

POLLARD'S RHO METHOD AND NON-POLYNOMIAL FUNCTIONS

Ronan O'Donnell
Department of Mathematics & Statistics, Maynooth University, Kildare, Ireland

David Malone
Hamilton Institute and Department of Mathematics & Statistics, Maynooth University, Kildare, Ireland
David.Malone@mu.ie

Received: 2/24/26, Accepted: 10/19/26, Published: 11/23/26

Abstract

In this paper, we are interested in the factoring method introduced by Pollard in 1975, which is simple to use, effective, and often covered in undergraduate cryptography courses. The method uses a function to generate a sequence in $\mathbb{Z}_n$, which is usually taken to be a polynomial. We begin by describing Pollard's method. We then show that non-polynomial choices may lead to poor performance of the method and identify conditions for better performance. We show that while there are non-polynomial functions that satisfy these conditions, polynomials are the general pragmatic choice.

1. Introduction

Pollard's Rho method is a technique for factoring integers that is efficient compared to basic methods such as trial division, without requiring the more complex theory used by algorithms such as the number field sieves [11, 16]. This, combined with its simple implementation, makes it suitable for factoring modest sized numbers and also for teaching as part of a first cryptography course or algorithms course. It was first published in 1975 by Pollard [14], where it was described as a *Monte Carlo method for factorization*. In 1980, Brent [2] made some improvements on the performance, while keeping the same core idea. Brent and Pollard [3] applied the method in 1981 to factor the eighth Fermat number, $2^{256}+1$, which was known to be composite but the factors were unknown. Pollard [15] also applied a similar idea to solving discrete log problems.

This method makes use of a function to generate sequences in $\mathbb{Z}_n$, where n is the number to be factored. This function is usually taken to be a simple polynomial,

DOI: 10.5281/zenodo.24473406

such as $f(x) = x^2 + 1$. In this paper we highlight two versions of a condition on f that is desirable if Pollard's method is going to be efficient in factoring n. We show that for square-free n this condition is equivalent to f being a polynomial, but in general a wider class of functions satisfy the conditions than just the set of polynomials on $\mathbb{Z}_n$.

We will begin by introducing Pollard's method, and then give an example to show that the choice of function can make a significant difference to the efficiency of the method. We then identify the conditions on f and prove our main result.

2. Pollard's Method

Here, we will describe the base version of Pollard's technique [14]. The aim of Pollard's method is to factor some composite number n. He proposes that we build a sequence x_i by first choosing a starting point $x_0 \in \mathbb{Z}_n$ and a polynomial $f : \mathbb{Z}_n \to \mathbb{Z}_n$. Then we generate the sequence by setting

$$x_{i+1} \equiv f(x_i) \pmod{n}.$$

He then suggests that we track $\gcd(x_i - x_{2i}, n)$, and if we get some number that is neither 1 nor n, then we have found a factor of n. Rather than store the whole sequence, just the current values of x_i and x_{2i} need to be stored, as shown in Algorithm 1. In the original paper $f(x)$ is taken to be $x^2 - 1$; however, the choice of $f(x) = x^2 + 1$ is commonly used today.

While the algorithm is quite simple, the idea that motivates it is cunning. It depends on the following observations. First, the algorithm searches for when $x_i \equiv x_j \pmod{p}$, for *any* factor p of n, through the use of the greatest common divisor. Second, since there are only p different values that $x_i \pmod{p}$ can take, it must repeat in at worst p steps. In fact, since the behavior of a polynomial on $\mathbb{Z}_p$ is complicated, if we are willing to believe they behave like a randomly selected function, then we expect to find a repeat in about $\sqrt{p}$ steps, by the Birthday Paradox[1]. Pollard's original paper observes that all polynomials $x^2 + b$ seem to work equally well (empirically), except $b = 0$ or $b = -2$, and Knuth [10] explains why these two values should be avoided.

Thus, we know that there must be a loop in these values $x_i \pmod{p}$, and consequently the sketch in our mind looks something like Figure 1. Indeed, the similarity of the shape of Figure 1 to the Greek letter ρ is the origin of the method's name[2].

[1]The Birthday Paradox says that if we are choosing from N different items uniformly at random, we only expect to have to choose about $\sqrt{N}$ items before we see a repeated value.

