AN ABSTRACT OF THE THESIS OF

Max F. Brugger for the degree of Honors Baccalaureate of Science in Mathematics presented on May 29, 2008. Title: Exploring the Discrete Logarithm through Ternary Functional Graphs.

Abstract approved:

Paul Cull

Encryption is essential to the security of transactions and communications, but the algorithms on which they rely might not be as secure as we all assume. In this paper, we investigate the randomness of the discrete exponentiation function used frequently in encryption. We show how we used exponential generating functions to gain theoretical data for mapping statistics in ternary functional graphs. Then, we compare mapping statistics of discrete exponentiation functional graphs, for a range of primes, with mapping statistics of the respective ternary functional graphs.

Key Words: Discrete Logarithm, Encryption, Random Mappings

Corresponding E-mail Address: bruggerm@onid.orst.edu

Exploring the Discrete Logarithm with Random Ternary Graphs

by

Max Francis Josef Brugger

A PROJECT

submitted to

Oregon State University

University Honors College

in partial fulfillment of

the requirements for the

degree of

Honors Baccalaureate of Science in Mathematics

Presented May 29, 2008

Commencement June 2008

Acknowledgements

I would like to take this opportunity to thank my first advisor, Dr. Joshua Holden, for first inspiring my interest in this project and encouraging me through the tough early months. I am grateful for his patience and insights. I would also like to thank my second advisor, Dr. Paul Cull, for his tireless efforts to challenge me to write more thorough and professional work. I would like to thank both for keeping me motivated during the academic year in balancing research and class work. Lastly, I would like to thank my parents for their countless sacrifices to help me achieve what I have accomplished today.

Honors Baccalaureate of Science in Mathematics project of Max F. Brugger presented on May 29, 2008.

APPROVED:

Mentor, representing Mathematics

Mentor, representing Electrical Engineering and Computer Science

---

Committee Member, representing Mathematics

Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any reader upon request.

Max F. Brugger, Author

Contents

List of Figures

1 Introduction

Encryption is a battlefield. The cryptographic protocols that we use today have evolved since the introduction of DES thirty-five years ago both to accomodate the dramatic increase in public need and to balk the attacks of those who would like to render the private communications of bank transfers and e-mail conversations insecure. Despite all the changes and developments that the field has seen the basic principles remain the same — secure public-key encryption requires a *one-way function*, described in Diffie and Hellman's seminal paper, "New Directions in Cryptography" [3] as a function with a property that "for any argument x in the domain of f, it is easy to compute the corresponding value $f(x)$, yet, for almost all y in the range of f, it is computationally infeasible to solve the equation $y = f(x)$ for any suitable argument x" ([3], 650).

In the same paper, Diffie and Hellman introduce a technique that "makes use of the apparent difficulty of computing logarithms over a finite field $GF(p)$ with a prime number p of elements" ([3], 649). A logarithm over a finite field, also called a *discrete logarithm*, is the inverse of the *discrete exponentiation function* described by

$$f(x) = \gamma^x \mod p, \qquad \text{for } 1 \leq x \leq p-1,$$

where γ is usually a primitive element of $GF(p)$ and x is referred to as the logarithm of y to the base γ, mod p. The corresponding inverse equation reads

$$x = \log_\gamma y \mod p, \qquad \text{for } 1 \leq y \leq p-1.$$

In choosing the discrete logarithm, Diffie and Hellman refer to the classical understanding of its difficulty, but let its status as a hard problem suffice as justification for its use as the secure backbone to their cryptographic algorithm. To this day, the

discrete logarithm is used in some of the most important cryptographic algorithms. In "A Method of Obtaining Digital Signatures and Public-Key Cryptosystems" [5] Rivest, Shamir, and Adleman introduce the RSA encryption protocol, which relies on a version of the discrete logarithm. We will demonstrate this with a quick example in Section 2. In [5], the authors make several attempts at justifying their system, stating that "since no techniques exist to *prove* that an encryption scheme is secure, the only test available is to see whether anyone can think of a way to break it" ([5], 125). They then show that "all the obvious approaches for breaking our system are at least as difficult as factoring n," ([5], 125), another problem known for its difficulty. They then consider ways to determine the private key from knowledge of the public key.

Since the discrete logarithm is widely used, and because its security remains unproven, we attempt a unique approach to its investigation. In this paper we compare properties of functional graphs generated by the discrete logarithm to properties of all functional graphs. A *functional graph* is a directed graph with an associated function, $f(x)$. The nodes of the graph represent the elements of the function's domain. For each x, there is a directed edge (an arrow) from the node representing x to the node representing $f(x)$. The out-degree of a node a is the number of elements b such that $f(a) = b$. Since we study only functional graphs, the out-degree of each node is exactly 1. The in-degree of a node b is defined as the number of elements a such that $b = f(a)$.

Dan Cloutier [2] has compared the set of unary and binary discrete exponentiation functional graphs (DEFGs) to the set of all unary and binary functional graphs. Thus for nine primes p, we compared statistics of the set of ternary DEFGs over $GF(p)$ with statistics of the set of all ternary functional graphs. We obtained observed values for the ternary DEFGs by running one of the C++ programs described in [2] and obtained theoretical values for the functional graphs by using generating function

methods on the ternary functional graphs. In a *ternary functional graph* each node has in-degree equal to exactly 0 or 3.

We compare ternary DEFGs to the average behavior of ternary functional graphs in order to ensure that the ternary DEFGs are secure. If the discrete logarithm has a property that sets it apart from the set of all other functions, for example if its cycles can be expected to be longer than the cycles of a function picked at random, then this property might allow for a rapid computation of the inverse, and might allow an attack on a cryptographic protocol.

2 Motivation

In 2005, Daniel Cloutier worked on a similar project involving unary and binary functional graphs [2]. For some m, all DEFGs are m-ary, meaning that each node in a graph has in-degree equal to exactly m or 0. He wrote two C++ programs; each constructed a set of DEFGs. One program cycled through each element γ in $GF(p)$ and generated the corresponding DEFG. The other constructed the subset of binary DEFGs. He then used each program to measure the observed statistics of the DEFGs. From the methods described in [4], he was able to construct theoretical data for the set of all binary functional graphs and compare those with the observed data that he generated from his programs.

