Analysis of Pollard's Rho Factoring Method

Soud K. Mohamed

NTNU (Nowegian university of Science and Technology)
Corresponding: soudmohamed124@gmail.com

Abstract. A comprehensive heuristic analysis of the Pollard's Rho Method (PRM) is given. The analysis is based on ultimate periods and tails distribution of sequences. If n is the composite number to be factored, then an improved version of PRM is developed which has an expected run time of $O(\sqrt[8]{n}\ln n)$ on a multi-core architecture which utilized a clever way of evaluating polynomials.

Keywords: ultimate period distribution; parallelization; tail distribution

Introduction

The security of cryptographic schemes based on RSA, the most widely used cryptosystem for secure data transmission on the Internet, relies on the factorization of large integers. Hence integer factorization algorithms must play a very importance role in secure communication on the Internet.

There are several algorithms of factorizing integers, the most common ones being: trivial division, Pollard's rho and $(p-1)$ methods, Fermat's method, Index calculus method, Elliptic curve method, Quadratic sieve method and number field sieve method. A good survey of modern factorization method is given by Lenstra [12]. Good analyses for most of the methods are found in the literature [1, 2, 3, 12, 16]. Trivial division is the most inefficient one of all the factorization methods. To find a factor p of a composite number n by trivial division, one needs to divide n by all primes less than $\sqrt{n}$. The best factorization algorithm so far is the number field sieve, which has sub-exponential run time as indicated by Lenstra [12]. The Pollard's rho method has a heuristic expected run time of $O(\sqrt{p})$, where p is the smallest factor of a composite number n [2, 3, 11, 12].

Pollard's rho method is one of the earliest factoring method in contemporary world which was developed by Pollard [14]. Since is invention in the seventies, there are very few analyses found in the literature: the well-known heuristic analysis based on the Birthday Paradox and the analysis which determining the probability of success discussed by Bach [2]. This paper proposed a new analysis of PRM, which is based on ultimate periods and

 doi:10.20944/preprints201911.0044.v1

tails distribution of sequences in $\mathbb{Z}_p$. An improve parallel implementation of PRM is proposed, which has expected run time $O((\ln p)\sqrt[4]{p})$.

Let p be a prime number ≥ 5 and $f(x)$ be a polynomial function over $\mathbb{Z}_p$, and $a_k = f(a_{k-1})$ be a sequence in $\mathbb{Z}_p$. Fixing a choice of coefficients of $f(x)$, then the average of the ultimate periods of a_k as a_0 runs over all element in $\mathbb{Z}_p$ is called the ultimate period of f with respect to p, and is denoted by $\mathrm{up}_p(f)$. We conjecture that for a polynomial $f(x) \in \mathbb{Z}[x]$ which is not a permutation, we have that $\mathrm{up}_p(f) \leq \sqrt{p}$.

If $a_k = f(a_{k-1})$ is a sequence, then Floyd Cycle Detection Algorithm (FCDA), as explained by Pollard [14] and Brent [3], picks every other element in a_k by using the (sub)sequence $b_k = f^2(b_{k-1})$, where $b_0 = a_0$. Every sequence in $\mathbb{Z}_p$ is ultimately periodic, hence it has a tail. The presence of tail makes the use of FCDA algorithm to detect collision inevitable as one is not sure when the sequence will enter a circle. With an appropriate selection of sequence generator, we show that the distribution of of the tails can be approximated by the function $\ln(p)\sqrt[4]{p}$. This can be used to improve the run time of PRM.

On the other hand, if the period of the sequence a_k is even, them application of FCDA reduces it by half. Referring to original FCDA as the FCDA *of index* 2, one can introduce an FCDA *of index* t. Then using the algorithm, the period of a_k is reduced by a factor of t, provided t divides the period of a_k. We show that by utilizing suitable FCDA of length t, one can obtain sequences b_k with expected period $\ln(p)\sqrt[4]{p}$.

Parallel implementation of factoring algorithms have been use in order to increase the efficiency of factoring algorithms [5, 6, 7]. For the PRM, two parallel implementation have been consider which have run time of $O(\sqrt{p})$ and $O(\sqrt{p}(\log m)^2/m)$ [6, 7]. An improved parallel implementation which with the help of fast polynomial multiplication is proposed.

A brief review of the literature is given in Section 1, while in Section 2 an investigation of ultimate periods of some special sequences and subsequences is discussed. In Section 3, the analysis of PRM is provided together with an improve parallelization of PRM. Finally conclusion and further research areas are given in Section 4.

