Fall 2014 Directed study in Elliptic curve Cryptography with Dr. Kaveh

T. Ian Martiny

December 20, 2014

1 Introduction

The directed study was on elliptic curve cryptography. We used Hankerson, Menezes, and Vanstone's book "Guide to Elliptic Curve Cryptography". We covered chapters 1 though 4, focusing most of our time on chapter 2 (Finite Field Arithmetic) and chapter 3 (Elliptic Curve Arithmetic). The book is written from a crash course standpoint with an emphasis on how rather than why. Thus the book was very light on theoretic justifications, so a good deal of the semester was spent justifying and understanding the material presented in the book.

In order to supplement the Hankerson et al book we used Silverman's "The Arithmetic of Elliptic Curves". This book provided a much more rigorous and theoretical approach to elliptic curves.

Dr. Kaveh and I met weekly to discuss the section(s) I had read, I presented the material as if I were lecturing to a class; Dr. Kaveh would listen and interject with any questions or comments, often filling in details left out of the book.

My interest in the subject spawned from a class the previous semester in Math 3600 - Mathematical Cryptography which was a very theoretic approach to cryptography taught by Dr. Hales. I have had a background in cryptography from my undergraduate career at Virginia Commonwealth University, but had never interacted much with elliptic curve cryptography. Thus a few of the topics in the Math 3600 course discussing algebraic geometry and elliptic curves were not as well understood as I would have preferred.

2 Overview

2.1 Basics

At its very basic cryptography is a way to have secure communication between two entities, when a third party is actively trying to eavesdrop. The standard model is as follows: entity A (Alice) and entity B (Bob) are trying to communicate over an unsecured channel; that is a channel that a third entity E (Eve) can eavesdrop over.

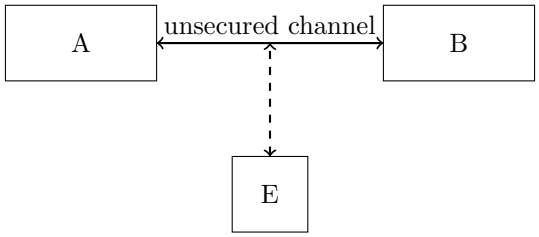

Figure 1: Basic communication model

The actual applications of this example of numerous, A and B could be actual people talking over a cellular network and E is attempting to eavesdrop, or A could be a web

browser and B could be an online store's website with E attempting to learn the credit card information used in transactions, etc.

As stated in [2], we have many goals for a secure cryptosystem as follows:

1. *Confidentiality*: no one but the desired readers should be able to read the sent message.
2. *Data integrity*: the receiver should be able to detect any unauthorized alterations made to the message.
3. *Data origin authentication*: the receiver should be able to verify the message was sent from the expected sender.
4. *Entity authentication*: the receiver should know who is sending the message.
5. *Non-repudiation*: the receiver can convince a neutral third party that the sender did in fact send the message.

Not all of these characteristics are met (or even desired) in cryptosystems. For example *Non-repudiation* would not be desired in a system designed in order to allow deniability.

We also assume that our eavesdropper (E) has considerable capabilities. We always give the adversary the benefit of the doubt and assume they can:

1. Read all data transmitted over the open channel.
2. Modify transmitted data and inject her own data.
3. Use substantial and advanced computational resources.
4. Know the complete descriptions of the protocols and any cryptographic mechanisms deployed (except private keys).

There are two main schools of cryptography, Symmetric-key cryptography and Public-key cryptography. We deal exclusively with the latter but for the sake of completeness describe the former. Symmetric-key cryptography requires that both parties involved have the same key that no one else has access to. This requires a secret and authenticated channel, i.e. a method of communication that is not able to be eavesdropped on and both parties can easily verify who they are interacting with, in order to share this private key. This is generally difficult to attain. Public-key cryptography is more general and only requires access to an authenticated channel. This can be easily attained via a website or an email. But every communication between the two entities is under observation, anything mention that is not encrypted can be used by adversaries.

2.2 Public-key cryptography

Compared to Symmetric-key cryptography, where the difficulty of deciphering encrypted messages lies in knowing a secret key, Public-key cryptography is more about solving problems. Public-key cryptography requires a key *pair* usually one private key and one public key where the public key is used to encrypt messages and the private key is used in some way to decrypt messages. The private key is generally required to be related to the public key via a computational problem that is, or is believed to be, intractable, i.e., difficult to solve. We discuss three problems whose difficulty is required for some main crypto-systems. It should be mentioned that the methods discussed are basic "textbook" versions, many enhancements and modified methods exist and are more reliable, but they are based on the methods presented here. Our basic problems are:

1. The integer factorization problem (RSA).
2. The discrete logarithm problem (DH, ElGamal).
3. The Elliptic Curve discrete log problem (ECDH).

2.2.1 RSA systems

The RSA system, named after Rivest, Shamir, and Adleman, was proposed in 1977[2].

The first step of any system is the generation of the key pair used for further communication. For RSA the key pair is generated via two prime numbers p, and q of roughly the same bit-length ($\log_2 p \approx \log_2 q$). Then for the public key we choose the pair, (n, e) where $n = pq$ and the encryption exponent e is an integer with $1 < e < \phi$ and $\gcd(e, \phi) = 1$ where $\phi = (p-1)(q-1)$ (the Euler Totient of n). The private key d, also called the decryption exponent, is an integer with $1 < d < \phi$ such that $ed \equiv 1 \mod \phi$ which exists since $\gcd(e, \phi) = 1$. It has been shown that the computation of d from n and e is equivalent to the problem of computing p and q given n; the latter is known as the integer factorization problem, which is intractable.

Once a key pair is generated the private key, d, is kept secret and told to no one, whereas the public key, (n, e), is published online, or in a journal, or where ever appropriate. Thus if Alice wishes to communicate with Bob she must choose Bob's public key, (n_B, e_B), and use it to encrypt a message and Bob can use his private key, d_B, to decrypt the message.

For the sake of simplicity we deal with messages that have already been converted into integers, this process can be complicated and very involved depending on the system, and as such is outside the scope of this paper.

Thus our RSA encryption hinges on the the fact that by design if the message we send is the integer m:

$$m^{ed} \equiv m \mod n$$

As such for encryption Alice can choose her message m and then send the integer $c = m^{e_B} \mod n_B$ to Bob and then Bob can decrypt the message as $m = c^{d_B} \mod n_B$. The encryption

method here is assumed to be secure since computing m from $m^e \mod n$ is assumed to be as difficult as the integer factorization problem.

One might notice that the process described above does not satisfy all (or in fact many) of the security goals we described before. Indeed all we achieve here is confidentiality, we obscure the message from any eavesdropper but Bob can't verify where the message came from or if it was modified. We should think of the above RSA encryption as but one part of the scheme. The part above handled confidentiality, and if so desired we can use the RSA key we generated to also provide a signature to help ensure Data origin authentication, that is to ensure that Alice did in fact send this message.

First we need the notion of a hash function. As mentioned before we think of our message as already being an integer, in some cases it is easier to think of integers as being binary sequences, i.e., $13 = (1, 1, 0, 1)_2$. Often the messages we want to send correspond to very large integers and very long binary sequences. As such it would be easier for our computations to work on smaller numbers or shorter binary sequences, this is where a hash function comes in.

Definition: A **cryptographic hash function** is a deterministic algorithm H that maps binary sequences of arbitrary finite length to binary sequences of a fixed length l (often $l = 160$ or $l = 256$) [1].

In addition we often require that hash functions have certain security properties:

1. **Preimage resistance:** Given a binary sequence, y, of length l it should be computationally infeasible to compute a binary string x with $H(x) = y$.

2. **Second-preimage resistance:** Given a binary sequence, x, and a binary sequence $y = H(x)$ it should be computationally infeasible to compute a binary sequence $x' \neq x$ with $H(x') = y$.

