can have at most three points of intersection. Furthermore, all three points of intersection occur when we consider all points in $\mathbf{P}^2_{\overline{F}}$, and also count points of intersection according to their multiplicity.

If E is defined over a field F, then we can apply the Chord and Tangent Method to $E(F)$ even if F is not algebraically closed. Since we begin with two points $P, Q \in E(F)$, the third point of intersection between the line and E will also have coordinates in F, as the following example exhibits.

Example 2.1 The curve in Figure 2.2 is defined over $\mathbb{Q}$ and is given by the equation $y^2 = x^3 - \frac{5}{2}x + 4$. Observe that the points $P = (0, 2)$ and $Q = (-2, 1)$ are on the curve. To compute $P + Q$, we must first

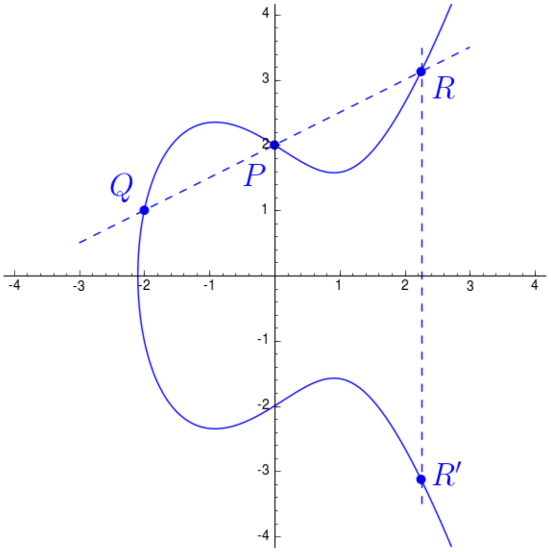

Figure 2.2: Illustration of the group law for elliptic curves.

compute the equation for the line L through P and Q, which is $y = \frac{x}{2} + 2$. Then, we substitute into the Weierstrass equation:

$$\begin{aligned}\left(\frac{x}{2}+2\right)^2 &= x^3 - \frac{5}{2}x + 4\\ \frac{x^2}{4} + 2x + 4 &= x^3 - \frac{5}{2}x + 4\\ 0 &= x^3 - \frac{x^2}{4} - \frac{9}{2}x\end{aligned}$$

The roots to this polynomial are the x-coordinates of points that are in the intersection of L and E, and we already know that the x-coordinates of P and Q are two of those roots. Recall that c is a root to a polynomial $p(x)$ if and only if $(x - c)$ divides $p(x)$. So, we may divide by $x(x+2) = x^2 + 2x$ to obtain

$$\frac{x^3 + \frac{x^2}{4} - \frac{9}{2}x}{x^2 + 2x} = x - \frac{9}{4}.$$

Because we divided a cubic polynomial by a quadratic polynomial, we obtained a linear term that identifies a third root. Therefore, the x-coordinate of the third point of intersection is $^{9}/_{4}$, and we may substitute into L's equation to solve for the y-coordinate, which is $^{25}/_{8}$. Finally, we reflect about the x-axis, which negates the y-coordinate, and obtain $P + Q = (^{9}/_{4}, -^{25}/_{8})$. $\boxtimes$

▶ **Remark** For a point P on an elliptic curve, we take "the line through both P and P" to mean the line tangent to E at P, so that the line intersects E at P with a multiplicity of 2. If E is given by the Weierstrass equation $y^2 = x^3 + ax + b$, then we may use implicit differentiation to obtain:

$$\frac{dy}{dx} = \frac{3x^2 + a}{2y}.$$

Therefore, the line tangent to E at $P = (x_1, y_1)$ is vertical if and only if $y_1 = 0$. Otherwise, if $m = \frac{dy}{dx}$, then the tangent line is given by $y = m(x - x_1) + y_1$.

Example 2.2 Consider the curve $E(\mathbb{Q}) : y^2 = x^3 - 3x + 3$, and observe that $P = (1,1) \in E$. Furthermore, the line tangent to E at P is horizontal. The other point on E with a y-coordinate of 1 is $(-2, 1)$. Therefore, $P + P = (-2, -1)$. $\boxtimes$

▶ **Remark** When a line between two points, or the tangent line at a point, is vertical, the third point of intersection is ∞. This is because the equation for a vertical line in the affine plane looks like $x = c$. This becomes $x = cz$ when homogenized. The solutions where z is nonzero correspond to the points on the line in the affine plane. If $z = 0$, then $x = 0$ as well, and y is allowed to vary, whence $\infty = [0 : 1 : 0]$ is on any vertical line as well as any elliptic curve. Conversely, the line between ∞ and an affine point P is simply the vertical line through P.

Two more notes on ∞:

1. The reflection of ∞ about the x-axis is ∞, since $[0 : 1 : 0] = [0 : -1 : 0]$.
2. The "line through ∞ and ∞" actually intersects an elliptic curve at ∞ with a multiplicity of 3. Recall that when we homogenized the Weierstrass equation and substituted $z = 0$, we obtained $x^3 = 0$, which is where the third intersection at ∞ comes from.

In order for an elliptic curve E to satisfy the definition of a group, we must have an identity element, and also inverse elements. We claim that ∞ is the identity element under the Chord and Tangent Method. We have already shown that $\infty + \infty = \infty$. If $P = (x, y)$ is an affine point on E, then the line through P and ∞ is vertical. This implies that the third point of intersection between this line and E is $(x, -y)$. Once we reflect $(x, -y)$ about the x-axis, we get $(x, y) = P$. Therefore, $P + \infty = P$ for all $P \in E$.

Since ∞ is the identity, it is its own inverse. For an affine point $P = (x, y)$, we claim that the inverse of P, written $-P$, is $(x, -y)$. Observe that the line through P and $(x, -y)$ is vertical, so the third point of intersection with E is ∞. As previously stated, reflecting ∞ obtains ∞, so the inverse of $P = (x, y)$ is $-P = (x, -y)$.

Also, P and Q are symmetric in the definition of $P + Q$. That is, the "line through P and Q" is the same as the "line through Q and P". Therefore, the binary operation is commutative. We have defined $P + Q$ for all possible values for P and Q, so the operation is closed on an elliptic curve. Associativity can be shown, but it is a complicated proof that is not illuminating to the study of elliptic curve cryptography, so we omit it. A complete proof appears in [10, Section 2.4]. We can summarize everything that has been said in the following:

Theorem 1 *If E is an elliptic curve defined over a field F, then the Chord and Tangent Method defines a binary operation on $E(F)$ with respect to which $E(F)$ is an abelian group.*

▶ **Remark** There will be times when we wish to evaluate functions $f(x, y)$ at points on an elliptic curve, including ∞. When this happens, we use the convention that $\infty = (\infty, \infty)$, and evaluate the limit of $f(x, y)$ as x and y get arbitrarily large. For instance, if $f(x, y) = \frac{x+2}{y}$, then $f(\infty) = 1$, since the limit of $\frac{x+2}{y}$, as x and y tend to infinity, is 1.