1. Literature Review

In this section a literature review of the PRM analysis is given. Also a brief review of fast polynomial evaluation algorithms and it runtime is provided.

1.1. Pollard's Rho Factorization Method.

Let n be a square-free composite number, and let p be the smallest prime dividing n. If one picks numbers $a_0, a_1, a_2, \ldots$ from $\mathbb{Z}_n$ independently at random, then after at most $p+1$ such picks we have for the first time two numbers a_i, a_s with $i < s$ such

 doi:10.20944/preprints201911.0044.v1

that $a_i \equiv a_s \mod p$. Since n is composite, there is another prime factor q of n with $p < q$ such that $x_i \neq x_s \mod q$ with probability at least $1 - 1/q$ even when $a_i \equiv a_s \mod p$. Hence q must divide n, and so $\gcd(n, a_i - a_s) = q$ is a non-trivial divisor of n with probability $1-1/q$. The fact that $a_i \equiv a_s \mod p$ for $i < s$ will be referred to as **collision**. By using the Birthday Paradox, the expected number of cycles before one encounters a collision is roughly of the order of $\sqrt{p}$. This briefly explains the Pollard's Rho Factorization Method (PRM).

To generate a random sequence, a "random" function f on $\mathbb{Z}_p$ is used. Since all function on $\mathbb{Z}_p$ are polynomials, a polynomial $f(x)$ which is not a permutation polynomial can be use as a random map. The so called magical polynomial $f(x) = x^2 + c$, where $c \neq 0, -1, -2$ are widely used. The current analysis of the algorithm work under the assumption that $f(x)$ behave like a random function which has never been proved!

To find a factor p of composite integer n by PRM, a Floyd Cycle Detection Algorithm (FCDA) is used. The algorithm simultaneously compute $a_k = f(a_{k-1})$ and $b_k = f(f(b_{k-1}))$, where $b_0 = a_0$ and check whether $\gcd(a_j - b_j, n) > 1$ for some $j > 0$. In that situation we would have encounter a collision $\mod p$. The algorithm can be improved by accumulating the differences $a_j - b_j$ and compute the gcd in a batch, that is, by letting $z_j = \prod_{j=1}^{l} a_j - b_j$ and compute $\gcd(z_j, n)$ only when l is a multiple of m for $\log_2 n \leq m \leq \sqrt[4]{n}$ as shown by Pollard [14].

1.2. **Analysis of PRM in the literature.** As stated in the introduction the most well-known analysis of PRM found in the literature are the one based on the Birthday Paradox. Pollard [14] heuristically established that a prime factor p of a composite number n can be found in $O(\sqrt{p})$ arithmetics operations. The heuristics was based on the fact that the generating sequence, which was then the polynomial $x^2 + c$ for an integer $c \neq 0, 2$, is a random polynomial chosen among the p^p. The first rigorous analysis of PRM was done by Bach [2], who showed that the probability that a prime factor p is discovered before the kth iteration, for k fixed, is at least $\binom{k}{2}/p + O(p^{-3/2})$.

The original PRM uses FCDA to determine collision. Brant [3] proposed another cycle-finding algorithm which is on average about 36 percent faster than FCDA. He used his algorithm to improve PRM by about 24 percent. The improvement was obtained by omitting the terms $a_j - b_j$ in the product z_j if $k < 3r/2$, where r is the power of 2 less than or equal to sum of the tail and period [3, sec. 7]. However to find a r one has to find a cycle first!

Parallel implementation of PRM method was discussed by Brent [5]. He showed that if one uses m parallel machines, the expected heuristic run time of PRM is $O(\frac{\sqrt{p}}{\sqrt{m}})$, a nonlinear speed up! Crandall [7] came up with a new parallel implementation of PRM which is able to discover p using m machines

 doi:10.20944/preprints201911.0044.v1

in a heuristic time of $O(\frac{(\log_2 m)^2\sqrt{p}}{m})$. However, the heuristic has never been tested as noted by Brent [5].

