Mapping the Discrete Logarithm

Daniel R. Cloutier
Rose-Hulman Institute of Techonology
Terre Haute, IN 47803
Daniel.R.Cloutier@Rose-Hulman.edu

March 23, 2005

Abstract

The discrete logarithm is a problem that surfaces frequently in the field of cryptography. This results from the use of the transformation $g^a \mod n$ which is of interest here. We use this function to build functional graphs and explore many statistical properties of these mappings such as the number of components, average cycle size and average tail size. Using published theory on random mappings and permutations as well as theory developed here on binary functional graphs (formed by restricting the modulus to be a prime of the form $2q + 1$ with q prime) we compare our observed results to the expected values. The results show a strong relationship for the permutations and binary functional graphs, but the results were poor for the general case. This implies that the factorization of $n - 1$ (when n is prime) clearly impacts the expected shape of the graphs and more research should be done to generalize the results.

1 Introduction

Just a few decades ago, cryptography was considered a domain exclusive to national governments and militaries. However, the computer explosion has changed that. Every day, millions of people trust that their privacy will be protected as they make online purchases or communicate privately with a friend. Many of the cryptographic algorithms they will use are built upon a common transformation, namely

$$g^a \mod n \equiv y. \tag{1}$$

For instance, Diffie-Hellman key exchange, RSA and the Blum-Micali pseudorandom bit generator all use (1). This paper will examine some of the properties exhibited by this sort of transformation and provide theoretical and experimental data describing how choosing a modulus with specified properties can change the expected nature of the transformation. First, though, we give a brief look at each of the algorithms listed above to provide an illustration of the wide range of use (1) has found in the field of cryptography and the extent to which many of these algorithms base their security on the properties of (1).

In 1976, Whitfield Diffie and Martin Hellman proposed a way to use (1) as part of a method for key-exchange. They presented a scheme in [3] where two users compute a shared secret key over a public communication channel using (1). Using a publicly specified p and g, which is a primitive root (also referred to as a generator) of p, Alice computes $g^a \mod p$ and Bob computes $g^b \mod p$. They can exchange results and each can compute the same shared key, namely $g^{ab} \mod p$. The most obvious attack to this system involves solving

directly for a (or equivalently, b). Fortunately, this problem appears to be quite difficult and is generally known as the discrete logarithm problem. Schneier [10, Section 11.6] discusses the problem and the current state of research.

Two years after Diffie and Hellman announced their key-exchange system, Ronald Rivest, Adi Shamir and Leonard Adleman used (1) as the backbone of a new public-key encryption method dubbed RSA [9]. RSA is a simple and elegant algorithm, yet it appears to provide an extremely high level of security. If Alice intends to send Bob a message, then she looks up Bob's public key which consists of two numbers, e and n. She then computes $C = M^e \mod n$ where M is the message she wishes to encrypt. Bob then takes Alice's encrypted message C and combined with his secret key d, computes $M = C^d \mod n$. The most obvious attack on this system is to compute d, Bob's secret key. The authors argue that this is equivalent to the problem of factoring n since the prime factorization of n can be found using e and d. In general, however, factoring a large number has proven to be a hard problem. Pomerance [8], describes the evolution of our attempts to find an efficient solution to this problem.

In 1984, Blum and Micali introduced a pseudorandom bit generator [2] designed to be secure enough for cryptographic applications. Schneier [10, Section 17.9] gives the following explanation of the algorithm. Let g be a prime and p be an odd prime. A seed, x_0 is used to start the process with each following bit computed using $x_{i+1} = g^{x_i} \mod p$. The output is 1 if $x_i < (p-1)/2$ and 0 otherwise. Assuming that g and p are known, predicting the next bit with greater than 50% certainty appears to require solving the discrete logarithm problem.