2.2 Formula for the Group Law

Instead of actually performing polynomial division to add points, we can give a formula for the group law using pure field arithmetic. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be non-identity points on the elliptic curve $E : y^2 = x^3 + ax + b$. Suppose that $P + Q \neq \infty$, so that the slope m of the line through P and Q is defined. If $P \neq Q$, then $m = \dfrac{y_2 - y_1}{x_2 - x_1}$, and if $P = Q$, then $m = \dfrac{3x_1^2 + a}{2y_1}$. In either case, we have an linear equation $y = mx + c$, so we may substitute this equation for y into the equation for E and obtain:

$$x^3 - m^2x^2 + (a - 2cm)x + b - c^2 = 0.$$

The three roots to this cubic polynomial are x_1, x_2, and some $x_3 \in F$ that is the x-coordinate of $P + Q$. In other words, the polynomial is equal to

$$(x - x_1)(x - x_2)(x - x_3) = x^3 - (x_1 + x_2 + x_3)x^2 + \text{other terms}.$$

Therefore, $m^2 = x_1 + x_2 + x_3$, so we can solve for x_3. Then, we can use the equation of the line to find its third point of intersection with E, negate the value of the y-coordinate, and obtain $P + Q$.

```
Given  : An elliptic curve $E : y^2 = x^3 + ax + b$.
Input  : $P, Q \in E$, where $P = (x_1, y_1)$ and $Q = (x_2, y_2)$.
Output: $P + Q = (x_3, y_3)$
if $x_1 \neq x_2$ then // we can compute the slope $m$ directly
    $m = \frac{y_2 - y_1}{x_2 - x_1}$
    $x_3 = m^2 - (x_1 + x_2)$
    $y_3 = m(x_1 - x_3) - y_1$
    return $(x_3, y_3)$
else if $x_1 = x_2$ then
    if $y_1 \neq y_2$ then // the line through $P$ and $Q$ is vertical
        return $\infty$
    else if $y_1 = y_2$ then // $P = Q$
        if $y_1 \neq 0$ then // we compute the line tangent at $P$
            $m = \frac{3x_1^2 + a}{2y_1}$
            $x_3 = m^2 - 2x_1$
            $y_3 = m(x_1 - x_3) - y_1$
            return $(x_3, y_3)$
        else if $y_1 = 0$ then // the line tangent to $P$ is vertical
            return $\infty$
```

Algorithm 2: An algorithm to compute the group law for elliptic curves.

It will be useful to compute the sum of a point added to itself multiple times. We can naively do this in linear time, but the best method is called **successive doubling**, and returns nP in $O(\log(n))$ time.[3] For instance, if we wanted to compute $16P$, then we could begin by computing $2P = P + P$, then $4P = 2P + 2P$, and so on. Each doubling is an addition of the previous result to itself. In general, we can expand n in base-2, compute only the multiples $2^k P$, and sum them appropriately to obtain nP (See Algorithm 3).

2.3 Structure and Size of $E(\mathbb{F}_q)$

If F is a finite field, so that $F = \mathbb{F}_q$ for some prime power $q = p^e$, then there are only finitely many points in F^2, hence $E(F)$ is a finite abelian group. The structure theorem for finite abelian groups implies that $E(F) \cong \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}$ where $n_1, \ldots, n_k$ are positive integers satisfying $n_i \mid n_{i+1}$. In fact, from [10, Theorem 4.1], we know that $k \leq 2$. The following important theorem relates the order of $E(\mathbb{F}_q)$ to q.

Theorem 2 (Hasse) *Let E be an elliptic curve defined over the finite field $\mathbb{F}_q$. Then the order of $E(\mathbb{F}_q)$ satisfies*

$$|q + 1 - \#E(\mathbb{F}_q)| \leq 2\sqrt{q}.$$

We will detail an algorithm that exploits the bounds given by Hasse's Theorem to compute $\#E(F_q)$. Before that, we give two more results about $\#E(\mathbb{F}_q)$, which are Theorems 4.12 and 4.14 in [10].

Theorem 3 *Let $\#E(\mathbb{F}_q) = q + 1 - a$. Write $x^2 - ax + q = (x - \alpha)(x - \beta)$. Then*

$$\#E(\mathbb{F}_{q^n}) = q^n + 1 - (\alpha^n + \beta^n)$$

for all $n \geq 1$.

[3] There might be additional complexity given by the size (in binary representation) of the coordinates for curves over fields with characteristic 0, like $\mathbb{Q}$. However, if a finite field $\mathbb{F}_q$ has been fixed, then the space required to represent the coordinates has an upper bound, so it is of order $O(1)$.

```
Given  : An elliptic curve $E$.
Input  : $P \in E$, $n \in \mathbb{Z}$
Output: $nP$
if $n < 0$ then
    Rerun algorithm with $-n$ and $-P$
let $a = n$, $B = \infty$, and $C = P$
while $a \neq 0$ do
    if $a$ is even then
        $a \to a/2$
        $C \to 2C$
    else if $a$ is odd then
        $a \to a - 1$
        $B \to B + C$
end
return $B$
```

Algorithm 3: $O(\log(n))$ method to compute nP.

For all $x \in \mathbb{F}_q$, where q is odd, we define a generalized Legendre symbol by

$$\left(\frac{x}{\mathbb{F}_q}\right) = \begin{cases} 1 & \text{if } t^2 = x \text{ has a solution } t \in \mathbb{F}_q^\times, \\ -1 & \text{if } t^2 = x \text{ has no solution } t \in \mathbb{F}_q, \\ 0 & \text{if } x = 0. \end{cases}$$

Theorem 4 *Let $E(\mathbb{F}_q) : y^2 = x^3 + ax + b$ be an elliptic curve. Then*

$$\#E(\mathbb{F}_q) = q + 1 + \sum_{x \in \mathbb{F}_q} \left(\frac{x^3 + ax + b}{\mathbb{F}_q}\right).$$

Example 2.3 Suppose we want to compute the order of $E(\mathbb{F}_{125}) : y^2 = x^3 + x + 1$. We can do so by considering $E(\mathbb{F}_5)$ and applying Theorem 3. A straightforward computation using Theorem 4 shows that $\#E(\mathbb{F}_5) = 9 = 5 + 1 + 3$. By using the quadratic formula, we compute the roots of $x^2 + 3x + 5$, which are $\frac{-3 \pm \sqrt{-11}}{2}$. Then, the cubes of the roots are $9 \pm 2\sqrt{11}$, so the sum of their cubes is 18. Therefore, we may compute

$$\#E(\mathbb{F}_{125}) = 5^3 + 1 - 18 = 108.$$

⊠

While Theorem 4 depicts an explicit algorithm for computing the order of an elliptic curve group, its time complexity is $O(\alpha q)$, where α is the time complexity of computing $\left(\frac{x}{\mathbb{F}_q}\right)$. Although we can amortize α be squaring all the elements of $\mathbb{F}_q$ beforehand and holding a list of squares in memory, a better approach is to reduce the problem of computing $\#E(\mathbb{F}_q)$ to the problem of computing the order of a point on the curve.[4]

How does knowing the order of a point P, denoted $|P|$, help us compute $N = \#E(\mathbb{F}_q)$? Hasse's Theorem tells us that

$$(\sqrt{q} - 1)^2 = q + 1 - 2\sqrt{q} \leq N \leq q + 1 + 2\sqrt{q} = (\sqrt{q} + 1)^2,$$

so we have an interval of length $4\sqrt{q}$ which we know contains N. By Lagrange's Theorem, $|P|$ divides N for all $P \in E$. Suppose that we found a point $P_1 \in E$ such that $|P_1| > 4\sqrt{q}$. We know that at least one multiple

[4] See Appendix A for an explanation of how the Baby Step–Giant Step algorithm can be modified to compute the order of a point. For now, we will carry on the discussion of computing $\#E(\mathbb{F}_q)$, assuming that we have a black box that computes the order of a point.

of $|P_1|$, namely N, is in our interval. Furthermore, because $|P_1|$ is larger than the length of the interval, N is the only such multiple.