[2]Richard Guy [8], who seems to have been in correspondence with Pollard at the time, suggests that Pollard used the name "Monte Carlo" in the original paper because of the difficulty of getting typesetters to correctly type-set ρ. Guy's article notes that factoring "is almost devoid of applica-

Algorithm 1 A basic version of Pollard's Rho method.

1. We want to factor n.
2. Pick a polynomial such as $f(x) = x^2 + 1$ and a starting point x_0. Set $x = x_0$ and $y = x_0$.
3. Repeat the following:
   - Let $x \equiv f(x) \pmod{n}$, and $y \equiv f(f(y)) \pmod{n}$.
   - Find $r = \gcd(n, x - y)$.
   - If $1 < r < n$ then r is a factor of n. If $r = n$ go back to Step 1. Otherwise $r = 1$, and continue to repeat Step 3.

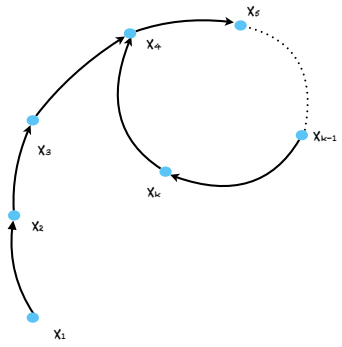

Figure 1: Searching for a loop in $\mathbb{Z}_p$.

The final piece of the algorithm is identifying this loop using x_i and x_{2i}. At some point, both x_i and x_{2i} will leave the tail and take values within the loop, though probably different values. From then on the gap between them will close by $2 - 1 = 1$ position on each step of the algorithm, eventually guaranteeing equality. This is sometimes called *Floyd's cycle-finding algorithm*[3] or *the tortoise and the hare algorithm*. One of Brent's improvements is to use a different cycle finding algorithm, which uses fewer evaluations of the function f.

The algorithm will fail in the situation that the loops for all factors of n are

tion, except that the calculation of many of the commonly occurring number-theoretic functions", reminding us that Pollard and Guy's works in 1975 and 1976, respectively, occurred just *before* the patent application for RSA in 1977 [18], which gave factoring numbers a new significance.

[3]Though, the attribution to Floyd may be a misattribution by Knuth. See the discussion in the Wikipedia page for cycle detection, `https://en.wikipedia.org/wiki/Cycle_detection`.

found at the same step of the algorithm, as in this case $x_i \equiv x_{2i} \pmod{n}$, and so the factor discovered will be n. If this occurs, it is common practice to run Algorithm 1 again either trying a different x_0 (the starting point) and/or changing f (the sequence/function/graph). If n is actually prime, then the only factor to be discovered is n, in which case changing x_0 or f will not help. To avoid this, one might use a primality-testing algorithm on n before attempting to use Pollard's Rho method.

3. Examples

3.1. Factoring 75 and 65

As a simple example, let us factor $n = 75$ following Algorithm 1. We will start with $x = y = 3$ and use the standard choice $f(x) = x^2 + 1$. We proceed as in Table 1, with all calculations being reduced modulo 75.

$n = 75$	step 1	step 2
x	3	10
y	3	26
$x = f(x)$	10	26
$y = f(f(y))$	26	5
$x - y$	-16	21
$\gcd(n, x - y)$	1	3

Table 1: Factorizing 75 using Pollard's Rho method.

In two steps, it has found that 3 is a factor of 75. We could actually observe what the algorithm is doing by drawing a directed graph representing the function $f(x)$ on $\mathbb{Z}_3$ and $\mathbb{Z}_5$, as shown in Figure 2. The sequence begins at the node labeled 0 in the graph on the left, and at the node labeled 3 in the graph in the middle. In two steps, the x value enters the loop in $\mathbb{Z}_3$, and because the loop consists of one value, it is identified immediately. While it might seem that starting at $x = y = 3$ is a favorable choice, discovering a factor in just two steps, in fact, this is as poor a starting point as possible for factoring 75, because of the structure of the graph on $\mathbb{Z}_3$.

To find an example that is one step more challenging, we can try factoring 65 starting at $x = y = 0$; see Table 2. In this case, we find a factor of 5 in three steps. Again, we can understand this as a walk on the graphs in the middle and right of Figure 2.

