An Introduction to Elliptic Curve Cryptography

The Ohio State University "What is. . . ?" Seminar

Miles Calabresi

21 June 2016

Abstract

After the discovery that secure encryption (of, for instance, a client's confidential data at a bank) does not require previous contact (if the client wanted to join online without first coming in person), several algorithms were proposed to protect our communications. A beautiful example combines tools from Number Theory, Group Theory, and Planar Algebraic Geometry. In this talk we'll explore the implementation of this algorithm and why it's so secure. In short, we'll take a look at the question of "What is... Elliptic Curve Cryptography?"

Acknowledgments

This paper and the accompanying presentation are both largely drawn from a final project I put together for an Algebraic Geometry course in the fall of 2014. My sincere thanks are due to the instructor of the course, Professor Dagan Karp, and to the other students in the class for their feedback on that project.

I say as a disclaimer that I am not a cryptographer, and I tried to write this paper with an eye towards the math involved. I apologize in advance, especially to anyone studying cryptography, for any fudges, omissions, or mistakes in this paper; they are my responsibility.

1 Introduction: Public-Key Cryptography

1.1 Classical and Modern Ciphers

We humans are always looking for ways to keep communications private. However, as communication lines become longer, it is impossible for someone sending a message to be sure that it will be delivered to the intended recipient without being intercepted or overheard. Out of this need to keep the message safe from prying eyes comes cryptography: encoding the message in such a way that only the sender and the recipient can understand the intended meaning of its contents.

Until recently, it was thought that (securely) encrypted communication could only be accomplished through the two parties sharing a secret, such as a codeword or device. This kind of cipher, which I will refer to as a *classical cipher*, can obscure the message to all but those who know the secret; indeed, there are classical ciphers[1] that do this job quite well.[2] One drawback to all classical ciphers, however, is that they punt on the problem if the shared secret needs to be changed or if the two parties can't have secure contact before they need to communicate the confidential message. In these cases, how can two people who have never spoken before (to give a practical and concrete example, a banker and a new online client) guarantee that the sensitive information they exchange will not be comprehensible to anyone who intercepts their communications?

[1] For readers new to cryptography, I'll give a simple example of a classical cipher. Suppose two parties know that three is the secret number. Then only they would know that the place to meet is Copenhagen when one tells the other to meet in "gcceaoospraegbnelhtaaongwcenan" and not Georgetown or Casablanca. Of course, this example is too simple to be useful if stakes are high or if an interceptor has the means to monitor and be ready for the meeting to happen in any of the three cities, but it should give an idea of what is possible with even a single digit as a shared secret.

[2] I won't go into the details right now of which ciphers work well, or what "well" entails in various settings, but I claim that there are plenty that can render a given piece of information intractably obfuscated to anyone who doesn't know the secret.

In 1976, Whitfield Diffie, Martin Hellman, and Ralph Merkle[3] answered this question. As opposed to the classical methods, where a shared secret was needed beforehand to encode future communications, their method generated such secrets without the need for prior secure communication.

In the classical method, Private-Key Cryptography, there is a single secret both the sender and the recipient of the message share. With this one secret, encoding and decoding are the same: any eavesdropping outsider who can manage to find out the encipher key will then know the decipher key, and vice-versa. In the example in the first footnote, the number of letters to skip when writing the message is the same as the number of letters that the reader skips when trying to decode it.

In the new scheme (Public-Key Cryptography), encoding and decoding are opposite, but not equal, in that knowing the encipher key is almost[4] useless to any eavesdropper who wants to decipher the message, so it is safe to share that key publicly (hence the name "public key"). This way, anyone wishing to communicate securely can use the public key to encrypt their message, and only the creator of the decipher key can read the encoded message. The existence of such dual keys is the basis for the Diffie-Hellman-Merkle Key Exchange.

I'll close this section with two loose analogies. Classical (private-key) cryptography is like two people meeting for the first time, making two copies of a key to the same lock that they will use to lock up boxes they send back and forth. Public-key cryptography is like two people each choosing their own lock and key. They trade locks (while both locks are open) and use the other person's lock to send boxes. Then the lock's original owner can open it with the key they kept. Since only the locks are exchanged, any spy who takes a picture of each lock would have to reverse-engineer a key (presumably a difficult task) since the keys themselves were never brought out in the public encounter.

Alternatively, one can think of public-key a encryption as a single lock with two keyholes, one of which only closes the lock, while the other only opens it. One person can safely receive anything from another while only revealing the "locking" (encipher) key to third parties.