1.3. **Fast Polynomial Evaluation.** Two type of fast polynomials evaluation are found in the literature. Sequential implementation uses a single core architecture, while parallel implementation utilizes a multi-core architecture. The well-known Horner's method, which is sequential, is often the default method for evaluating polynomials in many many computer libraries as noted by Reynolds [15]. The method has a runtime of $O(m)$, where m is the degree of the polynomial. There are two parallel algorithms which are the most common. Dorn's method, which is a generalized form of Horner's form, and has a run time of at least $2m/k + 2\log_2 k$, where k is the number of processors as indicated by Munro, Paterson [13]. The other one is Estrin's algorithm, also explained by Munro *et. al.* [13], which has run time of $O(\log_2 m)$, hence is very suitable for polynomial whose degree is a power of 2. The later was also discussed by Xu [17] and Reynolds [15].

2. Preliminaries

In this section we special sequence in $\mathbb{Z}_n$ and their properties, as well as the relation between sequence in $\mathbb{Z}_n$ and those in $\mathbb{Z}_p$ for a factor p of n.

Definition 2.1. [3, 8] A sequence a_k is a *ultimately periodic* if there are unique integers $\mu \geq 0$ and $\lambda \geq 1$ such that $a_k = a_{k+\lambda}$ for each $k \geq \mu$. The number λ is called the *ultimate period* or *period* of the sequence, while the number μ is the *tail* or *prepedriod* of the sequence. The sum $\lambda + \mu$ is the *length* of the sequence.

Example. The sequence $a_k : 1, 2, 5, 3, 10, 9, 13, 9, 13, 9, 13, 9, 13, 9, 13$ is ultimately periodic of ultimate period 2, tail 5 and length 7.

Let p be an odd prime. Then any natural sequence in $\mathbb{Z}_p$ is eventually periodic, since $\mathbb{Z}_p$ is a finite set. Consider a polynomial function $f(x)$ over $\mathbb{Z}_p$. If $a_j \in \mathbb{Z}_p^*$, then the sequence $a_k = f(a_{k-1})$ is called *the sequence in* $\mathbb{Z}_p$ *generated by* $f(x)$. Now if t be a positive integer, then $b_k = f^t(b_{k-1})$ is a subsequence of a_k. Note that the subsequences b_k defined above are obtained by picking every t-th element of a_k.

Example. Consider the sequence $a_k : 1, 2, 5, 3, 10, 9, 13, 9, 13, 9, 13, 9, 13, 9, 13$. Then $b_k : 1, 5, 10, 13, 13, 13, ...$ and $b_k : 2, 3, 9, 9, 9, ...$ are subsequences of a_k of ultimate period 1.

Consider the sequences $a_k = f(a_{k-1})$ in $\mathbb{Z}_p$ generated by polynomial f over $\mathbb{Z}_p$, and their corresponding subsequences $b_k = f^t(b_{k-1})$ for some positive integer t. Then the ultimate periods of a_k and b_k are related as shown in the following result.

Lemma 2.2. *Let $a_k = f(a_{k-1})$ be a sequence in $\mathbb{Z}_p$ generated by a polynomial f over $\mathbb{Z}_p$ with ultimate period λ and let $b_k = f^t(b_{k-1})$ for some positive integer t be a subsequence with ultimate period δ. Then*

i. *δ divides λ, and*
ii. *$\delta = m/\gcd(\lambda, t)$.*

Proof. i. b_k is obtained by picking elements with indices $0, \delta, 2\delta, 3\delta, \ldots$. Since $\delta \leq \lambda$ and b_k is ultimately periodic, it follows that $\delta|\lambda$.

ii. Since the sequences are ultimately periodic, there is an integer s such that $a_0 = b_\delta = a_{s\lambda} = a_{\delta t}$, so that $s\lambda = \delta t$, which implies that $\delta = \frac{\lambda}{t/s}$. We claim that $t/s = \gcd(\lambda, t)$. If $t/s = g$, then observe that $g|\lambda$, since δ is indeed an integer. On the other hand $\lambda = \delta g$, so that $s\delta g = \delta t$, which implies that $t = sg$. Hence g is a common divisor of δ and t. To show that $g = \gcd(\lambda, t)$ is left as an exercise. □

2.1. **Ultimate period of a Prime.** Let n be a square-free composite integer and p a prime factor of n. In this section an investigation of ultimate period of special sequences in $\mathbb{Z}_p$ is given. Also relationship of ultimate periods of sequences in $\mathbb{Z}_n$ and $\mathbb{Z}_p$ is discussed.

Let $f(x)$ is a fixed polynomial of degree m, then there are about p such sequences a_k. So as j runs from 1 to p we get p ultimate periods. Now if the coefficients of f are chosen at random, then there are p^m choices of coefficients, and we get p^{m+1} ultimate period. The average of these periods might be some thing useful.

