Pollard's Rho Algorithm for Discrete Logarithm Computation *

Franklin Harding

November 3, 2023

1 Introduction

Let G be a finite cyclic group for which g is a generator and h is a random group element. The *discrete logarithm problem* (DLP) is to find the smallest non-negative integer x so that $h = g^x$. The existence of an efficient algorithm for solving the DLP would render Diffie-Hellman (DH) key exchange, ElGamal encryption, and various other cryptosystems insecure. In practice, G is typically $\mathbb{Z}_p^*$ or the points of an elliptic curve over a finite field of order p (p prime).

Pollard suggested a method for solving DLPs in $\mathbb{Z}_p^*$ [Pol78] based on his "rho" idea initially exploited for factoring [Pol75]. The resulting algorithm takes roughly $O(\sqrt{|G|})$ group operations and has negligible space requirements. When combined with the methods of Pohlig and Hellman [PH78], the run time is reduced to $O(\sqrt{p})$ where p is the largest prime dividing $|G|$.

Shoup demonstrated that generic discrete log algorithms (that succeed with high probability) must perform at least $\Omega(\sqrt{p})$ group operations where p is the largest prime dividing the order of G [Sho97]. In other words, Pollard's rho algorithm is essentially the best we can do. While sub-exponential time algorithms exist for $\mathbb{Z}_p^*$, Pollard's rho algorithm is still the weapon of choice for elliptic curve DLPs. However, contemporary implementations differ in many ways from Pollard's original rho algorithm. We will discuss these variations and analyze their performance empirically.

2 Preliminaries

We use $\mathbb{N}_0$ to the denote the set of non-negative integers, i.e. $\mathbb{N} \bigcup \{0\}$. For fixed $n \in \mathbb{N}$, we write $\mathbb{Z}_n$ for the group of integers modulo n with multiplication

*This report was written for a single CS 406 projects credit and was supervised by Jiayu Xu. I did not end up having quite enough time to perform all of the experiments that I would have liked to, but at least one is given in this report. The source code for my implementation of Pollard's rho algorithm with Teske's 20-adding walk is available at https://github.com/fharding1/pollard-rho.

modulo n, and $\mathbb{Z}_n^\times$, denotes the group of units of $\mathbb{Z}_n$. We will mostly deal with $\mathbb{Z}_p$ where p is prime, in which case $\mathbb{Z}_p^\times = \mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$. When we deal with a generic group G, we assume that for all $g, h \in G$ we can efficiently compute gh, and represent each element as a unique binary string.

3 Pollard's rho method

Let W be a finite set and $f : W \to W$ be a mapping. For any $w_0 \in W$, the sequence $w_0, f(w_0), f(f(w_0)), \ldots$ is eventually periodic since W is finite. This idea can be used to solve many computational problems and is referred to as the *rho method*. As a warm up and to introduce some terminology, we will describe Pollard's original rho algorithm for solving the DLP in $\mathbb{Z}_p^*$; but first, some motivation.

Let G be a group for which g is a generator and h is an arbitrary element. Recall that the DLP is to find the smallest non-negative integer x so that $h = g^x$. If we have some $a, a', b, b' \in \mathbb{Z}$ so that $h^a g^b = h^{a'} g^{b'}$ then we can substitute $h = g^x$ yielding $(g^x)^{a-a'} = g^{b'-b}$ which implies

$$x(a - a') \equiv b' - b \pmod{|G|},$$

and we can easily compute x so long as $a - a'$ is relatively prime to $|G|$. A naive method for solving DLPs based on this idea is to repeatedly randomly sample $a, b \in \mathbb{Z}$ and store $h^a g^b$ in a table until you get a collision. However, the space requirements for this method are too significant for the large groups used in modern cryptography. As you might have guessed, one can combine this idea with Pollard's rho method to produce a more practical algorithm—but first we need to address the elephant in the room: *what if $|G|$ has small factors?*

At first blush this seems like a big issue. If we take $G = \mathbb{Z}_p^*$ for some prime $p > 2$, then $|G| = p-1$ always has even order, so our toy algorithm fails with high probability. Pollard originally used trial multiplication rather than inversion to solve the linear congruence, which is fairly efficient for most primes. We will use the Pohlig-Hellman algorithm, which allows us to solve a DLP in any group by first solving DLPs in its prime power order subgroups, and then reconstructing a solution to the original DLP via the Chinese Remainder Theorem [PH78]. The "strong primes" used in modern cryptography are picked so that $p - 1$ has no small prime factors, so $a - a'$ is relatively prime to those prime power subgroup orders with high probability.