1.2 The Diffie-Hellman-Merkle Key Exchange

There are many methods for public-key encryption, but the Diffie-Hellman-Merkle exchange (DHM, usually referred to only by the first two names) is an example that illustrates the essential ideas clearly.

For simplicity, let us begin by assuming that the shared secret to be generated is an integer. We can and will drop this assumption later. As is customary in cryptography, I will use the following three names to illustrate how communications play out: Alice wishes to send Bob a private message, but they have not had any previous contact. Eve is an evil eavesdropper who sees all of Alice's and Bob's communications and tries to decipher any encryptions they employ to protect their messages.

The goal of the DHM exchange is to allow Alice and Bob to generate a shared secret (kept from Eve) without needing an existing secure channel or prior communication between them. They are able to generate this secret by choosing their own private numbers, a and b, and then obscuring them in some systematic way, such as by some function $f\colon \mathbb{Z} \to \mathbb{Z}$ (which they publicly agree upon using for this purpose, and hence Eve knows how they will use it), such that f has two properties. First, there must be a way (a binary operation that we'll denote by $\star\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$) to combine the obfuscated numbers $f(a)$ and $f(b)$ with the original numbers b and a (respectively) that intertwines with f in the sense that the combination $a \star f(b)$ is equal to the combination $b \star f(a)$. Second, Eve must not be able to compute this quantity given only $f(a)$ and $f(b)$. In particular, $f(a) \star f(b)$ should not equal $a \star f(b)$, and f^{-1} must be computationally intensive to find. If Eve could quickly determine x given $f(x)$, then she could recover a from $f(a)$ and then compute $a \star f(b)$ as easily as Alice. This difficulty for Eve allows Alice and Bob to share the numbers $f(a)$ and $f(b)$ with each other (and thus with Eve), but the magic of this algorithm lies in the ability to use the unenciphered numbers a and b with the enciphered numbers $f(b)$ and $f(a)$ to recover the same number, $a \star f(b)$.

This number $a \star f(b) = b \star f(a)$ is Alice and Bob's shared secret, which allows them to use any classical system to encrypt and decrypt future messages between the two of them. Eve will then be unable to decrypt these communications because she does not know the number $a \star f(b)$. Alice can now

[3]The 1976 paper was written by Diffie and Hellman, but Merkle is generally given credit for some central ideas.

[4]In theory, it is possible to work backwards and determine the decipher key from the encipher key, but when the encryption is done well, it is impractical to do so; an eavesdropper would have a better chance (given how long it would take on average, to determine the decipher key by working backwards) of simply guessing (or using a chimpanzee with a typewriter) to determine the decipher key.

send messages in any format she chooses – it need not even be numerical – by any number of classical ciphers.

For instance, using the method from the first footnote, suppose $a \star f(b) = b \star f(a) = 9$. Since Alice and Bob both know this secret number is (but Eve does not), Alice can hide her intended message one letter at a time with eight numbers (ideally chosen to look random but be intentionally confusing upon inspection) in between. Bob simply reads off every ninth letter to see Alice's intended message.

Note that the number $a \star f(b)$ is *not* the message that Alice is trying to send; it is a secret key that she can use to send any message she desires. With no knowledge of b, it is not practical for Alice to arrange for $a \star f(b)$ to reflect her intended message.

One remaining question is how Alice and Bob might choose such mappings f and $\star$ with these properties. Do any even exist? In the following section, we explore an example, a variant of the popular RSA algorithm.

1.3 Example Implementation of the DHM Key Exchange

Some common implementations of this key exchange construct a function f by leveraging the ease of multiplying two primes versus the difficulty of factoring their product. The idea is that multiplying numbers is a computationally fast process, while factoring them is comparatively quite slow. This disparity is a good basis to start building the above function f. Now we'll explore this method in more detail with Alice, Bob, and Eve.

Alice and Bob agree upon an ordered pair of primes: 13 as a modulus and 7 as a base.[5] Since they have not had prior communication, this agreement is public, so Eve knows that Alice and Bob will use 13 and 7 for these respective purposes. They also agree publicly (and so Eve knows) that their function f will be given by $f(x) = 7^x \mod 13$ and that $\star$ will be given by $x \star y = y^x \mod 13$.

Next, Alice and Bob each choose a positive integer, which they do not share. These numbers need not be prime (but should also be large). Suppose Alice chooses $a = 4$, and Bob chooses $b = 14$. Alice calculates $f(a) = 7^4 \mod 13 = 9$, and Bob calculates $f(b) = 7^{14} \mod 13 = 10$. They share these results $f(a)$ and $f(b)$ with each other, so Eve hears them as well.