Definition 2.3. Let $p > 3$ be a prime, let $f(x)$ be a polynomial over $\mathbb{Z}_p$ of degree m and let $a_k = f(a_{k-1})$ be a sequence in $\in \mathbb{Z}_p$.

1. The average ultimate period of a_k for a given coefficients of f as the initial seed a_0 runs over $\{1, 2, 3, ..., p\}$ is called the **ultimate period of** $f(x)$ with respect to p, and will be denoted by $\mathrm{up}_p(f)$.
2. Then the average ultimate period of a_k as the coefficients of f and the initial seed a_0 run over $\{1, 2, 3, ..., p\}$ is called the **ultimate period of** p **with respect to** $f(x)$, and will be denoted by $\mathrm{up}_f(p)$.

Let $f(x) = x^2 + 1$, $g(x) = f^2(x)$ and $h(x) = g^2(x)$. Then the ultimate periods of f, g and h with respect to the primes p between 3 and 225 Figure 1. From the figure we can observe that the ultimate period of a polynomial with respect to a a prime p is a most $\sqrt{p}$. We have the following conjecture.

Conjecture 2.4. *Let $p > 3$ be a prime and let $f(x)$ be a polynomial over $\mathbb{Z}_p$ which is not a permutation. Then*

$$\mathrm{up}_p(f) < \sqrt{p}. \tag{1}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 November 2019 doi:10.20944/preprints201911.0044.v1

p	$\mathrm{up}_f(p)$	$\mathrm{up}_g(p)$	$\mathrm{up}_h(p)$	$\sqrt{p}$	p	$\mathrm{up}_f(p)$	$\mathrm{up}_g(p)$	$\mathrm{up}_h(p)$	$\sqrt{p}$
5	2.5	2.0	1.0	2.2	101	6.5	4.6	3.3	10.0
7	1.8	1.5	1.5	2.6	103	6.4	4.7	3.0	10.1
11	2.5	2.3	1.1	3.3	107	6.7	4.7	3.8	10.3
13	2.8	1.8	1.4	3.6	109	6.6	5.1	3.8	10.4
17	2.9	2.1	2.6	4.1	113	6.3	4.9	4.0	10.6
19	2.9	2.3	2.5	4.4	127	7.3	5.3	4.4	11.3
23	3.1	2.4	1.9	4.8	131	6.9	5.4	3.8	11.4
29	4.1	3.4	2.2	5.4	137	6.9	5.2	4.1	11.7
31	4.2	3.0	2.3	5.6	139	7.4	5.5	3.7	11.8
37	4.1	2.9	2.3	6.1	149	8.0	5.8	4.3	12.2
41	4.1	3.1	2.8	6.4	151	7.7	5.4	4.4	12.3
43	4.3	3.2	2.1	6.6	157	8.5	5.8	4.1	12.5
47	4.4	3.3	2.2	6.9	163	7.5	5.7	4.5	12.8
53	3.8	2.8	2.3	7.3	167	7.7	5.5	4.5	12.9
59	5.2	4.4	2.7	7.7	173	8.4	6.5	4.8	13.2
61	5.2	4.1	2.5	7.8	179	7.8	6.0	4.3	13.4
67	4.5	3.5	2.7	8.2	181	8.4	6.2	4.6	13.5
71	5.3	4.0	2.7	8.4	191	9.3	7.3	4.3	13.8
73	5.3	3.9	3.3	8.5	193	8.7	6.6	5.1	13.9
79	5.1	3.6	3.4	8.9	197	8.9	7.1	4.5	14.0
83	5.7	4.3	3.0	9.1	199	8.6	6.5	4.5	14.1
89	6.5	4.7	3.8	9.4	211	8.8	6.6	4.4	14.5
97	6.0	4.8	3.9	9.8	223	9.4	6.9	5.9	14.9

FIGURE 1. Ultimate Period for Primes less than 225

Remark 2.5. The ultimate period of a prime p with respect to a polynomial f, $\mathrm{up}_f(p)$ is more interesting, because the sound parallelization of PRM found in the literature [5] use random coefficients and initial seeds. However, it exact computation is very time consuming.

Suppose that $a_k = f(a_{k-1})$ for some polynomial f over $\mathbb{Z}_p$. We determine the relationship of ultimate period in $\mathbb{Z}_n$ and in $\mathbb{Z}_{p_i}$ for $i = 1, 2, ..., s$. This will be very useful in the coming sections when we analyze the Rho Factorization Method.

