

Discrete Log Attacks: The Square-Root Methods

Johan Strümpfer
STRJOH028

*MAM3000W Project, Department of Mathematics,
University of Cape Town*

(Submitted 05 August 2006)

Contents

	PG
1. Introduction	1
2. Public Key Cryptosystems	4
3. Number Theory	8
4. Discrete Logarithms	13
4.1 Enumeration	13
4.2 Shanks Baby-Step Giant-Step Algorithm	13
4.3 Pollard ρ -Algorithm	16
4.5 Pohlig-Hellman Algorithm	20
5. Conclusion	24
6. References	26

1. Introduction

Cryptography is used to communicate in private through public channels. Thus, by using cryptographic methods, communication is possible so that any interception by a third party is unimportant since the communications cannot be understood unless they are in possession of what is known as a 'private' key, which allows encryption and/or decryption of the communications. Encryption means to scramble the message so that it is unintelligible until it is decrypted back into legible communication.

Cryptography is an ancient art, the earliest record of which comes from around 1900 B.C. when an Egyptian scribe used Hieroglyphics that weren't part of the standard set of symbols in an inscription. The earliest record of a cipher comes from sometime in 500-600 B.C. and is called the ATBASH cipher. From then ciphers became more complex and cryptology arose as an interesting field of study.

Cryptography comes from the ancient Greek words *kryptós* meaning 'hidden' and *gráphein* which means 'to write'. Today cryptography refers to the art of making codes, while cryptanalysis refers to the art of breaking them and cryptology refers to the study of both fields.

With the invention of machines, electronics and computers came a great increase in complexity of the ciphers that were used to encrypt messages. The reason was that older ciphers were being broken with greater ease and with modern machines more complex ciphers could be used with ease. This led to researchers turning to mathematics and basing ciphers on mathematical problems which are considered difficult to solve.

In the newer cryptosystems two main ideas have been developed. They are symmetric and asymmetric cryptosystems. In a symmetric cryptosystem the encryption key is the same as the decryption key. Thus if a person is able to encrypt a message for someone else, they can also decrypt the message. The problem arises that if two people wish to communicate in secrecy, they must decide on a key first and somehow communicate this key to each other. This means that they must first use insecure channels to

establish how they will communicate securely, which is an easily exploitable weakness. In asymmetric cryptosystems this is overcome by using one key to encrypt messages and another to decrypt messages. Thus if two people wish to communicate securely they have only to exchange their encryption keys. The difficulty here lies in finding encryption keys from which it is infeasible to find out the decryption key. Asymmetric cryptosystems are also called public-key cryptosystems, where the encryption key is the public key and the decryption key is the private key.

This is where mathematics came to help. Two problems which are considered difficult to solve were those of finding prime factors of large numbers and of finding discrete logs. This report deals with the problem of finding discrete logs. If g , a and x are integers modulo p then x is the discrete log of a with base g if:

$$g^x \equiv a \pmod{p} \quad (1.1)$$

It is easy to determine values for a given x , but it is very difficult to find x given a .

2. Public Key Cryptosystems

The main problem, as outlined above, with symmetric (also known as private-key) cryptosystems is that secret keys must be exchanged over insecure channels. The solution to this problem was first introduced by Whitfield Diffie and Martin Hellman [DH76] in 1976. This became the start of public-key cryptography.

The main idea behind it was that solving discrete logs is difficult. The Diffie-Hellman key exchange protocol works with the two parties (say Alice and Bob) that wish to communicate securely deciding on a cyclic group G and generator g , where $|G| = n$. Then Alice chooses an exponent $a \in \{0, 1, 2, \dots, n-1\}$ and similarly Bob chooses an exponent $b \in \{0, 1, 2, \dots, n-1\}$. Alice then calculates her half-key:

$$A = g^a \quad (2.1)$$

Bob calculates his half key:

$$B = g^b \quad (2.2)$$

They then exchange half-keys, i.e. Alice sends Bob A and Bob sends Alice B . They then calculate the full key:

Alice calculates:

$$K = B^a = g^{ab} \quad (2.3)$$

Bob calculates:

$$K = A^b = g^{ab} \quad (2.4)$$

The common key is then K . Since they did not exchange any information about their exponents their key is private.

If a person were able to intercept the key exchange, gaining knowledge of A and B , and they could calculate discrete logs in the cyclic group effectively, they could calculate the exponents a or b and thus the secret key K .

The first public key cryptosystem that was developed is the RSA system by Ron Rivest, Adi Shamir and Len Adleman [RSA78]. This system is based on the difficulty of factoring large numbers into primes. This can be an easy problem if the prime factors are large. In the RSA system the number that needs factoring, called the *RSA modulus*, is the product of two large prime numbers, hence making the problem difficult.

The next major step in public key cryptography was the ElGamal cryptosystem. This system can be seen as an extension of the Diffie-Hellman key exchange protocol. It is based on the difficulty of solving discrete logs modulo large prime numbers p . Numbers modulo prime p form a cyclic group under multiplication modulo p .

The ElGamal system works as follows:

Alice chooses a large prime number p and a generator $g \bmod p$ that generates the cyclic group \mathbb{Z}_p^* with order $n = p - 1$. Alice then chooses a random exponent $a \in \{0, 1, 2, \dots, p - 2\}$ and calculates:

$$A = g^a \bmod p. \quad (2.5)$$

Her public key is then the triple (p, g, A) and her private key is the randomly chosen exponent a .