2 Terminology and Background

A mapping relates elements in one set, the domain, to elements in another set, the codomain. It does this by means of what will be referred to here as a transition function. For instance, let A be the domain and B the codomain. If φ is the transition function, then for each $a \in A$, $\varphi(a) = b$ for some $b \in B$. In this paper, we will examine mappings generated with (1) as the transition function and restrict the values of n to primes. For a more natural notation, p will hereafter denote the prime modulus. In some instances, it will prove to be useful to interpret the mappings as functional graphs. A graph is a set of vertices (or nodes) and a set of edges where each edge is a directed path from one vertex to another (or possibly the same) vertex. A functional graph simply restricts the edges such that each vertex must have exactly one edge directed out from it. Equivalently, the out-degree of each vertex must be one. The relationship between the mappings which interest us, and functional graphs is straightforward. In the mappings of interest here, the domain and codomain are the same set, namely S where

$$S = \{1, 2, ..., p-1\}.$$

Each element in S can then be interpreted as a vertex. The edges are defined simply for $a, b \in S$ where an edge $\langle a, b \rangle$ is in the graph if and only if $\varphi(a) = b$.

There are a number of statistics of interest derived from functional graphs. Following the convention of [5], which treats random mappings in detail, let $\varphi : S \to S$ be the transition function so that the edges in the functional graph can be expressed as the ordered pair $\langle x, \varphi(x) \rangle$ for $x, \varphi(x) \in S$. By applying the pigeonhole principle and noting that the cardinality of S is $p-1$ we can say that by starting at any random point u_0 and following the sequence $u_1 = \varphi(u_0)$, $u_2 = \varphi(u_1)$, ..., there must be a $u_i = u_j$ after at most p iterations.

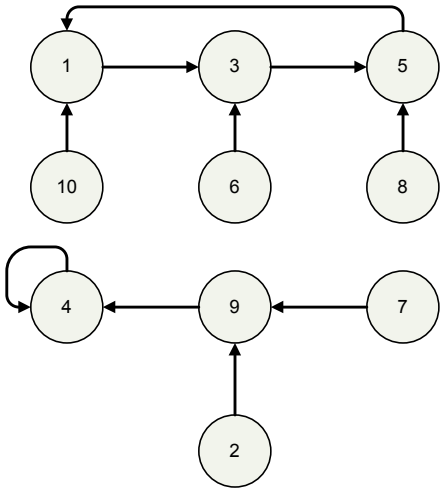

Figure 1: The graph generated using $\varphi(x) = 3^x \mod 11$. This graph has two connected components: one containing a cycle of length three (1,3,5) and the other containing a cycle of length one (fixed point) at 4.

Suppose u_i occurs before u_j in the sequence of nodes. In this case, the tail length is the number of iterations from u_0 to u_i. The cycle length is the number of iterations from u_i to u_j. In more natural graphical terms, the cycle length is the number of edges (or equivalently nodes) involved in the directed path from u_i to itself. The tail length is the number of edges from u_0 to u_i. Additionally, a terminal node is one with no pre-image, or more formally, x is a terminal node if $\varphi^{-1}(x) = \emptyset$. A node is an image node if it is not a terminal node. Since each node has an out-degree of exactly one, each cycle with the trees grafted onto its nodes will form a connected component. An example of a small functional graph can be found in Figure 1.

If g is a primitive root modulo p, the shape of the graph changes dramatically. A primitive root is any value of g where the smallest exponent, x, such that $g^x \mod p \equiv 1$ is $x = \phi(p)$ where ϕ refers to Euler's totient function. For a prime, p, $\phi(p) = p - 1$. For more information on Euler's totient function see [7, Section 2.1]. Using the functional graph interpretation, not only is each node restricted to an out-degree of one, but the in-degree is similarly restricted to one. This implies that every node is both an image node and a cyclic node for a mapping generated with g being a primitive root. Mappings of this sort will be referred to as permutations since for any $a \in S$ there exists a unique $b \in S$ such that $\varphi(a) = b$, or more naturally, φ produces all of the elements of S exactly once in some order. Therefore, the values of g will be split based on whether or not they are a primitive root and will be analyzed separately.