Proposition 2.6. *Let $n = p_1 p_2 \cdots p_s$ be a square-free integer and let $a_k = f(a_{k-1})$ for some polynomial f over $\mathbb{Z}$. If a_k has ultimate periods $\lambda_1, ..., \lambda_s$ respectively in $\mathbb{Z}_{p_1}, \mathbb{Z}_{p_2} \ldots, \mathbb{Z}_{p_s}$, then it has ultimate period $\mathrm{lcm}(\lambda_1, \cdots, \lambda_s)$ in $\mathbb{Z}_n$.*

Proof. It can be easily deduced that the sequence $(a_k \mod p_1, \ldots, a_k \mod p_t)$ has ultimate period $\mathrm{lcm}(\lambda_1, \ldots, \lambda_s)$ in $\mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_s}$. Then the result follows from the fact that $\mathbb{Z}_n \simeq \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_s}$ via the isomorphism $\varphi : \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_s} \to \mathbb{Z}_n$ given by $\varphi(a_1, \ldots, a_s) = e_1 a_1 + \cdots + e_s a_s$, where e_i are elements in $\mathbb{Z}_n$ such that $1 = e_1 + \cdots + e_s$. □

Remark 2.7. The converse of Proposition 2.6 is indeed true, namely if a_k is a sequence in $\mathbb{Z}_n$ with ultimate period $m = \mathrm{lcm}(\lambda_1, ..., \lambda_s)$, then a_k has

 doi:10.20944/preprints201911.0044.v1

ultimate period λ_i in $\mathbb{Z}_{p_i}$ for some i from $1, ..., s$. One thing to note here is that it happen sometimes that all λ_i are the same!

Remark 2.8. In the next sections, all the analysis will be based on a prime factor p of a composite n. Although one will be working in n, but because of the classification theory of finite abelian groups, the structure of p in n is kept intact.

3. Analysis of Pollard's Rho Method

In this section analysis of rho method using tails distribution of sequence a_k generated by a polynomial f over $\mathbb{Z}_p$ and ultimate period of the sequences a_k is given. The analysis is based on two experiments done the researcher.

3.1. **Tail and period distributions.** As mentioned in the literature review, Brant [3] proposed another cycle-finding algorithm which he used to improve PRM by about 24 percent. The improvement was obtained by omitting the terms $a_j - b_j$ in the product z_j if $k < 3r/2$, where r is the power of 2 less than $\lambda + \mu$ [3, Sec. 7]. The motive behind this move was to compute gcd using terms which lie in a cycle. This means that the knowledge of a position in a sequence where a cycle start (the end of the tail) may improve the PRM.

Let n be a composite number and $a_k = f(a_{k-1})$ be a sequence in $\mathbb{Z}_n$. Suppose that λ is a period of the sequence $a_k = f(a_{k-1})$ and μ is its tail. Then using the help of FCDA, the algorithm PRM finds the smallest prime factor p of n after exactly $\lambda+\mu$ terms.The subsequence b_k of a_k found during the execution of PRM has ultimate period $\lambda_1 = \lambda/\gcd(2, \lambda)$ by Lemma 2.2. Now if FCDA is applied on b_k, $\gcd(b_j, b_{2j}) > 1$ would be obtained after exactly $\lambda_2 = \mu/2 + \lambda_1$ terms, and if λ was even, one would need to compute even smaller number of terms, namely $(\lambda + \mu)/2$. Hence, applying several FCDA would reduce the number of terms required to get $\gcd(a_j, b_j) > 0$ even when the order λ is not even, as it would reduce the tail by a power 2.

Remark 3.1. 1. In the original PRM and and its subsequent variants, the choice $b_k = f^2(a_{k-1})$ was preferred only because of computation complexity of higher degree polynomials. In other word, any power $t \geq 2$ would have suffices for successful computation of period of a_k, with increased number of computation as t increases.

2. The nature of the sequences b_k are perfectly suitable for implementation in computers with t core processors using Estrin's algorithm. Since the degree 2^t, then the algorithm has a run time of $O(t)$.