More generally, if $P_1, \ldots, P_k \in E$, then each $|P_i|$ divides N, and so $M = \operatorname{lcm}(|P_1|, \ldots, |P_k|)$ divides N as well. We can therefore find the value of N by adding random points to our list. Each time, we compute the least common multiple of the orders of the points, call it M. Then, we see which multiples of M are in the interval $[(\sqrt{q}-1)^2, (\sqrt{q}+1)^2]$. If only one multiple of M is in the interval, then that value is N. Otherwise, we continue adding points to our list.

```
Input  : An elliptic curve $E(\mathbb{F}_q) : y^2 = x^3 + ax + b$.
Output: $\#E(\mathbb{F}_q)$.
let $M = 1$ // $\operatorname{lcm}(1, n) = n$ for all $n$, so we initialize $M$ as 1.
for $x \in \mathbb{F}_q$ do
    if $x^3 + ax + b$ is a square in $\mathbb{F}_q$ then
        // $P$ and $-P$ have the same order, so either value for $y$ works.
        let $y = \pm\sqrt{x^3 + ax + b}$
        let $m = |P|$, where $P = (x, y)$
        $M \to \operatorname{lcm}(M, m)$
        // Now we see if $M$ is large enough to determine $\#E(\mathbb{F}_q)$.  Observe that
        $a \leq kM \leq b$ if and only if ${}^{a}/_{M} \leq k \leq {}^{b}/_{M}$, so the number of multiples of $M$ in
        $[a, b]$ is the same as the number of integers in $[{}^{a}/_{M}, {}^{b}/_{M}]$.  Furthermore, there is
        a unique solution $k$ if and only if the ceiling of ${}^{a}/_{M}$ and the floor of ${}^{b}/_{M}$ are
        both $k$.
        let $k_1 = \lceil(\sqrt{q}-1)^2/M\rceil$, $k_2 = \lfloor(\sqrt{q}+1)^2/M\rfloor$
        if $k_1 = k_2$ then
            return $k_1 M$
end
```

Algorithm 4: Using $|P|$ for $P \in E(\mathbb{F}_q)$ to compute $\#E(\mathbb{F}_q)$.

3 Elliptic Curves and Cryptography

This section serves as a synthesis of the previous two sections by giving a concrete example of implementing and using a cryptosystem based on elliptic curves. First, we cover some encoding schemes proposed by Neal Koblitz in [6] that allow us to encode plaintext messages as points on an elliptic curve. Then, we will show how Adrian and Beth can use elliptic curves to perform the Three-pass protocol. Finally, we will compare the time it takes to perform the protocol with the time it takes for Eve to decrypt the message using the Baby Step–Giant Step algorithm. While the scale of the example is not akin to professional cryptographic standards, it will nevertheless illustrate the security of elliptic curve cryptography.

3.1 Koblitz Encodings

Recall that our model of cryptography involves a message space, such as all binary strings of some fixed length, and bijections on that message space that are inverses. Suppose that the cardinality of our message space is M, so that the messages can be bijected on the integers $\{m \in \mathbb{Z} : 0 \leq m < M\}$. Then, we may encode messages as points on an elliptic curve by encoding their associated integers. In [6], Koblitz offers three encoding schemes, of which we describe the first two.

Encoding Scheme 1 Pick a prime $p \geq 5$ and an even integer $n = 2n'$ such that $M \leq p^{n'}$. To encode a message m onto $E(\mathbb{F}_{p^n})$, write m in base-p: $m = m_0 + m_1 p + \cdots + m_{n'-1} p^{n'-1}$. Pick a basis $b_0, \ldots, b_{n'-1}$ for the field extension $\mathbb{F}_{p^{n'}}/\mathbb{F}_p$ (considered as an $\mathbb{F}_p$-vector space of dimension n'), and let

$$x = m_0 b_0 + m_1 b_1 + \cdots + m_{n'-1} b_{n'-1}.$$

Then, we can evaluate the righthand side of E's Weierstrass equation at x, denoting the obtained value by s. If s is a square in $\mathbb{F}_{p^{n'}}$, then we may let $y = \sqrt{s}$. Otherwise, $t^2 - s$ is irreducible in $\mathbb{F}_{p^{n'}}[t]$, and we may adjoin its roots to obtain a solution y in the quadratic extension $\mathbb{F}_{p^{2n'}} = \mathbb{F}_{p^n}$. Either way, we obtain a point $P = (x, y) \in E(\mathbb{F}_{p^n})$ to which we may encode m. To decode and obtain m from P, we determine the values $m_0, \ldots, m_{n'-1}$ using x and the basis, after which m can be computed.

Encoding Scheme 2 A simpler encoding scheme is to associate a message m with a point P such that the x-coordinate of P is m. However, such a point would only exist on a given curve $E(F) : y^2 = x^3 + ax + b$ if $m^3 + am + b$ is a square in the finite field F, which may not always be the case.

To remedy this, we give m multiple x-coordinates to try and find a point on the curve. We pick a prime integer $p > 100M$, and let E be an elliptic curve defined over $\mathbb{F}_p$. For each $n \in \{0, 1, \ldots, M-1\}$, we evaluate the righthand side of E's Weierstrass equation at $100m, 100m+1$, and so on, up to $100m+99$. This effectively obtains 100 random elements in $\mathbb{F}_p$. Since half of the elements in $\mathbb{F}_p^\times$ are squares, the probability that we will not obtain a point to which m can be encoded is 2^{-100}. Finally, we can decode a point P by dividing its x-coordinate by 100 and taking the floor of that quotient.

▶ **Remark** Since we are relying on the DLP to provide security, we want the points that encode our messages to have large order. So, if $(100n+i)^3 + a(100n+i) + b = 0$, then we should skip onto $100n+i+1$ and not use $P = (100n+i, 0)$, which has order 2.

3.2 Example: Three-pass Protocol with Elliptic Curves

Suppose that Adrian wants to send a message to Beth. We assume that Adrian's message is represented in binary, and that we will use the second of the two aforementioned encoding schemes. Since the size of Adrian's message could be arbitrarily large, we pick a positive integer s and encrypt the message s bytes at a time. Therefore, the size of our message space is 2^{8s}.

Say $s = 1$, and Adrian wants to use the Three-pass Protocol to send the single ASCII character "A", which is 0x41 in hexadecimal, or 65 in decimal, to Beth. They publicly agree on a prime integer p greater than $100 \cdot 2^8 = 25600$ and an elliptic curve defined over $\mathbb{F}_p$. Let $p = 25601$ and $E(\mathbb{F}_p)$ be given by the equation $y^2 = x^3 + x + 1$. Using the Koblitz encoding scheme described earlier, Adrian obtains the point $P = (6500, 12257)$.

We can easily compute $|P| = 8512$. Adrian picks a random unit modulo $|P|$, say $m_A = 4401$. Then, he computes

$$M_1 = m_A P = (110, 8415),$$

and sends it to Beth. Because m_A is relatively prime to $|P|$, Beth can compute $|M_1| = |P|$ and pick her own random unit $m_B = 1331$. The following points are computed and transmitted accordingly:

$$M_2 = m_B M_1 = (20823, 20645)$$
$$M_3 = m_A^{-1} M_2 = (15429, 6895).$$

Finally, Beth computes $M_4 = m_B^{-1} M_3 = P$, and then decodes the original message A = 65 from P.