3.1 $\mathbb{Z}_p^*$ DLPs

Let p be a prime for which g is a primitive root, and h be a random element of $\mathbb{Z}_p^*$. Recall that the DLP is to find the smallest non-negative integer x so that

$h = g^x$. Pollard's original method is as follows. Let $f : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ be given by

$$f(y) = \begin{cases} hy & \text{if } 0 < y < \frac{1}{3}p \\ y^2 & \text{if } \frac{1}{3}p < y < \frac{2}{3}p \\ gy & \text{if } \frac{2}{3}p < y < p \end{cases}.$$

Define the sequence (y_k) via the rule $y_0 = 1$ and $y_{i+1} = f(y_i)$ for all $i \in \mathbb{N}_0$. We refer to f as Pollard's original *iterating function*, and (y_k) as a *psuedorandom walk* in $\mathbb{Z}_p^*$. Then there are sequences (a_i) and (b_i) so that $y_i = h^{a_i} g^{b_i}$ which satisfy $a_0 = b_0 = 0$ and the recurrence relations:

$$a_{i+1} = \begin{cases} a_i + 1 & \text{if } 0 < y < \frac{1}{3}p \\ 2 * a_i & \text{if } \frac{1}{3}p < y < \frac{2}{3}p \\ a_i & \text{if } \frac{2}{3}p < y < p \end{cases} \qquad b_{i+1} = \begin{cases} b_i & \text{if } 0 < y < \frac{1}{3}p \\ 2 * b_i & \text{if } \frac{1}{3}p < y < \frac{2}{3}p \\ b_i + 1 & \text{if } \frac{2}{3}p < y < p \end{cases}$$

As $\mathbb{Z}_p^*$ is finite (of order $p - 1$), the psuedorandom walk (y_k) is eventually periodic. That is, there exist distinct indices $i, j \in \mathbb{N}_0$ such that $y_i = y_j$. In order to find them, we apply Floyd's *cycle-finding* algorithm. Then we have $h^{a_i} g^{b_i} = h^{a_j} g^{b_j}$ and since $h = g^x$ we have $(g^x)^{a_i - a_j} = g^{b_j - b_i}$ thus

$$x(a_i - a_j) \equiv b_j - b_i \pmod{p-1}.$$

Pollard's original rho algorithm for solving the DLP in $\mathbb{Z}_p^*$ can easily be modified to work for other groups and indeed, his iterating function f is not unique. However, the fact that we can find sequences (a_i) and (b_i) as described is important. This motivates the following definitions:

Definition 1. *Let G be a finite cyclic group and $g, h, y \in G$. An **iterating function** on G is a function $f : G \to G$ such that for any $a, b \in \mathbb{N}_0$ such that $y = h^a g^b$ we can efficiently compute $a', b' \in \mathbb{N}_0$ such that $f(y) = h^{a'} g^{b'}$.*

Definition 2. *Let G be a finite cyclic group and f be an iterating function on G. A sequence (y_k) is a **walk** in G with F if $y_0 \in G$ and $y_{k+1} = f(y_k)$ for all $k \in \mathbb{N}_0$.*

Clearly f is an iterating function, and (y_k) is a walk in $\mathbb{Z}_p^*$ with f. In the next section, we will discuss new walks with new iterating functions as proposed by Teske.

4 Teske's *r-adding* and *r+q-mixed* walks

Teske came up with new walks and proved both empirically and analytically that they perform better than Pollard's original walk [Tes01]. The idea is essentially unsurprising: an iterating function with "more rules" ends up "looking more random" so we find a match in fewer steps. Walks without a squaring rule are called *r-adding*; formally:

Definition 3. *Let G be a finite cyclic group, $r \in \mathbb{N}$, and $m_1, \ldots, m_r \in G$. Furthermore, let $v : G \to \{1, 2, \ldots, r\}$ be a hash function and f be an iterating function on G. A walk (y_k) on G with f is called r-**adding** if f is of the form $f(y) = y * m_{v(y)}$ for all $y \in G$.*

Teske also generalized Pollard's original walk as *r+q-mixed walks*, which have both multiplication and squaring steps:

Definition 4. *Let G be a finite cyclic group, $r, q \in \mathbb{N}$, and $m_1, \ldots, m_r \in G$. Furthermore, let $v : G \to \{1, \ldots, r+q\}$ be a hash function and f be an iterating function on G. A walk (y_k) on G with f is called $r+q$-**mixed** if f is of the form:*

$$f(y) = \begin{cases} y * m_{v(y)} & \text{if } v(y) \in \{1, \ldots, r\} \\ y^2 & \text{if } v(y) \in \{r+1, \ldots, r+q\} \end{cases}.$$

So Pollard's original iterating function is a $2+1$-mixed walk. The difference between Teske's walks and Pollard's original walk is significant; she found that a 16-adding walk yields a speed-up by a factor of at least 1.25 compared to Pollard's original walk. Also, Teske used Knuth's multiplicative hash function, but wrote "in practical applications, simpler hash functions that can be evaluated faster are certainly preferable."