3. **Collision resistance:** It should be computationally infeasible to compute binary sequences $x \neq x'$ such that $H(x) = H(x')$.

The phrase "computational infeasible" showed up often in our requirements, this is often case dependent and difficult to describe generally. In our general case it suffices to say that it is not easy to compute.

Thus after using Bob's key, (n_B, e_B), to compute c, an encrypted message, Alice can use a (public) cryptographic hash function, H, to compute $h = H(c)$ then she can use her private key, d_A to compute $s = h^{d_A} \mod n_A$. Then Alice would send her message, c, and its signature, s, to Bob. Here Bob would first verify the signature by computing $h = H(c)$, using the same cryptographic hash function, and then computes $h' = s^{e_A} \mod n_A$ from Alice's signature s. And by the property that $e_A d_A \equiv 1 \mod n_A$ Bob knows the message was sent from Alice if $h = h'$ otherwise it was a fabrication. The security here relies on the inability of a forger to compute $h^{d_A} \mod n_A$. If Bob is satisfied the message is sent from Alice he can then proceed to decrypt the message c as explained above.

The use of a cryptographic hash function for the signature is to help speed up computations. Since the security lies on the difficulty of computing d from (n, e) we can use a

simpler (but not too simple) message "fingerprint" to help verify identity. The idea is that the message recipient should not spend precious computation time working with very large numbers to verify identity, since only someone who knows d could have signed it.

This method of signing messages is also secure between different messages. As opposed to a written signature (which does not often change) this signature changes for every message (even for the same key) since it is based off of $H(c)$. This is why we require our hash function to have such specific security requirements.

2.2.2 Discrete log systems

The first discrete log system was a key agreement protocol by Diffie and Hellman in 1976. In 1985 ElGamal described a discrete log encryption / decryption scheme.

As before we have need of a discrete log key pair. Here our public parameters (not to be confused with public key) will be a triple (p, q, g) where p is prime, q is a prime that divides $p-1$ and g is an integer in $[1, p-1]$ with order q, that is $g^q \equiv 1 \mod p$ and no smaller number makes this so. The idea is that g is the generator of the subgroup of $\mathbb{F}_p^*$ of order q. With this the private key, x, is chosen at random so that $x \in [1, q-1]$ and the public key is computed as $y = g^x \mod p$. The problem of discerning x given (p, q, g) and y is the discrete logarithm problem.

With our public and private keys we can now portray how Alice, with Bob's public key y_B, can send a message to Bob so that he can decrypt the message, all under the public parameters (p, q, g), but no one else can. To send her message, m, Alice randomly chooses $k \in [1, q-1]$ and computes $c_1 = g^k \mod p$ and $c_2 = m \cdot y_B^k \mod p$ and sends (c_1, c_2) to Bob. To decipher the message Bob computes $m = c_2 \cdot c_1^{-x_B} \mod p$. Since $c_1^{-x_B} \mod p = g^{-kx_B} \mod p = y_B^{-k} \mod p$ we have that $c_2 \cdot c_1 \mod p = m$.

We use a random component here since Alice only has access to Bob's public key, and our keys have the nice property that $y_B = g^{x_B} \mod p$ so if Alice uses only $y_B \cdot m$ as an encryption then decryption is simply determining the inverse of y_B; while this is a computationally difficult problem it is not secure enough for encryption. Thus she chooses a random component and provides $g^k \mod p$.

As before with RSA this method does not provide anything beyond confidentiality. We must consider this a small part of a bigger scheme, and thus we present a discrete log signature method as well.

The Digital Signature Algorithm (DSA) was proposed in 1991 by NIST, and is used as a standard signing algorithm.

To sign a message Alice would compute $h = H(m)$ using a cryptographic hash function, choose a (separate) random integer $k \in [1, q-1]$ then computes $T = g^k \mod p$, $r = T \mod q$ and

$$s = k^{-1}(h + x_A r) \mod q$$

Then Alice's signature for m is the pair (r, s). To verify the signature Bob must verify the equation above, but has no access to x_A nor k, thus solving the equation above for k we get

$$k \equiv s^{-1}(h + x_A r) \mod q$$

and exponentiating both sides with g yields:

$$T \equiv g^{hs^{-1}} y_A^{rs^{-1}} \mod p$$

Therefore, Bob can compute T and check that $r = T \mod q$ and accept or reject the message.

2.2.3 Elliptic curve systems

The details of elliptic curves and their associated groups is covered in more detail later, but for completeness we discuss some of the ideas of elliptic curve cryptographic systems here.

An Abelian group, $(G, \cdot)$ consists of a set G with a binary operation $\cdot : G \to G$ that satisfies:

1. (Associativity) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all $a, b, c \in G$.
2. (Existence of identity) There exists an element $e \in G$ such that $a \cdot e = e \cdot a = a$ for all $a \in G$.
3. (Existence of inverses) For each $a \in G$, there exists an element $b \in G$, called the inverse of a such that $a \cdot b = b \cdot a = e$.
4. (Commutativity) $a \cdot b = b \cdot a$ for all $a, b \in G$.

We call a group **cyclic** if there is an element $g \in G$ such that $\langle g \rangle = \{g^i : i \geq 0\} = G$, that is if there is an element of g which generates all of G.

In the setting of a cyclic group of order n (i.e., G has n elements) we can define the same notion of discrete log as we had above in our $[1, n-1]$ setting. With this idea we can define the notion of an elliptic curve group and an elliptic curve crypto-system.

Let p be a prime number and $\mathbb{F}_p$ denote the field of integers, that is the integers $[0, p-1]$, which form a group under addition and multiplication modulo p with no zero divisors and proper interactions between addition and multiplication. An elliptic curve E over $\mathbb{F}_p$ is defined by an equation of the form:

$$y^2 = x^3 + ax + b$$

with $a, b \in \mathbb{F}_p$ satisfying $4a^3 + 27b^2 \not\equiv 0 \mod p$. A pair $(x, y) \in \mathbb{F}_p^2$ is a point on the elliptic curve if (x, y) satisfies the given curve. There is also the point at infinity, which will serve as our identity, on every elliptic curve, denoted ∞. All of the points on E are denoted $E(\mathbb{F}_p)$. As an example if our base field is $\mathbb{F}_7$ and we use the equation

$$y^2 = x^3 + 2x + 4$$

then the points on E are:

$$E(\mathbb{F}_7) = \{\infty, (0,2), (0,5), (1,0), (2,3), (2,4), (3,3), (3,4), (6,1), (6,6)\}$$

The points on an elliptic curve form an Abelian group under addition. The addition of points on elliptic curve is more complex than simply adding components, but is well defined, it will be covered later. We now discuss some elliptic curve crypto-systems.

Let E be an elliptic curve defined over a finite field $\mathbb{F}_p$, now denoted $E/\mathbb{F}_p$, let $P \in E(\mathbb{F}_p)$ and suppose the order of P is a prime number n, that is $nP = \infty$. Then the cyclic subgroup of $E(\mathbb{F}_p)$ generated by P is

$$\langle P \rangle = \{\infty, P, 2P, 3P, \ldots, (n-1)P\}$$

We have that the prime p, the equation of the elliptic curve E and the point P, with its order n, are our public parameters of our crypto-system. The private key is a random integer $d \in [1, n-1]$ and the public key is $Q = dP$. The problem of finding d given the public parameters and Q is the elliptic curve discrete log problem (ECDLP).

Following the methods above we can also create the elliptic curve ElGamal encryption scheme: if Alice wants to send a message, m, to Bob she must use Bob's public key's and parameters as follows. First she represents the message m as a point $M \in E(\mathbb{F}_p)$ (not covered). She then chooses $k \in [1, n-1]$ and computes $C_1 = kP$ and $C_2 = M + kQ_B$ and sends (C_1, C_2) to Bob. To decrypt Bob computes $M = C_2 - d_B C_1$ and extracts m from M.