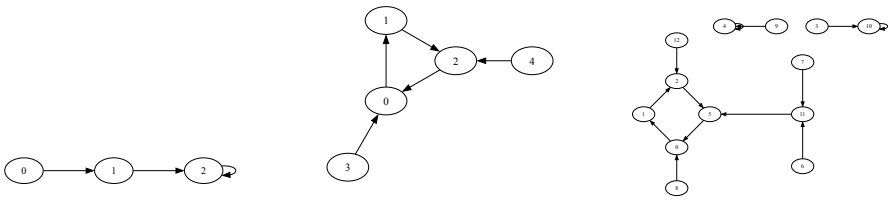

Figure 2: The graph induced by $x^2 + 1$ on $\mathbb{Z}_3$ (left), $\mathbb{Z}_5$ (middle), and $\mathbb{Z}_{13}$ (right).

$n = 65$	step 1	step 2	step 3
x	0	1	2
y	0	2	26
$x = f(x)$	1	2	5
$y = f(f(y))$	2	26	15
$x - y$	-1	-24	-10
$\gcd(n, x - y)$	1	1	5

Table 2: Factorizing 65 using Pollard's Rho method.

3.2. Average Behavior

Having seen how the algorithm works for two small examples, let us now look at what happens if we factor a selection of numbers with this technique. To produce examples over a range of values, we took the first 50,000 primes and discarded primes less than 600,000 to avoid smaller factors. We then took products of pairs of the remaining primes and used 100,000 of these for trial factorings. These numbers might be thought of as small RSA public keys, as they are a product of two similarly-sized primes.

In each case, we attempted to factor these numbers beginning with $x_0 = 2$. In some cases the method failed by finding n as a factor. In this event we picked a new starting x_0 value pseudorandomly in order to reduce the risk of exploring the same sequence[4]. The results are shown in Figure 3, where n is on the x-axis and the performance of Pollard's Rho method is shown at the y-axis.

Specifically, the graph shows the total number of steps required (including restarts, if needed) to factor each of our numbers. We can see that although there is some variation, the vast majority of numbers are factor in less than a couple of thousand steps. In fact, the mean number of steps is just over 545. For reference, we also

[4]Specifically, if we had to choose a new x_0 value K times, we used a function of K based on the cryptographic hash SHA256.

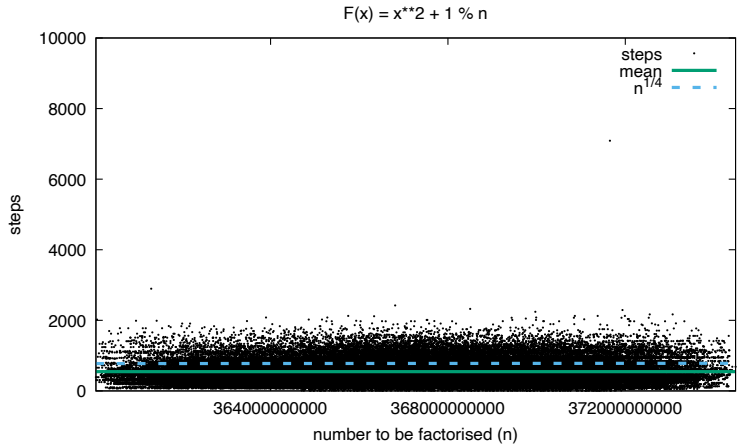

Figure 3: Performance of Pollard's Rho with $f(x) = x^2 + 1$ on a range of RSA-like numbers.

show $n^{1/4}$ where n is the number to be factored, which is a crude estimate for the number of steps based on both factors being roughly the same size and the Birthday Paradox.

4. Available Choices for f for Pollard's Rho

In 2022, the undergraduate cryptography class in Maynooth University raised the interesting question with the lecturer, *what functions can be used by Pollard's Rho method and why*? To answer this question, we checked a number of textbooks and found that many mention the idea that f should behave randomly, so random behavior of the sequence is regarded as important. In several of the textbooks, $f(x) = x^2 \pm 1$ is the only function explicitly mentioned (e.g., [7, 12, 17]), some mention quadratic or low degree polynomials (e.g., [19, 20]), while others mention that choosing any polynomial f will work (e.g., [8, 9, 10]). One source [4] suggests that the polynomial should be irreducible, though that is at odds with the original paper using $x^2 - 1$. With no clear statement on the permissible functions in the textbooks that we consulted, or in the original papers, we undertook to identify them. This is addressed in the following sections.