3.2. **Setting up of the experiments.** In the first experiment, tails of difference sequences in $\mathbb{Z}_p$ for different primes p were computed. For the prime between 4 and 5000 tails for the sequences generated by $f(x) = x^2 - 1$ and

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 November 2019 doi:10.20944/preprints201911.0044.v1

$g(x) = f^2(x)$ were computed for all initial seeds a_0 in $\mathbb{Z}_p$. For the primes in the other ranges a sample of size $\sqrt{p}$ was considered. The second experiment was conducted to determine the proportion of periods of sequences less than $\ln(p)\sqrt[4]{p}$, where p is a prime number. The sequence generator was $b_k = f^2(b_{k-1})$, where $f(x) = x^2 + c$. Two instances were used, where the first instance used random $a_0 = r_0$ and $c = r_2$, while the second one utilized random $a_0 = r_3$ and fixed $c = f_0$. For each prime p about $\sqrt{p}$ tails were computed.

Analysis of Experiment 1. Then tails mean of each prime p with respect to the polynomials f and g, $TM_f(p), TM_g(p)$, were computed. The the distribution of the TM were than compared to $LL(p) = \ln(p)\cdot\ln p, S(p) = \sqrt{p}$, and $LF(p) = \ln p \cdot \sqrt[4]{p}$. Figure 3.2 and 3.2 show comparisons of $TM_f(p), TM_g(p)$, $LL(p), S(p)$ and $LF(p)$.

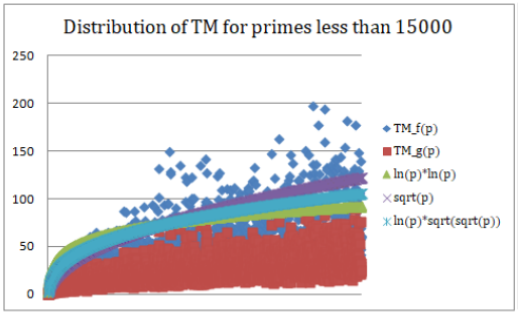
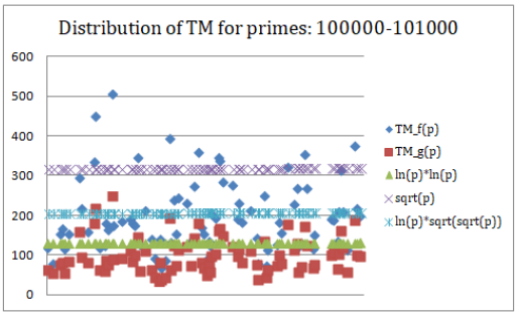
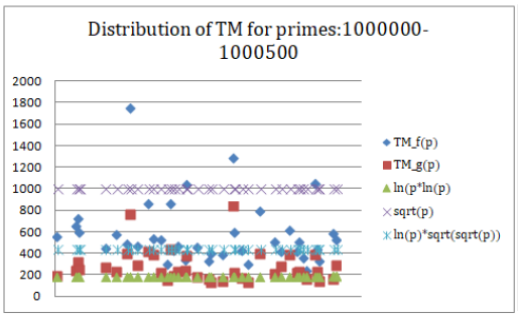


FIGURE 2





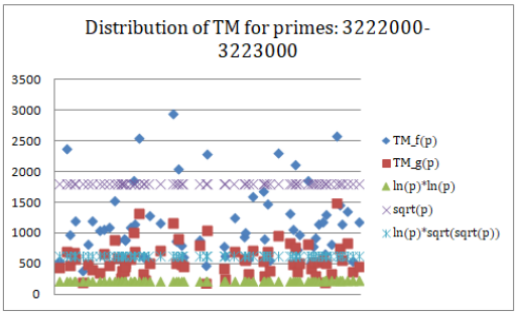


FIGURE 3

The top left diagram of Figure 3.2 shows that almost all $TM_g(p)$ lies bellow the function $LL(g), S(p)$, and $LF(p)$, while some of the $TM_f(p)$ are above the functions. But as p increases $TM_f(p)$ grows faster than $LL(p)$, but can be fitted into $LF(p)$, while more than half of $TM_f(p)$ lies below $LF(p)$, see top right diagram. As p moves to seven digits none of $TM_g(p)$ lies under $LL(p)$, while only a handful of $TM_g(p)$ are under $LL(p)$, see the bottom left diagram. In the bottom left diagram almost none of tails means are under $LL(p)$, a few of $TM_f(p)$ lies under $LF(p)$, while $TM_g(p)$ can be approximated by $LF(p)$.