Below is a table showing the average time it takes to complete the computations involved in this process as the block size s increases. To illustrate the security of protocol, we also list the average time it takes for Eve to decrypt the message using the Baby Step–Giant Step algorithm to solve discrete logarithms. The time it takes to transmit M_1, M_2, and M_3, as well as the time needed to confirm that m_A and m_B are not inverses, is not accounted for. I wrote my own code for elliptic curves and their algorithms using SageMath [8], and ran it on a virtual machine with about 13 gigabytes of RAM and 8 Intel i7 processors. The values shown are averages obtained by Python's timeit module, which times a script by returning the best result from 5 tests, where each test consists of looping the script 10 times and returning the average time per loop.

The script takes in the block size s as input and does the following:

1. Find the smallest prime p larger than $100 \cdot 2^{8s}$.

2. Initialize an integer m whose value is the ASCII encoding of the string consisting of s A's (i.e. if $s = 3$, then $m = 0\text{x}414141$).
3. Encode m as a point on the elliptic curve $E(\mathbb{F}_p) : y^2 = x^3 + x + 1$.
4. Perform the Three-pass protocol, which includes computing $\#E(\mathbb{F}_p)$.

Only the last two steps are timed.

Block Size s	Average Protocol Time	Average Baby Step–Giant Step Time
1	2.23 ms	2.51 ms
2	4.26 ms	20.7 ms
3	8.05 ms	271 ms
4	18.5 ms	9.23 s
5	69.2 ms	$>$ 20 min
6	370 ms	$>$ 20 min
7	1.85 s	$>$ 20 min
8	5.41 s	$>$ 20 min

Figure 3.1: Average computation times to perform/crack the Three-pass Protocol. When $s = 5$, so that $p > 100 \cdot 2^{40}$, the Baby Step–Giant Step algorithm exhausted the memory on my machine, which then halted the program after running for about half an hour.

4 Hyperelliptic Curves

Let g be a positive integer, let F be a field, and let f, h be polynomials in $F[x]$ such that f is monic of degree $2g + 1$ and $\deg(h) \leq g$. Then, the **hyperelliptic curve** C is the set of points in $\mathbf{P}^2_{\overline{F}}$ that satisfy the homogenized form of

$$y^2 + h(x)y = f(x),$$

provided that the curve is nonsingular. We call g the **genus** of C, and say that C is defined over F. When F is not of characteristic 2, then we can do as we did for generalized Weierstrass equations and define the same curve C with an equation $y^2 = f(x)$, where f is a monic polynomial in $F[x]$ with degree $2g+1$. We will only consider this case, and note that the non-singularity condition (for affine points) means that f is separable. Furthermore, $P = (x, y) \in C$ if and only if $Q = (x, -y) \in C$. For ordinary elliptic curves, $Q = -P$. For hyperelliptic curves, we define an involution $\omega : C \to C$ by $\omega(P) = Q$.

▶ **Remark** As with elliptic curves (which are hyperelliptic curves of genus $g = 1$), the only non affine point on C is $\infty = [0 : 1 : 0]$. However, when $g > 1$, ∞ is a singular point. We can treat C as a nonsingular curve by transforming it in such a way that the affine points remain fixed, and C obtains a new unique point on the hyperplane, which we denote using ∞.

The prefix "hyper-" in "hyperelliptic curve" refers to the fact that the polynomial defining the curve may be of higher degree than 3, which is the degree of the Weierstrass equations for elliptic curves. As a consequence of this, we cannot use the Chord and Tangent method to derive a group structure. If we take two points on a quintic curve and extend the line between them, it is not guaranteed that there will be a unique third point of intersection between the line and the curve.

Rather than form a group from the points on a hyperelliptic curve, we will begin with a large group based on formal sums of those points, and then take a quotient of that group to obtain something much more practical. We briefly discuss the theory of those formal sums for general nonsingular algebraic curves.

4.1 Divisors

Let C be a nonsingular algebraic curve defined over a field F. As with elliptic curves, we take this to mean that C is the set of points in $\mathbf{P}^2_{\overline{F}}$ that satisfy a polynomial equation in $F[x, y]$ such that the partial derivatives of x and y never simultaneously vanish. For each point $P \in C$, we define a symbol $[P]$. A **divisor** D on C is a finite linear combination of these symbols that have integer coefficients:

$$D = \sum_i n_i [P_i], \quad n_i \in \mathbb{Z}.$$

The set of divisors on C, denoted $\mathrm{Div}(C)$, form an abelian group where addition of divisors is defined by adding the coefficients of corresponding symbols, and the identity is the empty sum. Define the **degree** of a divisor to be the sum of its coefficients. The set of degree 0 divisors, $\mathrm{Div}^0(C)$, is a subgroup of $\mathrm{Div}(C)$.

▶ **Remark** When C is a hyperelliptic curve, $P \in C$ implies $\omega(P) \in C$, so $[\omega(P)] \in \mathrm{Div}(C)$. We can extend ω to $\mathrm{Div}(C)$ by defining $\omega\left(\sum_i [P_i]\right) = \sum_i [\omega(P_i)]$.

4.1.1 Divisors of Rational Functions

Let C be a nonsingular algebraic curve defined over F, and let f be a rational function on C. That is, $f = \dfrac{p(x,y)}{q(x,y)}$, where p and q are polynomials in $\overline{F}[x,y]$. Let $P = (x,y)$ be an affine point on C. We say that f has a **zero** at P if $f(x,y) = 0$. If $f(x,y) = \infty$, then we say that f has a **pole** at P. In other words, f has a zero at P when P is a root of p, and it has a pole at P when P is a root of q.

For every point $P \in C$, there exists a rational function u_P such that $u_P(P) = 0$ and every rational function $f \in \overline{F}(x,y)$ can be written as

$$f = u_P^r g, \quad \text{with } r \in \mathbb{Z} \text{ and } g(P) \neq 0, \infty.$$

We call u_P a **uniformizer** at P, and we define r to be the **order** of f at P, written $\mathrm{ord}_P(f)$. Essentially, uniformizers allow us to count the multiplicity of zeros and poles.

The following theorem (Proposition 11.1 in [10]) combines the previous discussions on divisors and rational functions.

Theorem 5 *Let f be a nonzero rational function on a nonsingular algebraic curve C.*

(a) f has only finitely many zero and poles,

(b) f has the same number of zeros and poles (counted with order), and

(c) if f has no zeros or poles, then f is a constant function.

In light of this theorem we may define, for every rational function f on C, a divisor

$$\mathrm{div}(f) = \sum_{P \in C} \mathrm{ord}_P(f)[P] \in \mathrm{Div}^0(C).$$

Part (a) of Theorem 5 makes $\mathrm{div}(f)$ well defined, since it implies that the sum is finite, and part (b) implies that $\deg(\mathrm{div}(f)) = 0$. If D is a divisor on C and $D = \mathrm{div}(f)$ for some rational function f then we say that D is a **principal divisor**.

Let P be any point on C, and let u_P be a uniformizer for P. For any two rational functions f, g on C, we have that

$$f = u_P^{r_1} h_1 \quad \text{and} \quad g = u_P^{r_2} h_2,$$

where $h_1(P), h_2(P) \neq 0, \infty$. This implies that $fg = u_P^{r_1 + r_2} h_1 h_2$, and that $h_1(P)h_2(P) \neq 0, \infty$. Therefore,

$$\mathrm{ord}_P(fg) = \mathrm{ord}_P(f) + \mathrm{ord}_P(g),$$

and so if $D_1 = \mathrm{div}(f)$ and $D_2 = \mathrm{div}(g)$, then $D_1 + D_2 = \mathrm{div}(fg)$. Consequently, the set of principal divisors on C, which we denote by $\mathrm{Div}^*(C)$, is a normal subgroup of $\mathrm{Div}^0(C)$.