In cryptography, it is common to look for primes where $p - 1$ has at least one large factor. In fact, primes of the form $p = 2q + 1$ where q is also prime have become known as safe primes for the seemingly enhanced security they provide (q is also known as a Sophie Germain prime). When p is a safe prime, the functional graphs produced take on a somewhat predictable shape. Theorem 1 describes the conditions that can be placed on g.

Theorem 1. *Let p be a safe prime, with q being the associated Sophie Germain prime, that is, $p = 2q + 1$. For every $g \in \{1, 2, ..., p - 1\}$, if g is not a perfect square modulo p then g is either a primitive root modulo p or $g \equiv -1 \mod p$.*

Proof. By [7, Corollary 2.38], $x^2 \equiv g \mod p$ has two solutions if

$$g^{(p-1)/2} \equiv 1 \mod p \tag{2}$$

and no solutions if

$$g^{(p-1)/2} \equiv -1 \mod p. \tag{3}$$

Showing that (2) and (3) are the only possibilities can be done by squaring both sides of each equation and invoking Fermat's Little Theorem [7, Theorem 2.7] which states that $g^{p-1} \equiv 1 \mod p$. If g is a perfect square then it satisfies Equation (2). Additionally, note that [7, Lemma 2.10]

$$x^2 \equiv 1 \mod p \text{ if and only if } x \equiv \pm 1 \mod p. \tag{4}$$

Since the prime factors of $p-1$ are 2 and q, for g to be a primitive root $g^{(p-1)/q} \equiv g^2 \not\equiv 1 \mod p$ and $g^{(p-1)/2} \equiv g^q \not\equiv 1 \mod p$. Therefore, if g satisfies Equation (3),and Equation (4) then $g \equiv -1 \mod p$. If g satisfies Equation (3), but does not satisfy (4), then g must be a primitive root by Lagrange's Theorem [7, Corollary 2.32] since the order of g is not any of the proper divisors of $p-1$. (Showing g does not have order one is trivial since 1 is a perfect square and $g \not\equiv 1 \mod p$ for all other values of g.) Then the order must be $p-1$, which is equivalent to saying g is a primitive root modulo p. □

Theorem 1 combined with elementary rules of modular arithmetic imply that the only image nodes that can be generated by g when g is neither a primitive root, nor congruent to -1 modulo p are perfect squares modulo p. Furthermore, Theorem 1 implies that the in-degree of each node is restricted to exactly zero or two. This fits with the definition of *binary functional graphs* given in [5]. We will apply the same name to these graphs in this paper. Figure 1 is an example of a binary functional graph since 11 is clearly a safe prime.

3 Theoretical Results

There are three main sections of theory that must be covered. We first need to determine the expected parameters for values of g that are a primitive root modulo p. Since these values are independent of whether p is a safe prime or not, they will apply to all instances where g is a primitive root. If g is not a primitive root, the parameters need to be estimated separately based on the constraints explicitly placed on p. Estimates are available in literature for unconstrained values of p. They are not available if p is a safe prime; however, the methods used in the unconstrained case can be adapted to the constrained case with relative ease. In all of these cases, let n be the size of the set acting as both the domain and the codomain. In our case, $n = p - 1$.

3.1 Permutations

If g is a primitive root modulo p, then rather than a functional graph, g generates a permutation of the numbers $1, 2, ..., p-1$. A number of the parameters of interest follow directly

from the definition of a permutation. These parameters are:

Number of cyclic nodes	n
Number of tail nodes	0
Number of terminal nodes	0
Number of image nodes	n
Average tail length	0
Maximum tail length	0

A moment's reflection should show that each of these statistics follow from the fact that g is generating a permutation of the numbers $1, 2, ..., p-1$. There are three non-trivial parameters of interest. They are expressed in Theorem 2.

