Rose-
Hulman
Undergraduate
Mathematics
Journal

Discrete Logarithms on Elliptic Curves

Aaron Blumenfeld[a]

Volume 12, no. 1, Spring 2011

Sponsored by

Rose-Hulman Institute of Technology
Department of Mathematics
Terre Haute, IN 47803
Email: mathjournal@rose-hulman.edu
http://www.rose-hulman.edu/mathjournal

[a]University of Rochester, New York, ablumenf@u.rochester.edu

Rose-Hulman Undergraduate Mathematics Journal
Volume 12, no. 1, Spring 2011

Discrete Logarithms on Elliptic Curves

Aaron Blumenfeld

Abstract. Cryptographic protocols often make use of the inherent hardness of the classical discrete logarithm problem, which is to solve $g^x \equiv y \pmod{p}$ for x. The hardness of this problem has been exploited in the Diffie-Hellman key exchange, as well as in cryptosystems such as ElGamal. There is a similar discrete logarithm problem on elliptic curves: solve $kB = P$ for k. Therefore, Diffie-Hellman and ElGamal have been adapted for elliptic curves. There is an abundance of evidence suggesting that elliptic curve cryptography is even more secure, which means that we can obtain the same security with fewer bits. In this paper, we investigate the discrete logarithm for elliptic curves over $\mathbb{F}_p$ for $p \geq 5$ by constructing a function and considering the induced functional graph and the implications for cryptography.

Acknowledgements: The author would like to extend thanks to Joshua Holden for his advice and helpful suggestions throughout this project.

1 Introduction

The two most studied number theoretic problems in cryptography are factoring integers and solving the discrete logarithm problem. In this paper, we focus on the latter. The process of modular exponentiation takes a base g (possibly a primitive root) and an integer x and computes $y = g^x \bmod p$. The inverse transformation is known as the discrete logarithm problem — that is, to solve $g^x \equiv y \pmod{p}$ for x. This is considered one of the hardest problems in cryptography, and it has led to many cryptographic protocols.

The best known such protocol that employs the hardness of the discrete logarithm problem is the Diffie-Hellman key exchange. In Diffie-Hellman, the two parties, Alice and Bob, agree on a (public) primitive root g and a large prime p. They choose secret integers, a and b, respectively. Then Alice sends $g^a \bmod p$ to Bob, and Bob sends $g^b \bmod p$ to Alice. Then they can both compute $g^{ab} \bmod p$ since $g^{ab} \equiv (g^a)^b \equiv (g^b)^a \pmod{p}$. They can use this number as their key or use some algorithm to extract a key out of this number. Given g^a, g^b, g and p, it seems impossible to find g^{ab} without finding either a or b, which amounts to solving the discrete logarithm problem (although this assertion has not been proved). Similar ideas are the basis of several cryptosystems, the most prominent of which is ElGamal.

In the past several decades, elliptic curves have entered the scene. Elliptic curves over finite fields contain finite cyclic groups that we can use for cryptography. There is no factorization problem for elliptic curves, but what is used is the discrete logarithm problem, which is to solve $kB = P$ for k. The analog of Diffie-Hellman, in particular, is as follows. Alice and Bob choose a public elliptic curve E (including a public prime p that determines $\mathbb{F}_p$) and a public generator B. They then choose secret integers, a and b, respectively. Alice sends Bob aB and Bob sends Alice bB. They can then both compute abB since $a(bB) = b(aB) = abB$. In this case, abB is a point, so they can use the x-coordinate (or perhaps run it through some algorithm) to obtain a key. Similarly, ElGamal and other cryptosystems have been adapted to elliptic curves.

In this paper, we introduce some of the technical details of elliptic curves over $\mathbb{F}_p$ for $p \geq 5$ and functional graphs, which we then use to study the discrete logarithm problem on elliptic curves. The ultimate goal is, of course, to show that these cryptographic protocols are indeed secure. It would make sense that they are secure if discrete "exponentiation" (written additively in this case) behaves like a "random" map. This map is easier to study through its associated functional graph. Therefore, we devise a functional graph from elliptic curve discrete "exponentiation" and consider some of its theoretical properties, as well as observed computational results.

In particular, we investigate the map $k \mapsto x(kB), \infty \mapsto \infty$, where $x(P)$ denotes the x-coordinate of the point P. We show that these graphs are binary as long as we have N points on the curve with $N \geq p$ odd, and we compare various statistics on these graphs to the expected asymptotic values for random binary functional graphs. Most of the statistics match up well. A few statistics deviate because there is a fixed point at infinity by construction. One statistic that deviates that is more difficult to explain, however, is maximum tail length. This seems to suggest some subtle structure in discrete exponentiation. We discuss this and

the implications for elliptic curve cryptography in detail in this paper.

2 Background

2.1 Elliptic Curves

Definition 2.1. An *elliptic curve* E over a field $\mathbb{K}$ is the set

$$\left\{(x, y) \in \mathbb{K}^2 : y^2 = x^3 + ax + b, a, b \in \mathbb{K}\right\} \cup \{\infty\}$$

with the restriction that $4a^3 + 27b^2 \neq 0$. Notationally, once we have specified the field $\mathbb{K}$, we refer to E as $E(\mathbb{K})$.

Technical Point: There is a more general form of the equation:

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

However, if char $\mathbb{K} \neq 2$ or 3, we can transform it into the above form; in this paper, we specifically consider curves over $\mathbb{F}_p$ for $p \geq 5$. Therefore, we may assume that our curve takes the form given in the definition above.

The requirement that $4a^3 + 27b^2 \neq 0$ simply means that the equation $x^3 + ax + b$ must have no multiple roots.

We may think of the point at infinity as taking on coordinates $\infty = (\infty, \infty)$. We will make the convention that all vertical lines pass through ∞. In any case, we will think of ∞ as a formal symbol computationally, but the idea can be made more rigorous by considerations from projective geometry. See [9, pp. 18 – 20] for more details.

Now that an elliptic curve E gives us a set of points, we define a binary operation $+$ on E. We will see that this will give us an abelian group. Initially, we define $P + \infty = \infty + P = P$ for all $P \in E$. Additionally, we define $Q = -P$ if the y-coordinates of P and Q are additive inverses. Otherwise, when computing $P + Q$, there are three different cases to consider:

- If $P = -Q$, we define $P + Q = \infty$.
- If $P \neq \pm Q$, we draw the line through P and Q. It will intersect E in a third point R. We define $P + Q = -R$.
- If $P = Q$, we draw the line tangent to E at P. It will intersect E in a second point R. We define $P + Q = -R$.

We can summarize the addition law by stating that P, Q and R are collinear if and only if $P + Q + R = \infty$ (recalling that every vertical line intersects ∞).

This operation of addition only seems to make sense if we have $\mathbb{K} = \mathbb{R}$. However, we can derive the formulas for addition [9, p. 14] and they work for any field (except that the formulas that follow must be altered if char $\mathbb{K} = 2$ or 3).

If we are in the second case — i.e., if $P \neq \pm Q$, put

$$P = (x_1, y_1),\ Q = (x_2, y_2),\ P + Q = (x_3, y_3).$$

Then

$$x_3 = m^2 - x_1 - x_2,\ y_3 = m(x_1 - x_3) - y_1,\ \text{where } m = \frac{y_2 - y_1}{x_2 - x_1}.$$

The formulas for the case where $P = Q$ (with nonzero y-coordinate — if the y-coordinate is zero, then $P + Q = \infty$) are the following:

$$x_3 = m^2 - 2x_1,\ y_3 = m(x_1 - x_3) - y_1,\ \text{where } m = \frac{3x_1^2 + a}{2y_1}.$$

As claimed before, this operation of addition makes E into an abelian group. The identity is ∞, and if $P = (x, y)$, then $-P = (x, -y)$. Since the coordinates are in a field, E is closed under addition. That E is abelian follows either from the formulas or from the fact that the line through P and Q is the same as the line through Q and P. The hard part is associativity, the proof of which we omit. It can be verified by using the formulas for all of the different cases. Alternatively, it can be verified by arguments using projective geometry [9, pp. 20 – 32] or complex analytic techniques.

Over $\mathbb{F}_q$, an elliptic curve is either a cyclic group or the direct sum of two cyclic groups (since $\mathbb{F}_q$ is finite, such an elliptic curve is always finite) [9, p. 97]. It is these curves that we will primarily be interested in throughout this paper (particularly for $q = p$). When considering curves which are finite cyclic groups, we will use B notationally to refer to a generator, which means that for every point P on the curve, we can write $P = kB = \underbrace{B + B + \ldots + B}_{\text{k terms}}$ for some $k \in \mathbb{Z}^+$. We will also use the notation $x(P)$ to denote the x-coordinate of the point P and $\#E(\mathbb{F}_p)$ to denote the number of points on E over $\mathbb{F}_p$.

We want to work with cyclic groups because given some group G, we are interested in elements that can be written as a power of a generator g, which are simply the elements of the cyclic subgroup generated by g. We want to work with a finite group because we then get "wrap-around." This disallows us from using tools such as calculus (not to be confused with the index calculus!) to compute discrete logarithms.