We continue Cloutier's work by comparing the observed data from the set of ternary DEFGs (by applying one of his programs) with the theoretical data for the set of ternary functional graphs (using the techniques in [1]). The generating function methods developed to generate the theoretical data are interesting and relate to cryptography in ways that will be demonstrated here.

2.1 Application to Cryptography

In arguing for our project's relevance to cryptography, we describe the use of ternary functional graphs in cryptographic protocols like Diffie-Hellman Key Agreement and RSA Encryption. For the reader without background in these protocols, we briefly introduce some details. The (experienced) reader may move ahead to Section 2.1.3.

2.1.1 Diffie-Hellman Key Agreement

This cryptographic protocol is used to generate a private key. The private key can be used by two parties to encrypt messages, send them to each other safely, and to easily decrypt the received messages. In describing a protocol, the two parties are

usually called Alice and Bob.

Alice comes up with some number α and Bob comes up with some number β and they keep them private from everyone, including each other. They publish two numbers publicly: γ and p. They then follow the processes below:

Alice		Process		Bob	Public
α				β	p, γ
α	$\rightarrow$	$r \equiv \gamma^{\alpha} \mod p$	$\rightarrow$	β, r	p, γ, r
α, s	$\leftarrow$	$s \equiv \gamma^{\beta} \mod p$	$\leftarrow$	β, r	p, γ, r, s

Alice and Bob then compute $s^{\alpha} \mod p$ and $r^{\beta} \mod p$ respectively, yielding in both cases $\gamma^{\alpha\beta} \mod p$. This is their private key, which they may use to encrypt their data via another cryptographic protocol.

As was mentioned in the introduction, Diffie-Hellman relies on the seemingly difficult problem of computing discrete logarithms for its security, since it is hard to find $\gamma^{\alpha\beta}$ given the public information r, s, γ, p.

2.1.2 RSA Encryption

The RSA method is fully elaborated in [5]. To show the relation of the RSA method to the computation of the discrete logarithm, we quote the encryption and decryptions steps here. Once one's message is prepared (see [5]), one can encrypt the message by raising it to the e^{th} power modulo n. The encryption and decryption algorithms E and D are:

$$C \equiv E(M) \equiv M^e \mod n \quad \text{for a message } M.$$
$$M \equiv D(C) \equiv C^d \mod n \quad \text{for a ciphertext } C.$$

Thus, solving for e given C, M, and n is analogous to solving for x in the discrete logarithm given y, γ, and p.

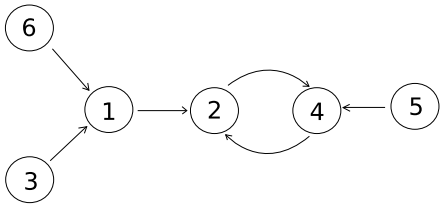

Figure 1: The DEFG corresponding to $p = 7$ and $\gamma = 2$.
It contains the subgroup $G := \{2^1 = 2,\ 2^2 = 4,\ 2^3 = 1\}$.

2.1.3 Connection to Ternary Functional Graphs

Diffie-Hellman Key Agreement is one of the most widely used, researched, and understood cryptographic protocols. As first presented in [3] it allows two parties to agree on an encryption key for use in private communication through completely public channels. To accomplish this, Diffie-Hellman uses the discrete logarithm function. However, instead of always using a primitive element, one sometimes makes use of a Sophie Germain prime, q, to calculate a "safe prime" $p = 2q + 1$. The base, γ, is then sometimes chosen to generate the subgroup of order $(p-1)/2$. This necessarily generates a binary DEFG.

Consider the simple example in Figure 1. Let $p = 7$ and $\gamma = 2$, so that $f(x) = 2^x \mod 7$. The group generated by γ, $G = \{2^1 = 2, 2^2 = 4, 2^3 = 1\}$ has order $(p-1)/2 = 3$, and consists of elements with in-degree not equal to 0, since 1, 2, and 3 map to them. Since order$(G) = 3$, there exist three other elements which map to elements of G. Since we have a functional graph, and every node maps to exactly one other node, these elements have in-degree zero and thus the elements of G have in-degree 2. So we have a binary functional graph.

Depending on the available processor power, it can be a difficult problem to generate large primes and primitive roots of those primes. To overcome this, some cryptographic protocols employ probabilistic pseudo-prime generation and probabilistic base generation. This allows the possibility of using a ternary DEFG in a cryptographic protocol like Diffie-Hellman Key Agreement. Since Diffie-Hellman does not

encrypt data that must later be decrypted (it merely generates a key) there is no problem with the existence of three pre-images to each image in a ternary graph. In practical use, no one would use a ternary graph in RSA, since each message block could be decrypted (correctly, even) in three different ways.

Much work has been done in the area of random combinatorial structures, but it would seem that our results regarding the theoretical statistics on ternary functional graphs are original.

3 Background

We are interested in examining certain statistics of ternary DEFGs and ternary functional graphs. The definitions of these statistics rely on graph theory concepts that we will define with a fair amount of rigor. We recall from the introduction that a function $f(x)$ on a finite set naturally defines a graph called the functional graph. The vertices are exactly the elements of the set and there exists a directed edge from x to y if and only if $y = f(x)$. A graph is a functional graph if and only if each vertex has out-degree equal to exactly one.

3.1 Definitions

Definition 1. *Let y be a node in a functional graph. It is considered to be an* image node *if there exists at least one x such that $y = f(x)$.*

Definition 2. *Let y be a node in a functional graph. If there exists no x such that $y = f(x)$, then it is a* terminal node.

The following definitions come from [4]. For each of the next definitions, we consider a function $f(x)$ and a directed graph whose nodes are the elements $[1 \dots n]$ and whose edges are the ordered pairs $\langle x, f(x) \rangle$, for all $x \in [1 \dots n]$. If we start from any x_0 and keep iterating f, we consider the sequence $x_1 = f(x_0)$, $x_2 = f(x_1)$, ..., we will find, before n iterations, a value x_j equal to one of x_0, x_1, ..., x_{j-1}. In graphical terms, starting from any x_0, the iteration structure of f is described by a path that connects to a cycle.