Theorem 2. *The asymptotic values for the number of components, the average cycle length as seen from a random node and the maximum cycle length have the following values:*

$$\textit{Number of components} \quad \sum_{i=1}^{n} \frac{1}{i} \tag{i}$$

$$\textit{Average cycle length} \quad \frac{n+1}{2} \tag{ii}$$

$$\textit{Maximum cycle length} \quad n \int_0^\infty \left[1 - \exp\left(-\int_v^\infty e^{-u}\frac{du}{u}\right)\right] dv \approx 0.62432965n \tag{iii}$$

Proof. In e.g., [1, Theorem 1], it is shown that the expected number of cycles of length i in a random permutation is asymptotic to $1/i$. Part (i) follows immediately as

$$\text{Number of components} = \sum_{i=1}^{n} \frac{1}{i}.$$

Part (ii) of the Theorem follows from the observation that for each cycle of length i, each of the i nodes will contribute a weight of i for the cycle. Thus, the solution follows

$$\text{Average cycle length} = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{i} i^2 = \frac{n+1}{2}.$$

Part (iii) seems to have first been solved by Shepp and Lloyd in 1966 [11]. An alternative solution and proof is offered by Flajolet and Odlyzko in [4]. □

3.2 Functional Graphs

Unlike the results found in the previous section, none of the parameters of interest are immediately accessible from the definition of a functional graph. However, Flajolet and Odlyzko do a thorough analysis of functional graphs in [5]. While none of these results are original to [5], Flajolet and Odlyzko demonstrate that all of these parameters can be estimated through a singularity analysis of generating functions as opposed to the variety of methods used to individually estimate them previously. The results are summarized below in Theorem 3.

Theorem 3. *The asymptotic values for the parameters of interest as $n \to \infty$ are:*

Number of components	$\frac{\log(2n)+\gamma}{2}$	(i)
Number of cyclic nodes	$\sqrt{\pi n/2} - \frac{1}{3}$	(ii)
Number of tail nodes	$n - \sqrt{\pi n/2} + \frac{1}{3}$	(iii)
Number of terminal nodes	$e^{-1}n$	(iv)
Number of image nodes	$(1-e^{-1})n$	(v)
Average cycle length	$\sqrt{\pi n/8}$	(vi)
Average tail length	$\sqrt{\pi n/8}$	(vii)
Maximum cycle length	$\sqrt{\frac{\pi n}{2}} \int_0^\infty \left[1 - \exp\left(-\int_v^\infty e^{-u}\frac{du}{u}\right)\right] dv$ $\approx 0.78248\sqrt{n}$	(viii)
Maximum tail length	$\sqrt{2\pi n}\log 2$	(ix)

In part (i), γ refers to Euler's constant which is approximately 0.57721566. The proofs for Theorem 3 are omitted and can be found in [5]. The second order terms for parts (i), (ii), and (iii) were not given in [5], but can be computed quickly from the methods contained there.

3.3 Binary Functional Graphs

Imitating the methods of [5], we first need to convert our ideas of a binary functional graph into corresponding generating functions. The machinery is fairly straightforward once we define the following as in [5]:

BinFunGraph	= set(Components)
Component	= cycle(Node*BinaryTree)
BinaryTree	= Node + Node*set(BinaryTree, cardinality = 2)
Node	= Atomic Unit