If Bob wishes to send Alice an encrypted message m , where m is an element in the set $\{0, 1, \dots, p - 1\}$, he also chooses a random exponent $b \in \{0, 1, \dots, p - 2\}$. He then computes her key part B :

$$B = g^b \bmod p. \quad (2.6)$$

He then calculates the encrypted message, c , as follows:

$$c = A^b m \text{ mod } p \quad (2.7)$$

Bob then sends Alice the pair (B, c) as the complete ElGamal encrypted message.

Once Alice has received the message pair (B, c) she has to decrypt it. To do this she calculates:

$$x = p - 1 - a \quad (2.8)$$

so that $g^x g^a = 1$. She then calculates:

$$\begin{aligned} B^x c \text{ mod } p &= g^{b(p-1-a)} A^b m \text{ mod } p \\ &= (g^{p-1})^b g^{a(-b)} g^{ab} m \text{ mod } p \\ &= 1 \cdot g^{-ab} g^{ab} m \text{ mod } p \\ &= m \text{ mod } p \end{aligned} \quad (2.9)$$

and so she recovers the original message m .

Example 1:

Bob wishes to send Alice her new bank pin number: 1234 (i.e. $m = 1234$).

Alice's public key is $(p, g, A) = (3701, 17, 15)$ and his secret key is $a = 1683$. Bob chooses an exponent $b = 1283$ and calculates:

$$B = g^b \text{ mod } p = 17^{1283} \text{ mod } 3701 = 1408 \quad (2.10)$$

She then calculates the encrypted message:

$$c = A^b m \text{ mod } p = (15^{1283})(1234) \text{ mod } 3701 = 3692, \quad (2.11)$$

and sends Alice the pair $(B, c) = (1408, 3692)$.

Alice then calculates:

$$x = 3701 - 1 - 1683 = 2017, \quad (2.12)$$

and thus can recover the original message by calculating:

$$B^x c \bmod p = (1408^{2017})(3692) \bmod 3701 = 1234. \quad (2.13)$$

If an attacker intercepted the message pair (B, c) and if he could calculate discrete logs efficiently, he could find x such that $g^x = A \bmod p$ (then $x = a \bmod p$, where a is Alice's private key) and thus follow the same calculations as in (2.12) and (2.13) to find the original message m . This is where the discrete log problem arises and the attacks described in chapter four find their application.

3. Number Theory

To understand what we are dealing with in encryption using mathematically difficult problems, such as discrete logs, we need to understand the mathematics behind these problems.

First, specification is needed of the system we are working with. The integers modulo p , for p prime, form cyclic groups with respect to multiplication. Since the group is cyclic it is generated by a single element in the group: the generator g . The group can thus also be specified as $G = \langle g \rangle = \{g^k \mid k \in \mathbb{Z}, 0 \leq k < n\}$, where n is the order of the group. To be more precise the elements of the group are the congruence classes $[x] = \{x + pz \mid z \in \mathbb{Z}\}$ where $x \in \{1, 2, \dots, p-1\}$ and the group $G = (\mathbb{Z}/p\mathbb{Z})^*$ is the multiplicative group of residues mod p . The congruence relation is thus specified as:

$$x \equiv y \pmod{p} \Leftrightarrow \exists_{m \in \mathbb{Z}} x - y = mp \quad (3.1)$$

If the congruence class of g generates the whole group, G , then g is called a *primitive root* mod p of G . When working with the elements of the group, $[x]$, we rather work with their representatives, x , that are reduced modulo p for succinctness and simplicity.

The order, n , of G , is determined by the Euler φ -function and is given as:

$$|G| = n = \varphi(p), \quad (3.2)$$

where $\varphi(p)$ is the number of elements in $\{1, 2, \dots, p\}$ with $\gcd(x, p) = 1$. For p prime we can see that $\varphi(p) = p - 1$.

The order of $g \in G$ (denoted as $|g|$) is defined as the smallest positive integer n such that $g^n \equiv 1 \pmod{p}$. If such an integer does not exist, then the order of the element is infinite. This, however, will not happen in our situation since we are in a finite group. This leads to an important theorem.

Theorem 1: Let $g \in G$ where $|g| = n$ and $e \in \mathbb{Z}$. Then:

$$g^e = 1 \Leftrightarrow \exists_{m \in \mathbb{Z}} e = nm.$$

Proof: Let $|g| = n$. If $e = mn$ then:

$$g^e = g^{mn} = (g^n)^m = 1^m = 1.$$

Conversely, let $g^e = 1$ and $e = mn + r$ with $0 \leq r < n$. Then:

$$g^r = g^{e-mn} = g^e (g^n)^{-m} = 1(1)^{-m} = 1,$$

but n is the least positive integer such that $g^n = 1$ and thus we conclude that $r = 0$ and thus $e = mn$.

This leads us to a very useful corollary:

Corollary 1: Let $g \in G$ and $k, l \in \mathbb{Z}$. Then:

$$g^k = g^l \pmod{p} \Leftrightarrow k \equiv l \pmod{n} \text{ where } n = |g|.$$

If $|g| = n = |G|$ then g is a generator of the group. We now have the tools to formulate Fermat's Little Theorem:

Theorem 2: Fermat's Little Theorem

If the greatest common divisor of a and m is 1 then $a^{\varphi(m)} \equiv 1 \pmod{m}$.