Alice can now compute $a \star f(b) = 4 \star 10 = 10^4 \mod 13 = 3$, and Bob can compute $b \star f(a) = 14 \star 9 = 9^{14} \mod 13 = 3$. (I leave it as an exercise to the reader to check that these quantities will always be equal for any choice of a, b, prime modulus, and prime base.) Crucially, Eve cannot compute this number in any obvious way given only $f(a) = 9$ and $f(b) = 10$ (and 7 and 13), even though she knew exactly how they were generated.

It is, of course, possible to factor these numbers given how small they are and check all thirteen residues, but for sufficiently large numbers, it becomes impractical, as we'll see in Section (1.4). Now Alice and Bob have a secret (the number 3) that Eve doesn't know, despite the fact that all information shared between Alice and Bob was also shared with Eve. (We are also assuming that Eve knows exactly what algorithm Alice and Bob are using; their initial communications about the choice of encryption method and the purposes of f and $\star$ might be gibberish to the untrained eavesdropper.) They can use this secret to employ any classical ciphers to obscure future communications and render them unintelligible to Eve.

Furthermore, should they ever want or need a new key, they can choose new numbers a and b and then repeat the DHM algorithm, even (or rather, especially) if their secure channel is compromised, or just to increase their level of security. (Imagine if Eve, after thousands of careful checks finally stumbled upon a factor that cracked the code, only to discover to her chagrin after a few final computations that the first encoded message Alice sent Bob, when decrypted, read "Hi, just to be safe let's do it again, but this time with $f(x) = \ldots$.")

1.4 Difficulty of Deciphering the Example Algorithm

In order to speak more precisely about the relative difficulty of enciphering and deciphering when the numbers in question get large, we will define and use the so-called *big oh* notation.

[5] In practice, when this algorithm is implemented with computers, they would want to choose much larger numbers.

Definition. Given two functions $f, g : U \subseteq \mathbb{R} \to \mathbb{R}$, we write $f = O(g)$ if and only if there are two real constants $c, x_0 \in \mathbb{R}$ such that for each $x \geq x_0$, we have $|f(x)| \leq c \cdot |g(x)|$. Informally, this notation says that f and g grow at the same rate, or that f can be regarded as approximately the same size as or bounded above by g for sufficiently large values of x. We will sometimes write $f \leq O(g)$ to emphasize when g is an upper bound.

Let us now consider how Eve might attempt to determine Alice's and Bob's secret number in order to decrypt future communications between them. Since she knows f, one naïve approach is for her to guess and check various values of x in the equation $x \star f(b) = x \star 10 = 10^x \mod 13$ until she finds a number x such that $f(x) = 9$, assume Alice's secret number a is this x, and repeat this process to find a candidate for b that satisfies the equation $f(b) = 10$. Then she would have to make sure that a and b satisfy the $\star$ equality and finally use the secret number $a \star f(b)$ to begin decoding the encrypted messages Alice and Bob have been sending all the while.

On a large scale, this task takes $O(\sqrt{a})$ operations,[6] while Alice's (and Bob's) task of multiplying numbers are smaller than $O(\log(a)^{1.6})$ operations,[7] which is strictly dominated by $O(\sqrt{a})$ for not-even-that-large values of a. Thus, Alice and Bob can easily compute their shared secret, while Eve cannot. This disparity in computational difficulty is what protects Alice's and Bob's shared secret number. We will explore these ideas more in Section (3).

One can ask whether its possible for Eve simply to guess what the secret numbers are (or just what their shared secret number is). The answer in both cases is yes, but if Alice and Bob set up the protocol correctly, Eve's odds of guessing correctly (and if she goes straight for the shared secret, she may or may not know if she's guessed correctly if Alice and Bob are sufficiently clever about their communications) are worse than her odds of just guessing (or asking her faithful typewriter-wielding chimpanzee to guess) what Alice and Bob are trying to say in the first place.

2 Elliptic Curve Cryptography

2.1 Preliminaries

In this section, we'll implement the DHM Key Exchange differently by replacing the task of factoring primes with an even tougher problem based on some properties of *elliptic curves* from Algebraic Geometry. First, we lay out some assumptions about the mathematical objects we will need for the encryption process.

In order to make this talk more accessible to readers unfamiliar with Algebraic Geometry or Abstract Algebra, I'll take the next subsection to give some common definitions that I will use repeatedly in this section. Readers familiar with fields and the group law on elliptic curves can skip to Section (2.1.2) without loss of continuity.