We now recall the basics of Legendre symbols. For $a \not\equiv 0 \pmod{p}$, the Legendre symbol is defined as follows:

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } x^2 \equiv a \pmod{p} \text{ has a solution,} \\ -1 & \text{if } x^2 \equiv a \pmod{p} \text{ has no solution.} \end{cases}$$

For $a \equiv 0 \pmod{p}$, we define $\left(\frac{a}{p}\right) = 0$. One standard fact is that $x^2 \equiv a \pmod{p}$ has $1 + \left(\frac{a}{p}\right)$ solutions. This can be seen as follows. For $a \equiv 0 \pmod{p}$, the one solution to $x^2 \equiv 0 \pmod{p}$ is 0; note that $1 + \left(\frac{0}{p}\right) = 1 + 0 = 1$. If $x^2 \equiv a \pmod{p}$ has zero solutions,

then $1+\left(\frac{a}{p}\right) = 1+-1 = 0$. Finally, if $x^2 \equiv a \pmod{p}$ has a solution x, then it has a second solution $-x$ (and no additional solutions); in this case, we have $1+\left(\frac{a}{p}\right) = 1+1 = 2$.

We can use this fact to count the number of points on an elliptic curve over $\mathbb{F}_p$. One way to count points is to include ∞, then to let x run over $\mathbb{F}_p$ and compute the number of distinct square roots of x^3+ax+b. This gives us the formula $1+\sum_{x=0}^{p-1}\left(1+\left(\frac{x^3+ax+b}{p}\right)\right) = p+1+\sum_{x=0}^{p-1}\left(\frac{x^3+ax+b}{p}\right)$. One other useful result is Hasse's theorem, which says that $\left|\sum_{x=0}^{p-1}\left(\frac{x^3+ax+b}{p}\right)\right| \leq 2\sqrt{p}$ [9, p. 97].

Now that we know that some elliptic curves over $\mathbb{F}_p$ are finite cyclic groups, recall that the discrete logarithm problem we are interested in is the following: given $kB = P$ on $E(\mathbb{F}_p)$, solve for k. Recall also the Diffie-Hellman key exchange: Alice and Bob choose secret integers a and b, Alice sends Bob aB, Bob sends Alice bB, and they each obtain the point abB. Since the x-coordinate encodes most of the information about a point (unless $y = 0$, there are exactly two square roots to choose from), some procedure is then followed to extract a key out of the x-coordinate.

It seems that elliptic curve cryptography provides more security than the classical discrete logarithm problem [6, p. 180]. What this means is that we can accomplish the same security with fewer bits. Another advantage is that we have more groups to choose from. For a fixed prime p, there are a large number of elliptic curves with around p elements. One other great advantage is that perhaps the best algorithm that works over finite fields, the index calculus, doesn't work over elliptic curves because points on elliptic curves can't be factored [9, p. 146]. This means that to solve discrete logs over elliptic curves, we typically need to use much slower algorithms that work in more general groups, such as Baby-Step Giant-Step or the Pohlig-Hellman algorithm. One thing to ensure, however, when choosing a curve over $\mathbb{F}_q$ is that the curve not be supersingular (because then the Weil pairing can be used to reduce the problem to solving discrete logs in $\mathbb{F}_{q^k}$ for some k [9, p. 154], which is usually easier, provided that k is not too large — in fact, k tends to be 2 in practice [9, p. 154]). Over $\mathbb{F}_p$, this just means that the curve should be restricted not to have $p+1$ points [9, p. 131]. But it is common practice to ensure that the curve has a prime number of points in the first place, so this issue should not be a huge concern.

2.2 Functional Graphs

If we have a function (especially one defined on a finite set), it is convenient to study its structure by considering a particular graph. We can do this by creating what is called a functional graph.

First, however, we introduce some graph theory notation. If $G = (V, E)$ is a directed graph, then edges in E are represented by ordered pairs (v, u) with $v, u \in V$. If $(v, u) \in E$, we will represent this notationally by $v \to u$.

Definition 2.2. We define the out-degree, denoted $\deg^+(v)$, of a vertex v to be the number

of vertices u such that $v \to u$. Similarly, the in-degree, denoted $\deg^-(v)$, of a vertex v is the number of vertices u such that $u \to v$. We can formalize this by defining $\deg^+(v) = |\{u \in V : (v,u) \in E\}|$ and $\deg^-(v) = |\{u \in V : (u,v) \in E\}|$.

Definition 2.3. A *functional graph* is a directed graph $G = (V, E)$ such that for every $v \in V$, $\deg^+(v) = 1$.

This gives a one-to-one correspondence between functional graphs and functions $f : V \to V$ by including an edge $v_1 \to v_2$ in E if and only if $f(v_1) = v_2$. Conversely, if we have a functional graph $G = (V, E)$, we will refer to the function that it represents by f. This allows us to use the notation $f(v)$, $f^{-1}(v)$ and $f^k(v)$ (where $k \in \mathbb{Z}^+$) for $v \in V$ in the context of functional graphs (we can also use the notation $v \mapsto u$ instead of $v \to u$ in the context of functional graphs). It should be noted that this requires us to define functions from a set V into itself. With the classical discrete logarithm problem, this is fairly straightforward since everything is an integer, but on elliptic curves, there are both points that lie on the elliptic curve and integer "exponents" (but written additively, of course).

It should be clear that the requirement that the out-degree of every vertex be exactly 1 is to mimic the requirement that a function send every element in the domain to exactly one element in the target space. One other idea associated with functions is whether or not they are one-to-one, or more generally, n-to-one. This motivation leads us to the following definition.

Definition 2.4. A functional graph $G = (V, E)$ is said to be *m-ary* if for every $v \in V$, we have $\deg^-(v) \in \{0, m\}$. We use the terms *unary* (or *permutation*) and *binary* for 1-ary and 2-ary, respectively.

Unary graphs are often called permutations because they simply represent bijective functions (assuming they are defined on a finite set). It can be shown [2] that the functional graph induced by $x \mapsto g^x \bmod p$ (where g is a primitive root modulo p) is a permutation. We will be particularly interested in binary functional graphs throughout this paper, however. We show in Section 4.1 that the functional graphs we create from elliptic curves are binary.

There are a few more definitions regarding graphs and functional graphs in particular.

Definition 2.5. We say a vertex v is a *terminal node* if $\deg^-(v) = 0$. Otherwise, we call v an *image node*.

We say a vertex v is a *cyclic node* if there is some $k \in \mathbb{Z}^+$ such that $f^k(v) = v$. Otherwise, we call v a *tail node*.

In terms of functions, a vertex v is terminal if $f^{-1}(v) = \emptyset$ (otherwise, it is an image node).

Given a graph $G = (V, E)$, we can break V up into its *components* (technically, its weakly connected components since it is a directed graph). We define an equivalence relation on V by $v_1 \sim v_2$ if there is a path from v_1 to v_2 in the underlying undirected graph. The equivalence classes of this equivalence relation are the components of the graph. Visually,

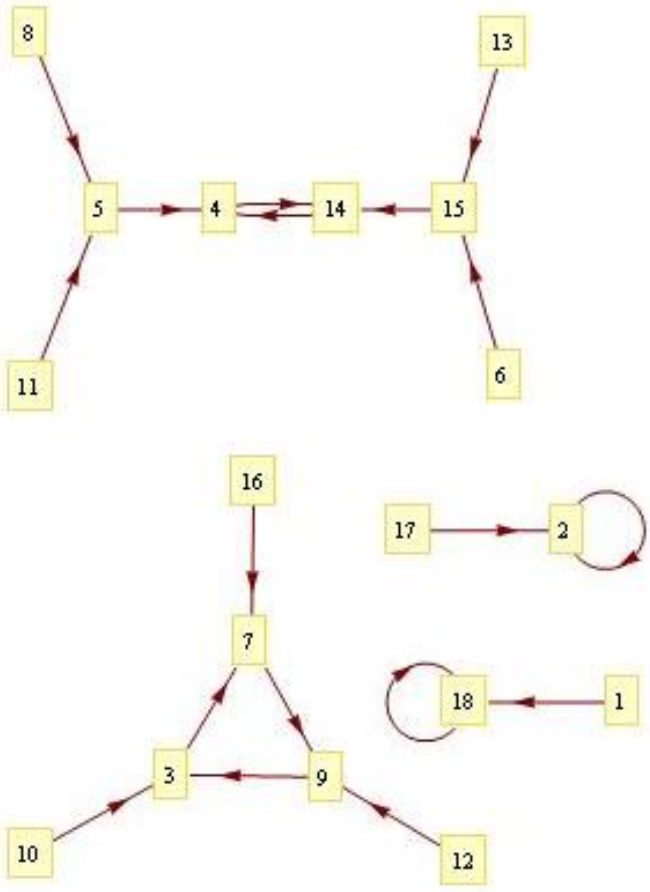

Figure 1: Functional graph of $f(x) = x^2 - 2 \bmod 19$ defined on $(\mathbb{Z}/19\mathbb{Z})^*$.