Theorem 3:

The congruence $ax \equiv 1 \pmod{p}$ has a solution if and only if $\gcd(a, p) = 1$, and then the solution x is uniquely determined mod p and is called the inverse of $a \pmod{p}$. This is then denoted as: $x = a^{-1}$.

Theorem 4: Chinese Remainder Theorem

Let m_1, \dots, m_n be positive integers that are pairwise coprime and let $m = m_1 m_2 \dots m_n$. Let a_1, \dots, a_n be integers such that: $x \equiv a_1 \pmod{m_1}$, $x \equiv a_2 \pmod{m_2}$, ..., $x \equiv a_n \pmod{m_n}$. Then the solution x exists and is unique mod m .

Proof:

Let m_1, \dots, m_n be positive integers that are pairwise coprime. Let a_1, \dots, a_n be integers such that: $x \equiv a_1 \pmod{m_1}$, $x \equiv a_2 \pmod{m_2}$, ..., $x \equiv a_n \pmod{m_n}$. To solve this simultaneous congruence we do the following:

Let:

$$m = \prod_{i=1}^n m_i \quad \text{and} \quad M_i = \frac{m}{m_i} \quad \forall i \in \{1, \dots, n\}.$$

Since all the m_i are coprime we have that $\gcd(M_i, m_i) = 1$ for all i in $\{1, \dots, n\}$. Thus we can compute all y_i such that:

$$y_i M_i \equiv 1 \pmod{m_i} \quad \forall i \in \{1, \dots, n\}.$$

We can thus see that:

$$a_i y_i M_i \equiv a_i \pmod{m_i} \quad \forall i \in \{1, \dots, n\}.$$

Since for each $i \neq j$, m_i is a divisor of M_j we have:

$$a_j y_j M_j \equiv 0 \pmod{m_i} \quad \forall i, j \in \{1, \dots, n\} \text{ and } i \neq j,$$

thus we see that:

$$\sum_{j=1}^n a_j y_j M_j \equiv a_i \pmod{m_i} \text{ for all } i.$$

We then set:

$$x = \left(\sum_{i=1}^n a_i y_i M_i \right) \pmod{m}.$$

Then x is a solution of the simultaneous congruence.

Finally, let x and x' be solutions to the simultaneous congruence. Then $x \equiv x' \pmod{m_i}$ for all $i \in \{1, \dots, n\}$ which means that m_i divides $x - x'$. Since all the m_i are coprime it means that m divides $x - x'$ and hence $x \equiv x' \pmod{m}$. Thus our solution exists and is unique.

Example 2

We use the Chinese Remainder Theorem to calculate a solution to the simultaneous congruence: $x \equiv 3 \pmod{4}$, $x \equiv 8 \pmod{25}$ and $x \equiv 18 \pmod{37}$.

Thus $m_1 = 4$, $m_2 = 25$, $m_3 = 37$, $a_1 = 3$, $a_2 = 8$ and $a_3 = 18$. We calculate $m = 4 \times 25 \times 37 = 3700$ and $M_1 = 3700/4 = 925$, $M_2 = 3700/25 = 148$ and $M_3 = 3700/37 = 100$.

Using the Extended Euclidean algorithm we solve for y_1 , y_2 and y_3 as solutions for:

$$y_1 * 925 \equiv 1 \pmod{4}, \quad y_2 * 148 \equiv 1 \pmod{25} \quad \text{and} \quad y_3 * 100 \equiv 1 \pmod{37}.$$

We get that $y_1 = 1$, $y_2 = 12$ and $y_3 = 10$.

Thus $x = (3 * 1 * 925 + 8 * 12 * 148 + 18 * 10 * 100) \pmod{3700}$, from which we get $x = 1683$.

The final theorem we have to prove is the birthday paradox, which states that if there are 23 people in a room there is a greater than 50% chance that two of them will share the same birthday.

This is proven by calculating what the probability is that all the people in the room have different birthdays. There are 365 days in the year. If we want all people in the room to have different birthdays: the first person in the room has 365/365 days to choose for a birthday, the second has 364/365 and the third has 363/365, etc. Person number x has $\left(1 - \frac{(x-1)}{365}\right)$ days to choose from.

The probability of x people in the room all having different birthdays is:

$$\bar{P}(x) = \left(\frac{365}{365}\right)\left(\frac{364}{365}\right)\left(\frac{363}{365}\right) \dots \left(1 - \frac{(1-x)}{365}\right) = \frac{365!}{365^x (365-x)!}. \quad (3.3)$$

The probability that any two people in the room will share a birthday is then given as $P(x) = 1 - \bar{P}(x)$. When we have $x = 23 \approx \sqrt{365}$, then the probability of any two people sharing a birthday is 50.7%.

This theorem, when applied to a general set of elements tell us that if we have a total of n elements to choose from and we select x random elements then the probability of two elements being identical is given as:

$$P(x) = 1 - \frac{n!}{n^x (n-x)!}. \quad (3.4)$$

The solution to the equation $P(x) = 0.50$ is $x \approx \sqrt{n}$.

We now have the tools necessary to help us solve discrete logs in cyclic groups.