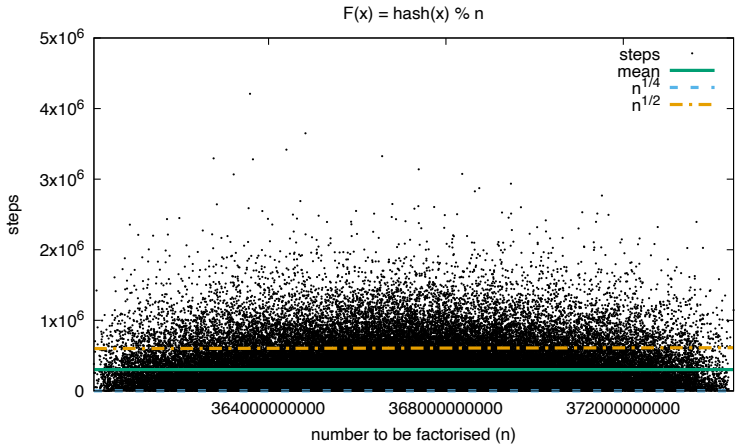

Figure 4: Performance of Pollard's Rho with a hash-based $f(x)$ on a range of RSA-like numbers. Note that $n^{1/4}$ is along the x-axis on this scale.

4.1. Requirements on f for Pollard's Rho

A significant part of the argument is that f appears like a random function. Cryptographers have specifically designed hash functions to behave as if their outputs are independent for two given inputs. Hence, one might consider that building the function f using a hash function might be a wise course of action. We attempt this, using the SHA256 hash function [13]. Our function converted $x \pmod n$ into (big-endian) binary, used SHA256 on the resulting sequence of bytes, converted the resulting bytes into an integer and then reduced it modulo n. That is, roughly speaking,

$$f(x) \equiv \texttt{SHA256}(x \pmod n) \pmod n.$$

The resulting performance on our RSA-like numbers is shown in Figure 4, and is terrible! Note that the y-axis of this graph has 500 times the range of Figure 3, and the mean number of steps is closer to $n^{1/2}$ than $n^{1/4}$. This is because there is a requirement on f for Pollard's argument to work, which is not always clearly stated in textbooks and papers. No matter which value of $x \pmod n$ we are using, f must give a well-defined function on $x \pmod p$ when p is a factor of n. In a sense, we are asking that there is a single outgoing arrow in the graph analogous to Figure 2 for $x \pmod p$, regardless of the value of $x \pmod n$ used to calculate it. If this is not the case, we are no longer searching for loops, and are really just hoping that $\gcd(x_i - x_{2i}, n) \neq 1$ by chance, which has performance similar to trial division.

For the function to be well defined, we require that for each factor $m \mid n$ we need that if $x \equiv y \pmod m$, then $f(x) \equiv f(y) \pmod m$, where we regard the function

as being evaluated on the equivalence classes in $\mathbb{Z}_n$, and then these are reduced to equivalence classes in $\mathbb{Z}_m$. Of our references, only two [9, 20] seem to explicitly mention this requirement. It is easy enough to see that all polynomials have this property, so other textbook authors may have been conscious of this requirement when they mention polynomials without explicitly articulating the requirement.

A weaker condition would be to require that for prime $p \mid n$ we need that if $x \equiv y \pmod{p}$ then $f(x) \equiv f(y) \pmod{p}$. Such functions would only find loops in the graph modulo p; however, this is still enough to factor a number, at the risk of possibly finding a factor p when we might otherwise have found a factor p^k.

4.2. Admissible f for Pollard's Rho for Square-Free n

In this section, we aim to identify the functions $f : \mathbb{Z}_n \to \mathbb{Z}_n$ so that for all $m \mid n$ we have $x \equiv y \pmod{m}$ implies $f(x) \equiv f(y) \pmod{m}$. For convenience, we make two definitions.

Definition 1. We say $f : \mathbb{Z}_n \to \mathbb{Z}_n$ satisfies the *Pollard Condition* if for all $m \mid n$ we have $x \equiv y \pmod{m}$ implies $f(x) \equiv f(y) \pmod{m}$.

Definition 2. We say $f : \mathbb{Z}_n \to \mathbb{Z}_n$ satisfies the *Weak Pollard Condition* if for all prime $p \mid n$ we have $x \equiv y \pmod{p}$ implies $f(x) \equiv f(y) \pmod{p}$.

The first of our two definitions is basically the idea of *congruence preserving* defined by Bhargava [1]. We will focus mainly on the Pollard Condition, for reasons that will become apparent.