In other words, a binary functional graph is a set of components. Each component is a cycle of nodes with each node having an attached binary tree to bring its in-degree to two. A binary tree is either a node (terminal node) or a node with two binary trees attached. Finally, a node is simply an atomic unit. A moment's reflection should indicate that this natural specification does, in fact, specify a binary functional graph. Imitating the transformations in [5, Section 2.1], the generating functions of interest are

$$f(z) = e^{c(z)} = \frac{1}{1 - zb(z)} \tag{5}$$

$$c(z) = \log \frac{1}{1 - zb(z)} \tag{6}$$

$$b(z) = z + \frac{1}{2}zb^2(z) \tag{7}$$

Where f generates the number of binary functional graphs, c generates the number of components, and b generates the number of binary trees of a given size. Solving for quadratic

formula for (7), we can produce the following equations f and c which simplify some of the cases

$$f^*(z) = \frac{1}{\sqrt{1-2z^2}} \tag{8}$$

$$c^*(z) = \log \frac{1}{\sqrt{1-2z^2}} \tag{9}$$

In order to compute asymptotic forms of any of the statistics of interest, we must first compute an asymptotic form for f or f^* to normalize results. The following derivations give only a highlight of the methods used by Flajolet and Odlyzko. The interested reader is encouraged to see [4, 5] for detailed proofs.

From equation (8) it is clear that there is a singularity at $z = 1/\sqrt{2}$. Performing the analysis as in [5, Section 2], the asymptotic form for f^* falls out quickly as[1]

$$f^*(z) \sim \frac{2^{n/2}}{\sqrt{2\pi n}}. \tag{10}$$

In at least one case, there are some second-order interactions with the error terms of the number of graphs and the appropriate statistic. In these cases, a more exact form of (10) must be used. Expanding one more term in the expansion of f^* gives

$$f^*(z) \sim \frac{2^{n/2}}{\sqrt{2\pi n}} - \frac{2^{n/2}}{4n\sqrt{2\pi n}} = \frac{2^{n/2}(4n-1)}{4n\sqrt{2\pi n}} \tag{11}$$

In most cases, using this more precise expansion of f is not necessary and does not change the results. Therefore, in all but the necessary cases, (10) will be used.

We begin by deriving the results for the most simple parameters.

Theorem 4. *The asymptotic forms for the number of components, number of cyclic nodes, number of tail nodes, number of terminal nodes and number of image nodes in a random binary functional graph of size n, as $n \to \infty$ are*

$$\textit{Number of components} \quad \frac{\log(2n) + \gamma}{2} \tag{i}$$

$$\textit{Number of cyclic nodes} \quad \sqrt{\pi n/2} - 1 \tag{ii}$$

$$\textit{Number of tail nodes} \quad n - \sqrt{\pi n/2} + 1 \tag{iii}$$

$$\textit{Number of terminal nodes} \quad n/2 \tag{iv}$$

$$\textit{Number of image nodes} \quad n/2 \tag{v}$$

In part (i), γ represents Euler's constant which is approximately 0.57721566. The highlights of the proofs as they differ from those in [5] follow.

Proof. It should first be noted that part (ii) and part (iii) are complements of each other. Likewise, parts (iv) and (v) must sum to n. The forms for parts (iii) and (v) follow from the derivation of their partners. As in [5], the following bivariate generating functions need to be defined with parameter u marking the elements of interest. The generating functions

[1] The analysis in this paper have been performed using the computer algebra program Maple and the packages created as part of the Algorithms Project at INRIA, Rocquencourt, France. The packages can be found online at http://pauillac.inria.fr/algo/libraries/software.html.

for the number of components, number of cyclic nodes and number of terminal nodes are respectively:

$$\xi_1(u,z) = \exp\left(u \log \frac{1}{1-zb(z)}\right) \tag{12}$$

$$\xi_2(u,z) = \frac{1}{1-uzb(z)} \tag{13}$$

$$\xi_3(u,z) = \frac{1}{\sqrt{1-2uz^2}} \tag{14}$$

Equation (14) follows from marking the appropriate element in (7), resolving the quadratic formula and substituting into (5). Imitating the methods in [5], the mean value generating function, $\Xi(z)$, is found by taking the partial derivative of $\xi(u,z)$ and evaluating at $u=1$. This yields the following results

$$\Xi_1(z) = \frac{1}{1-zb(z)} \log\left(\frac{1}{1-zb(z)}\right) \tag{15}$$

$$\Xi_2(z) = \frac{zb(z)}{(1-zb(z))^2} \tag{16}$$

$$\Xi_3(z) = \frac{z^2}{(1-2z^2)^{3/2}}. \tag{17}$$

Equations (15), (16), and (17) lead to the following expansion around the singularity $z = 1/\sqrt{2}$.