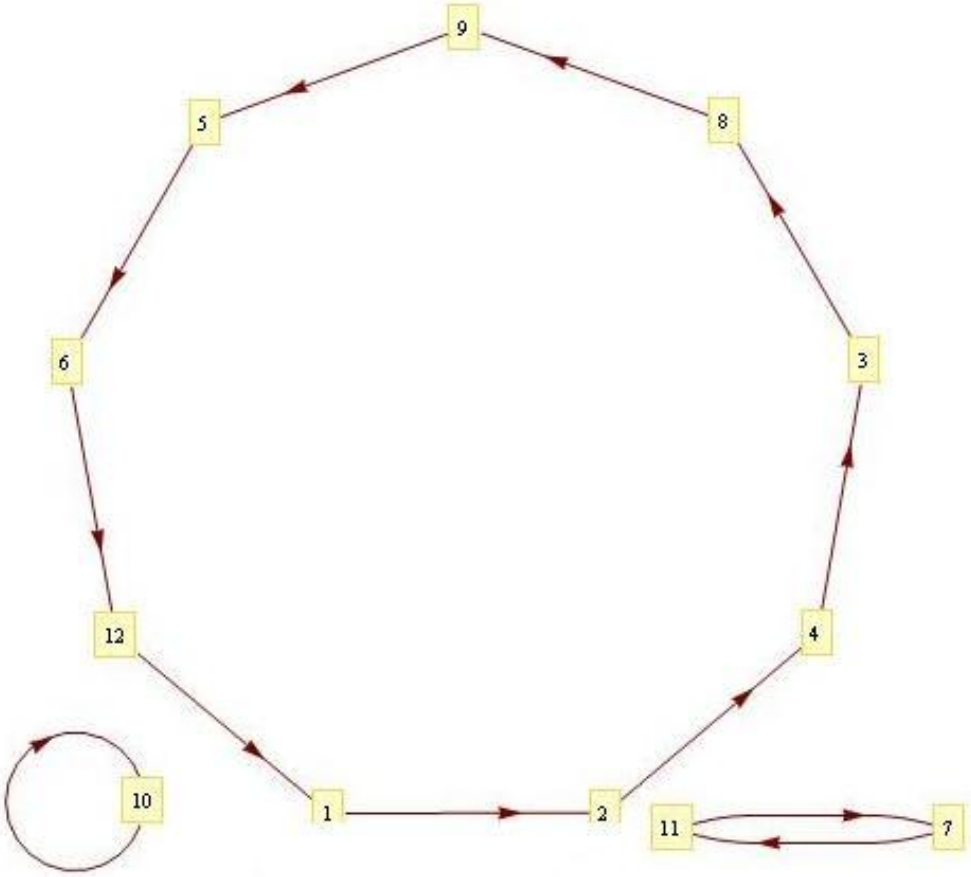

Figure 2: Functional graph of $f(x) = 2^x \bmod 13$ defined on $(\mathbb{Z}/13\mathbb{Z})^*$.

we can think of a component of a *functional* graph as a cycle with tail nodes mapping into the cycle (as we will shortly see in Theorem 2.6). This is illustrated below in Figure 1.

Given any vertex v, there exist distinct $i, j \in \mathbb{Z}_{\geq 0}$ such that $f^i(v) = f^j(v)$. To see this, just iterate f $|V|$ times. This gives us the $|V| + 1$ vertices $v, f(v), f^2(v), \ldots, f^{|V|}(v)$, but there are only $|V|$ vertices in V, so the claim pops out nicely by the pigeonhole principle. Assuming i and j are taken to be the smallest two such integers (with $i < j$), we define i to be the *tail length* of v and $j - i$ to be the *cycle length* of v. Graphically, the tail length is the number of vertices that v is away from entering a cycle, and the cycle length is the length of the cycle it maps into.

So the average cycle length of a graph could refer to the average size of the cycles contained in components or the average of the cycle lengths of all vertices. To avoid confusion, we refer to the former as *average cycle length* and the latter as *weighted average cycle length*.

We will also be interested in the maximum cycle and tail lengths (in this case, the maximum cycle length does not depend on whether we find the maximum cycle length of a vertex or the largest cycle contained in some component).

In Figure 2, we have a functional graph induced by $f(x) = 2^x \bmod 13$. 2 is a primitive

root modulo 13, so this is a permutation. Every vertex is therefore cyclic, so the tail length of every vertex is 0. Vertex 10 has cycle length 1, vertices 7 and 11 have cycle length 2, and so on. There are 3 components (and cycles), the average cycle length is 4, and the weighted average cycle length is $\frac{86}{12} \approx 7.17$. This example shows that average cycle length and weighted average cycle length encode different information about a graph.

We will need a couple of general results about functional graphs later in the paper, the first of which appears in the paper by Flajolet and Odlyzko [4]. We state them here.

Theorem 2.6. *If $f : V \to V$ is a function with V finite, and $G = (V, E)$ is the induced functional graph, then every component of G has exactly one cycle.*

Theorem 2.7. *If $G = (V, E)$ is a binary functional graph with n vertices, then G has exactly $n/2$ terminal nodes and $n/2$ image nodes.*

Proof. The Handshaking Theorem [8, p. 548] tells us that $\sum_{v \in V} \deg^-(v) = |E|$. Since there is one edge for every vertex, this is equal to n. The fact that each image node has exactly two preimages in a binary functional graph (and there can be no overlap since $\deg^+(v) = 1$ for all $v \in V$) implies that the number of image nodes is $n/2$, which also implies that there are $n/2$ terminal nodes. □

This can, in fact, be generalized to show that for an m-ary functional graph of order n, there are $(m-1)n/m$ terminal nodes and n/m image nodes. We can also conclude that the number of vertices in every component of an m-ary functional graph is divisible by m by observing that (after including the necessary edges) a component of an m-ary functional graph is an m-ary functional graph in its own right.

2.3 Random Functional Graphs

The whole reason for appealing to functional graphs is that graph theory is well studied, whereas discrete logs are somewhat mysterious by comparison. The goal is to create a map based on elliptic curve discrete exponentiation, from which we can create a functional graph in order to identify any noticeable patterns. These could lead us to attacks against discrete logs on elliptic curves, or convince us that such maps are indeed random.

Random functional graphs (where we consider the graph induced by each function $f : V \to V$ to be likely to occur with equal probability) have been fairly thoroughly studied. Flajolet and Odlyzko, for example, provide an analysis in [4]. They use exponential generating functions and give expected asymptotic values of various statistics for a random functional graph with n vertices. Similar results regarding binary functional graphs with n vertices appear in Cloutier and Holden's paper [3]. We summarize the results below.

Random Functional Graphs:

Statistic	Expected Asymptotic Value
Number of Components	$\frac{1}{2}(\log(2n)+\gamma)$
Number of Cyclic Nodes	$\sqrt{\pi n/2}-\frac{1}{3}$
Number of Tail Nodes	$n-\sqrt{\pi n/2}+\frac{1}{3}$
Number of Terminal Nodes	$e^{-1}n$
Number of Image Nodes	$(1-e^{-1})n$
Number of k-cycles	$1/k$
Average Cycle Length	$\frac{\sqrt{\pi n/2}-\frac{1}{3}}{2(\log(2n)+\gamma)}$
Weighted Average Cycle Length	$\sqrt{\pi n/8}$
Average Tail Length	$\sqrt{\pi n/8}$
Maximum Cycle Length	$c\sqrt{\frac{\pi n}{2}} \approx 0.78248\sqrt{n}$
Maximum Tail Length	$\sqrt{2\pi n}\log 2 \approx 1.73746\sqrt{n}$

Random Binary Functional Graphs:

Statistic	Expected Asymptotic Value
Number of Components	$\frac{1}{2}(\log(2n)+\gamma)$
Number of Cyclic Nodes	$\sqrt{\pi n/2}-1$
Number of Tail Nodes	$n-\sqrt{\pi n/2}+1$
Number of Terminal Nodes	$n/2$
Number of Image Nodes	$n/2$
Average Cycle Length	$\frac{\sqrt{\pi n/2}-1}{2(\log(2n)+\gamma)}$
Weighted Average Cycle Length	$\sqrt{\pi n/8}$
Average Tail Length	$\sqrt{\pi n/8}$
Maximum Cycle Length	$c\sqrt{\frac{\pi n}{2}} \approx 0.78248\sqrt{n}$
Maximum Tail Length	$\sqrt{2\pi n}\log 2 - 3 + 2\log 2 \approx 1.73746\sqrt{n} - 1.61371$

Here c refers to the constant

$$\int_0^\infty \left(1-\exp\left(-\int_v^\infty \frac{e^{-u}\,du}{u}\right)\right)\,dv.$$

It should be noted that cyclic and tail nodes are dual to each other, as are terminal and image nodes (i.e., knowledge of one is equivalent to knowledge of the other). Also, the average cycle length is the ratio of cyclic nodes to the number of components. The other observation to

make is that these are expected asymptotic values, which means we should compare data against these theoretical values when n is fairly large.

We now show that the expected number of k-cycles for a random binary functional graph is $1/k$, just as in the more general case. This result does not seem to appear in the literature, but the proof is very similar to those found in the papers by Cloutier [2] and by Flajolet and Odlyzko [4].

Theorem 2.8. *The expected number of k-cycles for a random binary functional graph is $1/k$.*

Proof. We first enumerate binary functional graphs combinatorially so that they can be converted into exponential generating functions. This is done as follows [4].

$$\begin{aligned} \text{BinFunGraph} &= \text{set(Component)};\\ \text{Component} &= \text{cycle(BinTree)};\\ \text{BinTree} &= \text{Node * set(BinTree)};\\ \text{Node} &= \text{Atomic Unit}. \end{aligned}$$

In other words, a binary functional graph is a set of components; a component is a cycle of binary trees; a binary tree is a node attached to a set of zero or two (possibly empty) binary trees; a node is an atomic unit.