First note that for factors m of n, if we have $g \equiv h \pmod{n}$ then $g \equiv h \pmod{m}$. This means that we can be a little careless of whether we regard f to be a function on $\mathbb{Z}_n$ or $\mathbb{Z}$, as the representative we choose will not make a difference when we work in $\mathbb{Z}_m$.

Theorem 1. *Let $n > 1$ be a square-free integer and suppose $f : \mathbb{Z}_n \to \mathbb{Z}_n$ is a function so that for all $m \mid n$ and for all $x, y \in \mathbb{Z}_n$ with $x \equiv y \pmod{m}$, we have $f(x) \equiv f(y) \pmod{m}$. Then f is a polynomial with integer coefficients.*

Proof. As n is square-free, we can write $n = p_1 p_2 \dots p_k$ where $p_1 < p_2 < \ldots < p_k$ are distinct primes. By the Pollard Condition on f, we know that for each p_i we have $f(x) \equiv f(y) \pmod{p_i}$ when $x \equiv y \pmod{p_i}$. So, we may define a function $f_{p_i} : \mathbb{Z}_{p_i} \to \mathbb{Z}_{p_i}$ to agree with $f \pmod{p_i}$.

Now, as $\mathbb{Z}_{p_i}$ is a field, every function $\mathbb{Z}_{p_i} \to \mathbb{Z}_{p_i}$ can be written as a polynomial of degree less than p_i by Lagrange interpolation. Say

$$f_{p_i}(x) = a_{i0} + a_{i1}x + a_{i2}x^2 + \ldots = \sum_{l=0}^{p_k - 1} a_{il}x^l,$$

where the $a_{ik} \in \mathbb{Z}_{p_i}$ and, for convenience, we have added zero coefficients so that each f_{p_i} has the same number of terms. As the p_i are relatively prime, using the Chinese Remainder Theorem, we can find $a_l \in \mathbb{Z}_n$ so that $a_l \equiv a_{il} \pmod{p_i}$.

Finally, consider the polynomial

$$p(x) = \sum_{l=0}^{p_k-1} a_l x^l.$$

If we consider $p(x) \pmod{p_i}$, we see that it must agree with $f_{p_i}(x)$, as its coefficients match. Thus, $p(x)$ agrees with $f_{p_i}(x)$ for each p_i and, by the Chinese Remainder Theorem, $p(x) \equiv f(x) \pmod{p_1 p_2 \dots p_k}$, or $p(x) \equiv f(x) \pmod{n}$. □

Hence, we see that in the case where n is a square-free number, it is both necessary and sufficient that f be a polynomial for f to satisfy the Pollard Condition. In fact, if n is square-free, the Pollard Condition and the Weak Pollard Condition are the same, thus this result also holds for the Weak Pollard Condition.

4.3. Admissible f for Pollard's Rho for Other n

Let us consider what would happen if n were not square-free. In this case, we could still factor the number, but it would have a more general form $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$. Here, if any of the $e_l > 1$ we will find that $\mathbb{Z}_{p_l{}^{e_l}}$ is not a field, and the presence of zero divisors means that we cannot write all functions as polynomials [6]. For example, on $\mathbb{Z}_4$ the function $f_4(0) = 0$, $f_4(1) = 0$, $f_4(2) = 0$, and $f_4(3) = 1$ cannot be written as a polynomial. However, this function does not satisfy even the Weak Pollard Condition with $p = 2$, as $1 \equiv 3 \pmod{2}$, but $f_4(1) \not\equiv f_4(3) \pmod{2}$. Exhaustively checking, it appears that (by chance?) all 64 functions on $\mathbb{Z}_4$ that satisfy our Weak Pollard Condition are also polynomials and so also satisfy our Pollard Condition.

However, the function on $\mathbb{Z}_8$ given by $f_8(x) = 0$ for $x = 0, 1, \dots, 6$ and $f_8(7) = 4$ satisfies both our Pollard Conditions and, again by exhaustively checking, can be seen to not be a polynomial in $\mathbb{Z}_8$. Likewise, the same can be verified for the function $f_9(x) = 0$ for $x = 0, 1, \dots, 7$ and $f_9(8) = 3$ on $\mathbb{Z}_9$. These example functions, shown in Table 3, can be combined via the Chinese Remainder Theorem with other functions that satisfy our Pollard Conditions to produce functions on $\mathbb{Z}_n$ for, say, $n = 8k$ or $n = 9k$. By a similar argument to that used in Theorem 1, these functions cannot be polynomials.