Definition 3. *The* cycle length of x_0 *is equal to the length of the cycle.*

Definition 4. *The length of the path (measured by the number of edges) is called the* tail length of x_0.

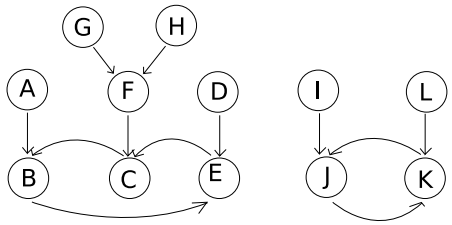

Figure 2: An example of a binary functional graph.
It contains two connected components. Nodes B, C, and E form a cycle of length 3 and nodes J and K form another cycle of length two. Nodes A, D, F, I, and L see a tail of length 1 and nodes G and H see a tail of length 2.

Definition 5. *A* cyclic node *is a node contained in a cycle. A* tail node *is a node that is not contained in a cycle.*

Definition 6. *A* component *is a set of all connected cyclic nodes and tail nodes.*

Definition 7. *The* total cycle length as seen from a node *is calculated by adding up the cycle length of each node. We consider each node to "see" a cycle and to record that cycle's length and then proceed to take the sum over all of the cycle lengths seen by the nodes.*

For example, consider a functional graph consisting of two components (as in Figure 2) of size eight and four. Within the component of size eight there is a cycle of length three and within the component of size four there is a cycle of length two, so the cycle length as seen from a node is $8 \cdot 3 + 4 \cdot 2 = 30$.

Definition 8. *The* total tail length as seen from a node *is calculated by adding up the tail length of each node.*

For example, consider a functional graph consisting of two components (see Figure 2) of size eight and four. Nodes A, D, F, I and L see a tail of length 1 and nodes G and H see a tail of length of 2, so the tail length as seen from a node is $5 \cdot 1 + 3 \cdot 2 = 11$. The other nodes see zero tail length.

3.2 Choosing our Primes

Consider Definition 6. Since we were using the program from [2] to obtain observed statistics for ternary DEFGs, we wanted to ensure we were using primes for which there existed bases γ that generated ternary DEFGs. We used the following theorem to guarantee this:

Theorem 1 ([2], Theorem 1)**.** *Let m be any positive integer that divides $p-1$. Then there are $\phi\left(\frac{p-1}{m}\right)$ m-ary functional graphs (where ϕ is the Euler Phi Function) produced by the map $x \to \gamma^x \mod p$ for a given γ and p. Furthermore, if r is any primitive root modulo p, and $g \equiv r^\alpha \mod p$, then the values of g that produce an m-ary graph are precisely those for which $\gcd(a, p-1) = m$.*

The second part of Theorem 1 can be explained to say that if we have a primitive root modulo p, denoted r, and we want an m-ary graph, then it suffices to find some positive integer $a \in [1, p-1]$ such that $\gcd(a,\, p-1) = m$. Then, the theorem states, $r^a \mod p$ will generate an m-ary graph.

So we chose nine primes of form $p = 3k+1$, since the theorem states that there will exist $\phi\left(\frac{p-1}{3}\right) = \phi(k)$ ternary DEFGs, guaranteeing a nonzero number of ternary DEFGs.

4 Ternary Functional Graphs

4.1 Exponential Generating Functions

> A generating function is a clothesline on which we hang up a sequence of numbers for display. [Herbert S. Wilf, *Generatingfunctionology*, [8]]

In our research, we use exponential generating functions to count certain features of ternary functional graphs. Before we give the precise definition, it is useful to impart an imprecise notion of what these things are used for. Exponential generating functions, or EGFs, are formal Taylor series derived so that the coefficient of the n^{th} degree term is equal to the number of things we have set up the EGF to count, up to a factor of $n!$. If we want our EGF to count the number of components in a ternary functional graph, then the number of components in all the ternary functional graphs of n nodes is given in the coefficient of the n^{th} term. The methods that we use to derive these EGFs are fun and beautiful; some basic examples will be given in this section, and some more complicated applications will be given in the next.

Definition 9. *Let $f(x)$ be an* exponential generating function. *Then $f(x)$ may be expressed as*

$$f(x) = \sum_{n=0}^{\infty} f_n \frac{x^n}{n!}$$

where the f_n are the elements of some sequence.

4.2 Building Ternary Functional Graphs

We would like to use these EGFs to give an expression for the number of ternary functional graphs with n nodes. To do this, we will build up to functional graphs by first considering ternary trees, then cycles, then connected components, and finally functional graphs. We begin with the penultimate example from [1], which demonstrates the ways in which two generating functions can be composed.

Theorem 2 ([1], Theorem 3.20)**.** *Let f_n be the number of ways to carry out a task on $\{1, \ldots, n\}$ and g_n be the number of ways to carry out another task on the same sequence. Let $F(x) = \sum_{i=0}^{\infty} f_i \frac{x^i}{i!}$ and $G(x) = \sum_{i=0}^{\infty} g_i \frac{x^i}{i!}$. Let h_n be the number of ways to:*

- *split $\{1, \ldots, n\}$ into the disjoint union of S and T,*
- *carry out Task 1 on S,*
- *carry out Task 2 on S.*

Let $H(x) = \sum_{i=0}^{\infty} h_i \frac{x^i}{i!}$. Then $H(x) = F(x)G(x)$.

Proof. Considering $F(x)G(x)$, as defined in the theorem,

$$F(x)G(x) = \left(f_0 + f_1 x + \frac{f_2 x^2}{2!} + \cdots\right)\left(g_0 + g_1 x + \frac{g_2 x^2}{2!} + \cdots\right)$$

$$= \sum_{i=0}^{\infty} \left(\sum_{k=0}^{i} \frac{f_k}{k!} \frac{g_{i-k}}{(i-k)!} \right) x^i$$

$$= \sum_{i=0}^{\infty} \left(\sum_{k=0}^{i} \binom{i}{k} f_k g_{i-k} \right) \frac{x^i}{i!}$$

Therefore, the coefficient of the i^{th} term is the product of the different ways to choose k ($\sum_{k=0}^{i}$), the number of ways to choose k objects from i for Task 1 $\binom{i}{k}$, the number of ways to perform Task 1 (f_k) and the number of ways to perform Task 2 (g_{i-k}),

$$= \sum_{i=0}^{\infty} h_i \frac{x^i}{i!} = H(x).$$

Therefore the EGF for $H(x)$ is the product of the EGFs for the first and second tasks, respectively $F(x)$ and $G(x)$. □

4.2.1 Making a Ternary Tree

We will use Theorem 2 to demonstrate how to use EGFs to make a ternary tree on $\{1, \ldots, n\}$.