$$\Xi_1(z) = \frac{\log\left(\frac{1}{\sqrt{2}\sqrt{1-z\sqrt{2}}}\right)}{\sqrt{2}\sqrt{1-z\sqrt{2}}} + O\left(\sqrt{1-z\sqrt{2}}\left(\log\left|\frac{1}{1-z\sqrt{2}}\right| + 2\right)\right) \tag{18}$$

$$\Xi_2(z) = \frac{1}{2(1-z\sqrt{2})} - \frac{\sqrt{2}}{2\sqrt{1-z\sqrt{2}}} + O(1) \tag{19}$$

$$\Xi_3(z) = \frac{\sqrt{2}}{8(1-z\sqrt{2})^{3/2}} - \frac{5\sqrt{2}}{32\sqrt{1-z\sqrt{2}}} + O\left(\sqrt{1-z\sqrt{2}}\right) \tag{20}$$

Applying singularity analysis as in [5], Equations (18) through (20), lead to the following:

$$\Xi_1(z) = \frac{2^{n/2}\log n}{2\sqrt{2\pi n}} + \frac{2^{n/2}(\gamma + \log 2)}{2\sqrt{2\pi n}} + O\left(\frac{2^{n/2}(\log|n| + 2)}{n^{3/2}}\right) \tag{21}$$

$$\Xi_2(z) = \frac{2^{n/2}(\sqrt{\pi n} - \sqrt{2})}{2\sqrt{\pi n}} + O\left(\frac{2^{n/2}}{n^{3/2}}\right) \tag{22}$$

$$\Xi_3(z) = \frac{2^{n/2}(4n-1)}{8\sqrt{2\pi n}} + O\left(\frac{2^{n/2}}{n^{3/2}}\right) \tag{23}$$

The forms in the statement of the proof follow by normalizing (21) and (22) by (10) and (23) by (11). Parts (iii) and (v) follow from parts (ii) and (iv) respectively since the respective pairs must sum to n.

□

The asymptotic values for the length of a cycle and tail as seen from a random point in the graph are also appropriate. The asymptotic forms of these values are given in Theorem 5.

Theorem 5. *The expected value for the cycle size and tail length as seen from a random node in a random binary functional graph are asymptotic to*

$$\text{Average cycle length} \quad \sqrt{\pi n/8} \tag{i}$$

$$\text{Average tail length} \quad \sqrt{\pi n/8} \tag{ii}$$

These values are the same as in the general case for a functional graph. Their proofs proceed in a similar fashion to the proof of Theorem 3 in [5] with the analogy closely following that for the proof of Theorem 4 given previously. The proofs are therefore omitted.

The maximum cycle length and maximum tail length can be generated from the methods in [5].

Theorem 6. *The expected length of the largest cycle and largest distance of a node to a cycle in a binary functional graph of size n have the following asymptotic forms*

$$\text{Maximum cycle length} \quad \sqrt{\frac{\pi n}{2}} \int_0^\infty \left[1 - \exp\left(-\int_v^\infty e^{-u}\frac{du}{u}\right)\right] dv$$

$$\approx 0.78248\sqrt{n} \tag{i}$$

$$\text{Maximum tail length} \quad \textit{Coming soon!} \tag{ii}$$

The proof of part (i) follows completely the proof of Theorem 5 in [5] with the substitution of the correct version of f. Therefore, the proof is omitted.