We can make this systematic by using a result of Carlitz [5] for checking if a function is a polynomial based on successive differences, though Bhargava [1] gives a detailed characterization of congruence preserving functions.

Theorem 2. *Let $n = p^e$ be a prime power, with either $p = 2, e \geq 3$ or $p > 2, e \geq 2$.*

x	0	1	2	3	4	5	6	7
$f_8(x)$	0	0	0	0	0	0	0	4
$f_8(x) \pmod 2$	0	0	0	0	0	0	0	0
$f_8(x) \pmod 4$	0	0	0	0	0	0	0	0

x	0	1	2	3	4	5	6	7	8
$f_9(x)$	0	0	0	0	0	0	0	0	3
$f_9(x) \pmod 3$	0	0	0	0	0	0	0	0	0

Table 3: Non-polynomial functions that satisfy the Pollard Condition.

Then the function $f : \mathbb{Z}_n \to \mathbb{Z}_n$ given by:

$$f(x) = \begin{cases} 0, & \text{for } x = 0, \ldots, n-2 \\ p^{e-1}, & \text{for } x = n-1 \end{cases},$$

satisfying the Pollard Condition but is not a polynomial.

Proof. The divisors of n are $m = p^k$ for $k \leq e$. If we check the Pollard Condition for $k < e$, then $f(x) \equiv 0 \pmod m$ for all x, and so the Pollard Condition holds. For the case $m = n$ the Pollard Condition just asks that f is well defined on $\mathbb{Z}_n$. Thus, we see f satisfies the Pollard Condition.

Theorem 1 of [5] gives a necessary and sufficient condition for f to be a polynomial. It requires that for all $x \in \mathbb{Z}_n$ and all $r \geq 0$ we have

$$\sum_{s=0}^{r} (-1)^{r-s} \binom{r}{s} f(x+s) \equiv 0 \pmod{p^{\nu(r)}},$$

where $\nu(r) = \min(e, \arg\max_k \{p^k | r!\})$. If we apply this test with $r = p^e - 1$ and $x = p^e - 1$, all the terms but one vanish, and we must check if

$$(-1)^r \binom{r}{0} f(p^e - 1) \equiv 0 \pmod{p^{\nu(r)}}.$$

Noting that none of the terms on the left are $0 \pmod{p^e}$, we want to check when $\nu(r) = p^e$. By counting the powers of p dividing $r! = (p^e - 1)!$, we find that

$$\arg\max_k \{p^k | r!\} = \frac{p^e - 1}{p - 1} - e.$$

If $p \geq 3$ then this is bigger than e when $e \geq 2$; however, if $p = 2$ we require $e \geq 3$. Thus, in these cases $\nu(r) = e$ and so f cannot be a polynomial. □

We can use Theorem 2 to construct non-polynomial examples for other n. For example, if we take $n = p^2 q$, for distinct odd primes p and q, and a is chosen so

that $a \equiv q^{-1} \pmod{p^2}$, then we can take

$$h(x) = \begin{cases} pqa & \text{when } x \equiv p^2 - 1 \pmod{p^2}, \\ 0 & \text{otherwise.} \end{cases}$$

Then we see that $h(x) \equiv 0 \pmod{q}$ and $h(x) \pmod{p^2}$ has the form required for Theorem 2. Thus, $h(x)$ will satisfy the Pollard Condition.

Though we need to know the factors of n to construct $h(x)$, we could nonetheless run Algorithm 1 with, say, $f(x) = x^2+1+h(x)$. We did this for a range of numbers with p in the first 1000 odd primes and q a prime of size just over 600,000. The performance is basically identical to taking $f(x) = x^2 + 1$ for two reasons. First, $h(x)$ only takes a non-zero value on $1/p^2$ of the integers, and the algorithm should run for about p steps. Second, even when $h(x) \neq 0$ we have that both $h(x) \equiv 0 \pmod{p}$ and $h(x) \equiv 0 \pmod{q}$. This means that on $\mathbb{Z}_p$ and $\mathbb{Z}_q$ we are generating the same sequences as $x^2 + 1$ would. Consequently, loops in these graphs will be found at the same time.

For this section, we conclude that in the non-square-free case, unless the only non-prime factor dividing n is 4, then there will be non-polynomial functions that satisfying the Pollard Condition.