In general, given a class of combinatorial objects $\mathcal{C}$, if we let C_n denote those of size n, there is a certain power series known as an *exponential generating function* $C(z) = \sum_{n\geq 0} C_n \frac{z^n}{n!}$ [4]. Therefore, we can obtain exponential generating functions for binary functional graphs, components, binary trees, and nodes. The exponential generating functions for binary functional graphs, components, and binary trees are denoted $f(z), c(z)$, and $b(z)$, respectively. These functions, as described in Cloutier's paper [2], are given below.

$$\begin{aligned} f(z) &= \frac{1}{\sqrt{1-2z^2}},\\ c(z) &= \log \frac{1}{\sqrt{1-2z^2}},\\ b(z) &= z + \frac{1}{2} z b^2(z). \end{aligned}$$

Now the closed-form expression for $b(z)$ is

$$b(z) = \frac{1-\sqrt{1-2z^2}}{z}.$$

The proof for the case of random functional graphs from Flajolet and Odlyzko's paper [4] uses a bivariate exponential generating function with the variable u marking k-cycles. This function is based on the formulas for $f(z), c(z)$, and $b(z)$. We imitate the proof by using

the formulas for *binary* functional graphs, as described above. The bivariate exponential generating function is the following.

$$g(u,z) = \exp\left(\log\left(\frac{1}{1-2z^2}\right) + (u-1)\frac{(zb(z))^k}{k}\right).$$

Proceeding as in Flajolet and Odlyzko's paper [4], we compute $g_u(1,z) = \dfrac{z^k b(z)^k}{k\sqrt{1-2z^2}}$. Singularity analysis using the Maple package Algolib reveals that the asymptotic behavior of the nth coefficient of this expression is

$$\frac{2^{1/2+n/2}\sqrt{\dfrac{1}{n}} + O\left(\dfrac{2^{n/2}}{n^{3/2}}\right)\sqrt{\pi}k}{\sqrt{\pi}k}.$$

Now this represents the number of k-cycles in all binary functional graphs with n vertices. We must normalize this, therefore, by dividing by the number of binary functional graphs with n vertices, which can be shown to be $n!\binom{n}{n/2}/2^{n/2}$ [7], and multiplying by $n!$ since in an exponential generating function for the sequence (a_n), each term is of the form $\frac{a_n}{n!}x^n$. Asymptotic analysis in Maple shows that the resulting expression is asymptotic to

$$\frac{1}{k} + O\left(\frac{1}{n}\right),$$

and the term that interests us falls out nicely as $\dfrac{1}{k}$. □

3 Prior Work

Daniel Cloutier [2], Nathan Lindle [7] and Andrew Hoffman [5] have worked on investigating the classical discrete logarithm problem by investigating related functional graphs. Their methods involved computing the number of components, sizes of cycles, numbers of tail and cyclic nodes, and so on, on thousands of functional graphs. They used exponential generating functions to calculate the expected values of random functional graphs, as well as particular classes of random functional graphs (e.g., random permutations and binary functional graphs). Cloutier chose to look at permutations and binary functional graphs in particular because the data he collected was quite inconsistent with random functional graphs, but in many cases turned out to be very accurate compared to the theoretical values for random permutations and binary functional graphs.

No previous work has been done on functional graphs associated with discrete logs on elliptic curves. Given an elliptic curve $E(\mathbb{F}_p)$ with N points, the author's goal has been to investigate them by the map

$$k \mapsto x(kB) \text{ for } k \in \{0,1,\ldots,N-1\},\ \infty \mapsto \infty.$$

It turns out these graphs are binary, so the author followed suit in heavy computation in order to compare the data to what would be expected of random binary functional graphs.

Ultimately, we would like to be assured that discrete exponentiation actually behaves like a random map for security reasons. If there is some way to exploit any structure it imposes, however, we would like to know what it is so that we can either abandon relying on the discrete log problem if it is breakable (which seems unlikely) or perhaps ensure that certain conditions are satisfied so that we can continue to communicate with one another securely (for example, in the classical discrete log problem, it is common practice to make sure that $p-1$ has a large prime factor).

4 Elliptic Curve Functional Graphs

It is relatively straightforward to devise a function to induce a functional graph that describes discrete exponentiation in the classical case. Simply map $x \mapsto g^x \bmod p$ for some nonzero g. With elliptic curves, however, the situation is rather subtle. Our "exponent" (i.e., the number k in the equation $kB = P$) is an integer, but our points are not, and we would like to take an integer k and consider kB. Two possibilities are the following:

- $P \mapsto x(P)B$
- $k \mapsto x(kB)$

In both cases, $x(P)$ denotes the x-coordinate of the point P, B is a generator, and we also map $\infty \mapsto \infty$.

In Appendix A, we show that the first map never induces a binary functional graph. Therefore, the map $P \mapsto x(P)B$ is not as interesting to study. However, we will shortly see that the map $k \mapsto x(kB)$ gives us binary functional graphs as long as $N = \#E(\mathbb{F}_p) \geq p$ is odd. Therefore, we investigate the map $k \mapsto x(kB), \infty \mapsto \infty$. We shall refer to these graphs as ECFGs (elliptic curve functional graphs) from now on.

We illustrate with an example in Figure 3.

4.1 Theoretical Results

We state one of the most important theoretical results about ECFGs in the next theorem. First, however, we require a lemma.

Lemma 4.1. *If $E(\mathbb{F}_p)$ is cyclic of order N (with N odd), then $x^3 + ax + b \equiv 0 \pmod{p}$ is insoluble.*

Proof. The order of $E(\mathbb{F}_p)$ is $p + 1 + \sum_{x=0}^{p-1} \left(\frac{x^3+ax+b}{p}\right)$. Since N and p are odd, the Legendre sum must also be odd. Therefore, it suffices to show that $\sum_{x=0}^{p-1} \left(\frac{x^3+ax+b}{p}\right)$ has no 0 terms in the sum. Since $\mathbb{F}_p$ is a field and the polynomial $x^3 + ax + b$ has degree 3, there must be at

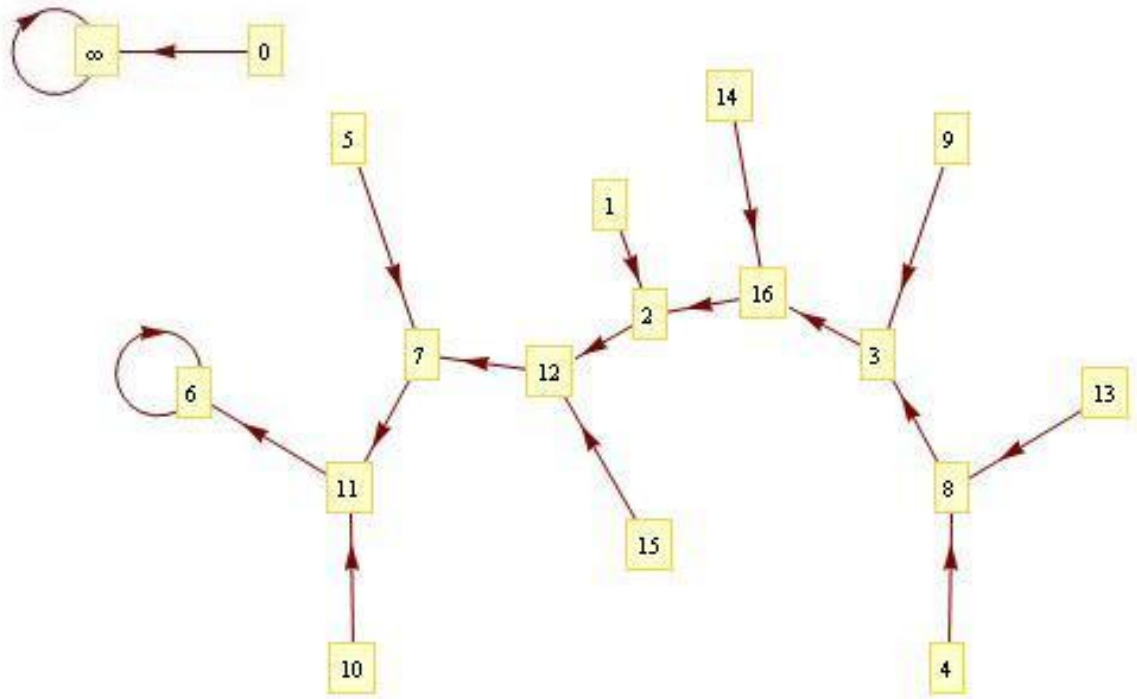

Figure 3: ECFG of $E(\mathbb{F}_{17}) : y^2 = x^3 + x + 3$ with generator $B = (2, 8)$ (this curve has order $N = p = 17$).

most 3 zeros. If there are no zeros, there is nothing to prove. Suppose we have one or three 0 terms in the sum. Then since $p-1$ and $p-3$ are even, a sum of $p-1$ or $p-3$ odd terms must be even, so the sum cannot be odd in this case. Now suppose x^3+ax+b has 2 zeros in $\mathbb{F}_p$. Put $x^3+ax+b=(x-\alpha)(x-\beta)g(x)$. But then $g(x)$ must be of degree one, which forces $f(x)$ to have a third zero, a contradiction. □

Proof. Here is an alternative proof. Let $x_0 \in \mathbb{F}_p$ such that $x_0^3+ax_0+b \equiv 0 \pmod{p}$. Then the point $P=(x_0,0)$ lies on E. Therefore $2P=\infty$. So P generates a subgroup of E of order 2. By Lagrange's theorem, N must be even. □

Now we are ready to prove our theorem.