The above works on the same principles of ElGamal encryption / decryption in $[1, n-1]$. An eavesdropper who wants to recover M from C_1 and C_2 needs to compute kQ the task of computing kQ from knowing Q and $C_1 = kP$ is the elliptic curve discrete logarithm problem.

2.3 Why use elliptic curve cryptography?

Given what we have so far a natural question is why we bother with elliptic curve crypto-systems at all if we have secure method just using $[1, n-1]$? Without going into the analysis of algorithms we mention that due to the unintuitive nature of point addition on elliptic curves [not that it is difficult to understand, but that the result is hard to predict] we can use smaller key and public parameters for the same level of security compared to the modulus n for our $[1, n-1]$ systems. See the table below from the Hankerson et al book:

	Security level (bits)				
	80 (SKIPJACK)	112 (Triple-DES)	128 (AES-Small)	192 (AES-Medium)	256 (AES-Large)
DL parameter q EC parameter n	160	224	256	384	512
RSA modulus n DL modulus p	1024	2048	3072	8192	15360

Figure 2: Here we see how large (in bits) the keys we use for our protocols need to be to get the same level of security. For example, in order to get 80 bits of security (using the SKIPJACK encryption scheme) if we use elliptic curves protocols our keys need to be about 160 bits long, for the same level of security in our RSA model we would need our keys to be 1024 bits long.

3 Finite Field Arithmetic

It is important that we discuss efficient means of performing computations, since all of the crypto systems that we will use require numerous computations. We first discuss computations in finite fields, this is broken in to three different categories: (i) Prime finite fields, (ii) binary finite fields, and (iii) optimal field extensions. Prime fields are simply fields of prime order, which give them means of doing computations quickly by exploiting their structure. Binary fields are field extensions of the field of two elements, due to their very simple base field they also have algorithms which significantly speed up computations. Optimal field extensions are generally chosen when the hardware is not suited for binary fields or prime fields, this is usually a specific choice.

We discuss in detail some of the prime field arithmetic and some of the similar algorithms for binary fields. First we begin with a brief introduction to fields:

3.1 Intoduction to Finite Fields

A field is an abstraction of number systems such as the rationals, or the real numbers that keep their basic qualities and nothing else. A field is a set $\mathbb{F}$ with two operations: addition (denoted $+$) and multiplication (denoted $\cdot$, or when understood, by juxtaposition). The operations with $\mathbb{F}$ must satisfy:

1. $(\mathbb{F}, +)$ is an Abelian group with (additive) identity denoted by 0.
2. $(\mathbb{F}\backslash\{0\}, \cdot)$ is an Abelian group with (multiplicative) identity denoted by 1.
3. The distributive law holds: $(a + b) \cdot c = a \cdot c + b \cdot c$ for all $a, b, c \in \mathbb{F}$

We deal with the case where $\mathbb{F}$ is finite, in which case we say that $\mathbb{F}$ is a finite field. We only need our two operations, $+$, $\cdot$, in our field; we can think of subtraction as addition:

$a - b = a + (-b)$ where $-b$ is the additive inverse of b. Similarly division can be thought of as multiplication: $a/b = a \cdot b^{-1}$ where b^{-1} is the multiplicative inverse of b.

The order of a finite field is the number of elements in $\mathbb{F}$. There exists a field of order q if and only if $q = p^m$ for some prime p and m a positive integer. In this case we call p the characteristic of the field. If $m = 1$ we say $\mathbb{F}$ is a prime field, otherwise it is an extension field.

In prime fields computation is done modulo the prime. For example, in the prime field $\mathbb{F}_{29} = \{0, 1, \ldots, 28\}$ we have:

(i) $17 + 20 = 8$ since $37 \mod 29 = 8$.

(ii) $17 - 20 = 26$ since $-3 \mod 29 = 26$.

(iii) $17 \cdot 20 = 21$ since $340 \mod 29 = 21$.

Binary fields are finite fields of order 2^m for $m \geq 1$, thus the characteristic of binary field is 2. A common way to construct these fields is to deal with polynomials. Note we cannot work with the numbers $\{0, \ldots 2^m - 1\}$ since this collection does not satisfy the conditions above (under usual definitions). In order to construct our field we must find an irreducible binary polynomial of degree m. This means that the polynomial cannot be factored (in $\mathbb{F}_2$) and has binary coefficients. There are algorithms which will construct irreducible polynomials of the chosen degree, but this topic is not covered here.

For example, in order to construct $\mathbb{F}_{2^4}$ we choose $f(x) = x^4 + x + 1$ which is irreducible over $\mathbb{F}_2$. Then the elements of our field are all possible binary polynomials of degree less than or equal to 3. The significance of our choice of irreducible polynomial is when we multiply polynomials:

$$\begin{aligned}(x^3 + x^2 + 1) \cdot (x^2 + x + 1) &= x^5 + x + 1 \\ &= x \cdot x^4 + x + 1 \\ &= x \cdot (x + 1) + x + 1 \\ &= x^2 + 1\end{aligned}$$

Where we used that $x^4 + x + 1 = 0 \implies x^4 = x + 1 \mod 2$. In general which irreducible polynomial we choose does not matter, there is only one field of each order. Any two with the same order are isomorphic.

The same procedure is done for any extension field with any base. If we wanted to construct a field of order p^m we would need to find an irreducible polynomial (over $\mathbb{F}_p$) of degree m with coefficients in $\mathbb{F}_p$. And construct our field as all polynomials of degree less than or equal to m, with coefficients in $\mathbb{F}_p$, using our chosen irreducible in order to reduce polynomials of higher degree.

We will denote the non-zero elements of a field as $\mathbb{F}_q^*$ these elements for a cyclic multiplicative group, and hence have a generator, so that:

$$\mathbb{F}_q^* = \{b^i : 0 \leq i \leq q-2\}$$

The order of an element of $\mathbb{F}_q^*$ must be a divisor of the groups order, thus it must be a divisor of $q-1$.

3.2 Prime field arithmetic

Hankerson et al's treatment of Prime field arithmetic is to present first basic then more advanced methods in order to compute the basic functions in a field. They first develop how an element of a prime field would be stored on a computer and then discuss methods for adding and subtracting these numbers. The methods they present involve working with registers of a computer of varying lengths depending on the architecture of the computer and the prime field being worked with. Rather than talk about these specific computations we move on to multiplication and discuss the Karatsuba-Ofman multiplication method which takes multiplication from a $O(n^2)$ operation to a $O(n^{\log_2 3})$ operation.

Here we consider the most efficient way to multiply two n-bit numbers, x, y, we assume $n = 2l$ for some l. Thus we rewrite our numbers by factoring out l powers of 2: $x = x_1 2^l + x_0$ and $y = y_1 2^l + y_0$. Then we can see:

$$\begin{aligned} xy &= (x_1 2^l + x_0)(y_1 2^l + y_0) \\ &= x_1 y_1 2^{2l} + [(x_0 + x_1)(y_0 + y_1) - x_1 y_1 - x_0 y_0]2^l + x_0 y_0 \end{aligned}$$

Where we have rewritten the cross terms $x_1 y_0 2^l$ and $y_1 x_0 2^l$ using $[(x_0 + x_1)(y_0 + y_1) - x_1 y_1 - x_0 y_0]2^l$. This helps because we already have to compute $x_1 y_1$ and $x_0 y_0$ for the first and last terms of the sum and computing $x_0 + x_1$ and $y_0 + y_1$ are simple. Thus instead of needing to compute four multiplications: $x_0 y_0$, $x_0 y_1$, $x_1 y_0$, and $x_1 y_1$ we only need to compute three: $x_0 y_0$, $x_1 y_1$, and $(x_0 + x_1)(y_0 + y_1)$ and then compute the correct summation to get the same end result.