4.4. Discussion

We can now make a number of interesting observations. First, having some form of the Pollard Condition is a requirement for the the random-function analysis of the method, which is not as well documented as it might be. Second, we have shown that for specific n, there may be non-polynomial functions that satisfy these conditions. However, for square-free n, we have also shown that only polynomials have these properties. This means that while the existence of these extra functions is interesting, we now know that we do not have to look outside the family of polynomials for general good choices of f. Finally, we note that satisfying the Pollard Condition is necessary but not sufficient to guarantee good behavior. The function $f(x) \equiv x+1 \pmod{n}$ is clearly a polynomial and has the Pollard Property; however, the graph on $\mathbb{Z}_p$ will be one long cycle and so the random-function analysis fails to apply.

5. Conclusion

We saw that a particularly careless choice of the function f for Pollard's Rho method can lead to poor performance, because f may not satisfy a condition that is often implicitly stated. We state two versions of this Pollard Condition and consider what choices of f are available. If n is a square-free number, then the two Pollard

Conditions are the same and the functions that satisfy them are *exactly* the polynomials on $\mathbb{Z}_n$. If n is not a square-free number, then, in addition to the polynomials, there usually exist non-polynomial functions that satisfy the Pollard Conditions. However, as the aim of Pollard's Rho method is to factor n, we do not know if n is square-free to begin with. Consequently, the only pragmatic choice for f in Pollard's Rho method is a polynomial.

Acknowledgments. Thanks to Ciarán Mac an Bhaird, the lecturer of the cryptography module who shared this question in a private communication. Thanks to Oliver Mason, Tom Dowling, Peter Clifford, Tony O'Farrell, and Gary McGuire for providing feedback on drafts. In particular Gary McGuire highlighted that Carlitz's condition could be used. Ronan O'Donnell was supported by a summer research internship from the Department of Mathematics and Statistics, Maynooth University.

References

[1] M. Bhargava, Congruence preservation and polynomial functions from $\mathbb{Z}_n$ to $\mathbb{Z}_m$, *Discrete Math.*, **173** (1–3) (1997), 15–21.

[2] R. Brent, An improved Monte Carlo factorization algorithm, *BIT*, **20** (2) (1980), 176–184.

[3] R. Brent and J. Pollard, Factorization of the eighth Fermat number, *Math. Comp.*, **36** (154) (1981), 627–630.

[4] D. Bressoud, Factorization and Primality Testing, *Undergrad. Texts Math.*, Springer Science & Business Media, New York, 2012.

[5] L. Carlitz, Functions and polynomials (mod p^n), *Acta Math.*, **9** (1) (1964), 67–78.

[6] Z. Chen, On polynomial functions from $\mathbb{Z}_n$ to $\mathbb{Z}_m$, *Discrete Math.*, **137** (1–3) (1995), 137–145.

[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, MIT press, Cambridge, MA, 2022.

[8] R. Guy, How to factor a number. In *Proc. Fifth Manitoba Conf. on Numerical Math., Winnipeg*, (1976), 49–89.

[9] J. Katz and Y. Lindell, *Introduction to Modern Cryptography: Principles and Protocols*, second edition, Chapman and Hall/CRC, Boca Raton, FL, 2015.

[10] D. Knuth, *The Art of Computer Programming: Seminumerical Algorithms, Volume 2*, Addison-Wesley Professional, Boston, MA, 2014.

[11] A. Lenstra and H. Lenstra, The Development of the Number Field Sieve, *Lecture Notes in Math.*, **1554** (1993).

[12] A. Menezes, P. Van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC press, Boca Raton, FL, 2018.

[13] National Institute of Standards and Technology, *Federal Information Processing Standards Publication 180-2: Secure Hash Standard*, Technical report, August 2002.

[14] J. Pollard, A Monte Carlo method for factorization, *BIT*, **15** (1975), 331–334.

[15] J. Pollard, Monte Carlo methods for index computation (mod p), *Math. Comp.*, **32** (143) (1978), 918–924.

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

[17] M. Purser, *Secure Data Networking*, Artech House, Inc., Boston, MA, 1993.

[18] R. Rivest, A. Shamir, and L. Adleman, Cryptographic communications system and method, 1977, US Patent 4,405,829.

[19] S. Wagstaff, *The Joy of Factoring*, Stud. Math. Libr. **68**. American Mathematical Society, Providence, RI, 2013.

[20] S. Wagstaff Jr., *Cryptanalysis of Number Theoretic Ciphers*, Chapman and Hall/CRC, Boca Raton, FL, 2019.