Theorem 4.2. *Let $E(\mathbb{F}_p)$ be cyclic of order N ($N \geq p$ odd). Let B be a generator, and $G=(V,E)$ be the induced ECFG. Then G is binary, and k and $-k$ map to the same vertex for all $k \in \mathbb{Z}/N\mathbb{Z}$.*

Proof. Let $x_0 \neq \infty \in V$. If $\deg^-(x_0)=0$, then there is nothing to prove. If $\deg^-(x_0) \geq 1$, then there is a $k \in \mathbb{Z}/N\mathbb{Z}$ with $kB=(x_0,y_0)$, so $(N-k)B=-kB=(x_0,-y_0)$. This means that both k and $-k$ map to x_0. These points are distinct as long as $y_0 \not\equiv 0 \pmod{p}$, which holds by Lemma 4.1. Now if there is a third integer r with $rB=(x_0,y_1)$, then we must have $rB \neq \pm kB$ since $N \geq p$ because after computing a multiple of a generator, we reduce the coordinates modulo p, then the exponent modulo N. Thus, $kB, -kB$, and rB are all distinct, so they lie on the vertical line $x=x_0$. This means that $kB+-kB+rB=\infty$, which forces $rB=\infty$. Therefore, rB is not a finite point. So we have shown the result for all finite points.

Now for $x_0=\infty$, by construction, we have $x_0 \mapsto x_0$. Furthermore, we have $0 \mapsto x_0$ since $0B=\infty$. Now suppose there is a third preimage l — i.e., $x(lB)=\infty$. This means that $lB=\infty$, which means that $l \equiv 0 \pmod{N}$, but l must belong to $\mathbb{Z}/N\mathbb{Z}$, so we actually have $l=0$. Therefore, ∞ cannot have a third preimage. □

An example has been shown previously where $N=p$. In Figure 4, we illustrate with an example of an ECFG with $N>p$.

This proof does not work, however, when $N<p$. A counterexample is given by $E(\mathbb{F}_{17})$: $y^2=x^3+2x+6$ with $B=(2,1)$. This curve is cyclic of order 11. The ECFG is shown in Figure 5.

What goes wrong in the proof? Before, reducing the coordinates mod p and then reducing the exponent mod N did no harm since $p \leq N$. However, when $N<p$, this changes the face of the situation. In particular, we run into trouble when we assume that $kB+-kB+rB=\infty$ implies that $rB=\infty$. (In our example, $f^{-1}(2)=\{1,4,7,10\}$. This is not binary since 1 and 10 map to 2, but 4 and 7 would map to 13, which is the same as 2 when working mod 11.)

But is it possible to salvage the situation by letting $k \in \{\infty,0,\ldots,p-1\}$? If we try this, then for $k=N,N+1,\ldots,p-1$, we get an in-degree of 3 for many vertices since a vertex has two preimages in $\{1,\ldots,N-1\}$ which are inverses of each other, but when enlarging

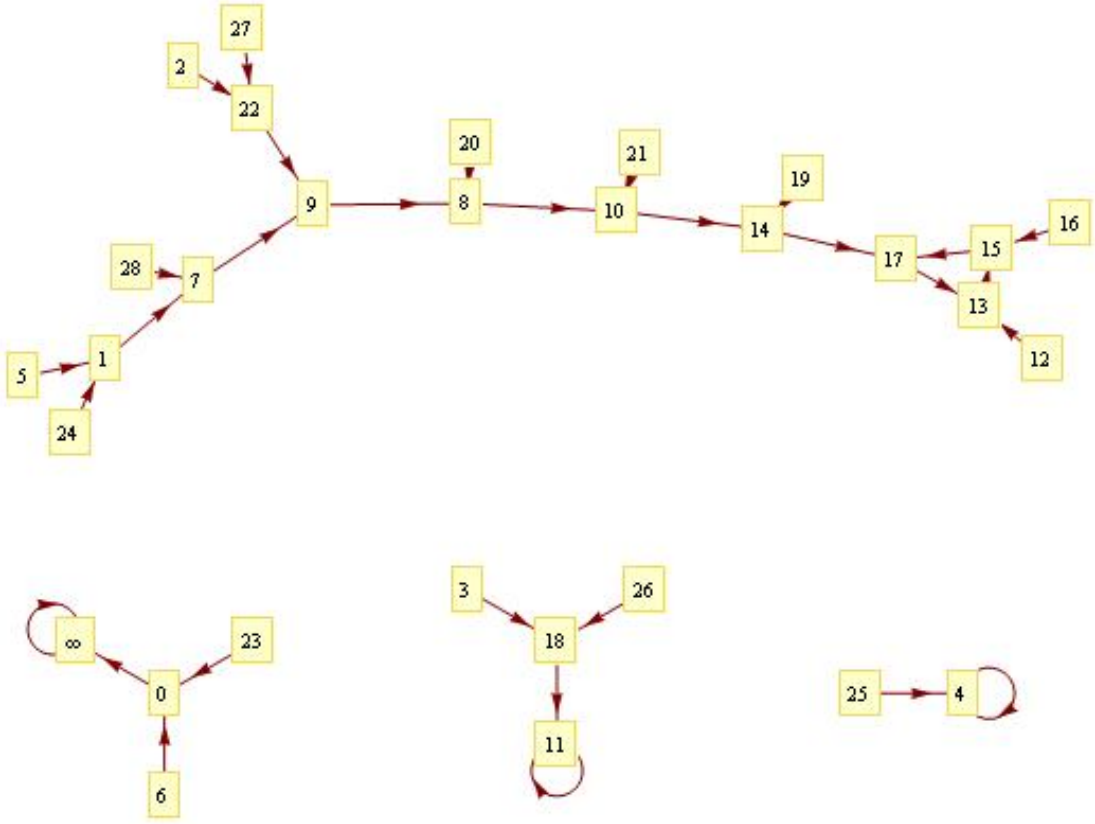

Figure 4: ECFG of $E(\mathbb{F}_{23}) : y^2 = x^3 + x + 4$ with generator $B = (7, 3)$ (this curve has order $N = 29 > p$).

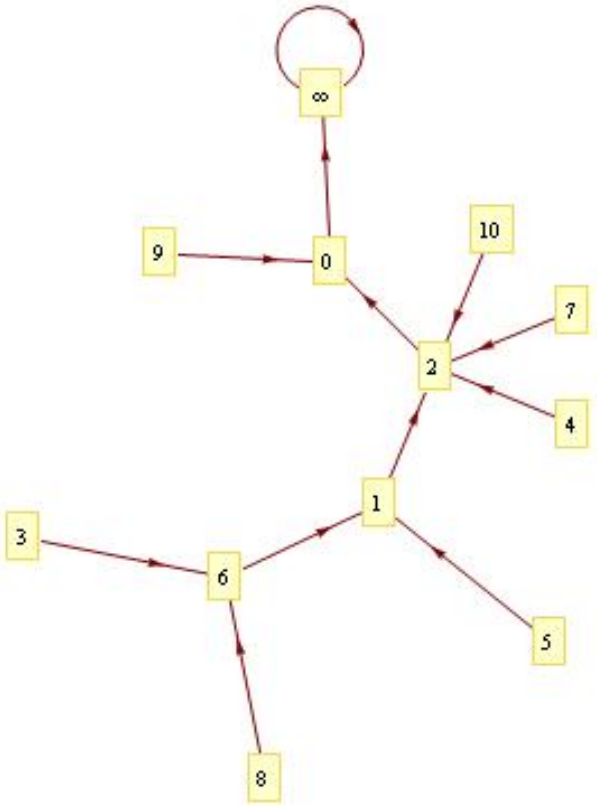

Figure 5: ECFG of $E(\mathbb{F}_{17}) : y^2 = x^3 + 2x + 6$ with generator $B = (2, 1)$ (this curve has order $N = 11 < p$).

