POLLARD'S RHO ALGORITHM FOR ELLIPTIC CURVES

AARON BLUMENFELD

Abstract. Elliptic curve cryptographic protocols often make use of the inherent hardness of the discrete logarithm problem, which is to solve $kG = P$ for k. There is an abundance of evidence suggesting that elliptic curve cryptography is more secure than the classical case. One reason for this is the best known general-purpose algorithm to solve the elliptic curve discrete logarithm problem is Pollard's Rho algorithm, which has exponential time complexity $O(\sqrt{n})$, where n is the order of the elliptic curve.

In this paper, we explore Pollard's Rho algorithm. In particular, we show that it only requires $O(1)$ space complexity. This is an astronomical improvement over the related Baby-Step Giant-Step algorithm, which requires $O(\sqrt{n})$ time *and* space complexity. We also investigate different methods of defining the sequence of points used in Pollard's Rho algorithm and discuss their effects on efficiency.

1. Introduction

Given a cyclic group G, the discrete logarithm problem asks us to solve $g^x = y$ for x, where $g, y \in G$ and $x \in \mathbb{Z}_{\geq 0}$. (In fact, x can be taken to be modulo $n = |G|$.) There are numerous general algorithms for solving the discrete logarithm problem in any cyclic group G. These are all applicable to elliptic curves, although not necessarily efficient. (Technically, an elliptic curve may not be cyclic, but the base point generates a cyclic subgroup.) The most naive method is an exhaustive search: simply iterate over the possible values of x and try each one.

An improved algorithm is Baby-Step Giant-Step. This algorithm creates lists of values g^i and yg^{-mj} for all $0 \leq i, j < m$, where $m = \lceil \sqrt{n} \rceil$. Once a match is found, $g^i = yg^{-mj}$, so $x \equiv i + mj \pmod{n}$. A match is guaranteed to be found because x can be written uniquely as $x = i + mj$ in base m. The problem with this algorithm is that it also requires $O(\sqrt{n})$ storage space.

An improvement is to use Pollard's Rho algorithm, which also has $O(\sqrt{n})$ time complexity, but only $O(1)$ space complexity. This algorithm is described in detail in the next section.

2. Pollard's Rho Algorithm

In this section, we describe the original version of Pollard's Rho algorithm. Although this algorithm is a general-purpose algorithm that can be applied to any cyclic group G, we will describe it using the notation of elliptic curves.

Let E be an elliptic curve over $\mathbb{F}_{2^k}$ with n points. Our goal is to solve for k in the equation $kP = Q$.

Date: December 5, 2015.

We should point out that unlike Baby-Step Giant-Step, Pollard's Rho algorithm is probabilistic, which means it will finish within the expected running time with a high probability. But there is no guarantee it will finish this quickly.

We first partition E into three subsets S_1, S_2, and S_3. The S_i should be roughly the same size. For example, we can do this by reducing the x-coordinate modulo 3 (and making an arbitrary choice for which set to place ∞ in). Or we could use projective coordinates and reduce the y-coordinate modulo 3 (since $\infty = (0 : 1 : 0)$ in projective coordinates).

We now choose a starting point $A_0 = \alpha P$ as a scalar multiple of the base point. α could simply be 1, or it could be randomly chosen modulo n. We now let

$$A_{i+1} = f(A_i) = \begin{cases} A_i + P \text{ if } A_i \in S_1, \\ 2A_i \text{ if } A_i \in S_2, \\ A_i + Q \text{ if } A_i \in S_3. \end{cases}$$

The terms of the sequence A_i then take the form $A_i = a_j P + b_j Q$. Once we discover an equality $A_{i_1} = A_{i_2}$, we have $a_{j_1} P + b_{j_1} Q = a_{j_2} P + b_{j_2} Q$, which means that $\frac{a_{j_1} - a_{j_2}}{b_{j_2} - b_{j_1}} P = Q$. We can then easily find k provided that $\gcd(b_{j_2} - b_{j_1}, n) = 1$.

In fact, even if $\gcd(b_{j_2} - b_{j_1}, n) = d > 1$, we can compute $\frac{a_{j_1} - a_{j_2}}{b_{j_2} - b_{j_1}} \pmod{N/d}$. There are then d possibilities for k, which is only intractable for large d. In practice, however, d is quite small, especially if E is chosen so that n is prime. [1]

Actually, it's not very important which set ∞ gets hashed to. Indeed, suppose, for example, that $A_i = 3P + 5Q \in S_2$ and $A_{i+1} = \infty$. This means that $6P + 10Q = 6P + 10kP = (10k + 6)P = \infty$. In this case, $10k + 6 \equiv 0 \pmod{n}$ (or modulo some proper divisor of n if P generates a proper subgroup of E), which means we can easily solve for k.

Unlike the Baby-Step Giant-Step algorithm, however, Pollard's Rho algorithm only requires $O(1)$ space complexity. The naive implementation would be to store all the results in a table of size $O(\sqrt{n})$, sort the table, and look for a match. But in fact, we only need to look at pairs $(A_i, A_{2i})_{i \geq 1}$.