2.1.1 Common Definitions

A *field* is an abstract setting consisting of a set of elements and two binary operations on that set (which we denote $+$ and $\cdot$). The requirements that make this set and operations into a field are that the set contain two (for us, distinct) elements, 0 and 1, and that these elements and the operations together satisfy the following relations: both $+$ and $\cdot$ are commutative and associative; the operation $\cdot$ distributes over $+$ on both sides; the elements 0 and 1 act as two-sided, neutral identities for $+$ and $\cdot$ respectively (that is, $0 + x = x + 0 = x$ and similarly $1 \cdot x = x \cdot 1 = x$ for all elements x); all elements x have an *additive* inverse (an element denoted $-x$ such that $-x + x = x + -x = 0$), and all elements x except 0 have a multiplicative inverse x^{-1} defined analogously.

A finite field is simply a field with finitely many elements in its set. To say that a field has *characteristic* two (or more generally characteristic k), means that $1 + 1 = 0$ in this field (or $1 + 1 + \cdots + 1 = 0$, where 1 is added together k times). If a field never satisfies $1 + \cdots + 1 = 0$ for any number k, then it is said to have characteristic zero. We will take for granted the existence of the kinds of fields described below (finite fields on a prime power of elements without characteristic two or three). We often refer to the set of elements itself as the field when the operations are clear from the context.

[6] To be clear, much faster algorithms exist, but they are still slower than algorithms to multiply two numbers.
[7] For instance, the Karatsuba Algorithm (see Karatsuba, 1995) meets this bound.

Two more quick definitions for readers unfamiliar with the (simplified) Group Law on Cubics. A *group* is an abstract object like a field but with fewer requirements: the group should have one (not necessarily commutative) binary operation that is associative, it should have an identity element with respect to that operation, and every element should have an inverse element.

Algebraic Geometry tells us that there is a group formed by the set of the points C (that we will specify below) in a plane with the following operation on any points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ belonging to the set C. Let L denote the (unique) line between P and Q (if $P = Q$, then we take L to be the line tangent to C passing through P, which exists because since C, as we define it below, will turn out to be a differentiable curve in the plane). In either case, L is well defined, and Algebraic Geometry tells us that there will be a unique point of C different from P and Q where L and C intersect again (even if $P = Q$). We let R' denote this other point of intersection. As it turns out, C will also be symmetric about the x-axis, so we know that the reflection of R' over that axis also belongs to C. We denote this point R and say that by definition, $P + Q = R$. For more details about this operation, including what it looks like on an example C, why it forms a group on C, and why this group law is a "simplified" version, see Reid (1988, p. 33). For an animation of the group law's action, see Sullivan (2013).

2.1.2 Setup

Armed with these definitions, we can being setting the scene for Elliptic Curve Cryptography (ECC). Suppose that F_q is a finite field of q elements, where q is some very large prime power $q = p^r$ for a prime p and a large positive integer r. Let us also assume that our field has characteristic not equal to two or three. The theory for fields of characteristic two or three is rich but requires some modifications to the setup we'll explore here.

We will study (cubic) elliptic curves, that is, plane curves of the form $y^2 = x^3 + ax + b$, together with the point at infinity, O. (Though it is not necessary for our purposes, I can't help but mention the intriguing fact that any cubic curve over F_q can be put in this so-called normal form.) For readers unfamiliar with the point at infinity, I will state its role when we need it; right now, we can think of it as just an abstract point that belongs to each elliptic curve.

We fix an arbitrary curve $C := \{(x, y) \in F_q^2 \mid y^2 = x^3 + ax + b\} \cup \{O\}$ by fixing two coefficients a and b (which I will promptly suppress to keep these letters available for our friends Alice and Bob to use). It is worth keeping in mind that since F_q is finite, the curve C is a discrete scattering of (very many) points among the discrete scattering of the (very, very many) q^2 points that make up the plane F_q^2. That said, not much is lost by imagining the continuous version of this curve living in $\mathbb{R}^2$. Sullivan (2013) has some helpful images for visualizing these discrete curves. Notice moreover that C is symmetric about the x-axis: the only y term in the equation is squared, so if $(x, y) \in C$, then so is $(x, -y)$.