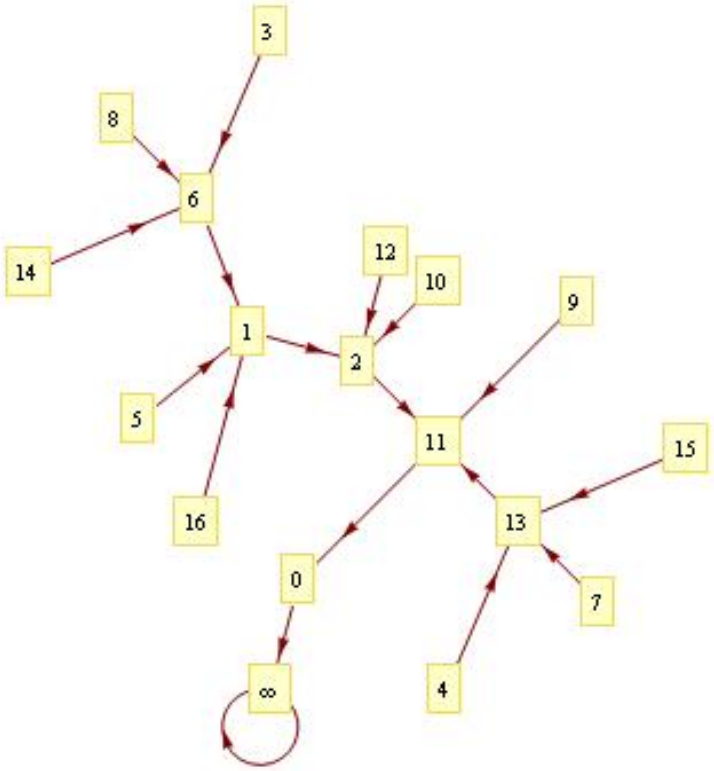

Figure 6: ECFG of $E(\mathbb{F}_{17}) : y^2 = x^3+2x+6$ with generator $B = (2, 1)$ with domain extended to $\mathbb{Z}/p\mathbb{Z} \cup \{\infty\}$ (this curve has order $N = 11 < p$).

the domain and computing kB, we reduce $k \mod N$. N is often large enough to get a third preimage for v. Let us make things concrete and display the functional graph in Figure 6.

In our example, 13 gets mapped to by 4 and 7, but when extending our domain, it also gets mapped to by 15. The next number congruent to 7 (mod 11), however, is 18, which is too large. As in the previous paragraph, we go wrong in this function definition when assuming $kB + -kB + rB = \infty$ implies that $rB = \infty$ since here $rB = \pm kB$ (i.e., $r \equiv \pm k \pmod N$). The problem here is that $r \equiv \pm k \pmod N$ does not imply that $r = \pm k$.

Therefore, in this paper, we consider elliptic curves over $\mathbb{F}_p$ of order $N \geq p$ (which is not to suggest that there is no interest in looking at curves of smaller order, but different techniques must then be applied since they will not result in binary functional graphs).

Proposition 4.3. *If* $\left(\frac{b}{p}\right) = -1$*, then* G *has a component* $\{0, \infty\}$*. If* $\left(\frac{b}{p}\right) = 1$*, then* ∞ *is part of a larger component.*

Proof. $\left(\frac{b}{p}\right) = -1$ means that $(0, y)$ cannot lie on the elliptic curve $E(\mathbb{F}_p)$ for any $y \in \mathbb{F}_p$. Therefore, there is no $k \in \mathbb{Z}/N\mathbb{Z}$ with $k \mapsto 0$. But it is apparent that $0 \mapsto \infty \mapsto \infty$.

If $\left(\frac{b}{p}\right) = 1$, however, then $(0, y)$ does lie on $E(\mathbb{F}_p)$ for some $y \in \mathbb{F}_p$, and since $E(\mathbb{F}_p)$ is generated by a point B, there is some $k \in \mathbb{Z}/n\mathbb{Z}$ such that $kB = (0, y)$. Thus, $k \mapsto 0 \mapsto \infty \mapsto \infty$. □

What happens if $\left(\frac{b}{p}\right) = 0$? This means $b \equiv 0 \pmod p$, or that E is described by $y^2 = x^3 + ax$. If $a = 0$, then this is not an elliptic curve; otherwise, we have the following.

Theorem 4.4. *If* E *is an elliptic curve over* $\mathbb{F}_p$ *with* N *points and* $b = 0$*, then* N *is even.*

Proof. Suppose that $b = 0$. Then the order of $E(\mathbb{F}_p)$ is $p + 1 + \sum_{x=0}^{p-1} \left(\frac{x^3+ax}{p}\right)$. This sum is even if and only if the sum $\sum_{x=1}^{p-1} \left(\frac{x^3+ax}{p}\right)$ is even (notice that $\left(\frac{0^3+a\cdot 0}{p}\right)$ contributes nothing to this sum).

Since $p \geq 5$, $p \equiv 1$ or $3 \pmod 4$. So there are two cases. Notice that $x^3 + ax$ is an odd function. The idea is to pair up the additive inverses in the Legendre symbols.

Case 1: If $p \equiv 1 \pmod 4$, then $\left(\frac{\alpha}{p}\right) + \left(\frac{-\alpha}{p}\right) = 2\left(\frac{\alpha}{p}\right)$, so when pairing up and adding the Legendre symbols for each pair of additive inverses, the resulting sum is even. Therefore, the sum $\sum_{x=1}^{p-1} \left(\frac{x^3+ax}{p}\right)$ is even.

Case 2: If $p \equiv 3 \pmod 4$, then $\left(\frac{\alpha}{p}\right) + \left(\frac{-\alpha}{p}\right) = 0$, so when pairing up and adding the Legendre symbols for each pair of additive inverses, the resulting sum is equal to 0. Therefore, $\sum_{x=1}^{p-1} \left(\frac{x^3+ax}{p}\right) = 0$.

We have shown that the sum

$$\sum_{x=1}^{p-1} \left(\frac{x^3 + ax}{p}\right)$$

is always even, which yields the result. □

Proof. Here is an alternative proof. Suppose $E(\mathbb{F}_p) : y^2 = x^3 + ax$ is an elliptic curve. Then $P = (0, 0) \in E$, and $2P = \infty$, so we have a subgroup of order 2, which, by Lagrange's theorem, implies that N is even. □

Remark. In cryptographic settings, it is typical to ensure that $\#E(\mathbb{F}_p) = q$, where q is a prime (not necessarily $q = p$). Thus, the importance of this result may become clearer when stated in the contrapositive: if $\#E(\mathbb{F}_p) = N$ is odd (perhaps even prime), then $b \not\equiv 0 \pmod{p}$, so we know there are exactly two cases for what sort of component ∞ can lie in. So while it is still of theoretical interest to consider curves of even order, we ignore the case $b \equiv 0 \pmod{p}$ in this paper.

One other thing to count on these graphs is the number of terminal (or equivalently, image) nodes. The reason for the in-degree being 0 could be that for a particular x_0, $x_0^3 + ax_0 + b$ is not a square in $\mathbb{F}_p$, that the base point B does not map to this value in this functional graph, or that $x > p - 1$ since we always reduce the coordinates mod p. However, the fact that B is a generator eliminates the second possibility.

It turns out that for $N \geq p$ odd, there are $(N + 1)/2$ terminal nodes, and therefore $(N + 1)/2$ image nodes. We prove this result first when the order of the curve is exactly p, then for the case where $N > p$. Note that the following proposition is equivalent to stating that when $N = p$, the ECFG of E has exactly $(p + 1)/2$ terminal nodes (and therefore also $(p + 1)/2$ image nodes).

Proposition 4.5. *If $x^3 + ax + b$ defines a cyclic elliptic curve $E(\mathbb{F}_p)$ of order p, then there are exactly $\frac{p+1}{2}$ values of $x \in \mathbb{F}_p$ such that $\left(\frac{x^3+ax+b}{p}\right) = -1$.*

Proof. Since p is odd, we know that $x^3 + ax + b \equiv 0 \pmod{p}$ has no solution by Lemma 4.1. We also know that $\sum_{x=0}^{p-1} \left(\frac{x^3+ax+b}{p}\right) = -1$. Therefore, each term in the sum contributes ± 1 to the sum. There are p terms in the sum and we need one more -1 than 1s in the sum. Let s count the number of 1s. Then $s + 1$ counts the number of -1s. This gives the equation $2s + 1 = p$, which means that $s = (p - 1)/2$. Therefore, $s + 1 = (p + 1)/2$. □

Now we prove the result for $N > p$ in the following theorem.

Theorem 4.6. *If $N > p$ is odd, there are always $(N + 1)/2$ terminal nodes (and therefore $(N + 1)/2$ image nodes).*

Proof. We know that $N = p + 1 + \sum_{x=0}^{p-1} \left(\frac{x^3+ax+b}{p}\right)$. Since $N > p$ is odd, it follows that $N = p+r$ for some r in $\mathbb{Z}_{\geq 2}$. This means $p+1+\sum_{x=0}^{p-1} \left(\frac{x^3+ax+b}{p}\right) = p+r$, or $\sum_{x=0}^{p-1} \left(\frac{x^3+ax+b}{p}\right) = r - 1$. We know that $x^3 + ax + b = 0$ is unsolvable in $\mathbb{F}_p$ by Lemma 4.1 (so there are only ± 1s in the Legendre sum) and that $\sum_{x=0}^{p-1} \left(\frac{x^3+ax+b}{p}\right)$ is positive. Therefore, there are at least $r - 1$ 1s in the Legendre sum, so we choose $r - 1$ (which is odd since N and p are odd, which means that r must be even) squares in $\mathbb{F}_p$, and let s count the number of -1s. Therefore,

$s + (r-1)$ counts the number of 1s. Then $s + (s + (r-1)) = 2s + (r-1) = p$. This gives $s = (p-(r-1))/2$. By the discussion preceding Proposition 4.5, we need to add the number of integers modulo N greater than $p-1$ to s to count the number of terminal nodes. This number is $(N-1) - (p-1) = N - p$. So the following computation yields the result.

$$\begin{aligned} s + (N-p) &= (p-(r-1))/2 + 2(N-p)/2 \\ &= (p-(r-1)+2(N-p))/2 \\ &= (p-(r-1)+2r)/2 \\ &= (p+r+1)/2 \\ &= (N+1)/2. \end{aligned}$$

□

In fact, we can prove this more generally for any binary functional graph. See Theorem 2.7 for the details.