We explicitly factored out l copies of 2 for $2l$-bit numbers but this is not always the most efficient way to do computations. The way presented is referred to as the $n/2$ split. In which case we factor out l copies of 2 and then multiply the coefficients (which could require another use of the Karatsuba-Ofman method) and continue on. Occasionally taking into account hardware considerations can speed up the process. For example if we are trying to multiply two 224-bit numbers, our $n/2$ method would have us first multiplying 112-bit numbers then 56-bit numbers etc. But if we are working on a machine with word size 32-bits it may be beneficial to split our numbers at multiples of 32 instead. For example we could split our 224-bit numbers as a 128-bit number and a 96-bit number.

This method is more difficult to write and highly dependent on the system using it. The usual $n/2$ split provides significant speed up from usual multiplication methods, but if working on specific system and multiplying many numbers it may be worth the hassle to invest in unusual splitting.

Besides multiplication one of the more expensive operations in field arithmetic is reduction. There are algorithms (covered in the book) that deal with reduction in fields of prime order where the prime has a specific form. They also discuss Barrett reduction which is a general reduction algorithm, which does not require the prime to have a specific form.

The idea for Barrett reduction is to compute the reduction of z in $\mathbb{F}_p$ that is we want to compute: $r = z \mod p = z - qp$, for some q. So first we recognize that $q = \frac{z}{p}$ then we can estimate q by computing (a bit more efficiently):

$$q \approx \left\lfloor \left\lfloor \frac{z}{b^{k-1}} \right\rfloor \cdot \left\lfloor \frac{b^{2k}}{p} \right\rfloor \cdot \frac{1}{b^{k+1}} \right\rfloor$$

and using this approximation and others we can quickly determine the reduction of numbers modulo a prime number.

Another computationally expensive component of field arithmetic is the inversion of elements. For this we use the extended Euclidean algorithm.

The Euclidean algorithm is the most efficient method to compute $\gcd(a, b)$, and relies on the fact that $\gcd(a, b) = \gcd(b - ca, a)$ for any integer c. This algorithm can be extended to find integers x, y such that $ax+by = \gcd(a, b)$. In fact there will be infinitely many solutions, x, y. This is helpful because the extended algorithm still follows the well known method of finding $\gcd(a, b)$. In particular if $\gcd(a, b) = 1$ then near the end of the computation we will reach a stage with:

$$u_n = q_n v_n + 1$$

where u_n and v_n are computed via previous steps in the Euclidean algorithm. The important point is that the extended Euclidean algorithm is always computing x_i and y_i such that at every step we have $ax_i + by_i = u_i$ and we will terminate when $u_i = 0$. Thus for our inversion we have a number a we would like to compute $a^{-1} \mod p$ for some prime p. So we desire an x such that $ax \equiv 1 \mod p$ or $ax + py = 1$. So we run the extended Euclidean algorithm on a and p, and stop one step early (when $u_i = 1$) to find $x \equiv a^{-1} \mod p$.

These are the main algorithms (or their ideas) which help significantly speed up computation in a field of prime order. The ideas presented hold for any arbitrary prime but can be significantly sped up with good choices of primes. NIST has recommended many primes for which even more speedups are available based on the how the number can be written as a sum and difference of powers of 2. And all but the last in the list involve powers of 2 which are multiples of 32 – meaning on machines with a 32-bit architecture there can be further speedups. The recommended NIST primes are:

$$\begin{aligned}
p_{192} &= 2^{192} - 2^{64} - 1 \\
p_{224} &= 2^{224} - 2^{96} + 1 \\
p_{256} &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \\
p_{384} &= 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1 \\
p_{521} &= 2^{521} - 1
\end{aligned}$$

3.3 Binary field arithmetic

For binary fields the storage of elements is very simple; since our elements are really just binary polynomials of a certain degree we need to only store whether each term is included or not. This also makes addition (which is the same as subtraction in $\mathbb{F}_2$) very simple. As for multiplication, Karatsuba-Ofman methods still exist and are still efficient. However it is worthwhile to examine a comb method as well. Here we consider multiplying two polynomials of degree m: $a(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0$ and $b(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1x + b_0$. One way to compute the multiplication is as follows:

$$\begin{aligned} a(x) \cdot b(x) &= a_{m-1}x^{m-1}b(x) + a_{m-2}x^{m-2}b(x) + \cdots + a_1xb(x) + a_0b(x) \\ &= ((\cdots(a_{m-1}b(x) \cdot x + a_{m-2}b(x))\, x + a_{m-3}b(x))\, x + \cdots + a_1b(x))\, x + a_0b(x) \end{aligned}$$

The idea is to compute this multiplication from the inside out. This stays simple since we are dealing with binary polynomials so $a_i \in \{0, 1\}$. Thus we would first take the highest degree term of $a(x)$ and multiply its coefficient by $b(x)$ and an additional x and add the next highest term then multiply that sum by an x. Thus the highest term continually accumulates x's and the lower degree terms collect they correct amount of x's.

Also worth noting is that for binary polynomials squaring is very simple. Since we are over a field of characteristic two all of the cross-terms will become zero thus $a^2(x) = a_{m-1}x^{2(m-1)} + a_{m-2}x^{2(m-2)} + \cdots + a_1x^2 + a_0$, we only have to square the variable terms in order to compute $a^2(x)$.

As for inversion we use the same ideas here as we did for prime fields. The specifics of the algorithms must change to adapt to the fact we are working with polynomials now instead of just numbers, but the same idea will still work and it will work in the same way. The main difference here is that the computations will actually be noticeably faster since we can take advantage of the binary nature of our elements.

3.4 Optimal field extensions

For the sake of completeness we discuss what makes an optimal field extension, though we leave out the algorithms associated with computations in the field, since they are used in very specific circumstances.

When we discuss optimal field extensions we already have a working base field $\mathbb{F}_p$. An extension field $\mathbb{F}_{p^m}$ is optimal if $p = 2^n - c$ for some integers n and c with $\log_2 |c| \leq n/2$ and there is an irreducible polynomial $f(x) = x^m - \omega$ in $\mathbb{F}_p[x]$ with $\omega \in \mathbb{F}_p$. Worth noting is that we call our optimal field extension Type 1 if $c \in \pm 1$ (if $c = 1$, p is a Mersenne prime). And if $\omega = 2$ then the optimal field extension is Type 2.

Type 1 fields allow for incredibly simple arithmetic in the subfield $\mathbb{F}_p$ and Type 2 fields allow for simplifications in computations in the field $\mathbb{F}_{p^m}$. Exploiting the structure created by optimal field extensions as well as the Type of the field extension can lead to significant speedups in computations. However unless necessary due to hardware constraints it is often more useful to use a binary field extension.

4 Elliptic Curve Arithmetic

This section will more formally introduce elliptic curves and their group structure. Worth keeping in mind is that the computations done on elliptic curves really take place in their base field. Thus the last section's work on finite field arithmetic will be put indirectly to work this section.

We first introduce elliptic curves and give their definitions and properties, explicitly describe the group law and group structure. We then consider different ways to represent points on the curve, introducing projective coordinates. Then we focus briefly on point multiplication, the main computational factor of elliptic curves. We finish with a discussion of curves with efficiently computable endomorphisms. This chapter of the book required a few supplemental topics from Silverman's "The Arithmetic of Elliptic Curves" specifically for Weierstrass equations and calculating the number of points on a curve.