The quotient group $\operatorname{Div}^0(C)/\operatorname{Div}^*(C)$ is called the **Jacobian variety** of C and denoted by J. It is the group we will eventually discuss using for cryptographic purposes. However, to see why the Jacobian is a natural generalization of the elliptic curve group derived from the Chord and Tangent method, it is worth showing that the Jacobian of an elliptic curve is isomorphic to its Chord and Tangent group.

Let E be an elliptic curve. We define the **sum** of a divisor on E as

$$\operatorname{sum}\left(\sum_i n_i[P_i]\right) = \sum_i n_i P_i.$$

In other words, the sum of a divisor is simply its evaluation as per the usual group operation for elliptic curves. Observe that the sum function defines a homomorphism between $\operatorname{Div}(E)$ and E. Furthermore, for any $P \in E$, we may define the divisor $D = [P] - [\infty]$. Since $D \in \operatorname{Div}^0(E)$ and $\operatorname{sum}(D) = P$, we conclude that $\operatorname{sum}|_{\operatorname{Div}^0(E)}$ is a surjective homomorphism onto E. It is true that $\operatorname{sum}(D) = \infty$ if and only if D is a principal divisor ([10, Theorem 11.2]), and therefore $\operatorname{Div}^*(E)$ is the kernel of $\operatorname{sum} : \operatorname{Div}^0(E) \to E$. So by the First Isomorphism Theorem for groups, E is isomorphic $\operatorname{Div}^0(E)/\operatorname{Div}^*(E)$, its own Jacobian.

4.2 Jacobians of Hyperelliptic Curves

In order to grasp the structure of a curve's Jacobian J, we will show how J's elements, which are cosets of $\operatorname{Div}^*(C)$, can be more practically represented. We begin by characterizing the divisors of polynomials. A polynomial $U(x) \in \overline{F}[x]$ can be split into its linear factors $\prod_i (x - a_i)^{c_i}$. Therefore, $\operatorname{div}(U) = \sum c_i \operatorname{div}(x - a_i)$. On the hyperelliptic curve $C : y^2 = f(x)$, there are two points on C where $x - a_i$ evaluates to 0: $P = (a_i, \sqrt{f(a_i)})$ and $\omega(P)$. If $f(a_i) = 0$, then $P = \omega(P)$, and the line $x - a_i$ is tangent to C at P, so that $\operatorname{ord}_P(x - a_i) = 2$. Regardless, there are two poles that must be accounted for, and since a polynomial can not have a pole at an affine point, it must be that $x - a_i$ has a double pole at ∞. So, $\operatorname{div}(x - a_i) = [P] + [\omega(P)] - 2[\infty]$. We have proven the first part of the following theorem, and a proof for the second half can be found in [10, Proposition 13.2].

Theorem 6

(a) Let $U(x) = \prod_i (x - a_i)^{c_i}$. Then

$$\operatorname{div}(U) = \sum_i c_i \left([P_i] + [\omega(P_i)] - 2[\infty]\right),$$

where $P_i = (a_i, \sqrt{f(a_i)}$.

(b) Let $V(x)$ be a polynomial and write $f(x) - V(x)^2 = \prod_i (x - a_i)^{d_i}$. Then

$$\operatorname{div}(y - V) = \sum_i d_i \left([(a_i, b_i)] - [\infty]\right),$$

where $b_i = V(a_i)$, and $b_i = 0$ implies that $d_i = 1$.

Next, we say that $D \in \operatorname{Div}^0(C)$ is **semi-reduced** if D is of the form $\sum_i c_i \left([P_i] - [\infty]\right)$, where $P_i = (a_i, b_i)$, and:

1. $c_i \geq 0$ for all i,
2. if $b_i = 0$, then $c_i = 0$ or 1, and
3. if $b_i \neq 0$, then $(a_i, -b_i) = \omega(P_i)$ does **not** occur in D.

Intuitively, a semi-reduced divisor is like taking the divisor of a polynomial and removing either $[P_i] - [\infty]$ or $[\omega(P_i)] - [\infty]$ for each of the polynomial's roots a_i. If $\sum_i c_i \leq g$, then we say that D is **reduced**. For all $D_1 = \sum_i c_i \left([P_i] - [\infty]\right)$ and $D_2 = \sum_i d_i \left([P_i] - [\infty]\right)$, the **greatest common divisor** of D_1 and D_2 is

$$\gcd(D_1, D_2) = \sum_i \min\{c_i, d_i\} \left([P_i] - [\infty]\right).$$

Corollary *For all polynomials $U(x)$ and $V(x)$,* $\operatorname{div}(y-V)$ *and* $\gcd(\operatorname{div}(U), \operatorname{div}(y-V))$ *are semi-reduced.*

Let $D = \gcd(\operatorname{div}(U), \operatorname{div}(y-V))$. To suppose that the coefficient of each $([P] - [\infty])$ term in D is the same as the multiplicity c_i of the corresponding root of U is equivalent to supposing that $c_i \leq d_i$ for all i. Therefore:

Theorem 7 (Proposition 13.4 in [10]) *Let $D = \sum_i c_i([P_i] - [\infty])$ be semi-reduced divisor, where $P_i = (a_i, b_i)$. Let $U(x) = \prod_i (x - a_i)^{c_i}$, and let $V(x)$ be a polynomial such that $V(a_i) = b_i$. Then*

$$D = \gcd(\operatorname{div}(U), \operatorname{div}(y-V)) \iff U \mid f - V^2.$$

This theorem establishes a relation between certain pairs of polynomials (U, V) and semi-reduced divisors, where $(U, V) \sim \gcd(\operatorname{div}(U), \operatorname{div}(y-V))$. Let $D = \sum_i c_i([P_i] - [\infty])$ be a semi-reduce divisor. If we require that U be monic such that $\deg(U) = \sum_i c_i$, and also that $\deg(V) < \deg(U)$, then there exists a unique pair (U, V) such that $U \mid f - V^2$, or equivalently, $D = \gcd(\operatorname{div}(U), \operatorname{div}(y-V))$ ([10, Theorem 13.5]).

If we restrict to pairs of polynomials (U, V) where $\deg(U) \leq g$, then we obtain a one-to-one correspondence with reduced divisors. [10, Theorem 13.6] states that for each $D \in \operatorname{Div}^0(C)$, there is a unique reduced divisor D' such that $D - D' \in \operatorname{Div}^*(C)$, so therefore each congruence class in C's Jacobian J has a unique reduced divisor, and we have a one-to-one correspondence between pairs of polynomials (satisfying all the properties mentioned) and the elements of $J = \operatorname{Div}^0(C)/\operatorname{Div}^*(C)$. We call (U, V) the **Mumford representation** of its associated congruence class of divisors. Note that we will sometimes abuse notation and use (U, V) to refer to $D = \gcd(\operatorname{div}(U), \operatorname{div}(y-V))$, even when D is only semi-reduced.

4.3 Cantor's Algorithm

The following algorithm, published by David Cantor [1], computes the group operation of a Jacobian J using the Mumford representation. We do not supply a proof, but do provide an intuition for how the algorithm works.