2.2 The Discrete Logarithm Problem

The engine of ECC is the *discrete log problem* (on the cubic[8]), whose empirical difficulty provides the security of the message. Suppose B (for "base") is a point on a curve $C = \left\{y^2 = x^3 + ax + b\right\} \cup \{O\}$. Let

$$P = xB = \underbrace{B + B + \cdots + B}_{x \text{ times}}$$

for some (large) positive integer x, where $+$ denotes the operation of the simplified group law (the identity being O) on C. The statement of the problem is to find the smallest natural number n, given the points B and P, such that $nB = P$.

A concrete analogy to this version of the discrete log problem on elliptic curves is to determine how many times a billiard ball hit the table, given only the starting position, the ending position, and the direction it was hit.

[8]More generally, the discrete logarithm returns the smallest positive integer n solving the equation $B^n = P$, where B and P are elements of some given finite group, written multiplicatively. (Since our group on the cubic is written additively, this "exponent" appears as $n \cdot B = P$.)

As of November 2014, no method significantly faster than simply adding the point B on the cubic to itself until P is reached had been published yet, despite decades of research. This fact makes for excellent cryptography. In the next sections, we will see how it can be put to work in a DHM Key Exchange.

2.3 The Encryption Algorithm

Elliptic Curve Encryption, like any algorithm, is a recipe. The ingredients are as follows: a finite field F_q, an elliptic cubic curve C over F_q, and an initial point B from which the key will be calculated. Alice and Bob will agree upon these parameters publicly. They will try to choose q, the two coefficients that determine the curve, and B in such a way that the order of B in the group C (the number of points in its orbit as it is added to itself repeatedly) is large – and that these parameters are not easily guessed. There are techniques of choosing (otherwise random) numbers to achieve this setup reliably, but they are beyond the scope of this paper.

Once Alice and Bob fix their field, curve, and base point, they agree to fix a function $f\colon F_q \to C$ to be given by $f(x) = xB$. (Notice that we have dropped the assumption that we work only with integers since f maps into the curve C.) They also agree that $\star\colon F_q \times C \to C$ will be given by $x \star P = xP$ for any integer x and any point $P \in C$. (If $x < 0$, then $xP = (-x)(-P)$, where $-P$ is the additive inverse of P in the group.) So far, all this information is available to any eavesdroppers. As before, Alice and Bob separately choose two very large, and unpredictable[9] integers a and b, respectively, which they do not share. (These are *not* the same as the public coefficients that determine C.)

The "cooking" process, as before, aims to generate a shared secret (now in the form of a point P on the curve) from the (public) base point B. Alice and Bob (still individually) compute the points $f(a) = aB$ and $f(b) = bB$ on C. They both keep their numbers a and b secret, but they openly share their points aB and bB with each other. At this stage, Alice, Bob, and Eve all know B, aB, and bB on C. However, in order for Eve to determine either a or b given aB and bB, she would have to solve the discrete logarithm problem. Thus, Alice's and Bob's numbers are protected by the difficulty of that problem.

Alice and Bob are now poised to have a shared secret. Alice, knowing her private number a and Bob's point bB, can compute $a \star f(b) = a(bB)$, and she does not need to know b to do so. Similarly, Bob can compute $b \star f(a) = b(aB)$. Since the group law defines an associative operation (and integer multiplication is commutative), f and $\star$ intertwine as desired, and Alice and Bob have computed the same point: $a(bB) = (ab)B = (ba)B = b(aB)$. This point is their shared secret because Eve cannot compute it, even given aB and bB, without solving the discrete log problem. (She can easily compute the point $aB + bB = (a + b)B \neq (ab)B$, but this point is of no help for determining abB and hence equally useless for deciphering future messages.) In fact, in order to perform $\star$ with any of the available points B, aB, or bB, Eve would need to know an integer, but she doesn't have access to either of the integers Alice or Bob used, unless she solves the discrete log problem.

Alice and Bob have won. They have established a shared secret, namely the point abB, over an insecure channel. Remember that while this point $(ab)B$ is not the intended message, it is a secret they share, which enables them to communicate securely with little fear of Eve's deciphering their communications.

As far as we know, Eve cannot get this point much faster than adding B to itself ab times times and checking for the first $ab - 1$ times that she is wrong, ignorant all the while as to when her search will end. Alice and Bob then can use their shared secret point to encrypt future communications from Eve. Next, we will see how Alice and Bob can compute these points efficiently and why Eve's task is almost impossible in comparison.