4.1 Introduction to Elliptic curves

Definition: An elliptic curve E over a field k is defined by the equation

$$E: \quad y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

with $a_i \in k$ and $\Delta \neq 0$ where:

$$\begin{aligned}
\Delta &= -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\
d_2 &= a_1^2 + 4a_2 \\
d_4 &= 2a_4 + a_1a_3 \\
d_6 &= a_3^2 + 4a_6 \\
d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2
\end{aligned}$$

Note that in general this is an implicit function of two variables, and delta represents the discriminant of the curve. If L is any field extension of k (where we can think of k being an extension of itself too) the set of L rational points on E is:

$$E(L) = \{(x, y) \in L \times L : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{\infty\}$$

The requirement that $\Delta \neq 0$ gives that we will have a *smooth* curve; we won't have points with two distinct tangent lines (like a sharp point) [3]. The way we have written our curves is the most general, but we can simplify the way we write the curve. Some example elliptic curves are given in the figures below.

4.1.1 Simplified Weierstrass equations

First we discuss when two equations represent the same curve, two equations E_1, E_2 over a field k given by:

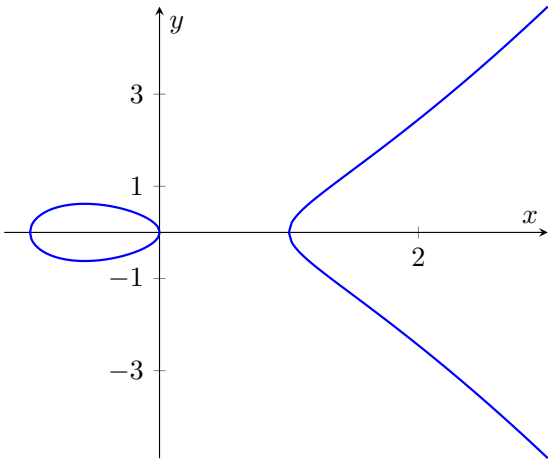

Figure 3: Graph of $y^2 = x^3 - x$

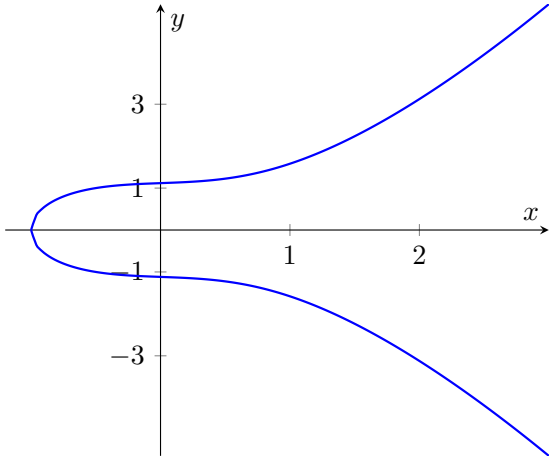

Figure 4: Graph of $y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$

$$E_1: \quad y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$
$$E_2: \quad y^2 + \overline{a_1}xy + \overline{a_3}y = x^3 + \overline{a_2}x^2 + \overline{a_4}x + \overline{a_6}$$

are isomorphic (over k) if there exist some $u, r, s, t \in k$ with $u \neq 0$ such that the change of variables from $(x, y) \to (u^2x + r, u^3y + u^2sx + t)$ transforms E_1 into E_2, this is referred to an admissible change of variables.

Using admissible changes of variables we can simplify our equations for an elliptic curve dramatically. In the case where the base field, k, has char $k \neq 2, 3$ then with the following curve:

$$E: \quad y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

we can use the following change of variables:

$$(x, y) \to \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - a_1x}{216} - \frac{a_1^3 + 4a_1a_2 - 12a_3}{24}\right)$$

to come out with the simplified curve:

$$E: \quad y^2 = x^3 + ax + b$$

where $a, b \in k$ and our requirement on Δ becomes: $\Delta \neq -16(4a^3 + 27b^2)$ [3].

There are similar transformations for when char $k = 2, 3$ to simplify the curve expression, though the end result is not as nice and simple as for the preceding case. For the sake of simplicity and succinctness we focus on the case where char $k \neq 2, 3$ in the following sections, though every thing stated also has an analogue in those cases.

4.1.2 Group Law

Back in Section 2.2.3 we mentioned that elliptic curves form a group under addition, but the addition of points is not as straight forward as adding components. We know introduce the group law of an elliptic curve and give the motivation of this group law.

The standard way to add points of the form (x, y) is to add the component-wise, the issue here is that this addition does not keep a group structure; indeed adding component wise would, in general, not even keep the resulting point on the elliptic curve. In order to keep our group structure we must define things differently. Let us consider, first, adding two distinct points, P and Q on an elliptic curve. The first step is to draw a line connecting the two points P and Q. Since this is a line $y = mx + b$ it must intersect a curve of the form $y^2 = x^3 + ax + b$ exactly three times (except in one case) thus find the third point of intersection of this line with our elliptic curve call this point R'. Then the point reflected across the x-axis is to be the summation of our points P and Q sometimes denoted $P \oplus Q$, as seen below. In the case where P and Q are connected via a vertical line then the line between them intersects the curve only at P and Q, in which case we give the third intersection point as the point at ∞. And we define $P \oplus Q = \infty$.

In the case where we wish to add P to itself we proceed in the same manner choosing instead the tangent line at P.

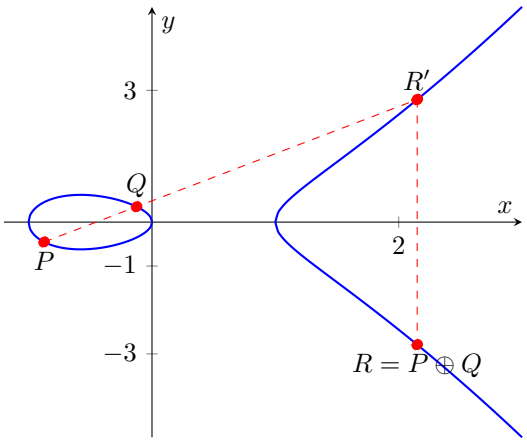

Figure 5: Example of addition of two points P and Q on the elliptic curve $y^2 = x^3 - x$.

We present the group law for the case where our base field, k, has char $k \neq 2, 3$. For $E/k : y^2 = x^3 + ax + b$ we have:

1. Identity: $P + \infty = \infty + P = P$, $\forall\ P \in E(k)$.

2. Negatives: If $P = (x, y) \in E(k)$ then $(x, y) + (x, -y) = \infty$. The point $(x, -y)$ is denoted $-P$ and is called the negative of P. Note that $-P$ is indeed a point in $E(k)$, due to our equation $y^2 = \cdots$. Also $-\infty = \infty$.

3. Point addition: Let $P = (x_1, y_1)$, $Q = (x_2, y_2) \in E(k)$, with $P \neq \pm Q$ then $P + Q = (x_3, y_3)$ with:

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \qquad y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1$$

4. Point doubling: Let $P = (x_1, y_1) \in E(k)$ where $P \neq -P$. Then $2P = (x_3, y_3)$ where:

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \qquad y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1$$

4.1.3 Group order

One of the main characteristics of a group is it's order, how many elements it has. Unfortunately it is not always straight forward to determine the order of an elliptic curve group. However we can give a good approximation to the order due to a theorem from Hasse:

Theorem 1. *Let E be an elliptic curve defined over $\mathbb{F}_q$ then:*

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

Hasse's Theorem can also be stated as: $\#E(\mathbb{F}_q) = q + 1 - t$ where $|t| \leq 2\sqrt{q}$. t is called the trace of E over $\mathbb{F}_q$. For a proof of Hasse's Theorem see [3].