Suppose we have the Mumford pairs (U_1, V_1) and (U_2, V_2), which correspond to congruence classes of divisors in $\operatorname{Div}^0(C)$ that respectively contain reduced divisors D_1, D_2, where $D_i = \gcd(\operatorname{div}(U_i), \operatorname{div}(y - V_i))$. Clearly, the sum in J of these divisor classes is the class that contains $D_1 + D_2$, so the first phase of Cantor's Algorithm is to combine the polynomials U_i and V_i in such a way that obtains a new pair (U', V') that is the Mumford representation of a semi-reduced divisor that lies in the same congruence class as $D_1 + D_2$. In more detail, we inductively apply the Euclidean Algorithm to find the the greatest common divisor d of U_1, U_2, and $V_1 + V_2$, and find polynomials h_1, h_2, and h_3 such that

$$d = h_1 U_1 + h_2 U_2 + h_3(V_1 + V_2).$$

Then, we let

$$\begin{aligned} U' &= \frac{U_1 U_2}{d^2}, \\ V_0 &= \frac{h_1 U_1 V_2 + h_2 U_2 V_1 + h_3(V_1 V_2 + f)}{d}, \text{ and} \\ V' &\equiv V_0 \pmod{U'} \text{ where } \deg(V') < \deg(U'). \end{aligned}$$

It can be shown that (U', V') corresponds to a semi-reduced divisor that is in the same congruence class as $D_1 + D_2$.

The second and final phase of the algorithm is a reduction procedure that modifies U' and V' until (U', V') corresponds to the reduced divisor that is congruent to $D_1 + D_2$. Recall that $\deg(V') < \deg(U')$ and $U' \mid f - V'^2$. So, if $\deg(U') > g = \deg(f)$, then $\deg\left(\frac{f - V'^2}{U'}\right) < \deg(U')$, so we reassign U' to be this quotient and compute a new value for V' where $\deg(V') < \deg(U')$ and (U', V') corresponds to a semi-reduced divisor that is congruent to the divisor corresponding to the pair of original values for U' and V'. This process is applied recursively until $\deg(U') \leq g$.

▶ **Remark** In Cantor's Algorithm and later on in this text, we will reduce a polynomials modulo another polynomial. Given a field F two polynomials $U(x), V(x) \in F[x]$, we define $V \ \% \ U$ to be the unique polynomial $V'(x)$ such that $V \equiv V' \pmod{U}$ and $\deg(V') < \deg(U)$.

```
Given  : A hyperelliptic curve $C : y^2 = f(x)$ of genus $g$ with Jacobian $J$.
Input  : Mumford representations $(U_1, V_1), (U_2, V_2)$ of two divisor classes of $J$.
Output: The Mumford representation $(U', V')$ of the sum of those divisor classes.
let $d_0, a_1, b_1 = \text{xgcd}(U_1, U_2)$
let $d, a_2, b_2 = \text{xgcd}(d_0, V_1 + V_2)$
let $h_1 = a_1 a_2$, $h_2 = a_2 b_1$, and $h_3 = b_2$
let $V_0 = \dfrac{h_1 U_1 V_2 + h_2 U_2 V_1 + h_3(V_1 V_2 + f)}{d}$
let $U' = \dfrac{U_1 U_2}{d^2}$ and $V' = V_0 \ \% \ U'$
Scale $U'$ so that it is monic
while $\deg(U') > g$ do // Reduction procedure
    $U' \to \dfrac{f - V'^2}{U'}$
    $V' \to -V' \ \% \ U'$
    Scale $U'$ so that it is monic
end
return $(U', V')$
```

Algorithm 5: Cantor's Algorithm. The function xgcd(f, g) refers to the Extended Euclidean Algorithm, which returns a triple (d, a, b) where $d = \gcd(f, g)$ and $d = af + bg$.

5 Hyperelliptic Curve Cryptography

In order to make use of hyperelliptic curve of genus $g \geq 2$ for cryptographic applications, we must obtain finite groups from their Jacobians. For an elliptic curve that was defined over a finite field $\mathbb{F}_q$, we could restrict the curve to the set of points whose coordinates are in $\mathbb{F}_q$.

5.1 The Frobenius Map and Finite Subgroups of Jacobians

Let $C : y^2 = f(x)$ be a hyperelliptic curve defined over a finite field $\mathbb{F}_q$, with Jacobian J. Let ϕ be the Frobenius endomorphism for $\overline{\mathbb{F}_q}$, i.e. $\phi(x) = x^q$. We extend ϕ to divisors by defining $\phi\left(\sum_i [P_i]\right) = \sum_i \phi([P_i])$, where $\phi([\infty]) = [\infty]$ and $\phi([(x, y)]) = [(x^q, y^q)]$.

A divisor $D \in \text{Div}(C)$ is **defined over** $\mathbb{F}_q$ if $\phi(D) = D$. Note that D can be defined over $\mathbb{F}_q$ even if it contains points with coordinates outside $\mathbb{F}_q$. The points that ϕ does not fix are instead permuted to other points in D. We say that a divisor class $\mathcal{A} \in J$ is defined over $\mathbb{F}_q$ if there exists some divisor $D \in \mathcal{A}$ such that $\phi(D) \in \mathcal{A}$ as well, or equivalently, $\phi(D) - D$ is a principal divisor. It can be shown that the principal divisors $\text{Div}^*(C)$ are closed under ϕ, from which it follows that for all other $D' \in \mathcal{A}$, $\phi(D') \in \mathcal{A}$. In other words, each divisor class in J is either closed under ϕ or gets mapped completely outside itself. We define $J(\mathbb{F}_q)$ to be the set of all divisor classes that are defined over $\mathbb{F}_q$.

If R is a reduced divisor, then so is $\phi(R)$, since $\deg(R) = \deg(\phi(R))$. Every divisor class $\mathcal{A} \in J$ contains a unique reduced divisor R, so if $\mathcal{A}$ is defined over $\mathbb{F}_q$, then so is R. In general, if (U, V) is the Mumford representation of a reduced divisor R, then the representation of $\phi(R)$ is (U^ϕ, V^ϕ), where the superscripts denote applying ϕ to the coefficients of U and V. So if R is defined over $\mathbb{F}_q$, then it must be that $U^\phi = U$ and $V^\phi = V$, which is to say that the coefficients of U and V lie in the fixed field $\mathbb{F}_q$ of ϕ. Conversely if $U, V \in \mathbb{F}_q[x]$, then the divisor corresponding to (U, V) must be defined over $\mathbb{F}_q$. Therefore, the divisor classes in $J(\mathbb{F}_q)$ are exactly those represented by polynomials in $\mathbb{F}_q[x]$.

Theorem 8 *Let C be a hyperelliptic curve of genus g defined over $\mathbb{F}_q$ with Jacobian J. Then $J(\mathbb{F}_q)$ is a finite subgroup of J.*

Proof. $J(\mathbb{F}_q)$ consists of the divisor classes in J that are represented by pairs of polynomials $U, V, \in \mathbb{F}_q[x]$. Since $\deg(V) < \deg(U) \leq g$, there is a finite number of representations (U, V), and so $J(\mathbb{F}_q)$ is finite.