4.1 Hash functions

Teske's r-adding and $r+q$-mixed walks require the use of a hash function in order to partition the group into roughly equal sized sets. In the case of Pollard's original rho algorithm as a $2+1$-mixed walk, this "hash function" is essentially $v : \mathbb{Z}_p^* \to \{1, 2, 3\}$ given by

$$v(y) = \begin{cases} 1 \text{ if } 0 < y < p/3 \\ 2 \text{ if } 2/3p < y < p \\ 3 \text{ if } 1/3p < y < 2/3p \end{cases}.$$

There is a subtlety of the manner in which a group G is partitioned and the iterating function f is designed in the case of $r+q$-mixed walks; if e is the identity for G, then we would like $v(e) \in \{1, \ldots, r\}$ and $m_{v(e)} \neq e$, otherwise $f(e) = e^2 = e$ and the walk becomes constant.

Hash functions suitable for this application should be fast and produce a roughly even distribution, but do not need to be cryptographic. Examples include FNV32 and Murmur3.

5 Cycle-finding

Let W be a finite set, and $f : W \to W$ be a mapping. Define the sequence $(w_k)_{k \in \mathbb{N}_0}$ via the rule $w_0 \in W$ and $w_{i+1} = f(w_i)$. Then (w_k) is eventually

periodic, so there exist distinct indices $i, j \in \mathbb{N}_0$ such that $w_i = w_j$. The task of finding such indices is called *finding a match*. More generally, there exist $\lambda \in \mathbb{N}$ and $\mu \in \mathbb{N}_0$ such that $w_0, \ldots, w_{\mu+\lambda-1}$ are distinct and $w_k = w_{k+\lambda}$ for all integers $k \geq \mu$. λ is called the *period* and μ is called the *preperiod*. The process of finding the preperiod and period is called *cycle-finding*. Finding just one match suffices for our purposes (that is, we don't care about λ and μ) but most match-finding algorithms are cycle-finding algorithms and vice versa. In this paper, both refer to finding a match.

One of the most basic cycle-finding algorithms (used in Pollard's original rho algorithm) is known as Floyd's cycle-finding algorithm, and works as described in figure 1.

1. Define $a := w_0$, $b := w_0$, and $i := 0$
2. Repeat $i := i + 1$, $a := f(w_0)$, $b := f(f(w_0))$ until $a = b$
3. Return $(i, 2i)$

Figure 1: Floyd's cycle-finding algorithm [Bre80]

If it does not cost much to evaluate f, then this is a fairly space and time efficient algorithm. On the other hand, if evaluating f is expensive, then there are better algorithms such as Brent's [Bre80]. Schnorr and Lenstra furter improved upon Brent's algorithm for a specific application of cycle-finding, and their method was generalized by Teske [Tes98].

The method of *distinguished points* proposed by van Oorschot and Wiener [OW99] allows for the paralellization of Pollard's rho algorithm. In this method, each processor chooses a random group element to start its walk. Upon each iteration the processor checks whether the group element meets some efficiently testable property, such as the number of leading zeroes in its binary representation. If the group element satisfies that criteria it is considered a distinguished point, and it along with the initial group element is sent to a central repository shared by all processors (otherwise the walk just continues). If the same group element occurs twice in the central repository, then their walks can be reconstructed to find a collision. Given the parallel speed-up, this method essentially renders conventional cycle-finding obsolete.

6 Group automorphisms

Let $G = E(\mathbb{F}_q)$ where $q = p^n$ for some prime p and $n \in \mathbb{N}$. The negation map $(x, y) \mapsto (x, -y)$ is an automorphism on G of order 2, and if an iterating function f on G is defined so that $f(y) = f(-y)$, then it is actually defined on the equivalence classes of G under $\pm$. This reduces the average number of iterations to find a match by a factor of $\sqrt{2}$ [BLS11]. If $p = 2$ (e.g. Koblitz curves), then a similar idea is to use the Frobenius map $(x, y) \to (x^2, y^2)$, which

results in a speed up by a factor of $\sqrt{2n}$ [GLV00]. Either method introduces some issues with "fruitless cycles" which require some additional work to avoid [BKL]. These are typically the only easily computed automorphisms of small order for elliptic curves.

7 Experiments

We implemented a number of variations on Pollard's rho algorithm in C using the GNU Multiple Precision (https://gmplib.org/) and Permuted Congruential Generator (https://www.pcg-random.org/) libraries for arbitrary-precision math and random number generation respectively. We tested each variation against prime-order subgroups of $\mathbb{Z}_p^*$ and groups of points of an elliptic curve over prime-order finite fields (denoted $E(\mathbb{F}_p)$). For each type of group, we work with group orders between 10^{n-1} and 10^n for $3 \leq n \leq 13$.