One question of theoretical interest is the following: can we also look at functional graphs of proper subgroups of a given elliptic curve? The answer is a resounding no.

We know we want to look at curves with $N \geq p$, but if we want a subgroup, we want a proper divisor r of N such that $r \geq p$. However, Hasse's Theorem tells us that an elliptic curve over $\mathbb{F}_p$ has $N \leq p + 1 + 2\sqrt{p}$ points [9, p. 97]. Suppose that $r \mid N, N > r \geq p$. Then $N \geq 2r \geq 2p$. But for $p > 5$,

$$\begin{aligned} &\sqrt{p} - 2 > \frac{1}{\sqrt{p}} \\ &\sqrt{p}(\sqrt{p} - 2) = p - 2\sqrt{p} > 1 \\ &p > 1 + 2\sqrt{p} \\ &2p > p + 1 + 2\sqrt{p} \geq N \\ &N \geq 2r \geq 2p > N. \end{aligned}$$

This shows that considering such subgroups is fruitless.

4.2 Computational Results

One fact is that B and $-B$ generate the same group. But they also induce the same ECFG. This can be seen as follows: $kB = (x, y)$ if and only if $k(-B) = -kB = (x, -y)$, so $k \mapsto x$ in the graph with generator B if and only if $-k \mapsto x$ in the graph with generator $-B$. By Theorem 4.2, in both graphs, $k \mapsto x$ and $-k \mapsto x$ for all $x \neq \infty$. The only preimages of ∞ are 0 and ∞ regardless of the generator (in fact, regardless of the curve). This fact allows us to analyze only half as many graphs as we would otherwise have to.

There is another theoretical idea we can use to speed up computation. The largest possible cycle is of size $N - 1$ since there is always a fixed point at ∞ and we also have

$0 \mapsto \infty$. By Theorem 2.6, every component in a functional graph contains exactly one cycle, so the number of cycles is equal to the number of components. Therefore, if C_i denotes the number of i-cycles, and r denotes the number of components, then $\sum_{i=1}^{N-1} C_i = r$. It is much quicker to use a cycle-finding algorithm to compute the number of cycles than it is to construct a graph and use an algorithm such as depth-first search to partition the graph into its components.

In order to gather statistics, the author wrote three computer programs. The first is a Java program that takes in a prime N and finds all elliptic curves of order N over fields $\mathbb{F}_p$ with $p \leq N$. This uses an array of prime numbers, the bounds on N implied by Hasse's Theorem, and a simple search (for each curve, it also finds the generators for the curve by iterating over $\mathbb{F}_p$ and computing square roots). It then outputs the values of a, b, p and B (the generator) to a file for each curve-generator pair. The second program, written in C, reads in this file, and for each curve-generator pair, computes the number of terminal nodes, image nodes, fixed points, two-cycles, three-cycles, five-cycles, components (or equivalently, cycles), cyclic nodes, tail nodes, as well as the average cycle length, weighted average cycle length, average tail length, the maximum cycle length, and the maximum tail length. These computations are performed by using Brent's algorithm for finding cycles in iterated functions and by computing Jacobi symbols (a slightly faster method of computing Legendre symbols). These statistics for each ECFG are then output to a file.

The third program is a Java program that reads in the file output from the second program and for each statistic, prints out the expected value, mean and variation for each statistic (this is all within a particular prime N).

One problem with gathering statistics for all ECFGs of some fixed order N is that there is an astronomical number of graphs to create. With the classical discrete log problem, there are $p-1$ different graphs (and only $\varphi\left(\frac{p-1}{2}\right)$ different binary graphs). However, in the ECFG case, for a fixed N, p varies when finding the prime fields $\mathbb{F}_p$, and for each prime field, there seem to be on the order of N different elliptic curves. Furthermore, for each elliptic curve, there are $\varphi(N)$ graphs to create (which, when N is prime, is equal to $N-1$). Even at a quite small value of $N = 83$, there are already 34,112 graphs. For $N = 211$, there are 549,780 graphs. We know that a given curve with generators B and $-B$ induce the same ECFG, so this cuts down on the computation we have to do drastically, but unless we can find deeper results about when various generators induce isomorphic ECFGs, it seems highly unlikely that we can record very many asymptotic statistics for these graphs. In any case, some results are tabulated below.

$N = 167$ (208,496 graphs)			
Statistic	Expected	Observed	Variance
Terminal Nodes	84	84	0
Image Nodes	84	84	0
Fixed Points	1	1.992422	0.976786
2-cycles	0.5	0.475012	0.475874
3-cycles	$0.\overline{3}$	0.318932	0.312430
5-cycles	0.2	0.182804	0.177953
Components	3.197163	4.085517	1.972782
Average Cycle Length	4.768229	3.852527	4.191673
Cyclic Nodes	15.244808	15.126151	57.433792
Tail Nodes	152.755192	152.873849	57.433792
Weighted Average Cycle Length	8.122404	7.014287	25.312406
Average Tail Length	8.122404	7.125777	9.990484
Max Cycle Length	10.142100	9.076117	31.394953
Max Tail Length	24.133765	18.032260	34.678483

$N = 211$ (549,780 graphs)			
Statistic	Expected	Observed	Variance
Terminal Nodes	106	106	0
Image Nodes	106	106	0
Fixed Points	1	1.994940	0.976805
2-cycles	0.5	0.502165	0.493947
3-cycles	$0.\overline{3}$	0.316221	0.315596
5-cycles	0.2	0.189228	0.185972
Components	3.313475	4.238230	2.122886
Average Cycle Length	5.205572	4.279185	5.455133
Cyclic Nodes	17.248529	17.339449	74.948891
Tail Nodes	194.751471	194.660551	74.948891
Weighted Average Cycle Length	9.124265	8.161955	33.787259
Average Tail Length	9.124265	8.109176	13.359563
Max Cycle Length	11.393081	10.445054	41.073544
Max Tail Length	26.911509	20.660497	46.404844

$N = 227$ (292,896 graphs)			
Statistic	Expected	Observed	Variance
Terminal Nodes	114	114	0
Image Nodes	114	114	0
Fixed Points	1	1.996238	1.012967
2-cycles	0.5	0.501106	0.503713
3-cycles	$0.\bar{3}$	0.340052	0.340567
5-cycles	0.2	0.176861	0.171720
Components	3.349854	4.303398	2.220580
Average Cycle Length	5.350868	4.449070	5.928930
Cyclic Nodes	17.924628	18.176977	80.148224
Tail Nodes	210.075372	209.823023	80.148224
Weighted Average Cycle Length	9.462314	8.647097	38.137383
Average Tail Length	9.462314	8.453528	14.086671
Max Cycle Length	11.815189	10.967579	45.399227
Max Tail Length	27.848781	21.570455	48.971410

Although we cannot compute these statistics over all ECFGs for N large, we can try to pick a random selection of ECFGs for larger values of N and see how the limited statistics match up to the expected values. The author modified the program to find elliptic curves of a given order to find randomly chosen curve-generator pairs. Displayed below are limited statistics for somewhat larger primes.

$N = 419$ (100,000 graphs)			
Statistic	Expected	Observed	Variance
Terminal Nodes	210	210	0
Image Nodes	210	210	0
Fixed Points	1	1.994600	0.995091
2-cycles	0.5	0.493040	0.494272
3-cycles	$0.\bar{3}$	0.331530	0.328578
5-cycles	0.2	0.193690	0.190394
Components	3.655309	4.601670	2.433203
Average Cycle Length	6.753273	5.718504	10.352899
Cyclic Nodes	24.685297	24.939260	157.386731
Tail Nodes	395.314703	395.060740	157.386731
Weighted Average Cycle Length	12.842648	12.055864	75.266458
Average Tail Length	12.842648	11.880105	28.272666
Max Cycle Length	16.036068	15.247660	89.399045
Max Tail Length	37.221044	30.789860	98.998861

$N = 997$ (15,000 graphs)			
Statistic	Expected	Observed	Variance
Terminal Nodes	499	499	0
Image Nodes	499	499	0
Fixed Points	1	2.001400	1.012065
2-cycles	0.5	0.500400	0.491600
3-cycles	$0.\bar{3}$	0.338333	0.347864
5-cycles	0.2	0.191933	0.194162
Components	4.088058	5.056400	2.904686
Average Cycle Length	9.440575	8.060594	21.242696
Cyclic Nodes	38.593620	38.610733	400.831871
Tail Nodes	959.406380	959.389267	400.831871
Weighted Average Cycle Length	19.796810	18.692177	188.266545
Average Tail Length	19.796810	18.772267	68.699347
Max Cycle Length	24.719434	23.758533	222.702094
Max Tail Length	56.502049	49.439933	243.368259