We already know that the identity in J, $\text{Div}^*(C)$, is defined over $\mathbb{F}_q$. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be divisor classes in $J(\mathbb{F}_q)$ with respective reduced divisors R_1 and R_2. Clearly, $\omega(R_1)$ is a reduced divisor, and because $R_1 + \omega(R_1) \in \text{Div}^*(C)$ ([10, Proposition 13.3]), the inverse of $\mathcal{A}_1$ is the divisor class containing $\omega(R_1)$. Observe that for any affine point $(x, y) \in C$,

$$\phi(\omega([(x, y)])) = [(x^q, -y^q)] = \omega(\phi([(x, y)])),$$

so $\omega(R_1)$ is also defined over $\mathbb{F}_q$, implying that $-\mathcal{A}_1 \in J(\mathbb{F}_q)$. Finally, $\phi(R_1+R_2) = \phi(R_1)+\phi(R_2) = R_1+R_2$, so $J(\mathbb{F}_q)$ is closed under the Jacobian group operation. □

Example 5.1 Consider the genus 2 hyperelliptic curve $C : y^2 = x^5 + x + 1$, defined over $\mathbb{F}_5$, and let J denote the Jacobian. The pairs $(x+1, 3)$ and $(x^2+x, x+4)$ represent divisor classes in $J(\mathbb{F}_5)$. The greatest common divisor of $x+1$, x^2+x, and $3+(x+4) = x+2$ is 1, and our Euclidean coefficients are $h_1 = 4$, $h_2 = 0$, and $h_3 = 1$. Therefore, we let

$$V_0 = 4(x+1)(x+4) + 3(x+4) + (x^5+x+1) = x^5 + 4x^2 + 4x + 4.$$

We compute

$$U' = (x+1)(x^2+x) = x^3 + 2x^2 + x \text{ and } V' = V_0 \ \% \ U' = x + 4.$$

Since $\deg(U') = 3$, we must apply the reduction procedure. Observe that $\frac{f-V'^2}{U'} = x^2+3x+3$, so we reassign U' to this polynomial. Then we reassign V' to $-V' \ \% \ U' = 4x + 1$, and finally return the pair (U', V). ⊠

5.2 Concluding Remarks: Complexity, Discrete Logarithms, and Encodings

Now that we have obtained a finite group $J(\mathbb{F}_q)$ from the Jacobian of a hyperelliptic curve C, we may postulate about DLP-based cryptosystems implemented with these groups. Perhaps the most important factor to consider is the size of $J(\mathbb{F}_q)$. In [11], André Weil proved a generalization of Hasse's Theorem which states that the size of $J(\mathbb{F}_q)$ satisfies

$$(\sqrt{q} - 1)^{2g} \leq \#J(\mathbb{F}_q) \leq (\sqrt{q} + 1)^{2g}.$$

That is, given a hyperelliptic curve of genus g and a finite field of size q, we may obtain a finite abelian group whose size is of order $\Theta(q^g)$.

While the size of the group must be large in order to provide security, its operation must be easily computable so that the cryptosystem is practical. In this finite context, the inputs to Cantor's Algorithm are two pairs of polynomials, all of which lie in $\mathbb{F}_q[x]$ and have degree at most g. The algorithm consists wholly of polynomial arithmetic, mainly finding quotients and remainders, and this can be done quickly, especially since the coefficients have a fixed size. Overall, Cantor's Algorithm as depicted above requires $O(g^3)$ operations in $\mathbb{F}_q$ ([4, Section 2.5]).

We may also consider the space complexity of a point on $J(\mathbb{F}_q)$. For an elliptic curve, whose genus is 1, an affine point is identified by two coordinates in $\mathbb{F}_q$. In general, if C has genus g, then as previously mentioned, each point in $J(\mathbb{F}_q)$ is represented by two polynomials $U, V, \in \mathbb{F}_q[x]$ where U is monic and $\deg(V) < \deg(U) \leq g$. We store a polynomial by storing the values of its coefficients. Because U is assumed to be monic, we only need to store at most $g - 1$ values in $\mathbb{F}_q$. Similarly, $\deg(V) \leq g$, so we store at most $g - 1$ values to represent V as well. If $q = p^e$, and if we represent a value in $\mathbb{F}_q$ using e coordinates in $\mathbb{F}_p$ and a basis for the vector space $\mathbb{F}_q/\mathbb{F}_p$, where each coordinate is a binary integer between 0 and $p - 1$, then the storage required to represent a point on $J(\mathbb{F}_q)$ is of order $O(ge\log(p)) = O(g\log(q))$.

In the notation of hyperelliptic curves, the discrete logarithm problem is: given $\mathcal{A}, \mathcal{B} \in J(\mathbb{F}_q)$, where $\mathcal{A} = k\mathcal{B}$, determine k. Alternatively, we could say that we are given two divisors D_1 and D_2 that represent classes in $J(\mathbb{F}_q)$, where $D_1 \equiv kD_2$. The Baby Step–Giant Step algorithm works for an arbitrary group of order N, and returns an answer in time $O(\sqrt{N})$. Since $\#J(\mathbb{F}_q)$ is approximately q^g, we should expect Baby Step–Giant Step to complete in time $O(q^{g/2})$.

However, when g is large, there is another algorithm, based on index calculus, that performs better than baby Step–Giant Step. In §1.1.1 we used index calculus to solve discrete logarithms in $\mathbb{F}_p^\times$ by factoring group elements using factorizations in $\mathbb{Z}$. A linear system of congruence relations solved the DLP for all elements in our chosen factor base B, which allowed us to compute the logarithms of any element that factored completely over B.

Let C be a hyperelliptic curve of genus $g \geq 2$ that is defined over $\mathbb{F}_q$, and let $D \in \mathrm{Div}^0(C)$ be a semi-reduced divisor that corresponds to the pair of polynomials (U, V) where $U, V \in \mathbb{F}_q[x]$. [10, Proposition 13.12] states that if we can factor $U(x) = \prod_i U_i(x)$ in $\mathbb{F}_q[x]$, then we can separate D into $\sum_i D_i$, where D_i corresponds to (U_i, V_i) and $V_i = V \ \% \ U_i$. A semi-reduced divisor $D \sim (U, V)$ is **prime** if $\deg(U) \geq 1$ and U is irreducible in $\mathbb{F}_q[x]$.

Because we may factor divisors, and therefore also factor divisor classes in $J(\mathbb{F}_q)$, by factoring polynomials, we may apply index calculus. Pick a positive integer B, then enumerate all Mumford representations (T_j, W_j) such that T_j is irreducible. This will be our factor base. Let $D_i \sim (U_i, V_i)$ for $i = 1, 2$. To compute the logarithm, base D_2, of D_1, compute the Mumford representation (U, V) for $mD_1 + nD_2$ for random integers m, n. If (U, V) factors in our factor base, then we obtain a congruence $mD_1 + nD_2 \equiv \sum_j c_j(T_j, W_j)$. After accumulating enough of these relations, we may obtain a relation $m_0D_1 + n_0D_2 \equiv 0$ and solve for D_1.

There exists index calculus algorithms for Jacobians of hyperelliptic curves which run in time $O\left(g^5 q^{2-\frac{2}{g+1}+\varepsilon}\right)$, where $\varepsilon > 0$ is arbitrarily small (see [9]). For all $g \geq 3$, we have

$$2 - \frac{2}{g+1} + \varepsilon < \frac{g}{2}.$$

If we fix g to be a constant greater than 2, then the running time for index calculus is less than the running time for Baby Step–Giant Step (with respect to q). That is,

$$O\left(q^{2-\frac{2}{g+1}+\varepsilon}\right) < O\left(q^{g/2}\right).$$

In other words, for a given hyperelliptic curve C of genus $g \geq 3$, index calculus is more efficient than Baby Step–Giant Step for computing discrete logarithms in $J(\mathbb{F}_q)$. For this reason, genus 2 hyperelliptic curves are considered to have the most potential for cryptographic applications.