There are theorems presented in [2] that show the existence of elliptic curve groups with a given order. But a more interesting characteristic is that given an elliptic curve, E, defined over $\mathbb{F}_q$ then E is also defined over any extension of $\mathbb{F}_q$, i.e., $\mathbb{F}_{q^n}$. The group $E(\mathbb{F}_q)$ of $\mathbb{F}_q$-rational points is a *subgroup* of $E(\mathbb{F}_{q^n})$ which by Lagrange's Theorem gives that $\#E(\mathbb{F}_q)|\#E(\mathbb{F}_{q^n})$ and leads to a theorem which gives that if $\#E(\mathbb{F}_q)$ is known then $\#E(\mathbb{F}_{q^n})$ can be computed:

Theorem 2. *Let E be an elliptic curve defined over $\mathbb{F}_q$ and let $\#E(\mathbb{F}_q) = q+1-t$ be known. Then $\#E(\mathbb{F}_{q^n}) = q^n + 1 - V_n$ for all $n \geq 2$ where $\{V_n\}$ is the recursively defined sequence $V_0 = 2$, $V_1 = t$, $V_n = V_1 V_{n-1} - qV_{n-2}$ for $n \geq 2$.*

4.1.4 Group structure and Isomorphism Classes

The order of our elliptic curve group does not completely determine the group, indeed we need to know more about the structure of the group, the following theorem helps with that:

Theorem 3. *Let E be an elliptic curve defined over $\mathbb{F}_q$. Then $E(\mathbb{F}_q)$ is isomorphic to $\mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}$ where n_1 and n_2 are uniquely determined positive integers such that $n_2|n_1$, $n_2|(q-1)$ and $\#E(\mathbb{F}_q) = n_2 n_1$.*

This gives a very simple consequence, if $n_2 = 1$ then our group $E(\mathbb{F}_q)$ is cyclic. Indeed if n_2 is small ($n_2 = 2, 3, 4$) we still say that $E(\mathbb{F}_q)$ is *almost* cyclic. Based on the conditions on n_1, n_2 we expect most $E(\mathbb{F}_q)$ to be cyclic or almost cyclic often [2].

Recall that two elliptic curves are isomorphic if there exists a change of variables between them. Naturally this leads to the conclusion that if two curves our isomorphic their corresponding groups are also isomorphic. The converse of this is not true. The following theorem can help give an example of this.

Theorem 4. *Let $k = \mathbb{F}_q$ and char $k \neq 2, 3$.*

(i) The elliptic curves:

$$E_1 : \quad y^2 = x^3 + ax + b \tag{1}$$
$$E_2 : \quad y^2 = x^3 + \overline{a}x + \overline{b} \tag{2}$$

defined over k are isomorphic (over k) if and only if $\exists\ u \in k^*$ such that $u^4\,\overline{a} = a$ and $u^6\,\overline{b} = b$. If such a u exists then the change of variables:

$$(x, y) \to (u^2x, u^3y)$$

transforms (1) *into* (2).

(ii) The number of isomorphism classes of elliptic curves over k is either $2q + 6$, $2q + 2$, $2q + 4$, or $2q$ for $q \equiv 1, 5, 7, 11 \mod 12$ respectively.

Using the theorems presented above we can give an example of two curves whose groups are isomorphic but the curves are not. We look at curves defined over $\mathbb{F}_5$. Using Theorem 3 we can see that the curves given by:

$$E_1 : \quad y^2 = x^3 + 4$$
$$E_2 : \quad y^2 = x^3 + 3$$

both have $E_i(\mathbb{F}_5) = \mathbb{Z}_6$ (given that they both have six points on them). However, by Theorem 4 we can see that there is no $u \in \mathbb{F}_5$ that gives $3u^6 = 4$, this can be easily seen since $u^6 = u^2$ in $\mathbb{F}_5$ and $3(1)^2 = 3$, $3(2)^2 \equiv 2 \mod 5$, $3(3)^2 \equiv 2 \mod 5$ and $3(4)^2 \equiv 3 \mod 5$. So there is no u that will correctly transform $3 \to 4$. Thus the curves are not isomorphic but their associated groups are.

4.2 Point Representation and the group law

The equations given above for adding and doubling points all involve inverting elements of the base field k. Since inversion is a significant factor in computation time we can benefit by representing our points in a coordinate system and developing different formulas for addition and doubling which avoid the costly use of inversion. The coordinate system we refer to is projective coordinates.

For a given field k, and $c, d \in \mathbb{Z}^{>0}$ we define an equivalence relation $\sim$ on $k^3 \backslash \{(0, 0, 0)\}$ as $(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2)$ if $X_1 = \lambda^c X_2$, $Y_1 = \lambda^d Y_2$ and $Z_1 = \lambda Z_2$. We write the equivalence class containing (X, Y, Z) in $k^3 \backslash \{(0, 0, 0)\}$ as

$$(X : Y : Z) = \{(\lambda^c X, \lambda^d Y, \lambda Z) : \lambda \in k^*\}$$

$(X : Y : Z)$ is called a **projective point** and (X, Y, Z) it's representative. The set of all projective points is denoted $\mathbb{P}(k)$. We also have the concept: if $(X_1, Y_1, Z_1) \in (X : Y : Z)$ then $(X_1 : Y_1 : Z_1) = (X : Y : Z)$ in particular if $Z \neq 0$ we can write: $(X/Z^c, Y/Z^d, 1)$ as a representative of $(X : Y : Z)$, this is the only representative in this class with the Z component equal to 1. This gives a correspondence between the projective points with non-zero Z terms and affine points (as we have been using):

$$\mathbb{P}(k)^* = \{(X : Y : Z) : X, Y, Z \in k, Z \neq 0\} \leftrightarrow \mathbb{A}(k) = \{(x, y) : x, y \in k\}$$

Note that we call

$$\mathbb{P}(k)^\circ = \{(X : Y : 0) : X, Y \in k\}$$

the line at ∞.

Given this new coordinate system we can rewrite our equations in projective coordinates by replacing x with X/Z^c and y with Y/Z^d and clearing the denominator. In particular if $(X, Y, Z) \in k^3 \backslash \{(0, 0, 0)\}$ satisfies our projective equation then so does all of $(X : Y : Z)$ so we can say that $(X : Y : Z)$ lies on E. This method gives the correspondence described above and $\mathbb{P}(k)^\circ$ is the "point" at ∞ in our original system.

While the above holds for general $c, d \in k$ there are some standard choices, for simplicity we only consider the curve given by $y^2 = x^3 + ax + b$.

1. Standard projective coordinates, $c, d = 1$: In this case $(X : Y : Z)$, $Z \neq 0$ corresponds to $(X/Z, Y/Z)$ and our curve becomes $Y^2 Z = X^3 + aXZ^2 + bZ^3$. In this system $(0 : 1 : 0)$ corresponds to ∞ and $-(X : Y : Z) = (X : -Y : Z)$.

2. Jacobian projective coordinates, $c = 2$, $d = 3$: [2] uses this system most, providing formulas for point doubling. Here $(X : Y : Z)$, $Z \neq 0$ corresponds to the point $(X/Z^2, Y/Z^3)$ and the curve becomes: $Y^2 = X^3 + aXZ^4 + bZ^6$. The point $(1 : 1 : 0)$ corresponds to ∞ and $-(X : Y : Z) = (X : -Y : Z)$.

3. Chudnovsky coordinates: Here we use the same $c = 2$, $d = 3$ as in Jacobian coordinates but now we represent the point $(X : Y : Z)$ as $(X : Y : Z : Z^2 : Z^3)$. The redundancy sometimes helps speed up multiplication of points [2].

As mentioned above [2] provides the formula for point doubling (finding $2P$ given P) in Jacobian coordinates. Rather than restate it, it would be more beneficial to describe the process for determining the addition formulas in new coordinate from the affine coordinate system.

To add two points $(X : Y : Z)$ and $(X' : Y' : Z')$ that are both on an elliptic curve, you would first find their corresponding affine points: $(X/Z^c, Y/Z^d)$ and $(X'/(Z')^c, Y'/(Z')^d)$. Then using the given formula for adding two points on an elliptic curve (which use the inversion we are trying to avoid). This will give expression for the X and Y term of our new

point. After finding the common denominator we let the denominator be the Z term and clear it out of the X and Y expressions and then what we are left with is three formulas for the X, Y, and Z term of our new point with no inversions. See [2] for an example using Jacobian coordinates.