4. Discrete Logarithms

The problem we are trying to solve is:

In the finite cyclic group G , where $|G| = n$ and g is a generator of the group, given $a \in G$, what is the discrete logarithm of a to the base g ? i.e. what is the smallest non-negative integer x such that:

$$g^x = a ? \tag{4.1}$$

We will describe various algorithms to calculate x in this section.

4.1 Enumeration

The first and most basic attack is to try all possible values for x till the correct exponent is found. Thus given g and a , we try $x = 1, 2, 3, \dots$ till we find an x that satisfies equation (4.1). This will take $x-1$ multiplications of g and x comparisons in the group. We only ever need to store three items: g^x , g and a . Thus if x were very large, this would take a very long time and is infeasible to use in current cryptographic systems.

4.2 Shanks Baby-Step Giant-Step Algorithm

This is the first feasible attack on the discrete log problem. The algorithm works by calculating a set of baby steps from the public key a using the generator g , as shown in step 3. If it finds a match within the baby steps set then the exponent is found and the discrete logarithm is solved. Otherwise, it calculates a sequence of giant steps, starting from g^m this time, till one of the larger steps lands on a baby step, shown in step 5. It was devised by D. Shanks [SHK71] in 1971 and given a , g and m , it is used as follows:

1. let $m = \lceil \sqrt{n} \rceil$
2. Let $x = qm + r$ for some r and q where $0 \leq r < m$ and $0 \leq q < m$

Thus: $a = g^x = g^{qm+r}$ and hence $(g^m)^q = ag^{-r}$. We next compute the set of m baby steps:

$$B = \{(ag^{-s}, s) : 0 \leq s < m\}$$

If $(1, s) \in B$ then $s = r, q = 0$ and we have found the discrete logarithm $x = s$, otherwise we continue with the giant steps:

3. let $d = g^m$
4. Calculate: d^k for $k = 1, 2, 3, \dots, m$ till we find a pair $(d^k, s) \in B$.

Once we have found this pair we have in essence landed on a baby step while taking our giant steps. We then see that: $ag^{-s} = d^k = (g^m)^k$ and thus $a = g^{km+s}$. Then $k = q, s = r$ and hence $km + s$ is a solution to the discrete logarithm:

5. $x = km + s$

The algorithm takes m multiplications modulo p to calculate the baby-steps and stores them all. The giant-steps are calculated using k multiplications of $d = g^m$, but $k \leq m$ and thus we have to make at most m multiplications modulo p . Thus the giant-steps requires at most m multiplications and comparisons modulo p . Since $n = |G|$ and $m \approx \sqrt{n}$ we can conclude that the algorithm requires $O(\sqrt{|G|})$ multiplications and comparisons in the group and storage of $O(\sqrt{|G|})$ elements of the group.

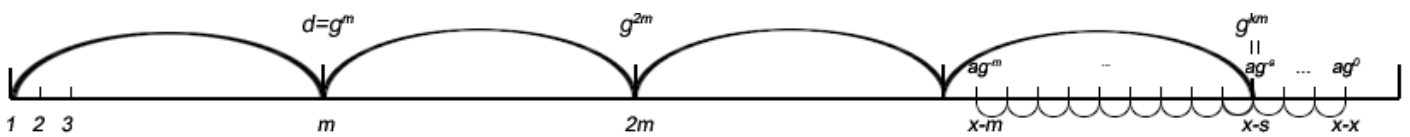


Figure 1: The giant-steps are shown on top of the number line and the baby-steps on the bottom. The algorithm stops when a giant step lands on a baby step.

Example 3:

We use the same situation as in Example 1. Suppose an attacker can intercept messages in the communications to Alice. He also knows Alice's public key $(p, g, A) = (3701, 17, 15)$. He then needs to calculate the exponent x such that $17^x \equiv 15 \pmod{3701}$ to be able to decrypt the messages meant for Alice. Using Shanks Baby-Step Giant-Step algorithm:

He first calculates $m = \lceil \sqrt{3700} \rceil = 61$. He then calculates the baby-step set

$B = \{(ag^{-s}, s) : 0 \leq s < m\}$, thus:

$B = \{(15,0), (654,1), (1127,2), (284,3), (3500,4), (859,5), (2663,6), (2116,7), (1213,8), (1813,9), (2066,10), (2734,11), (2991,12), (2353,13), (3404,14), (2595,15), (2112,16), (2519,17), (1019,18), (2237,19), (567,20), (1775,21), (3370,22), (2593,23), (2765,24), (2122,25), (2955,26), (3004,27), (3660,28), (433,29), (1114,30), (2678,31), (2770,32), (2340,33), (2097,34), (1865,35), (3593,36), (1953,37), (768,38), (916,39), (707,40), (477,41), (1552,42), (309,43), (889,44), (270,45), (669,46), (1781,47), (1411,48), (83,49), (658,50), (3522,51), (1078,52), (3329,53), (3026,54), (178,55), (1099,56), (2024,57), (1643,58), (2056,59), (2298,60)\}.$

The giant steps are $(g^m)^k = (1696)^k$ for $k = 1, 2, \dots, m$:

1696, 739, 2406, 2074, 1554, 472, 1096, 914, 3126, 1864, 690, 724, 2873, 2092, 2474, 2671, 3693, 1236, 1490, 2958, 1913, 2372, 3626, 2335, 90, 899, 3593.