4 Observed Results

In [6], heuristics and observed values for the number of small cycles (fixed points and two-cycles) in graphs of the type investigated here are given. Our methods build on this to generate the parameters for which theoretical results have been derived above. The method of data collection was straightforward. A prime was chosen for p and then for each $g \in \{2, 3, ..., p-2\}$, the corresponding map or permutation was generated. The values $g = 1$ and $g = p - 1$ were omitted for their completely predicable shape which would only serve to skew the data. That the shape is predicable is obvious for $g = 1$ since one to any power is one. For $g = p - 1$ the relationship is slightly less obvious. That the shape is predictable is obvious, though, once the following fact is identified: $p - 1 \equiv -1 \mod p$. The only two image nodes then become one and $p - 1$ depending on whether the exponent is even or odd. The statistics of interest could then be counted for each graph and permutation. Thus, the final results give averages over $p - 3$ graphs and permutations. There will be $\phi(p-1)$ permutations and $p - \phi(p-1) - 3$ functional graphs to consider. The generation of graphs and permutations was done in C code written by the author. The primes chosen were $p = 100,057$ which is not a safe prime and $p = 100,043$ which is a safe prime. The number of permutations and graphs associated to each prime are give in Table 1. The predicted and observed values for the number of components, number of cyclic nodes and number of tail nodes can be found in Table 2 for the functional graphs and binary functional graphs. The data for the binary functional graphs is a near perfect fit. The data for the functional graphs, however, does not seem to fit as nicely with the predicted values. While the number

	100,057	**100,043**
Permutations	30,240	50,020
Functional Graphs	69,814	50,020
Total Maps and Graphs	100,054	100,040

Table 1: The number of permutations and functional graphs associated with $p = 100,057$ and $p = 100,043$.

	100,057			**100,043**		
	Predicted	**Observed**	**Error**	**Predicted**	**Observed**	**Error**
Components	6.392	5.675	11.217%	6.392	6.389	0.047%
Cyclic Nodes	396.111	228.389	42.342%	395.416	395.303	0.029%
Tail Nodes	99,659.889	99,827.611	0.168%	99,646.584	99,646.697	0.00011%
Terminal	36,808.545	74,833.743	103.305%	50,021	50,021	0.00%
Image Nodes	63,247.455	25,222.257	60.121%	50,021	50,021	0.00%

Table 2: The predicted and observed number of components, cyclic nodes, tail nodes, terminal nodes and image nodes for the functional graphs and binary functional graphs.

of components correlates well with the predicted value, there is clearly a notable shortage of cyclic nodes. Since the number of cycles seems well correlated with the prediction, but the number of cyclic nodes is smaller than expected, we would expect the average cycle length of these graphs to also be smaller than predicted. Table 3 shows that this is, in fact, the case. In fact, not only are the average cycles smaller than expected by approximately

	100,057			**100,043**		
	Predicted	**Observed**	**Error**	**Predicted**	**Observed**	**Error**
Avg Cycle	198.222	114.832	42.069%	198.208	198.319	0.056%
Max Cycle	247.511	143.023	42.215%	247.494	247.261	0.094%
Avg Tail	198.222	114.218	42.379%	198.208	197.961	0.125%
Max Tail	549.587	312.207	43.192%	??	541.827	??

Table 3: The predicted and observed average lengths for cycles and tails as well as the maximum observed and predicted lengths for cycles and tails in the functional graphs and binary functional graphs.

40%, the maximum cycle length is also equivalently smaller, and the tails, both average and maximum, are shorter by the same margin. The data for the binary functional graphs, however, correlates well in each category.

The data on permutations also fits well with the corresponding predicted values. The values for the parameters of interest can be found in Table 4. This is not unexpected since the prime factorization of $p - 1$ does not impact the expected shape of the permutation.[2]

[2]It is interesting to note that for $p = 100,057$, the largest cycle had a length of 100,052 ($g = 58,303$).