First, we split $\{1, \ldots, n\}$ into the disjoint union of two sets, S and T. T will be the root and S will be everything else. The real meat comes when we split up the elements of S into three trees (since we are making a ternary tree) and then, within each tree, repeat the process until we run out of elements. We split this up into Task 1 and Task 2.

Task 1 is the task of making T into a root. Since there is only one element in T the EGF is simple: x. After all, the only value we want hung on this clothesline is 1.

Task 2 is the task of breaking S into three pieces, S_1, S_2 and S_3. We consider this to be three subtasks. Task 2a is the task of assigning S_1 to the left ternary tree with EGF denoted $t(x)$ (the t is for ternary). Task 2b is the task of assigning S_2 to the middle ternary tree, with EGF $t(x)$. Naturally, Task 2c is the task of assigning S_3 to the right ternary tree, with EGF $t(x)$.

Therefore, we want the number of ways to make a ternary tree, $t(x)$. Note that, since we do not care which subtree is the left, middle, or right, we must divide the number of ways to perform Task 2 by the number of ways to permute 3 things: 3!.

By Theorem 2,

$$t(x) = \frac{xt(x)t(x)t(x)}{3!} + x. \tag{1}$$

The addition sign may be read as an "or" symbol. Either we have at least one node, in which case we will have a root and three subtrees, as described by the $\frac{xt(x)t(x)t(x)}{3!}$ term, *or* we will have one node, the root, denoted by x.

The function $t(x)$ is implicitly defined. Rearranging,

$$xt(x)^3 - 6t(x) + 6x = 0, \tag{2}$$

we observe that we have a cubic polynomial in terms of $t(x)$ with one real and two complex roots. For our purposes, it suffices to keep $t(x)$ defined as in (1).

4.2.2 Making Cycles

We begin with the following theorem, basic to an understanding of how we use EGFs to describe cycles.

Theorem 3 ([1], Theorem 3.27)**.** *Let $A(x)$ be the EGF for some task:*

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}$$

with $a_0 = 0$.

Let $H(x) = \sum_{i=0}^{\infty} h_i \frac{x^i}{i!}$ be the EGF for the number of ways to partition $\{1, \ldots, n\}$ into subsets and carry out Task 1 on each subset. Set $h_0 = 1$. Then,

$$H(x) = e^{A(x)}.$$

Proof. I omit the proof, which may be found in [1]. □

We want to count the number of ways to make the sequence $\{1, \ldots, n\}$ into cycles. The EGF for this is:

- the number of ways to partition $\{1, \ldots, n\}$ into subsets, *and*

- the number of ways to make each subset into a cycle.

Let $c(x)$ be the EGF for the number of ways to make the sequence $\{1, \ldots, n\}$ into a cycle,

$$\begin{aligned} c(x) &= \sum_{n=1}^{\infty} a_n \frac{x^n}{n!} \\ a_n &= \text{number of ways to make } n \text{ things into a cycle} \end{aligned}$$

which is the number of ways to place n things into n slots ($n!$), without any concept of starting points (so we divide by n, since in a cycle of n elements each element is a possible starting point). Therefore,

$$a_n = \frac{n!}{n} = (n-1)!$$

$$c(x) = \sum_{n=1}^{\infty} \frac{x^n}{n} = \log\left(\frac{1}{1-x}\right) = -\log(1-x). \tag{3}$$

Consider the EGF for the number of ways to make a sequence into cycles to be Task 1 from Theorem 3 and, letting $H(x)$ be the desired EGF for the number of ways to partition $\{1, \ldots, n\}$ into cycles, we obtain:

$$H(x) = e^{\log\left(\frac{1}{1-x}\right)} = \frac{1}{1-x} = 1 + x + x^2 + \cdots$$

$$= 1 + x + 2!\frac{x^2}{2!} + 3!\frac{x^3}{3!} + \cdots,$$

which implies that there are $n!$ ways to partition $\{1, \ldots, n\}$ into cycles, which is equivalent to the number of ways to permute n things, which is of course equal to $(n!)$.

4.2.3 Ternary Functional Graphs

We recall the EGF for the number of ternary trees,

$$t(x) = x + x\frac{t(x)^3}{3!} \tag{4}$$

and, since two trees hang off of each cyclic node, we apply equation (3) to $t(x)$ as follows:

$$c(x) = \log\left(\frac{1}{1 - \frac{1}{2}xt(x)^2}\right). \tag{5}$$

Now we apply Theorem 3, which partitions the functional graph into connected components of cycles and their attached trees, yielding

$$f(x) = e^{c(x)} \tag{6}$$

which we could simplify by writing the EGF out for $c(x)$, but it is important to see the relationship between the functional graph and its connected components.

In the next section we will use equations (4), (5), and (6) at length to count mapping statistics and derive our theoretical data.

5 Counting Mapping Statistics

Methods in [1] can be used to "mark" nodes of interest in EGFs for, in our case, ternary functional graphs. These markings allow us to calculate total mapping statistics for ternary functional graphs with n nodes. As far as we know, the EGFs we derive here for the *number of components*, *number of cyclic nodes*, *component size*, *cycle length*, and *tail length* are original to our research.