The element (3593, 36) is in the set of baby-steps and 3593 is the 27th giant step, thus the discrete logarithm is $x = km + s = 27 \times 61 + 36 = 1683$.

This process required 61 multiplications modulo 3659 for the baby-steps, 1 exponentiation and 26 multiplications modulo 3659 for the giant-steps. Enumeration would have taken many more, namely 1682 multiplications modulo 3701, to find the correct solution. We needed to store all the baby-steps as well as m, A, g, p and k , a total of 68 elements.

4.3 Pollard ρ -Algorithm

This algorithm uses the cyclic nature of our group to solve discrete logs. We first split the set of elements we are working with into three disjoint subsets G_1 , G_2 and G_3 of roughly equal size. In other words we need G_1, G_2, G_3 , such that $G_1 \cap G_2 = G_2 \cap G_3 = G_1 \cap G_3 = \emptyset$ and $G_1 \cup G_2 \cup G_3 = G$. Then we define a function $f : G \rightarrow G$ as:

$$f(\beta) = \begin{cases} g\beta & \text{if } \beta \in G_1 \\ \beta^2 & \text{if } \beta \in G_2 \\ a\beta & \text{if } \beta \in G_3 \end{cases}.$$

Now let $\beta_0 = g^{x_0} a^{y_0}$, where x_0 is a random element in the set $\{1, \dots, n\}$ and β_0 is a random element of G . Then define a sequence $\beta_{i+1} = f(\beta_i)$. Each element in the sequence can then be written as:

$$\beta_i = g^{x_i} a^{y_i}$$

with

$$x_{i+1} = \begin{cases} x_i + 1 \bmod n & \text{if } \beta_i \in G_1 \\ 2x_i \bmod n & \text{if } \beta_i \in G_2 \\ x_i & \text{if } \beta_i \in G_3 \end{cases} \quad \text{and} \quad y_{i+1} = \begin{cases} y_i & \text{if } \beta_i \in G_1 \\ 2y_i \bmod n & \text{if } \beta_i \in G_2 \\ y_i + 1 \bmod n & \text{if } \beta_i \in G_3 \end{cases}.$$

Since G is cyclic there will eventually be two elements, β_i and β_{i+l} for some $i \geq 0$ and $l > 0$, where $\beta_i = \beta_{i+l}$. Hence:

$$g^{x_i} a^{y_i} = g^{x_{i+l}} a^{y_{i+l}}.$$

This implies then that:

$$g^{x_i - x_{i+l}} = a^{y_{i+l} - y_i}.$$

But we know that $a = g^x$ thus:

$$g^{x_i - x_{i+l}} = g^{x(y_{i+l} - y_i)}.$$

By Corollary 1 we know then that:

$$x_i - x_{i+l} \equiv x(y_{i+l} - y_i) \pmod{n}.$$

By Theorem 3 $(y_{i+l} - y_i)$ is invertible if $\gcd(y_{i+l} - y_i, n) = 1$ and thus:

$$x \equiv (x_i - x_{i+l})(y_{i+l} - y_i)^{-1} \pmod{n},$$

and we have solved the discrete logarithm.

If $(y_{i+l} - y_i)$ is not invertible then we reduce the congruence by dividing it by the greatest common divisor, d , of all three $(x_i - x_{i+l})$, $(y_{i+l} - y_i)$ and n and finding z such that:

$$\frac{(x_i - x_{i+l})}{d} \equiv z \frac{(y_{i+l} - y_i)}{d} \pmod{\left(\frac{n}{d}\right)},$$

then:

$$x = z + k \left(\frac{n}{d}\right) \text{ where } 0 \leq k < d.$$

We test all values for x till we find one that solves the discrete log. If there are too many values to test, we start the algorithm again using a different value for x_0 .

This method however will require storage for $i + l$ number of triples of the form (β_j, x_j, y_j) . To reduce the storage usage of the algorithm we initially only store (β_i, x_i, y_i) for $i = 1$ and compute (β_j, x_j, y_j) till either a match is found or $j = 2i$. In the latter case we replace (β_i, x_i, y_i) with (β_j, x_j, y_j) and set $i = j$ and continue as before. Thus only the triple (β_i, x_i, y_i) , with $i = 2^k$ for $k = 0, 1, 2, \dots$, is stored.

The basis of why the Pollard- ρ algorithm works lies in the birthday paradox. The birthday paradox states that if there are 23 or more people in a room then there is a chance of more than 50% that at least two of them will have the same birthday. This translated into slightly more familiar terms states that probability of finding two numbers in a randomly chosen sequence that are congruent modulo p is greater than 50% once we have chosen approximately \sqrt{p} numbers. Applying this to the Pollard-Rho algorithm gives us that the probability of finding a match $\beta_i = \beta_{i+l}$ is greater than 50% when we have $O(\sqrt{|G|})$ group elements in the sequence of β_i 's. This corresponds to having gone once around the loop of the ρ , which is of period l , and is depicted below:

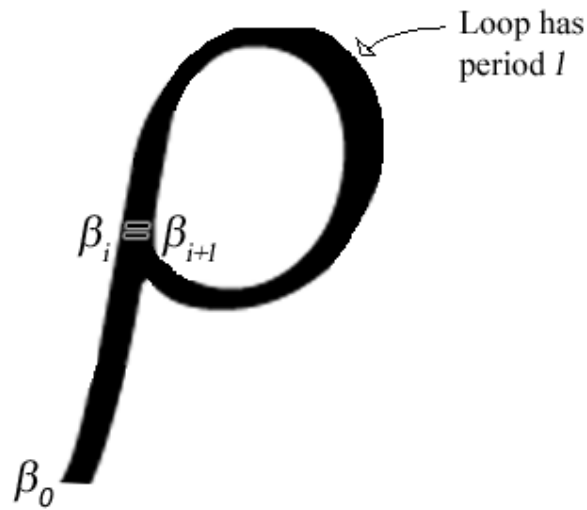


Figure 2: The Pollard-Rho method gets its name from this depiction of the process.

This implies that $i+l$ is $O(\sqrt{|G|})$. This, however, is based on storing every β_i , which can take up a lot of space, as explained above. If we, as described above, store only β_{2^k} we need to find β_{2^k+l} such that $\beta_{2^k} = \beta_{2^k+l}$. We know that $2^{k-1} \leq l \leq 2^k$. This implies that $2^k \leq 2l$. So if we find a match within $2^k + l$ steps it implies that we find a match in less than or equal to $3l$ steps. In other words, the number of steps till we find a match is: $2^k + l \leq 3l \leq O(\sqrt{|G|})$ since $l \leq O(\sqrt{|G|})$. Hence the method still requires only $O(\sqrt{|G|})$ steps.

Example 4

If the attacker in Example 3 used the Pollard ρ -Algorithm to solve the discrete log: $17^x = 15 \pmod{3701}$, he would first choose 3 disjoint subsets, say:

$$G_1 = \{1, \dots, 1233\} \quad G_2 = \{1234, \dots, 2467\} \quad G_3 = \{2468, \dots, 3700\},$$

and randomly: $x_0 = 1280$. He then computes the sequence of triples (β_j, x_j, y_j)

for $j = 1, 2, \dots$ and stores the following:

j	β_j	x_j	y_j
0	2725	1280	0
1	164	1280	1
2	2788	1281	1
4	348	1282	2
8	518	2864	16
16	184	2884	514
32	1244	912	448

For $j = 57$ he finds $\beta_j = 1244$, $x_j = 1072$, $y_j = 28$. Now we have $\beta_{32} = \beta_{57} \pmod{p}$ which implies that $g^{x_{32}-x_{57}} = a^{y_{57}-y_{32}} \pmod{p}$ and thus $17^{3540} = 15^{3280} \pmod{3701}$ and so to find the discrete log he needs to solve the congruence:

$$3540 \equiv x \ 3280 \pmod{3700}.$$

The greatest common divisor of all three 3540, 3280 and 3700 is 20, thus the congruence reduces to:

$$177 \equiv z \ 164 \pmod{185}.$$

Here $\gcd(164, 185) = 1$, thus 164 is invertible mod 185. He finds the inverse of 164 is 44 and $z = 18$. Thus $a = 18 + k(185)$ for $k \in \{0, 1, \dots, 19\}$. For $k = 9$ he finds that $x = 1683$ solves the discrete logarithm.

4.4 Pohlig-Hellman Algorithm

If we know the prime factorization of the group order then using the Pohlig-Hellman algorithm, we reduce our problem of finding one discrete logarithm in a large group of composite order to a problem of finding a number of discrete logs in smaller groups. We then calculate discrete logs in smaller groups of prime order and solve a simultaneous congruence to find the solution x of the initial discrete log, $a = g^x$.

Let:

$$|G| = n = \prod_{i=1}^k p_i^{e_i},$$

where p_i are all prime and e_i are positive integers. For each $1 \leq i < k$ we then

calculate: $n_i = \frac{n}{p_i^{e_i}}$, $g_i = g^{n_i}$ and $a_i = a^{n_i}$. Note that in the group G , the order of g_i is $p_i^{e_i}$ and we have $g_i^{x_i} = a_i$. We denote the discrete log of a_i to the base g_i as x_i ; so $x \equiv x_i \pmod{p_i^{e_i}}$. Then the solution of (4.1) is the unique $x \in \{0, 1, \dots, n-1\}$ such that $x \equiv x_i \pmod{p_i^{e_i}}$ for $1 \leq i < k$, which we find using the Chinese Remainder Theorem.

Proof: Let $x \equiv x_i \pmod{p_i^{e_i}}$ for $1 \leq i < k$. Then: $(ag^{-x})^{n_i} = a_i g_i^{-x_i} = 1$ for $1 \leq i < k$. This implies (by Theorem 1) that the order of (ag^{-x}) is a divisor of n_i for $1 \leq i < k$. Thus the order of (ag^{-x}) is a divisor of the greatest common divisor of all n_i , but this is 1. Thus the order of (ag^{-x}) is 1 which shows that $g^x = a$.

We show now that to solve for x_i we need to solve e_i discrete logs in a group of order p_i . Raise $g_i^{x_i} = a_i$ to the power of $p_i^{e_i-1}$ thus:

$$g_i^{x_i p_i^{e_i-1}} = a_i^{p_i^{e_i-1}}.$$

Now let $x_i = \sum_{j=0}^{e_i-1} x_{i,j} p_i^j$ where $0 \leq x_{i,j} < p_i$.