2.1. **Proof of Correctness.**

Why should there be a match in this narrow sequence of points? Certainly there will be an equality of the form $A_i = A_j$ at some point. This is because the elliptic curve has only finitely many points, so our sequence of points must be ultimately cyclic. Given the existence of such an equality, it follows that $A_{i+k} = A_{j+k}$ for all $k \geq 0$. Since we want an equality of the form $A_i = A_{2i}$, we set $2(i + k) = j + k$ and solve for k. This has the solution $j - 2i$. Indeed, we then have $A_{i+j-2i} = A_{j+j-2i}$, or $A_{j-i} = A_{2(j-i)}$, as desired.

We require $j \geq 2i$, however. But this assumption is not stifling. Indeed, if $A_i = A_j$ but $j < 2i$, then we know we have a cycle of length $j - i$ (or a divisor of $j - i$). We can find an integer r with $r(j - i) \geq i$. In this case, we know $A_j = A_{j+r(j-i)}$, which must also be equal to A_i. Since $r(j - i) \geq i$ and $j \geq i$, it follows that $j + r(j - i) \geq 2i$, and we can assume without loss of generality that our initial match $A_i = A_j$ satisfies $j \geq 2i$. What we are doing here is traversing the cycle multiple times.

To give a concrete example, suppose we find that $A_5 = A_8$. In this case, our original $j - 2i$ would be -2, which does us no good. But we see that there must be a cycle of length 3 (or possibly 1), so $A_5 = A_8 = A_{11}$. Since $11 \geq 2 \cdot 5$, we can then let $k = 11 - 2 \cdot 5 = 1$, and see that $A_{5+1} = A_{11+1}$, or $A_6 = A_{12}$.

3. Increasing the Number of Partitions

How fast should a match be found? If we view our function as producing a graph, where the vertices are the points of E, and there is an edge $P_i \to P_j$ whenever $f(P_i) = P_j$, then this graph will look like the Greek letter ρ: there will be a tail followed by a cycle. It can be shown that, for a random map f, the tail length and the cycle length each have expectation $\sqrt{\pi n/8}$ [2]. Therefore, a match should be found within $2\sqrt{\pi n/8} = \sqrt{\pi n/2}$ iterations.

Unfortunately, there is evidence to show that with our choice of function f, the number of iterations exceeds the expected $\sqrt{\pi n/2}$ required iterations by about 28-35% [3]. One possible explanation for this is provided by an analysis of the different cases. The rule $A_i + P$ only takes one step, and the rule $A_i + Q$ only takes k steps, which could be fairly small. With such small steps, it could take considerably longer to walk through the the tail and the cycle and find a match. [3]

Research has indicated, however, that increasing the number of partitions improves the randomness of the function f, which improves the performance of the algorithm.

In order to do this, we first need to hash the points $(x, y) \in E$ to the set $\{1, \ldots, m\}$, where there are m partition elements. Again, it is not important what set ∞ gets hashed to. It turns out that hashing based on the x-coordinate is just as effective as using the y-coordinate. Since the x-coordinate is a polynomial, we can represent it as a binary vector and view it as a non-negative integer (for the purposes of hashing). We then partition the points evenly into m subsets of size $2^k/m$.

We define $M_j = a_jP + b_jQ$, where the a_j's and b_j's are randomly chosen modulo n. We finally define $f(A_i) = A_i + M_j$ when $A_i \in S_j$, where S_j is the jth partition element.

The best choice for m in simulating a random function f seems to be in between 20 and 30 [3]. However, there is evidence that for m around 60, the function f performs more efficiently than a random map by about 6% [3].

4. Future Work

This research topic could be continued in a couple of different ways. First, the data referenced in this paper was gathered for curves over $\mathbb{F}_{2^8}$. Of course, these curves are too small to be considered for real cryptographic applications. Unfortunately, curves over, say, $\mathbb{F}(2^{163})$ would be too large to gather statistics on. But certainly fields larger than $\mathbb{F}_{2^8}$ could be used for further analysis.

Secondly, this research could be extended to curves over prime fields $\mathbb{F}_p$. Most of the details would be the same, except for how to hash the points. But hashing would actually be simpler. For example, one could just reduce the x-coordinate modulo m, and pick a random set S_i to place ∞ in (or ignore ∞ altogether).

References

[1] Washington, Lawrence C., *Elliptic Curves: Number Theory and Cryptography*, Chapman & Hall, Boca Raton, FL, 2nd. Ed., 2008.

[2] P. Flajolet and A. Odlyzko, Random Mapping Statistics. In *Advanced in Cryptology—EUROCRYPT '89 (Houthalen, 1989)*, volume 434 of *Lecture Notes in Comput. Sci.*, pages 329-354. Springer, Berlin, 1990.

[3] Lamb, Nicholas, An Investigation into Pollard's Rho Method for Attacking Elliptic Curve Cryptosystems. 2002.

E-mail address: `ablumenf@u.rochester.edu`