The above analysis draws the following conclusion: If $a_k = f(a_{k-1})$ is a sequence generated by a non-permutation polynomial f over $\mathbb{Z}_p$, then, for

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 November 2019 doi:10.20944/preprints201911.0044.v1

some positive integer t, the expected tail of the sequence $b_k = f^t(b_{k-1})$, where $b_0 = a_0$ is $\ln(p) \cdot \sqrt[4]{p}$ with reasonable probability.

Remark 3.2. As p increases $LF(p)$ is moving below $TM_g(p)$, in other word probability of having a good number of tails below $LF(p)$ decreases. However, in the analysis we used $t = 2$, so an increase of t will remedy this problem.

Analysis of Experiment 2. The outcome of the experiment is shown in Table 1. Row two of the table shows probabilities for different primes when

TABLE 1. Probability of period $\leq \ln(p)\sqrt[4]{p}$

p	$P($ period $\leq \ln(p)\sqrt[4]{p}$ $)$	
	$a_0 = r_1, c = r_1$	$a_0 = r_3, c = f_0$
41149	0.822	1
61129	0.773	0.012
100189	0.797	0.092
111143	0.771	0
211177	0.692	0.847
411259	0.692	1
511279	0.660	0.172
611189	0.635	1
821295	0.628	0.101

random a_0, c are used. They are all greater than 0.5, hence good probabilities. On the other hand, when the parameter c is fixed (row 3), there are all kind of probabilities, 0, very small and very big.

There are at least few of inferences that can be drown from the table. First, it highlight the concern J. M. Pollard himself raised by Crandall [7] about the consequence of fixing c, as there will be then very few periods $O(\leq \lceil \ln p \rceil)$. Moreover, it strengthen the idea of using random parameters a_0 and c in the construction and execution of PRM, as one has a very wide iterative cycles to work with noted by Crandall [7]. In all the cases considered there were more than $\ln(p)^2$ different cycles.

One can then infer that the use of random parameters a_0 and c are important in implementation of PRM, both single and parallel modes. Moreover, $\ln(p)\sqrt[4]{p}$ is a good estimate of the expected period of a sequence b_k in heuristic terms.

Remark 3.3. From row 2 one can observe that as p increases the proportion of periods less than $\ln(p)\sqrt[4]{p}$ decreases. This can be offset by taking appropriate value of t. Note that in this example $t = 2$.

3.3. **A Parallelization.** Let n be a composite whole number to be factored, and $a_k = f(a_{k-1})$ be a sequence generated by the polynomial $f(x) = x^2 + c$,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 5 November 2019 doi:10.20944/preprints201911.0044.v1

in $\mathbb{Z}_n$, where $c \neq 0, -2$ is an integer. Choose t such that $2^t \leq \lceil \ln \sqrt{n} \rceil$. A set of t independently machines will be used and in each one of them, a PRM will be run using the sequences $b_k = f^t(b_{k-1})$ for $t = 2, ..., t$.

The algorithm. Each machine j should perform the following steps:

(1) Set the sequence b_k^j for $j = 2, ..., t$, and generate random parameters a_0 and c.
(2) Compute the first $M = \lceil \sqrt[8]{n} \ln \sqrt{n} \rceil$ terms, and store the term b_M^j.
(3) Set $N = \lceil \ln \sqrt{n} \rceil$, for $l = 0, N, 2N, ...,$ compute the terms b_k^j for $k > M$ and store the product

$$P_{l+N} = \prod_{i=l}^{l+N} (b_M^j - b_{M+i}^j) \mod n$$

(4) Compute $\gcd(P_{l+N}, n)$. If $\gcd(P_{l+N}, n) > 1$ in one of the machines, stop. Otherwise, repeat Step 3 until $i = M$.

Remark 3.4. Observe that the algorithm does not utilize a cycle detection algorithm in Step 3, instead it computes the difference $b_M - b_{M+i}$ for $i = 0, ..., M$. This move will improve the efficiency of the algorithm as it avoid computation of tow sequences and comparison associated to cycle detection algorithms.

Analysis of the algorithm. The analysis will be based on two implementation, namely a single core architecture and a multi-core architecture. The first thing to note is that the algorithm is a Monte Carlo algorithms, it has a probability of success, which we will try to determine. In Step 2 the algorithm is trying to escape the tail and enter a circle, an idea which was use with success by Brent [3]. Using Remark 3.2, Step 2 will reach its goal with a very good probability.