5.1 Direct Mapping Statistics

Theorem 4. *The exponential generating functions for the total number of components and total number of cyclic nodes in a ternary functional graph of size n are*

$$\begin{aligned} \textit{Number of components} &= \left[\frac{d}{du}e^{uc(x)}\right]_{u=1} \\ &= \frac{1}{2}x^3 + \frac{13}{24}x^6 + \frac{83}{144}x^9 + \frac{355}{576}x^{12} + O(x^{15}) \end{aligned} \tag{7}$$

$$\begin{aligned} \textit{Number of cyclic nodes} &= \left[\frac{d}{du}e^{\ln\frac{1}{1-\frac{1}{2}ux\,t(x)^2}}\right]_{u=1} \\ &= \frac{1}{2}x^3 + \frac{2}{3}x^6 + \frac{29}{36}x^9 + \frac{17}{18}x^{12} + O(x^{15}) \end{aligned} \tag{8}$$

Proof. (7) The exponential generating function in Equation (6) counts the number of ways to partition the nodes in a ternary functional graph into components. Inserting a u marks each component in all of the graphs. The marked generating function results in a Taylor series:

$$1 + \frac{1}{2}ux^3 + \left(\frac{7}{24}u + \frac{1}{8}u^2\right)x^6 + \left(\frac{2}{9}u + \frac{7}{48}u^2 + \frac{1}{48}u^3\right)x^9 + O(x^{12}).$$

The coefficients of u and u^2 in the term corresponding to $n = 6$ tells us that $\frac{7}{24} \cdot 6!$

possible ternary functional graphs with 6 nodes have 1 component and $\frac{1}{8} \cdot 6!$ have 2 components. Differentiating with respect to u and evaluating at $u = 1$ yields the exponential generating function for the total number of components in all ternary functional graphs of size n.

(8) Within each component, Equation (5) is an exponential generating function for the number of ways to make a cycle out of ternary trees. Inserting the u marks the root of each tree in the cycle, therefore counting the number of cyclic nodes. Exponentiating this function accounts for the number of possible components. The marked generating function results in a Taylor series:

$$1 + \frac{1}{2}ux^3 + \left(\frac{1}{6}u + \frac{1}{4}u^2\right)x^6 + O(x^9).$$

Again, the coefficients of u and u^2 in the term corresponding to $n = 6$ tells us $\frac{1}{6} \cdot 6!$ possible ternary functional graphs with 6 nodes have 1 cyclic node and $\frac{1}{4} \cdot 6!$ have 2 cyclic nodes. Differentiating with respect to u and evaluating at $u = 1$ yields the exponential generating function for the total number of cyclic nodes in all ternary functional graphs of size n. □

In order to calculate the average mapping statistics from the total mapping statistics we need to use the exponential generating function for the number of graphs. The following formulas refer to the statistics for a number of nodes n, not the exponential generating functions themselves.

$$\text{avg. number of components} = \frac{\text{total number of components}}{\text{total number of graphs}}$$

$$\text{avg. number of cyclic nodes} = \frac{\text{total number of cyclic nodes}}{\text{total number of graphs}}$$

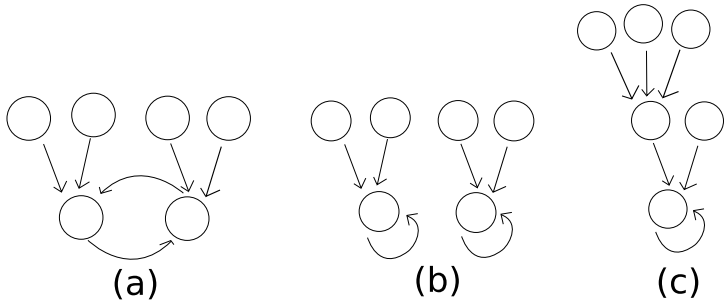

Figure 3: All of the possible ternary functional graphs with 6 nodes.
In "as seen from a node" mapping statistics, the count is weighted by the number of nodes that can "see." For example, in (a), all 6 nodes "see" a cycle length of 2. In (b), 3 nodes "see" a cycle length of 1 and another 3 nodes "see" a cycle of length 1. In (c), all 6 nodes "see" a cycle length of 1. The 6^{th} term in the Taylor series for (10) reflects this.

Now, the theoretical values for average number of components and average number of cyclic nodes can be computed. They can then be compared with the observed values generated from DEFGs (see Section 6).

5.2 "As Seen From a Node" Mapping Statistics

The double marking process is used to count mapping statistics "as seen from a node." The two markings, u and w mark the statistic of interest and the number of nodes that "see" the object respectively.

Theorem 5. *The exponential generating functions for the total component size and*

total cycle length as seen from a node in a random ternary functional graph are

$$\begin{aligned} \textit{Component size} &= \left[\frac{d^2}{du\,dw}e^{c(x)}\ln\frac{1}{1-\frac{1}{2}uwx\,t(uwx)^2}\right]_{u=1,w=1} \\ &= \frac{9}{2}x^3+\frac{51}{4}x^6+\frac{201}{8}x^9+\frac{339}{8}x^{12}+O(x^{15}) \end{aligned} \tag{9}$$

$$\begin{aligned} \textit{Cycle length} &= \left[\frac{d^2}{du\,dw}e^{c(x)}\ln\frac{1}{1-\frac{1}{2}uwx\,t(wx)^2}\right]_{u=1,w=1} \\ &= \frac{3}{2}x^3+\frac{13}{4}x^6+\frac{43}{8}x^9+\frac{191}{24}x^{12}+O(x^{15}) \end{aligned} \tag{10}$$

Proof. (9) The EGF $e^{c(x)}$ counts the number of ways to partition the nodes in a ternary functional graph into components. Equation (9) is the product of the EGF for this task and the task of marking components of connected ternary trees:

$$\ln\frac{1}{1-\frac{1}{2}uwxt(uwx)^2}.$$

Within the component, u acts as a counter for the size of the component by marking all of the nodes in the tree for each tree. Then w acts as a counter for the number of nodes total in the component that "see" the component size. The combined marked generating functions result in a Taylor series:

$$\frac{1}{2}u^3w^3x^3+\left(\frac{7}{24}u^6w^6+\frac{1}{4}u^3w^3\right)x^6+O(x^9).$$

The coefficient of the term corresponding to $n=3$ tells us that there are $\frac{1}{2}\cdot 3!$ possible ternary functional graphs with 3 nodes, and in all of them the three nodes "see" a component size of three. In the case of $n=6$, the coefficients mean that $\frac{7}{24}\cdot 6!$ possible ternary functional graphs with 6 nodes have 6 nodes that "see" a component size of 6 ((a) and (c) in Figure 3), and $\frac{1}{4}\cdot 6!$ have 3 nodes that see a component size of 3

((b) in Figure 3). Differentiating with respect to u and w and evaluating at $u = 1$ and $w = 1$ yields the exponential generating function for the component size as seen from a node in all ternary functional graphs of size n.