Thus:

$$\begin{aligned} g_i^{x_i p_i^{e_i-1}} &= g_i^{(x_{i,0} + x_{i,1} p_i + \dots + x_{i,e_i-1} p_i^{e_i-1}) p_i^{e_i-1}} \\ &= g_i^{x_{i,0} p_i^{e_i-1} + x_{i,1} p_i^{e_i} + \dots + x_{i,e_i-1} p_i^{2e_i-2}} \\ &= g_i^{x_{i,0} p_i^{e_i-1} + p_i^{e_i} (x_{i,1} + \dots + x_{i,e_i-1} p_i^{e_i-2})} \\ &= g_i^{x_{i,0} p_i^{e_i-1}}, \end{aligned}$$

since g_i has order $p_i^{e_i}$. Thus we see that $(g_i^{p_i^{e_i-1}})^{x_{i,0}} = a_i^{p_i^{e_i-1}}$. The order of $(g_i^{p_i^{e_i-1}})$ is p_i and thus $x_{i,0}$ is the solution of a discrete log in a group of order p_i . We thus determine $x_{i,0}$. The other coefficients are determined recursively as follows:

Suppose we know all $x_{i,0}, \dots, x_{i,j}$, then:

$$g_i^{x_{i,j+1} p_i^{j+1} + x_{i,j+2} p_i^{e_i+2} + \dots + x_{i,e_i-1} p_i^{e_i-1}} = a_i g_i^{-(x_{i,0} + x_{i,1} p_i + \dots + x_{i,j} p_i^j)}.$$

Thus we can see that:

$$\begin{aligned} \left(g_i^{x_{i,j+1} p_i^{j+1} + x_{i,j+2} p_i^{e_i+2} + \dots + x_{i,e_i-1} p_i^{e_i-1}} \right)^{p_i^{e_i-j-2}} &= g_i^{x_{i,j+1} p_i^{e_i-1} + p_i^{e_i} (x_{i,j+2} p_i^{-j} + \dots + x_{i,e_i-1} p_i^{e_i-j-3})} \\ &= g_i^{x_{i,j+1} p_i^{e_i-1}}. \end{aligned}$$

Then $x_{i,j+1}$ satisfies:

$$\left(g_i^{p_i^{e_i-1}} \right)^{x_{i,j+1}} = \left(a_i g_i^{-(x_{i,0} + x_{i,1} p_i + \dots + x_{i,j} p_i^j)} \right)^{p_i^{e_i-j-2}}.$$

Thus $x_{i,j+1}$ is the discrete log of $\left(a_i g_i^{-(x_{i,0} + x_{i,1} p_i + \dots + x_{i,j} p_i^j)} \right)^{p_i^{e_i-j-2}}$ to the base $(g_i^{p_i^{e_i-1}})$ in a group of order p_i . So we can determine all the $x_{i,j}$ hence all the x_i and then, by the Chinese Remainder Theorem, x .

Algorithm:

Given $p, g, a, n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$

1. set : $n_i = \frac{n}{p_i^{e_i}}, g_i = g^{n_i}, a_i = a^{n_i}$
2. For $i = 1, \dots, k$ do the following:
 - 2.1. For $j = 0, \dots, e_i - 1$ do the following:
 - 2.1.1. Set $x_m = \sum_{k=0}^j x_k p_i^k$
 - 2.1.2. Set $a_{i,j} = (a_i g_i^{-x_m})^{p_i^{e_i-j-2}}$
 - 2.1.3. Find $x_{i,j}$ such that $(g_i^{p_i^{e_i-1}})^{x_{i,j}} = a_{i,j}$ using Shank's Baby-Step Giant-Step algorithm or the Pollard ρ -Algorithm
 - 2.1.4. Store $x_{i,j}$
 - 2.2. Set $x_i = x_{i,0} + x_{i,1} p_i + \dots + x_{i,e_i-1} p_i^{e_i-1}$
 - 2.3. Store x_i
3. Find x such that $x \equiv x_i \pmod{p_i^{e_i}}$ for $1 \leq i < k$ using the Chinese Remainder Theorem.

To compute each digit of x_i (i.e. each $x_{i,j}$) we have to do one exponentiation in the group and apply Shank's Baby-Step Giant-Step algorithm or the Pollard ρ -Algorithm once. Thus, for the exponentiation and either algorithm, we need $O(\log n)$ and $O(\sqrt{p_i})$ group operations for each digit of x_i . Hence to calculate each x_i , we require $O(e_i(\log n + \sqrt{p_i}))$ group operations. To calculate the discrete logarithm x we need to calculate k number of x_i 's, since there are k prime divisors of n . Since $n = |G|$, we can see that to find the discrete logarithm x requires $O\left(\sum_{i=1}^k e_i(\log |G| + \sqrt{p_i})\right)$ group operations in total. This is much less than $O(\sqrt{n})$ if n has at least two large prime factors.