Case 1. For one core architecture, computation of the terms of b_k in Step 3 can be estimated as follow. By using the Horner's form, each term of b_k for $k > 0$ can be computed in $O(\ln \sqrt{n})$ arithmetic operations, since its degree is less than $\ln(\sqrt{n})$. Step 3 will require $O(\sqrt[8]{n}(\ln \sqrt{n})^2)$ arithmetic operations from each machine. Apart from computation of b_k^i in Step 3 and 4, there is multiplication of terms and computation of gcd. But it is plausible to consider the cost of only a_k and ignore the other. The motive behind this move is the fact that computation of b_k is more costly than multiplication and gcd computations. Moreover, the cost bound set on b_k is the worst-case scenario, in other word, very few machine will come near the bound. Now the expected cost of each machine is $O(\sqrt[8]{n} \ln \sqrt{n} \cdot \ln \sqrt{n} + \sqrt[8]{n} \ln \sqrt{n} \cdot \ln \sqrt{n}) = O(\sqrt[8]{n} \ln \sqrt{n} \cdot \ln \sqrt{n})$, where $\ln \sqrt{n}$ is the cost of computation of terms as in the sense of [6].

Case 2. Using Estrin's method on multi-core architecture, each b_k in Step 3 can be computed in $O(t)$ arithmetic operations. Hence the step will require $O(\sqrt[8]{n} \ln \sqrt{n})$ arithmetics operations from each machine. By using the

same assumption as in Case 1, the expected cost of each machine is now $O(\sqrt[8]{n}\ln\sqrt{n})$, vast improvement from Case 1.

4. Summary and Conclusion

Analysis of Pollard's rho factorization method is given which is based on the distribution of periods and tails of sequences. An improved parallelization of the method is given with a run time of $O(\sqrt[8]{n}\ln\sqrt{n})$. The parallelization can be improved with an improve in the polynomial evaluation algorithms and an increase in the number of cores in a machine.

References

[1] B. R. Ambedkar, S. S. Bedi (2011), *A New Factorization Method to Factor RSA Public Key Encryption*, IJCSI, **90** (6.1), 130-155.
[2] E. Bach (1991), *Toward a Theory of Pollard's Rho Method*, Information and Computation **90**, 130-155.
[3] R. P. Brent (1980), *An Improved Monte Carlo Factorization Algorithm*, BIT **20**, 176-184.
[4] R. P. Brent (1990), *Number Theory and Cryptography*, Cambridge University Press.
[5] R. P. Brent (1990), *Parallel Algorithms for Integer Factorization*,Number theory and cryptography 154, 26-37.
[6] R. P. Brent (1999), *Ssome Parallel Algorithms for Integer Factorization*, Porc. Europar'99, Toulouse, Sept 1999.
[7] R.E. Crandall, Parallelization of Pollard-rho factorization, preprint, 23 April 1999
[8] A. Dubickas, P. Plankis (2008), *Periodicity of Some Recurrence Sequences Modulo M*, Combinatorial Journal of Number Theory, **8**, # A42.
[9] G. ESTRIN (1960), *Organization of Computer System–The Fixed Plus Variable Structure Computer*, Proceedings Western Joint Computer Conference, May, 1960, AFIPS Press, Montvale, N J, pp. 33-40.
[10] A. K. Koundinya, G. Harish, N. K. Srinath, G. E. Raghavendra, Y. V. Pramod, R. Sandeep, P. G. Kumar (2013), *Performance Analysis of Parallel Pollard's Rho Factoring Algorithm* , International Journal of Computer Science & Information Technology (IJCSIT), **5** (2), 157-164 .
[11] N. Koblitz (1994), *A Course in Number Theory and Cryptography*, 2nd Ed., Springer, USA.
[12] A. K. Lenstra (2000), *Integer Factoring*, Designs, Codes Cryptography, **19**, 101-128.
[13] I. Munro, M. Paterson (1973), *Optimal Algorithms for Parallel Polynomial Evaluation*, JCSS, **7**, 189-198.
[14] J. M. Pollard (1975), *A Monte Carlo Method for Factorization*, BIT **15**, 331-334.
[15] G. S. Reynolds (2010), *Investigation of different methods of fast polynomial evaluation*, Master thesis, The University of Edinburgh.
[16] R. D. Silverman (1987), *The Multiple Polynomial Quadratic Sieve*, Mathematics of Computation, **48**(177), 329-339.
[17] S. Xu (2013), *Eficient Polynomial Evaluation Algorithm and Implementation on FPGA*, Mater thesis, Nanyang Technological University.

Email address: shkhalifa@hotmail.com, soudkm@stud.ntnu.no