(10) The double marking process is similar to the process to construct (9). For all of the unmarked components, $e^{c(x)}$, there is a marked component of connected ternary trees:

$$\ln \frac{1}{1 - \frac{1}{2}uwx\, t(wx)^2}.$$

Within the component, u acts as a counter for the size of the cycle by marking the number of trees rooted on the cycle. Then, w acts as a counter for the number of nodes total in the component that "see" the cycle length. The marked generating function results in the Taylor series:

$$\frac{1}{2}uw^3x^3 + \left(\frac{1}{6}uw^6 + \frac{1}{8}u^2w^6 + \frac{1}{4}uw^3\right)x^6 + O(x^9).$$

The coefficient of the term corresponding to $n = 3$ tells us that there are $\frac{1}{2} \cdot 3!$ possible ternary functional graphs with 3 nodes, and in all of them the three nodes "see" a cycle of length 1. In the case of $n = 6$, the coefficients mean that $\frac{1}{6} \cdot 6!$ possible ternary functional graphs with 6 nodes have 6 nodes that "see" a cycle of length 1 ((c) in Figure 3), $\frac{1}{8} \cdot 6!$ have 6 nodes that see a cycle of length 2 ((a) in Figure 3), and $\frac{1}{4} \cdot 6!$ have 3 nodes that see a cycle of length 1 ((b) in Figure 3). Differentiating with respect to u and w and evaluating at $u = 1$ and $w = 1$ yields the exponential generating function for the cycle length as seen from a node in all ternary functional graphs of size n. □

Theorem 6. *The exponential generating function for the total length of all paths to a cycle, henceforth known as the* total tail length as seen from a node*, is*

$$\textit{Tail length} = \left[\frac{d}{du}\frac{zut(z)}{\left(1 - \frac{1}{2}zt(z)^2\right)^2\left(1 - \frac{1}{2}uz\, t(z)^2\right)}\right]_{u=1} \tag{11}$$

Proof. We begin to build up the EGF for total tail length as seen from a node by deriving two functions to count two different types of trees: unmarked and marked. The unmarked trees are the same as in Equation (4),

$$t(z) = \frac{zt(z)^3}{3!} + z$$

We want the EGF for the number of marked trees (so marked because the tree contains the length that we are counting) to be read as follows:

- $c(z) =$
  - one unmarked node,

    or
  - a marked node attached to a tree, one branch of which is marked,

    or
  - an unmarked node attached to a tree, no branches of which are marked,

  which is translated as

$$t(z,u) = z + \frac{1}{2}uzt(z)^2t(z,u) + \frac{1}{6}zt(z)^3.$$

Solving for $t(z,u)$,

$$t(z,u)\left(1 - \frac{1}{2}uzt(z)^2\right) = t(z)$$

$$t(z,u) = \frac{t(z)}{1 - \frac{1}{2}uzt(z)^2}. \tag{12}$$

Now, we know from Theorem 2 that the EGF that counts the total tail length as seen from a node in a functional graph is the product of two EGFs, one that counts the total tail length as seen from a node in each component, and another that counts

the number of corresponding unmarked components, so

$$f(z,u) = f(z) \cdot c(z,u)$$

where $f(z)$ is the same as in Equation (6). To construct $c(z,u)$, we need to derive two surrogate functions $T(z,u)$ and $T(z)$ to express whether a cyclic node is part of a tail length (and thus marked) or not. Since each component is made up of a certain number of nodes and there is one tail length connecting to the cycle in each marked component, we have

- $c(z,u) =$
  - one marked cyclic node,

    or
  - one marked cyclic node and one unmarked cyclic node,

    or
  - one marked cyclic node and two unmarked cyclic nodes,

    or ...

which is translated as,

$$c(z,u) = T(z,u) + T(z,u)T(z) + T(z,u)T(z)^2 + \cdots$$

Summing the geometric series,

$$c(z,u) = \frac{T(z,u)}{1 - T(z)}.$$

A marked cyclic node sees two trees, one of which is marked, so its EGF is:

$$T(z,u) = zut(z,u)t(z) \tag{13}$$

and an unmarked cyclic node sees two trees, neither of which is marked, thus:

$$T(z) = \frac{1}{2} z t(z)^2 \tag{14}$$

Therefore, putting it all together,

$$f(z,u) = f(z)c(z,u) = f(z)\frac{T(z,u)}{1-T(z)} = \frac{zut(z)}{\left(1-\frac{1}{2}zt(z)^2\right)^2\left(1-\frac{1}{2}uzt(z)^2\right)}.$$

□

As in the previous section, in order to calculate average mapping statistics from the total mapping statistics, we need to use a "normalizer." Unlike the previous case, in "as seen from a node" mapping statistics, the number of graphs *and* the number of nodes need to be taken into account. The *average component size*, *cycle length*, and *tail length* are:

$$\text{avg. component size} = \frac{\text{component size}}{\text{total number of graphs} \cdot \text{number of nodes}}$$

$$\text{avg. cycle length} = \frac{\text{cycle length}}{\text{total number of graphs} \cdot \text{number of nodes}}$$

$$\text{avg. tail length} = \frac{\text{total tail length as seen from a node}}{\text{total number of graphs} \cdot \text{number of nodes}}$$

Now, the theoretical values for average cycle length and average tail length can be computed and compared with the observed values generated from the DEFGs (see Section 6).

6 Observed Results

Using a C++ program written by Dan Cloutier in [2], we collected data for the number of ternary discrete exponentiation functional graphs, average number of components, average cycle length, average cycle length as seen from a node, and average tail length as seen from a node for nine primes of the form $p = 3k + 1$.