The final requirement for implementing a hyperelliptic curve cryptosystem is an encoding procedure. Rather than encode messages as points on a curve, we require a method that maps a message to a divisor class in the Jacobian of the curve. Much work has been done in recent years to develop encoding procedures (see [3] and [5]). As it turns out, encodings onto the curve C itself can be useful for encodings onto $J(\mathbb{F}_q)$. The basic idea is to associate a message m with a set of points, which can then be used to construct a divisor D that represents a divisor class in $J(\mathbb{F}_q)$.

However, these results are relatively new. Although there exist efficient encoding algorithms, there does not yet seem to be an industry standard with which to implement hyperelliptic curve cryptosystems.

Appendix

A Baby Step–Giant Step, and the order of $P \in E(\mathbb{F}_q)$

We can use the Baby Step–Giant Step algorithm to solve the discrete logarithm problem. If P and Q are points on an elliptic curve $E(\mathbb{F}_q)$, then this algorithm either finds an integer n such that $nP = Q$ or determines that no such integer n exists. Let $P \in E(\mathbb{F}_q)$, and suppose we want to find the order of P, denoted $|P|$, which is the smallest positive integer k such that $kP = \infty$. In a sense, finding the order of P is like computing the discrete logarithm of ∞, base-P. However, Algorithm 1 will return the trivial solution $n = 0$, so we will modify it so that it returns a positive value.

▶ **Remark** The modified Baby Step–Giant Step algorithm does not return $k = |P|$, but only *some* positive integer n such that $nP = \infty$. By Lagrange's Theorem, $nP = \infty$ if and only if $k \mid n$, so we know that $n = k$ if n has no proper divisor d such that $dP = \infty$. Furthermore, we only need to test n/p for each prime p that divides n. Consider the contrapositive: Suppose $k \mid n$, but $k \neq n$. Let p be a prime divisor of n/k, which implies that $p \mid n$. Therefore $pk \mid n$, and we may divide out by p to obtain $k \mid n/p$, so that $(n/p)\,P = \infty$. If n fails the test, so $(n/p)\,P = \infty$ for some prime $p \mid n$, then we recurse to n/p.

We still need to find some $n > 0$ such that $nP = \infty$. To do this, we compute two lists of multiples of P such that there is a guaranteed match between them $k_1P = k_2P$, where $k_1 \neq k_2$. Then, we may let $n = |k_1 - k_2| > 0$, and move onto the recursive part of the algorithm. The guarantee that such a match occurs comes from the following lemma (4.20 in [10]).

Lemma 1. *Let a and m be integers with $|a| \leq 2m^2$. Then there exist integers a_0 and a_1 with $-m < a_0 \leq m$ and $-m \leq a_1 \leq m$ such that*

$$a = a_0 + 2ma_1.$$

Proof. Let $a_0 \equiv a \pmod{2m}$, with $-m < a_0 \leq m$, and $a_1 = \dfrac{a - a_0}{2m}$. Then by the triangle inequality,

$$|a_1| \leq \frac{|a| + |-a_0|}{|2m|} \leq \frac{2m^2 + m}{2m} < m + 1.$$

□

Let $N = \#E(\mathbb{F}_q)$ (recall that this means $NP = \infty$), and then let $a = q + 1 - N$. If $m > \sqrt[4]{q}$, then Hasse's Theorem implies that $|a| \leq 2\sqrt{q} < 2m^2$. By the previous lemma, there exist integers a_0 and a_1 in the specified range such that

$$a = q + 1 - N = a_0 + 2ma_1 \implies q + 1 - 2ma_1 = a_0 + N.$$

Therefore, we obtain $(q + 1 - 2ma_1)P = (a_0 + N)P = a_0P$, so we may pick $|q + 1 - a_0 - 2ma_1|$ as our initial value in the recursive procedure to compute $|P|$.

Our algorithm will find a_0 and a_1 by virtually trying all possible pairs of values. The first step of the algorithm will be to compute all multiples iP for $i = 0, 1, \ldots, m$, where i corresponds to a_0. In truth, a_0 varies from $-m$ to m, but $-iP$ is just iP with the y-coordinate flipped, so it's easier to check for a match using the x-coordinate and then deduce whether the matching i value should be negated. Next, we compute $(q+1+2mj)P$ for $j = -m, -(m-1), \ldots, m$. (Here, j corresponds to a_1, and the $+2mj$ vs. $-2ma_1$ difference doesn't matter because of how a_1 varies.) Our choice of m guarantees that some value of j will obtain a point that matches, or is the inverse of, one of the iP's.

▶ **Remark** How do we know that we do not stumble on a trivial match? It does not help us if $q+1-2ma_1 = a_0$. We will never have this problem if $q + 1 - 2mj \neq i$ for all possible values of i and j. Say that we always pick the smallest value possible for m, which is $\lfloor \sqrt[4]{q} + 1 \rfloor$. Then, we will have no trivial matches as long as

$$q + 1 - 2(\sqrt[4]{q} + 1)^2 > \sqrt[4]{q} + 1$$

because the lefthand side is a lower bound for $q + 1 - 2mj$ and the righthand side is an upper bound for i. Fortunately, this inequality is satisfied for all $q \geq 23$.

▶ **Remark** Notice that the explicit value of $N = \#E(\mathbb{F}_q)$ was only needed for the proof of correctness of this algorithm, not the algorithm itself. In other words, we can run this algorithm even when N is unknown. The only value we need to know is q.

References

[1] David G. Cantor. Computing in the Jacobian of a hyperelliptic curve. *Math. Comp.*, 48(177):95–101, 1987.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, third edition, 2009.

[3] Pierre-Alain Fouque and Mehdi Tibouchi. Deterministic encoding and hashing to odd hyperelliptic curves. In *Pairing-based cryptography—Pairing 2010*, volume 6487 of *Lecture Notes in Comput. Sci.*, pages 265–277. Springer, Berlin, 2010.

[4] Michael Jacobson, Jr., Alfred Menezes, and Andreas Stein. Hyperelliptic curves and cryptography. In *High primes and misdemeanours: lectures in honour of the 60th birthday of Hugh Cowie Williams*, volume 41 of *Fields Inst. Commun.*, pages 255–282. Amer. Math. Soc., Providence, RI, 2004.

[5] Jean-Gabriel Kammerer, Reynald Lercier, and Guénaël Renault. Encoding points on hyperelliptic curves over finite fields in deterministic polynomial time. In *Pairing-based cryptography—Pairing 2010*, volume 6487 of *Lecture Notes in Comput. Sci.*, pages 278–297. Springer, Berlin, 2010.

[6] Neal Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48(177):203–209, 1987.

[7] Daniel Shanks. Class number, a theory of factorization, and genera. pages 415–440, 1971.

[8] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 7.5.1)*, 2017. `http://www.sagemath.org`.

[9] Nicolas Thériault. Index calculus attack for hyperelliptic curves of small genus. In *Advances in cryptology—ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Comput. Sci.*, pages 75–92. Springer, Berlin, 2003.

[10] Lawrence C. Washington. *Elliptic curves*. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, second edition, 2008. Number theory and cryptography.

[11] André Weil. Numbers of solutions of equations in finite fields. *Bull. Amer. Math. Soc.*, 55:497–508, 1949.