3 The Strength of ECC: Computational Complexity

The power of ECC lies in two facts. First, it is provably easy to compute xP given any integer x and any point P on the curve C. Second, it is apparently (though not provably) quite difficult in comparison to compute x given P and xP. In this section, we will formalize these notions of "easy" and "difficult." We will determine the *computational complexity* of these two tasks. That is, we will bound the number

[9]Such a number can be generated, for instance, a pseudo-random number generator or an environmental process.

of operations required to perform them (approximately, using big oh notation) as a function of the the parameters that Alice and Bob choose for their encryption. In fact, once we look at magnitudes, the relative difficulty of these tasks will only depend on one parameter, q. In short, we will show that Eve's task to compute abB from aB and bB requires significantly more operations than the task Alice and Bob must do to find abB from a and bB (or b and aB).

3.1 Point-Multiplication (Enciphering) Complexity

The goal of this section is to show that, regardless of other parameters, the number of operations Alice and Bob must perform on the individual digits of numbers written in binary (base two) is bounded above by $O(\log(q)^4)$. Much of this section consists of (in my opinion, beautiful) computations in plane geometry and counting the number of operations performed along the way. Readers who want to avoid these details can skip the proof of the following theorem without much loss of continuity.

Theorem. Given a point P on an elliptic curve C over a finite field F_q and an integer x, the coordinates of the point $xP = P + P + \cdots + P$ (where $+$ denotes the operation of the group law on cubics) can be computed in $O(\log x \cdot \log^3 q)$ bit (short for "binary digit") operations.

Proof. Given the coordinates of $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, we can algebraically compute the coordinates of their sum $R = P + Q$ using the definition of the $+$ operation and various facts about the group law and planer geometry.

We use three claims, given below, in this proof. The first is that this computation of adding any two points takes under 20 *arithmetic* computations on the four numbers x_1, y_2, x_2, and y_2. This gives a (negligibly small) constant value on the number of steps to add any two points. Second, each of those steps to add two points on C takes $O\left((\log q)^3\right)$ bit operations. Third, in calculating xP from P, we add P to itself x times in only as many steps as there are digits of x, that is, $O(\log x)$ operations. Therefore, adding P to itself x times takes $O(\log x)$ steps, each of which takes $O\left((\log q)^3\right)$ operations, then the calculation of xP is $O\left((\log x)\,(\log q)^3\right)$.

Claim 1: Computing the sum of two points takes at most thirteen arithmetic operations.
As above, let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$. We wish to find $P + Q = (x_3, y_3)$ under the simplified group law for the curve $C = \left\{(x,\, y) \in F_q^2 : y^2 = x^3 + ax + b\right\}$.
If the line connecting P and Q is vertical (that is, if $x_1 = x_2$ but $y_1 \neq y_2$), then their sum is the point at infinity, O, where the vertical line through P and Q again intersects C. Checking equality of both coordinates counts as two arithmetic operations to return O.)

For P and Q with distinct x-coordinates, the unique line L through P and Q is the set of points defined by $L = \{(x,\, y) \in F_q^2 \mid y = \alpha x + \beta\}$, where $\alpha = \frac{y_2 - y_1}{x_2 - x_1}$ and $\beta = y_1 - \alpha x_1$. Now we can solve the equations defining the intersection $L \cap C$. The first condition is that $(x,\, y) \in F_q^2$ satisfy the equation $y^2 = x^3 + ax + b$ (where now a and b are the parameters defining the curve C, not Alice's and Bob's private numbers), while the second condition is that $(x,\, y)$ also satisfy $y = \alpha x + \beta$. We can solve this system of equations to determine the x-coordinates of the points of intersection to be the solutions to the equation $(\alpha x + \beta)^2 = x^3 + ax + b$. We already know the first coordinates (x_1 and x_2) of two of the roots (P and Q), so we can use the algebraic trick that the sum of the roots of a monic polynomial is $-a_{d-1}$, the opposite of the coefficient of x^{d-1}, where d is the degree. Thus, $x_1 + x_2 + x_3 = -(-\alpha^2)$, so $x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_2 - x_1$. The y-coordinate of this point is then $\alpha x_3 + \beta$. Finally, the group law takes the reflection of the third point of intersection over the x-axis, so that $y_3 = -\alpha x_3 - \beta = \alpha(x_1 - x_3) - y_1$. Therefore, we have found

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

in terms of the coordinates of P and Q. Using this formula, we can now compute an arbitrary $P + Q$ in no more than nine arithmetic operations (additions, subtractions, multiplications, and divisions, all of which are possible in the field F_q) on four known quantities (x_1, y_1, x_2, and y_2) and an intermediate quantity (α, which we hold onto after computing it in x_3 for use in computing y_3).