4.3 Point Multiplication

The most expensive operation in our crypto-systems will be computing nP for P on an elliptic curve and $n \in \mathbb{Z}$. We refer to this operation as point (or scalar) multiplication. Thus we will talk briefly about the runtime of this operation with the material we have covered and then discuss a new method to help give some speedups. We assume here that our elliptic curve group has $\#E(\mathbb{F}_q) = nh$ where n is prime and h is small, forcing $n \approx q$. We also assume $|P| = n$, that is $nP = \infty$. Then for any $k \in [1, n-1]$ we can write $k = (k_{t-1}, k_{t-2}, \ldots, k_1, k_0)_2$ the binary expansion of k. Using this we can write the standard double and add formula for computing kP as:

Algorithm: LTR binary method for point multiplication
Input: $k = (k_{t-1}, \ldots k_0)_2$, $P \in E(\mathbb{F}_q)$
Output: $kP \in E(\mathbb{F}_q)$.

$Q \leftarrow \infty$
for $i = t - 1$ **to** 0 *by* -1 **do**
  $Q \leftarrow 2Q$
  if $k_i = 1$ **then**
    $Q \leftarrow Q + P$
  end
end
return Q

Figure 6: The double-and-add algorithm for evaluating kP

To give an expected run time of this algorithm we note that the expected number of 1's in the binary representation is roughly $t/2$ (where t is the length of the binary expansion).Thus the expected run time of the algorithm is $t/2$ additions and t point doublings. As mentioned before we can speed this up using different coordinate systems. Indeed the "quickest" presented method would be to have our point Q stored in Jacobian coordinates and P in affine coordinates (standard (x, y) form). Then we can put all of our computations in terms of base field computations and our end result will require $8t$ field multiplications and $5.5t$ field squarings. Additionally we will need to do 1 field inversion 3 field multiplications and 1 field squaring in order to convert our results back to affine coordinates.

This is about the best we can hope for with what we have presented. It turns out that if we store our number k in a more efficient manner we can speed up our overall computation. The disadvantage of binary is that for a random number k the binary expansion tends to have a lot of 1s which will require more work in our algorithm. But we can notice that since we are dealing with a field such that char $\mathbb{F}_q \neq 2, 3$ we get that for $P \in E(\mathbb{F}_q)$, if $P = (x, y)$

then $-P = (x, -y)$. That is subtraction of points is just as efficient as addition, it is easy to switch from $P - Q = P + -Q$. Thus to compute kP we can benefit from writing k as a signed bit representation.

A **non-adjacent form** (NAF) of a positive integer k is an expression $k = \sum_{i=0}^{l-1} k_i 2^i$ where $k_i \in \{0, \pm 1\}$, $k_{l-1} \neq 0$ and no consecutive k_i are non-zero. The **length** of the NAF(k) is l. We present the following theorem of properties of the NAF:

Theorem 5. *(i) $k \in \mathbb{Z}$ has a unique NAF denoted NAF(k).*

(ii) NAF(k) has the fewest non-zero digits of any signed digit representation of k.

(iii) The length of NAF(k) is at most one more than the binary representation length.

(iv) If the length of NAF(k) is l then $2^l/3 < k < 2^{l+1}/3$.

(v) The average density of non-zero digits among all NAFs of length l is approximately $\frac{1}{3}$.

In order to compute the NAF(k) we repeatedly divide k by 2 and force a remainder $r \in \{0, \pm 1\}$ where if k is odd then $r \in \{\pm 1\}$ is chosen so that $(k - r)/2$ is even, forcing the next digit to be 0. We provide the algorithm below:

```
Algorithm: Construction of NAF(k).
Input: Positive integer k.
Output: NAF(k).

i ← 0
while k ≥ 1 do
    if k is odd then
        k_i ← 2 − (k_i mod 4)
        k ← k − k_i
    end
    else
        k_i = 0
    end
    k ← k/2
    i ← i + 1
end
return (k_{i−1}, k_{i−2}, ..., k_1, k_0)
```

Figure 7: Algorithm for computing NAF(k).

This constructs our representation by using $k \mod 4$; if k is odd then it is either almost divisible by 2 ($k \mod 4 \equiv 1$) or it is almost divisible by 2 twice ($k \mod 4 \equiv 3$). In the latter case we use a higher power of 2 and subtract 1. As a simple example we would represent 3 as $(1, 0, -1)_{\text{NAF}}$ as opposed to $(0, 1, 1)_2$.

Algorithm: LTR binary method for point multiplication with NAF representation.
Input: $k = (k_{t-1}, \ldots k_0)_{\text{NAF}}$, $P \in E(\mathbb{F}_q)$
Output: $kP \in E(\mathbb{F}_q)$.

$Q \leftarrow \infty$
for $i = t - 1$ **to** 0 *by* -1 **do**
  $Q \leftarrow 2Q$
  if $k_i = 1$ **then**
    $Q \leftarrow Q + P$
  end
  if $k_i = -1$ **then**
    $Q \leftarrow Q - P$
  end
end
return Q

Figure 8: The double and add algorithm for computing kP with NAF(k).

Using the NAF representation of integers we recall Theorem 5 part (v), since we only have roughly 1/3 of the entries filled (as opposed to 1/2 for binary) we can expect our computation to lessen, slightly. Indeed the addition algorithm is seen in Figure 8.

Using this algorithm our expected run time (in elliptic curve computations) is $t/3$ additions and t point doublings. This leads to even less base field computations, as would be expected. There are further speedups available which are presented in [2].

4.4 Curves with efficiently computable endomorphisms

In order to further speed up computations when calculating kP we look for efficiently computable endomorphisms. We do this by defining the integer representation of an endomorphism and using that to speed up point multiplication, but first the necessary definitions.

Definition: Let E/k be an elliptic curve. The set of all points on E whose coordinates also lie in **any** field extension of k is also called E. An endomorphism, ϕ of E over k is a map, $\phi : E \to E$ such that $\phi(\infty) = \infty$ and $\phi(P) = (g(P), h(P))$, $\forall\, P \in E$ and g, h are rational functions with coefficients in k.

Note that ϕ is a group homomorphism as well. Also the set of all endomorphisms forms a ring.

Definition: The characteristic polynomial of an endomorphism, ϕ, is the monic polynomial, $f(x)$, of least degree in $\mathbb{Z}[x]$ such that $f(\phi) = 0$, that is $f(\phi)(P) = \infty$ for all $P \in E$.

We examine some examples of endomorphisms and their characteristic polynomials:

1. Let E be an elliptic curve defined over $\mathbb{F}_q$. For each integer m the multiplication-by-m map $[m] : E \to E$ defined by:
$$[m] : P \mapsto mP$$

is an endomorphism of E defined over $\mathbb{F}_q$. The characteristic polynomial of $[m]$ is $f(x) = x - m$.

2. Let E be an elliptic curve defined over $\mathbb{F}_q$. The qth power map $\phi : E \to E$ defined by:

$$\phi : (x, y) \mapsto (x^q, y^q) \quad \phi : \infty \mapsto \infty$$

is the Frobenius endomorphism. The characteristic polynomial of ϕ is $f(x) = x^2 - tx + q$ where t is the trace of E, $t = q + 1 - \#E(\mathbb{F}_q)$.

In order to use endomorphisms to help speed up computations we need the idea of the integer representation of an endomorphism. Suppose that E is an elliptic curve defined over $\mathbb{F}_q$ and $\#E(\mathbb{F}_q)$ is divisible by a prime n but not divisible by n^2. Then by Sylow's Theorem $E(\mathbb{F}_q)$ contains exactly one subgroup of order n, say $\langle P \rangle$ where $P \in E(\mathbb{F}_q)$ is of order n. If ϕ is an endomorphism of E then $\phi(P) \in E(\mathbb{F}_q)$ and in particular $\phi(P) \in \langle P \rangle$. Thus if $\phi(P) \neq \infty$ then $\phi(P) = \lambda P$ some $\lambda \in [1, n-1]$.