The functional graphs associated with each prime p contain $p - 1$ nodes (because we do not include 0 mod p in the functional mapping). The theoretical values for the mapping statistics associated with each prime p are the $(p-1)^{th}$ term in the EGFs for the average number of components (described by Equation (7)), the number of cyclic nodes (Equation 8), the cycle length (Equation (10)) and tail length (Equation (11)). Since we are considering the set of ternary functional graphs which must have $3k$ nodes (for some integer k) we use primes of the form $3k+1$. Thus our EGF will have a nonzero coefficient c_n only when n is a multiple of 3. We also note that Cloutier's program does not calculate component size.

6.1 Theoretical Results via Maple and GFUN

Because of the complexity of the generating functions, we were not able to obtain a closed form for the coefficients of the EGFs for the mapping statistics. To obtain the theoretical values for the average mapping statistics, we used Maple to convert the EGFs into procedures that could rapidly calculate the desired coefficients. We used three procedures from the GFUN package. A brief explanation of each procedure follows. These descriptions come from [6].

holexprtodiffeq This procedure converts the closed form expression for our EGF into a differential equation. *holexprtodiffeq* stands for *Holonomic Expression to Differential Equation*. A holonomic expression is defined in [6] as follows: "A function of one variable is said to be *holonomic* when it satisfies an ordinary linear differen-

tial equation with polynomial coefficients" ([6], 168). Many common functions are holonomic: exp, ln, sin, cos, x^a, etc. Also, the class of holonomic functions is closed under finite addition, multiplication and even composition of algebraic functions [7]. Since our EGFs are made up of the building blocks of orthogonal polynomials, exp, and ln through the operations over which the class of holonomic functions is closed, our EGFs are holonomic.

Applying the procedure to the EGF for total number of cyclic nodes, we observe that Maple can express this EGF as an ordinary linear differential equation with polynomial coefficients. We input the EGF as follows, with "terntree" defined as in Equation (2),

```
> numcycnodes := eval(diff(1/(1 - u * (1/2) * x * terntree ^ 2), u), u = 1);
> DEQnumcycnodes := holexprtodiffeq(numcycnodes, y(x));
```

and the procedure outputs

$$
\begin{aligned}
DEQnumcycnodes \;:=\; & \Big\{ y(0) = 0, D(y)(0) = 0, (D^{(2)})(y)(0) = 0, \\
& (D^{(3)})(y)(0) = 3, (D^{(4)})(y)(0) = 0, (D^{(5)})(y)(0) = 0, \\
& (D^{(6)})(y)(0) = 480, (D^{(7)})(y)(0) = 0, (D^{(8)})(y)(0) = 0, \\
& (405x^5 - 36x^2)y(x) + (567x^6 - 468x^3 - 32)\left(\frac{d}{dx}y(x)\right) \\
& +(81x^7 - 144x^4 + 64x)\left(\frac{d^2}{dx^2}y(x)\right) - 144x^2 \Big\}.
\end{aligned}
$$

which is a sequence of initial conditions and the left side of a differential equation whose right side is zero. Our second-order differential equation defines a sequence wherein every third term is nonzero, thus the procedure gives eight initial conditions (the terms prior to the 9^{th}) of which only two are nonzero.

diffeqtorec The procedure *diffeqtorec* translates a holonomic equation $c_k(z)y^{(k)}(z)+\cdots+c_1(z)y'(z)+c_0(z)y(z)+b(z)=0$ for the function $y(z)$ into a holonomic recurrence for the coefficients $u(n)$ of the Taylor series for the unique solution to the differential equation. The generating function (either ordinary or exponential) of a holonomic sequence is holonomic and reciprocally the sequence of Taylor coefficients of a holonomic function is holonomic [7]. The procedure uses this correspondence. For example, we have:

```
> RECnumcycnodes := diffeqtorec(DEQnumcycnodes, y(x), u(n));
```

$$
\begin{aligned}
RECnumcycnodes \;:=\; & \{u(0)=0, u(1)=0, u(2)=0, u(3)=1/2, u(4)=0, \\
& u(5)=0, u(6)=2/3, u(7)=0, u(8)=0, \\
& (486n+405+81n^2)u(n)+(-2304-1188n-144n^2)u(n+3) \\
& +(672n+1728+64n^2)u(n+6)\}
\end{aligned}
$$

rectoproc Given a holonomic recurrence, the *rectoproc* procedure returns a Maple procedure that computes the n^{th} term of the sequence that satisfies the recursion. The initial terms of the sequence, if provided, are stored in the *remember* table of the procedure; the other terms are computed one by one, according to the recurrence, and using the memo-mechanism provided by Maple [6].

The actual procedure is long and oblique, so we do not produce it here. However, we demonstrate it for a small value of n,

```
> PROCnumcycnodes(12);
```

$$\frac{17}{18}$$

which agrees with the coefficient of x^{12} in the series following Equation (8).

We derive the theoretical statistics by running the obtained Maple procedures for each statistic for each prime and apply the normalizers discussed in Section 5. We compare these values with the observed values from Cloutier's program.

In the following tables the first column is the chosen prime, p, as used in the equation $y = \gamma^x \bmod p$. Cloutier's program then analyzes the functional graph corresponding to $f(x) = \gamma^x \bmod p$ for each base, γ, but only if the resulting graph is ternary. The second column, labelled "Number of DEFGs," is the size of the set of ternary discrete exponentiation functional graphs. It is derived by computing $\phi(\frac{p-1}{3})$ as in Theorem 1. The third column is the output of Cloutier's program for each prime p (and all of the acceptable bases γ for each prime p). The fourth column is the $(p-1)^{th}$ coefficient of the EGF for each statistic, calculated using the Maple procedures discussed above. The fifth column, the relative error, is calculated by taking the difference of the theoretical value and the observed value for each prime p and dividing by the theoretical value.