Teske showed that if the iterating function on a finite group G behaves like a random mapping and her cycle-finding algorithm based on Schnorr and Lenstra's is used, then we expect a match in approximately $1.416\sqrt{|G|}$ iterations. Thus, she used the metric

$$L := \frac{\text{\# of iterations until a cycle is found}}{\sqrt{|G|}}$$

to evaluate the performance of her walk functions experimentally. We will do the same. Our code is available on Github at https://github.com/fharding1/pollard-rho.

7.1 Walk method

We compared Pollard's original walk method to a Teske 20-adding walk with Floyd's cycle-finding algorithm and FNV32 hashing.

digits of subgroup order	average L	DLPs
3	3.105	10000
4	3.121	10000
5	3.158	10000
6	3.167	10000
7	3.149	10000
8	3.176	10000
9	3.191	10000
10	3.179	10000
11	3.176	10000
12	3.187	10000
13	3.159	10000

(a) Teske 20-adding, FNV32, Floyd

digits of subgroup order	average L	DLPs
3	4.138	10000
4	4.031	10000
5	4.043	10000
6	4.003	10000
7	3.990	10000
8	3.980	10000
9	4.023	10000
10	4.022	10000
11	4.017	10000
12	4.008	10000
13	4.073	10000

(b) Pollard

Figure 2: Solving DLPs in prime-order subgroups of $\mathbb{Z}_p^*$

8 Conclusion

Pollard's rho algorithm is known to be one of the fastest and most practical algorithms for elliptic-curve DLPs, however contemporary implementations take advantage of numerous lesser-known optimizations. Teske came up with new methods of taking random walks in the group called r-adding and $r+q$-mixed walks. We implemented Pollard's rho algorithm in C and compared a 20-adding walk with FNV32 hashing to Pollard's original rho algorithm, and showed that Teske's method is significantly faster. Future work would be to study the use of the negation map, other cycle-finding algorithms and hash functions, and the method of distinguished points.

References

[BKL] Joppe W. Bos, Thorsten Kleinjung, and Arjen K. Lenstra. "On the Use of the Negation Map in the Pollard Rho Method". In: *Algorithmic Number Theory*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 66–82. ISBN: 3642145175.

[BLS11] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. *On the correct use of the negation map in the Pollard rho method*. Cryptology ePrint Archive, Paper 2011/003. https://eprint.iacr.org/2011/003. 2011. URL: https://eprint.iacr.org/2011/003.

[Bre80] Richard P. Brent. "An improved Monte Carlo factorization algorithm". In: *BIT* 20.2 (1980), pp. 176–184. ISSN: 0006-3835.

[GLV00] Robert Gallant, Robert Lambert, and Scott Vanstone. "Improving the parallelized Pollard lambda search on anomalous binary curves". In: *Mathematics of computation* 69.232 (2000), pp. 1699–1705. ISSN: 0025-5718.

[OW99] Paul C van Oorschot and Michael J Wiener. "Parallel collision search with cryptanalytic applications". In: *J. Cryptology* 12.1 (Jan. 1999), pp. 1–28.

[PH78] S. Pohlig and M. Hellman. "An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (Corresp.)" In: *IEEE transactions on information theory* 24.1 (1978), pp. 106–110. ISSN: 0018-9448.

[Pol75] J. M. Pollard. "A monte carlo method for factorization". In: *BIT* 15.3 (1975), pp. 331–334. ISSN: 0006-3835.

[Pol78] J. M. Pollard. "Monte Carlo Methods for Index Computation (mod p)". In: *Mathematics of computation* 32.143 (1978), pp. 918–. ISSN: 0025-5718.

[Sho97] Victor Shoup. "Lower Bounds for Discrete Logarithms and Related Problems". In: *Advances in Cryptology — EUROCRYPT '97*. Springer Berlin Heidelberg, 1997, pp. 256–266. DOI: 10.1007/3-540-69053-0_18. URL: https://doi.org/10.1007/3-540-69053-0_18.

[Tes01] Edlyn Teske. "On Random Walks for Pollard's Rho Method". In: *Mathematics of Computation* 70.234 (2001), pp. 809–825. ISSN: 00255718, 10886842. URL: http://www.jstor.org/stable/2698783 (visited on 12/06/2022).

[Tes98] Edlyn Teske. "Speeding up Pollard's rho method for computing discrete logarithms". In: *Algorithmic Number Theory*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 541–554. ISBN: 9783540646570.