We call λ the integer representation of ϕ on the subgroup $\langle P \rangle$. We can use this to speedup the computations of kP as follows:

Given an endomorphism ϕ that is easily computable for all $Q \in \langle P \rangle$ and we know that $\phi(Q) = \lambda Q$ for all $Q \in \langle P \rangle$ then we can compute kP by writing $k = k_1 + k_2\lambda \mod n$ (where n is the order of P). Then we compute:

$$\begin{aligned} kP &= k_1 P + k_2 \lambda P \\ &= k_1 P + k_2 \phi(P) \end{aligned}$$

Using this method can (in some cases) speed up calculations of kP. Using this along with the previous speed up can be very worthwhile.

5 Cryptographic Protocols

With the background from the last section we now present the Elliptic curve discrete log problem (ECDLP) on which many of the elliptic curve crypto-systems are based. We also examine one of the popular attacks on the ECDLP which helps demonstrate how large public keys should be chosen.

Definition: The ECDLP is: given an elliptic curve E defined over $\mathbb{F}_q$, $P \in E(\mathbb{F}_q)$ of order n and $Q \in \langle P \rangle$ find the integer $l \in [0, n-1]$ such that $Q = lP$. We call l the discrete log of Q base P and denote it $l = \log_P Q$.

We want the parameters for our crypto-systems to resist all known attacks on the ECDLP. The naïve approach is to exhaustively search $P, 2P, 3P, \ldots$ until we find Q. On average this takes $\frac{n}{2}$ steps. Thus we require n to be sufficiently large that this takes prohibitively long, often $n \geq 2^{80}$.

The best known general purpose attack on the ECDLP is the combination of the Pohlig-Hellman and Pollard's rho algorithm which has a fully exponential running time (in bits) of $O(\sqrt{p})$ where p is the largest prime factor of n. To resist this attack we need n to be divisible by a sufficiently large prime to make this infeasible, often ($p \geq 2^{160}$).

5.1 Pohlig-Hellman attack

We now focus in on the Pohlig-Hellman attack on the ECDLP. It reduces the computation of $l = \log_P Q$ to the discrete log problem in prime order subgroups, hence why we need n to have a large prime factor.

Suppose that $|P| = n = p_1^{e_1} \cdots p_r^{e_r}$ (the prime factorization of n) the strategy is to compute $l_i = l \mod p_i^{e_i}$ for $1 \leq i \leq r$ and then solve the system:

$$\begin{aligned} l &\equiv l_1 \mod p_1^{e_1} \\ l &\equiv l_2 \mod p_2^{e_2} \\ &\vdots \\ l &\equiv l_r \mod p_r^{e_r} \end{aligned}$$

for $l \in [0, n-1]$ (this exists uniquely by the Chinese Remainder Theorem). In order for this computation to be reduced to the computation on prime order subgroups, for each i we write:

$$l_i = z_0 + z_1 p_i + z_2 p_i^2 + \cdots + z_{e_i - 1} p_i^{e_i - 1}$$

where $z_j \in [0, p_i - 1]$ (this is the base p_i expansion of l_i). Then we compute z_j one at a time as follows: First compute $P_0 = \left(\frac{n}{p_i}\right) P$, and $Q_0 = \left(\frac{n}{p_i}\right) Q$, then $|P_0| = p_i$ which gives that:

$$\begin{aligned} Q_0 &= \frac{n}{p_i} Q \\ &= l \left(\frac{n}{p_i}\right) P \\ &= l P_0 \\ &= z_0 P_0 \end{aligned}$$

the last line following since $|P_0| = p_i$ and $l = l_i \mod p_i^{e_i}$ which gives that $lP_0 = l_i P_0 \mod p_i = z_0 P_0$.

Thus $z_0 = \log_{P_0} Q_0$ can be obtained by solving the ECDLP in $\langle P_0 \rangle$. Similarly define

$$\begin{aligned} Q_1 &= \left(\frac{n}{p_i^2}\right)(Q - z_0 P) \\ &= \frac{n}{p_i^2}(l - z_0) P \\ &= (l - z_0)\left(\frac{n}{p_i^2} P\right) \end{aligned}$$

$\frac{n}{p_i^2}P$ has order p_i^2 thus $l\frac{n}{p_i^2}P = (z_0 + z_1 p_i)\frac{n}{p_i^2}$

$$\begin{aligned} &= (z_0 + z_1 p_i - z_0)\left(\frac{n}{p_i^2}P\right) \\ &= z_1\left(\frac{n}{p_i}P\right) \\ &= z_1 P_0 \end{aligned}$$

Hence $z_1 = \log_{P_0} Q_1$ can be found by solving the ECDLP in $\langle P_0 \rangle$. In general if we have computed $z_0, z_1, \ldots z_{j-1}$ then $z_j = \log_{P_0} Q_j$ where

$$Q_j = \frac{n}{p_i^{j+1}}\left(Q - z_0 P - z_1 p_i P - z_2 p_i^2 P - \cdots - z_{j-1} p_i^{j-1} P\right).$$

We now finish an example of the Pohlig-Hellman attack. Consider the elliptic curve E defined over $\mathbb{F}_{7919}$ by the equation:

$$E: \quad y^2 = x^3 + 1001x + 75$$

Let $P = (4023, 6036) \in E(\mathbb{F}_{7919})$ then $|P| = n = 7889 = 7^3 \cdot 23$. Let $Q = (4135, 3169) \in \langle P \rangle$ we try to determine $l = \log_P Q$.

(i) First we write $l_1 = l \mod 7^3$ and $l_1 = z_0 + z_1 7 + z_2 7^2$ and

$$\begin{aligned} P_0 &= (7^2 \cdot 23)P = (7801, 2071) \\ Q_0 &= (7^2 \cdot 23)Q = (7801, 2071) \end{aligned}$$

Since $Q_0 = P_0$ we have that $z_0 = 1$. Now we compute:

$$Q_1 = 7 \cdot 23(Q - P) = (7285, 14)$$

and it can be found that $Q_1 = 3P_0$ which gives that $z_1 = 3$. Again we compute

$$Q_2 = 23(Q - P - (3 \cdot 7)P) = (7285, 7905)$$

from which it can be found that $Q_2 = 4P_0$ so that $z_2 = 4$. Altogether this gives that $l_1 = 1 + 3 \cdot 7 + 4 \cdot 7^2 = 218$.

(ii) Next we determine $l_2 = l \mod 23$ and compute

$$\begin{aligned} P_0 &= 7^3 P = (7190, 7003) \\ Q_0 &= 7^3 Q = (2599, 759) \end{aligned}$$

which determines $Q_0 = 10P_0$ and thus $l_2 = 10$.

(iii) Finally we solve the system:

$$\begin{aligned} l &\equiv 218 \mod 7^3 \\ l &\equiv 10 \mod 23 \end{aligned}$$

which yields the result that $l = 4334$.

The moral of the example is that even in a large field and with a point that has a large order ($n = 7889$) the computation of l can be simplified based on the prime factorization of n. Thus to ensure security we require n have a large prime factor.

6 Conclusion

Overall this was a very enlightening semester. This directed study was a great refresher on the standard cryptographic protocols and a very good covering of the algebraic geometry behind elliptic curve cryptography. I would like to thank Dr. Kaveh for working with me the semester of Fall 2014 providing a lot of help and being a great resource for the material.

References

[1] Steven D. Galbraith. *Mathematics of Public Key Cryptography.* Cambridge University Press, 2012.

[2] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography.* Springer, 2004.

[3] Joseph H. Silverman. *The Arithmetic of Elliptic Curves.* Springer, 2009.