$N = 1103$ (10,000 graphs)			
Statistic	Expected	Observed	Variance
Terminal Nodes	552	552	0
Image Nodes	552	552	0
Fixed Points	1	1.998900	1.006699
2-cycles	0.5	0.494000	0.489564
3-cycles	$0.\bar{3}$	0.336100	0.332137
5-cycles	0.2	0.202100	0.201256
Components	4.138529	5.106400	2.912079
Average Cycle Length	9.820696	8.433923	23.836005
Cyclic Nodes	40.643236	40.682000	437.580876
Tail Nodes	1063.356764	1063.318000	437.580876
Weighted Average Cycle Length	20.821618	19.902411	215.481783
Average Tail Length	20.821618	19.803899	75.592366
Max Cycle Length	25.999068	25.149100	253.147869
Max Tail Length	59.343417	52.370000	269.637500

Even though N is fairly small here, most of the statistics are already converging to the expected values for random binary functional graphs. There are a few statistics, however, that deviate from our expectations. For example, we would expect one fixed point on average, but we get about two. This is easy to explain, however, since there is always a fixed point at ∞. Another statistic that seems to be off is the number of components. It appears that the number of components is about one more than what we would expect. It is likely that there is a connection with the fact that $0 \mapsto \infty \mapsto \infty$. We know when $\left(\frac{b}{p}\right) = -1$, $\{0, \infty\}$

is its own component. This happens with probability about 1/2. But even if b is a square mod p, each of 0's two preimages has a probability of about 1/2 (actually, $N/2p$) of having a preimage. In other words, if $r \mapsto 0$, then r has a preimage only if $r^3 + ra + b$ is a square mod p. To go backwards i iterations, the probability of the preimage being nonempty is about $\left(\frac{N}{2p}\right)^i \approx \frac{1}{2^i}$. Therefore, it is quite likely that the component containing ∞ is fairly small. This suggests that there should be about one more component than we would expect from a random binary functional graph.

The other three statistics that seem off are average cycle length, maximum cycle length and maximum tail length. Average cycle length tends to be off by about 0.9. However, since the average cycle length can be computed by cyclic nodes / components, this deviation is equivalent to the number of components deviating (notice that the number of cyclic nodes matches up quite accurately compared to the expected value).

One possible explanation of why the maximum cycle length is off is because of the fixed point at ∞. If we consider the binary functional graph by removing ∞'s component, we get a binary graph with fewer vertices, and the maximum cycle length might match up better to the expected value for this subgraph. An explanation of why the maximum tail length is off seems more elusive, however. It is consistently under the expected value by about 6 or 7. This suggests that discrete exponentiation on elliptic curves is slightly less secure than one might hope for two reasons. First, it suggests that there's less of a chance for a longer tail, which could prove detrimental for certain pseudorandom number generators [1]. Secondly, it seems to suggest that there is some extra structure in the maps we've been considering, which could lead to an attack on the discrete logarithm problem.

5 Conclusions and Future Work

The hardness of the discrete logarithm problem on elliptic curves has offered an advance in cryptography, and there is computational evidence that suggests that it is even more secure than classical techniques. It is assumed to be secure because of the belief that discrete exponentiation behaves like a random map. If, however, there is some structure imposed by discrete exponentiation for elliptic curves, we want to know the details so that we can either abandon relying on the discrete logarithm problem or else ensure that certain conditions are met so that the discrete logarithm problem remains hard.

Most of the statistics collected for ECFGs do match up with what would be expected of a random map. There are, however, a couple of areas where there does seem to be some extra structure imposed by discrete exponentiation. One area is the maximum tail length. This seems to be consistently lower than the expected theoretical value of a random binary map by about 6 or 7. The deviation seems too extreme to be caused just by the small component containing ∞. Indeed, Lindle [7] seemed to be stuck with the same anomaly when investigating binary functional graphs in the discrete logarithm problem modulo p. Perhaps this information could be used to devise an attack on the discrete logarithm problem.

More significantly, however, most of the other deviations in the statistics seem to be

caused by the small component containing the cycle at infinity. In addition to applications of the hardness of the discrete logarithm problem on elliptic curves to Diffie-Hellman and ElGamal, there are also random number generators that exploit the discrete log's hardness. For example, in [1], a map is specified by $k \mapsto t(x(kB))$, where $t(x)$ processes the x-coordinate in some way. One thing that we want to ensure when using this random number generator is that we don't quickly enter a short cycle. Although for a curve with a large number of points, it is relatively unlikely that an initial seed k belongs to a small component, it can happen. If it does, then the random number generator will no longer be secure. One small component in particular is the one containing ∞. We can iterate our map to see if we get a fixed point (which would give rise to an eventually constant sequence). This weakness doesn't seem to be as exploitable in Diffie-Hellman, however, because knowing that there is an incoming edge to x_0, where $P = (x_0, y_0)$ (we can also figure this out by computing a Legendre symbol) doesn't seem to give us extra information about how to solve the discrete logarithm $kB = P$.

There are a number of ways in which this project can be continued. One quite specific continuation is investigating why the maximum cycle and tail length deviate from our expectation, particularly maximum tail length.

Another way to continue this project would be simply to collect more data and perform more thorough statistical analyses. Yet another idea would be to investigate curves of order $N < p$, which I have chosen to neglect since these curves do not induce binary functional graphs. Similarly, curves of even order could be investigated, perhaps even supersingular curves. Other related functions such as $P \mapsto x(P)B$ could also be considered (these do not induce binary functional graphs).

I have also chosen to ignore elliptic curves over non-prime fields $\mathbb{F}_q$. One reason for ignoring other curves is that it is simpler to implement the arithmetic in the prime case. I have also ignored curves over $\mathbb{F}_2$ and $\mathbb{F}_3$. There is not a huge number of curves over these fields, so this is not a huge limitation; however, elliptic curves over $\mathbb{F}_{2^k}$ in particular might be worth investigating since these binary fields are often used in cryptography (and arithmetic over $\mathbb{F}_{2^k}$ would be easier to implement than arithmetic in $\mathbb{F}_{p^k}$ for arbitrary p).

A The Map $P \mapsto x(P)B$

In this appendix, we prove a couple of properties of the map mentioned earlier in this paper: $P \mapsto x(P)B, \infty \mapsto \infty$, where P is a point on an elliptic curve $E(\mathbb{F}_p)$, B is a generator, and $x(P)$ denotes the x-coordinate of the point P.

Proposition A.1. *Let $E(\mathbb{F}_p) : y^2 = x^3 + ax + b$ be an elliptic curve with N points. Then $P \mapsto x(P)B$ does not induce a binary functional graph (regardless of the generator B).*

Proof. If we have an odd number of points on the curve, then the graph cannot be binary by the remarks that follow Theorem 2.7.

If we have an even number of points, then since $E(\mathbb{F}_p)$ is abelian, there is a subgroup of order 2, and the generator of this subgroup has to have $y-$coordinate equal to 0. Let this

generator be denoted by $Q = (x, 0)$. Now let $P = x(Q)B = xB$. If there were a second preimage, then we would have $P = x(R)B$, and thus $x(R) = x(Q)$. But there can only be one point on the curve with x-coordinate $x(Q)$ since $Q = -Q$.

Therefore, the map $P \mapsto x(P)B$ never induces a binary functional graph. □

Remark. Many of the vertices in these graphs do have two preimages, however (and ∞ can have three preimages in some cases). Therefore, these do not produce m-ary functional graphs for any m.

Proposition A.2. *$\{\infty\}$ is its own component if and only if $\left(\frac{b}{p}\right) = -1$.*

Proof. Once we have specified our generator B, we can identify any point on the curve uniquely by an integer modulo N. Thus, ∞ has a preimage if and only if we can solve $0 = x(P)$ for some $P \in E$, which is solvable if and only if $\left(\frac{b}{p}\right) = 1$. □

References

[1] Brown, Daniel R. L. and Gjøsteen, Kristian, A Security Analysis of the NIST SP 800-90 Elliptic Curve Random Number Generator. In *CRYPTO 2007, LNCS 4622*, pages 466-481. 2007.

[2] Cloutier, Daniel. R., Mapping the Discrete Logarithm. Senior thesis, Rose-Hulman Institute of Technology, 2005.

[3] Cloutier, Daniel R. and Holden, Joshua, Mapping the Discrete Logarithm. 2006. http://xxx.lanl.gov/abs/math.NT/0605024

[4] P. Flajolet and A. Odlyzko, Random Mapping Statistics. In *Advanced in Cryptology—EUROCRYPT '89 (Houthalen, 1989)*, volume 434 of *Lecture Notes in Comput. Sci.*, pages 329-354. Springer, Berlin, 1990.

[5] Hoffman, Andrew, Statistical Investigation of Structure in the Discrete Logarithm, Rose-Hulman Institute of Technology REU Report, 2009.

[6] Koblitz, Neal, *A Course in Number Theory and Cryptography*, Springer-Verlag, New York, NY, 2nd. Ed., 1994.

[7] Lindle, Nathan W., A Statistical Look at Maps of the Discrete Logarithm. Senior thesis, Rose-Hulman Institute of Technology, 2008.

[8] Rosen, Kenneth H., *Discrete Mathematics and Its Applications*, McGraw-Hill, Boston, MA, 5th Ed., 2003.

[9] Washington, Lawrence C., *Elliptic Curves: Number Theory and Cryptography*, Chapman & Hall, Boca Raton, FL, 2nd. Ed., 2008.