Finally in the special case when $P = Q$, we calculate the tangent line to C at P instead of the line connecting P and Q. Implicit differentiation of the polynomial $y^2 = x^3 + ax + b$ gives us the slope $\alpha = \left(3x_1^2 + a\right)/(2y_1)$. In this case, if $P = (x_1, y_1)$, then we can calculate

$$P + P = 2P = \left(\left(\frac{3x_1^2 + a}{2y_1} \right)^2 - 2x_1,\ \left(\frac{3x_1^2 + a}{2y_1} \right)(x_1 - x_3) - y_1 \right)$$

using only eleven arithmetic operations (when we reuse α), and the others required two and nine (respectively), so adding our two checks for whether $x_1 = x_2$ and $y_1 = y_2$ we can then use thirteen as our upper bound on the number of arithmetic operations required to calculate the point $P + Q$ from P and Q.

Claim 2: Recall that a bit operation is an arithmetic operation performed on the individual digits of two numbers written in binary (base two). The number of bit operations required to be perform any arithmetic operation on (to add, subtract, multiply, or divide) two elements of a finite field F_q is no more than $O((\log q)^3)$. Each arithmetic operation on large numbers will require many bit operations, depending on how many binary digits there are to operate on. This claim tells us that "many" is bounded above by $O\left((\log q)^3\right)$. The proof of this claim, as given in Koblitz (1987, p. 37), is rather long and mechanical, so I omit it.

Corollary: Using claim 1, we can conclude that adding two points of C takes $O\left((\log q)^3\right)$ binary operations since we can absorb the (at most) thirteen arithmetic operations required to add two points into the constant c in the formal definition of big oh notation (for reference, number $\log q$ will often have a magnitude upwards of 10^8).

Claim 3: We can compute the coordinates of a point xP given $x \in \mathbb{Z}$ and $P \in C$ in $\log x$ arithmetic operations.
There are $\log x$ binary digits of any integer x, each digit either 0 or 1. The essence of this proof is that if we always reduce by a suitable modulus after any multiplication, we can operate directly on the binary digits of x and thus perform $O(\log x)$ group operations to compute xP (instead of x point-additions, as one might expect). The details of the proof can be found in Koblitz (1987, p. 22).

In summary, we are trying to show that given $x \in \mathbb{Z}$ and $P \in C$, we can calculate the coordinates of xP in at most $O\left(\log x \cdot (\log q)^3\right)$ operations. First, we showed that calculating $P + Q$ takes a small (and thus negligible) constant number of arithmetic operations for any points P and Q. Then, each such operation takes $O\left((\log q)^3\right)$ binary operations. Finally, we calculate xP in $O(\log x)$ many of the above calculations, which means finding xP takes a total of $O\left((\log x)(\log q)^3\right)$ binary operations. ■

There is one more simplification we can make given what we know about Alice's and Bob's goal. In the above theorem, the numbers for x will be a and b (referring again to the private integers, not coefficients defining C). We may assume in both cases that $x < 2q + 1$ because choosing $x \geq 2q + 1$ will just lead to recycling the finitely many points of C. There are *at most* $2q + 1$ points belonging to C because there are q points total along the x-axis, and C, which is not necessarily defined for all values of x, contains at most two points, plus the lone O, for any given x value.

Since computing points in the group only requires the *smallest* number x that gives the right point, Alice and Bob should not choose numbers larger than $2q + 1$. The constants can be ignored next to the comparatively huge parameter q, so replacing x by q, we have derived the upper bound $O\left((\log q)^4\right)$ on the number of binary operations one need to perform to compute a point xP given x and P. Next, we wish to show that this number of operations is small compared to the time to solve the discrete log problem (to calculate x given only P and xP), which is the (naïve) deciphering algorithm.

3.2 Discrete Log (Deciphering) Complexity

While no proof of the difficulty of solving the discrete log problem for elliptic curves is currently[10] known, decades of research suggest that it is very difficult. With various technical improvements[11] on

[10] I made this claim in November 2014, when I wrote the first version of this paper. To my knowledge, it is still the case, but I have not looked far into the frontier of this inquiry since then.

[11] For examples, see Hoffstein, Pipher, and Silverman (2008, p.315).

the naïve approach of simply adding the base point B to itself until the result is P, one can compute x from $xP$and P in $O\left(\sqrt{q}\right)$ binary operations. From calculus, we know that $\lim_{q\to\infty} \sqrt{q}/\log^4(q) = \infty$. That is, for large values of q, the difference between enciphering and deciphering becomes arbitrarily large. In practice, it does not take a huge q to achieve a satisfactory disparity given what we know about today's computers.