To understand the data, it is useful to look at the asymptotic estimates that more generally describe the growth of these structures. Theorem 2 from [4] gives the asymptotic estimates of a number of mapping statistics, and in particular the average number of components and cyclic nodes in a random functional graph. Cloutier performed a similar analysis on binary functional graphs, (Theorem 5, [2]) giving the approximation in Equation (15), below, which seems to be a little higher than the expected number of components in a ternary functional graph. This intuitively implies that the average number of components decreases when each cyclic node requires two

trees to hang off of it (see Figure 3).

$$\begin{aligned} \textit{Avg. Number of components} &= \frac{\log(2n)+\gamma}{2} \\ &= \frac{\log(2\cdot 100297)+\gamma}{2} = 6.39312 \end{aligned} \tag{15}$$

Cloutier gives $\sqrt{\pi n/2}-1$ as an asymptotic estimate for the average number of cyclic nodes (Theorem 5, (ii) [2]). We made an adjustment for the ternary case and obtained

$$\begin{aligned} \textit{Avg. Number of Cyclic Nodes} &= \sqrt{\frac{\pi n}{4}}-\frac{2}{3} \\ &= \sqrt{\frac{\pi 100297}{4}}-\frac{2}{3} = 279.998 \end{aligned} \tag{16}$$

For the purpose of concision, we do not include the asymptotic estimates in the following tables.

Avg. Number of Components				
Prime Number	Number of Ternary γ's	Theoretical Values	Observed Values	Relative Error
100297	9504	6.09055	6.03188	0.009632
100333	11136	6.09086	6.01419	0.012588
100549	9072	6.09273	6.03616	0.009286
100621	8064	6.09336	6.10379	0.001713
100693	11184	6.09398	6.01967	0.012194
100801	7680	6.09492	6.04922	0.007499
100981	7680	6.09650	6.04089	0.009122
101089	10368	6.09745	6.01987	0.012723
101161	8960	6.09808	6.06641	0.005194

Avg. Number of Cyclic Nodes				
Prime Number	Number of Ternary γ's	Theoretical Values	Observed Values	Relative Error
100297	9504	279.998	281.555	0.005559
100333	11136	280.049	282.662	0.009332
100549	9072	280.351	280.018	0.001184
100621	8064	280.451	279.935	0.001839
100693	11184	280.552	280.444	0.000385
100801	7680	280.702	280.444	0.000922
100981	7680	280.954	280.373	0.002066
101089	10368	281.104	279.932	0.004168
101161	8960	281.204	281.447	0.000862

Avg. Cycle Length (as seen from a node)				
Prime Number	Number of Ternary γ's	Theoretical Values	Observed Values	Relative Error
100297	9504	140.498	139.191	0.009298
100333	11136	140.523	139.432	0.007763
100549	9072	140.674	141.117	0.003146
100621	8064	140.724	140.438	0.002033
100693	11184	140.774	140.215	0.003970
100801	7680	140.850	140.269	0.004122
100981	7680	140.975	141.540	0.004007
101089	10368	141.051	140.958	0.000657
101161	8960	141.101	140.716	0.002726

Avg. Tail Length (as seen from a node)				
Prime Number	Number of Ternary γ's	Theoretical Values	Observed Values	Relative Error
100297	9504	140.001	141.673	0.011947
100333	11136	140.026	142.219	0.015664
100549	9072	140.177	140.499	0.002303
100621	8064	140.227	141.167	0.006708
100693	11184	140.277	139.944	0.002372
100801	7680	140.353	142.385	0.014486
100981	7680	140.478	140.610	0.000943
101089	10368	140.553	140.152	0.006349
101161	8960	140.604	139.711	0.000862

Theorem 3 of [4] gives $\sqrt{\pi n/8}$ as the asymptotic estimate for both the average cycle length and average tail length as seen from a node. We made a small adjustment for the average cycle length and tail length in a ternary graph, and obtained

$$\begin{aligned} \textit{Avg. Cycle Length} = \textit{Avg. Tail Length} &= \sqrt{\frac{\pi n}{16}} - \frac{2}{3} \\ &= \sqrt{\frac{\pi 100297}{16}} - \frac{2}{3} = 140.333 \end{aligned} \tag{17}$$

The theoretical values appear to be very close to the observed values for the DEFGs generated by the primes. The number of ternary DEFGs generated for each prime ranged from 7680 to 11184 graphs. The results for average cycle length had all of the relative errors less than 1%. For average number of components and average number of cyclic nodes, all of the relative errors are less than 2%.

There does not appear to be a correlation between number of ternary DEFGs and relative error in our results. In the data set for the cycle length, the highest error of .9% occurred for a prime that generated 9504 graphs, however the second highest error of .1% occured for a prime that generated 11136 graphs. In the same data set, the least two errors of .06% and .2% occured for primes that generated 10368 and 8064 graphs respectively.

The results from the average number of components and average number of cyclic nodes also show that the number of ternary DEFGs does not appear to be correlated with relative error.

7 Conclusions

Our observed results showed that ternary DEFGs, with respect to the four mapping statistics (average number of cyclic nodes, average cycle length as seen from a node, average number of components, and average tail length as seen from a node), behaved very similarly to the "average" ternary functional graph. So, we conclude that the discrete log behaves like a "random" function. This means that, for these four mapping statistics, the discrete logarithm does not appear to show any characteristic behavior or pattern that could later be exploited.

Future research could include derivation of EGFs for more mapping statistics such as tree size, maximum cycle length, and maximum tail length. These statistics could then be included in the existing analysis of unary, binary and ternary DEFGs, or in an extrapolation to the more general m-ary case, (possibly) through use of the Lagrange Inversion Formula. In addition, higher moments of these statistics (variance, skewness, etc.) could be analyzed to detect differences between the observed statistics of DEFGs and the theoretical statistics of "average" functional graphs.

References

[1] Miklos Bona. *Introduction to Enumerative Combinatorics*. McGraw-Hill Science & Engineering, 2005.

[2] Daniel Cloutier. Mapping the discrete logarithm. Senior Thesis, Rose-Hulman Institute of Technology, 2005.

[3] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, November 1976.

[4] Philip Flajolet and Andrew Odlyzko. Random mapping statistics. *Advances in Cryptology, Proc. Eurocrypt '89*, 1990.

[5] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, February 1978.

[6] Bruno Salvy and Paul Zimmerman. Gfun: a maple package for the manipulation of generating and holonomic functions in one variable. *ACM Trans. Math. Softw.*, 20(2):163–177, 1994.

[7] R. P. Stanley. Differentiably finite power series. *European Journal of Combinatorics*, 1(2):175–188, 1980.

[8] Herbert S. Wilf. *Generatingfunctionology*. Academic Press, University of Pennsylvania, 1994.