	100,057			100,043		
	Predicted	**Observed**	**Error**	**Predicted**	**Observed**	**Error**
Components	12.091	12.054	0.306%	12.091	12.081	0.083%
Average Cycle	50,028.5	50,191.352	0.326%	50,021.5	49,980.551	0.082%
Maximum Cycle	62,467.927	62,627.745	0.256%	62,459.187	62,395.488	0.102%

Table 4: The predicted and observed number of components, average cycle length, and maximum cycle length for the permutations generated.

5 Conclusions and Future Work

The transformation used here to generate functional graphs and permutations is an exceedingly important transformation in cryptography. If the output the function were to fall into a predictable pattern, it could be an exploitable flaw in many algorithms considered secure today. The fact that the cycles in the functional graphs associated with $p = 100,057$ are smaller than would otherwise be predicted could be of concern if the trend were to continue with increasing complexity of the factorization of $p-1$. The data for the binary functional graphs, however, corresponds well to the theory developed here, as does the data for values of g that are a primitive root modulo p. Since primes with $p-1$ having one large factor are usually sought out, this seems to provide more reason to choose a prime with that trait.

The fact that the data for $p = 100,057$ does not correlate well to the expected values for a random functional graph seems to indicate that the prime factors of $p-1$ are noticeably impacting the shape of the graph, much as the simple factorization of $p-1$ on the safe prime impacted the expected shape of the graph. More research should be done to further explore, in a more general sense, how the prime factorization of $p-1$ impacts the functional graphs generated. Since many algorithms use a modulus that is not prime, e.g. RSA, research should be extended to remove the restriction that the modulus be prime.

The data collected here gives an indication that when averaged over a number of graphs and permutations, the results correlate well to a random binary functional graph or permutation. However, no data has been collected to analyze the distribution of the different parameters. Clearly, if the values are distributed widely, simply comparing the average values to expected average values is less powerful than if the values are all very closely distributed. Data should certainly be collected to, at a minimum, determine the standard deviation of the values from graph to graph.

Acknowledgements The author would like to thank his thesis advisor, Joshua Holden, for his help and support throughout this project.

The longest tail observed had a length of 1,589 ($g = 18,115$) and for 24 different values of g, there were no cycles with a length greater than one. The interesting, maximal statistics for $p = 100,043$ were a largest cycle of length 100,042 ($g = 20,812$ and $g = 94,034$), a longest tail of 1,448 ($g = 89,339$) and three instances where the longest cycle was a fixed point ($g = 72,116$, $g = 91,980$, and $g = 95,997$).

References

[1] R. Arratia and S. Tavaré. The cycle structure of random permutations. *Ann. Probab.*, 20(3):1567–1591, 1992.

[2] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[3] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[4] P. Flajolet and A. Odlyzko. Singularity analysis of generating functions. *SIAM J. Discrete Math.*, 3(2):216–240, 1990.

[5] P. Flajolet and A. M. Odlyzko. Random mapping statistics. In *Advances in cryptology—EUROCRYPT '89 (Houthalen, 1989)*, volume 434 of *Lecture Notes in Comput. Sci.*, pages 329–354. Springer, Berlin, 1990.

[6] J. Holden. Fixed points and two-cycles of the discrete logarithm. In *Algorithmic number theory (Sydney, 2002)*, volume 2369 of *Lecture Notes in Comput. Sci.*, pages 405–415. Springer, Berlin, 2002.

[7] I. Niven, H. S. Zuckerman, and H. L. Montgomery. *An Introduction to the Theory of Numbers*. John Wiley & Sons, Inc, 1991.

[8] C. Pomerance. A tale of two sieves. *Notices Amer. Math. Soc.*, 43(12):1473–1485, 1996.

[9] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21(2):120–126, 1978.

[10] B. Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. John Wiley & Sons, Inc, 2 edition, 1996.

[11] L. A. Shepp and S. P. Lloyd. Ordered cycle lengths in a random permutation. *Trans. Amer. Math. Soc.*, 121:340–357, 1966.