3.3 Comparing Algorithms: Why ECC?

One might ask why go to all the trouble of using elliptic curves and group laws – even if the calculations will be done by a computer – when simple integer operations, such as the ones described in Section (1.3), can be used for a DHM Key Exchange? The answer is that ECC performs better than some other implementations of the key exchange[12] in a few ways.

First, the ease of enciphering does not change much, while the difficulty of deciphering increases substantially for a given increase in the magnitude of the parameters (roughly, the "speed" of the limit from the previous section). Other considerations include relative abundance of (and ease of finding) "good" parameters, how often and how closely the various bounds are actually approached, the details of the many times I asserted there are ways of making things happen, and the probability of various (un)lucky things happening to our intrepid Alice and Bob. I'll focus on the first improvement.

For readers recalling the $O(\sqrt{x})$ versus $O(\log(x)^{1.6})$ figure from Section (1.4), there is an important clarification: $O(\sqrt{x})$ is not a good estimate of how fast Eve can solve the problem if she used a more advanced algorithm than the very naïve one I described in that section. Indeed, Hoffstein, Pipher, and Silverman (2008, pp. 81, 169) claim that the best known algorithm to solve the discrete log problem for the multiplicative group of F_q, as in Section (1.4), is subexponential, while the best known algorithms for the elliptic curve discrete log problem take $O(\sqrt{q})$ steps, which yields a fully exponential time solution with respect to the parameter r. Recall that q is a prime *power* with respect to r, so $\sqrt{q} = p^{r/2}$ while in comparison, $(\log q)^4 = r^4 \log^4(p)$. For a given p, the exponential expression of the form c^r grows much faster with respect to r than the monomial $c'r^4$.

Hence, for a given q, the elliptic curve discrete log problem appears to be orders of magnitude more difficult to solve than the integer factorization discrete log problem used in Section (1.3), and the increase of the exponent of $\log(q)$ from 1.6 to 4 in the enciphering difficulty is not nearly enough to counteract a sub- versus fully exponential disparity in deciphering difficulty.

To close, let us try to make this difference more intuitively meaningful. Sullivan (2013) describes a wonderful comparison of the levels of difficulty to decipher the two encryptions. He cites work by Arjen Lenstra, Thorsten Kleinjung, and Emmanuel Thomé (2013) that introduces a way to estimate how much energy would be required (presumably by some given kind of computer, though Sullivan does not say) to break a given encryption.

By these estimates, it would take less energy to break an 228-bit RSA encryption (a common integer factorization implementation) than to boil a teaspoon of water. On the other hand, it would take more energy to break a 228-bit ECC key than to boil all the water on earth. The same level of security using RSA would require a key with 2,380 bits, a full order of magnitude higher. Unless and until a faster deciphering algorithm is found for ECC, it will be provide a greater level of security.

[12]Interestingly, Adrian et al. (2015) have suggested that the abstract DHM Key Exchange is more vulnerable to sufficiently powerful attacks than previously thought. One of their recommendations for strengthening the security of the key exchange is using the elliptic curve implementation.

Bibliography

[1] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *22nd ACM Conference on Computer and Communications Security*, October 2015.

[2] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[3] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *An Introduction to Mathematical Cryptography*. Springer, 2008.

[4] Anatoly A. Karatsuba. The Complexity of Computations. *Proceedings of the Steklov Institute of Mathematics-Interperiodica Translation*, 211:169–183, 1995.

[5] Neal Koblitz. *A Course in Number Theory and Cryptography*. Springer-Verlag New York, Inc., 1987.

[6] Arjen K. Lenstra, Thorsten Kleinjung, and Emmanuel Thomé. Universal security; from bits and mips to pools, lakes – and beyond. Cryptology ePrint Archive, Report 2013/635, 2013. <`http://eprint.iacr.org/2013/635.pdf`>.

[7] Miles Reid. *Undergraduate Algebraic Geometry*. Cambridge University Press Cambridge, 1988.

[8] Nick Sullivan. "A (Relatively Easy to Understand) Primer on Elliptic Curve Cryptography". *CloudFlare*, 24 Oct. 2013. Web. 16 Nov. 2014. <`https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/`>.

[9] Wikipedia. Elliptic curve cryptography — Wikipedia, The Free Encyclopedia, 2014. [Online; accessed 16-November-2014] <`https://en.wikipedia.org/w/index.php?title=Elliptic_curve_cryptography`>.