Example 5:

We look at how an attacker would solve $17^x = 15 \pmod{3701}$ using the Pohlig-Hellman Algorithm. The factorization of $n = p - 1$ is $3700 = 2^2 * 5^2 * 37$, and thus

$$\begin{aligned} p_1 &= 2, & e_1 &= 2 \\ p_2 &= 5, & e_2 &= 2 \\ p_3 &= 37, & e_3 &= 1. \end{aligned}$$

For the factor 2^2 he computes the following:

$$n_1 = 925, \quad a_1 = 1279, \quad g_1 = 2422 \quad g_1^{p_1^{e_1-1}} = 3700.$$

He calculates $x_m = 0$, $a_{1,0} = (1279 * 2422^0)^{2^1} = 3700$. He obtains and stores $x_{1,0} = 1$ as a solution for $3700^{x_{1,0}} = 3700$.

He recalculates $x_m = 1$, computes $a_{1,1} = (1279 * 1279^1)^{2^0} = 3700$ and obtains $x_{1,1} = 1$ as a solution for $3700^{x_{1,1}} = 3700$. Thus he calculates $x_1 = 1 + 1 * 2 = 3$.

For the factor 5^2 he then computes:

$$n_2 = 148 \quad a_2 = 3129 \quad g_2 = 3066 \quad g_2^{p_2^{e_2-1}} = 549.$$

He computes again $x_m = 0$, $a_{2,0} = 1140$ and finds $x_{2,0} = 3$ as a solution to $549^{x_{2,0}} = 1140$. He recalculates $x_m = 3$ and computes $a_{2,1} = 549$ and finds $x_{2,1} = 1$ as a solution to $549^{x_{2,1}} = 549$ and thus $x_2 = 8$.

For the factor 37^1 he computes the following:

$$n_3 = 100 \quad a_3 = 1930 \quad g_3 = 1951 \quad g_3^{p_3^{e_3-1}} = 1951.$$

He calculates $x_m = 0$, $a_{3,0} = 1930$ and finds $x_{3,0} = 18$ as a solution to $1951^{x_{3,0}} = 1930$ and thus $x_3 = 18$.

Using the Chinese Remainder Theorem he then finds $x = 1683$ as a solution to the simultaneous congruence: $x \equiv 3 \pmod{4}$, $x \equiv 8 \pmod{25}$ and $x \equiv 18 \pmod{37}$ (as shown in Example 2). Thus $x = 1683$ is the solution to the discrete log.

5. Conclusion

All three algorithms have only used properties of general cyclic groups. Thus they all work the same in any finite cyclic Abelian group as long as you can perform the group operation. These algorithms all use $O(\sqrt{|G|})$ group operations and it was proven by Victor Shoup [SHP97] that this is a lower bound to the efficiency of algorithms for calculating discrete logarithms in a generic group. Another algorithm of this sort does exist and is not covered in this project. It called the Kangaroo Method and is described by Edlyn Teske in [TSKE].

Of all three algorithms, the Pohlig-Hellman algorithm is by far the fastest for random group orders. The main determining factor in how long the algorithm will take is the size of the group order's largest prime divisor. For large group orders which are the products of two and a large prime number, this algorithm can also become infeasible. The Pollard- ρ algorithm then usually runs faster than Shanks Baby-Step Giant-Step algorithm, depending on its starting value.

As an illustration to see the differences in time I employed, using Python (an interpreted programming language), all three algorithms on the same problem. I tried to find x where:

$$212251^x = 19611985510 \pmod{112345679051} \quad (4.2)$$

All three algorithms gave the same answer (as expected) of $x = 18258522538$.

A breakdown of the time taken by each algorithm and the storage usage is given in the table below:

Algorithm	Time Taken	Group Operations	Group Elements Stored
Shanks Baby-Step Giant-Step	3.256 hrs	735992	335185
Pollard- ρ	1.260 min	389653	27
Pohlig-Hellman	14.16 sec	907	812

For reference, this was done on an AMD Athlon XP 2700+ processor with 512MB of RAM and

$p = 112345679051$ is a 38 bit prime number.

Running the algorithms with a larger prime modulus (58 bit); the Pohlig-Hellman algorithm solved it in 0.016 seconds, the Pollard- ρ algorithm took 23.53 hours and Shanks Baby-Step Giant-Step algorithm ran out of memory. Compared with the previous run (shown in the table above), the Pohlig-Hellman algorithm took much less time in calculating the discrete log. This was due to the fact that the group order of the larger prime number consisted of many small prime factors.

The default for current encryption technologies is to use 128 bit and larger primes, which makes it impractical to use any of these algorithms. Faster algorithms, such as the Index Calculus methods, do exist, the fastest of which is the number field sieve. They are, however, beyond the scope of this project.

6. References:

- [CRPT1] Buchmann, J. A., *Introduction To Cryptography*. Springer-Verlag, New York, 2001. ISBN: 0-387-95034-6
- [FREY] G. Frey, et. al., *Mathematical background of public key cryptography. Arithmetic, geometry and coding theory (AGCT 2003)*, S'emin. Congr., vol. 11, Soc.Math. France, Paris, 2005, pp. 41–73.
- [ODL99] Odlyzko, A., *Discrete logarithms: The past and the future*, 1999
- [MNZ] Menezes, A. J., van Oorschot, P. C. & Vanstone, S. A. *Handbook of Applied Cryptography (5th Printing)*. CRC Press, 2001, ISBN: 0-8493-8523-7
- [DH76] Diffie, W., Hellman, M., *New Directions in Cryptography*, IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976, pp: 644-654.
- [SHK71] D. Shanks. *Class number, a theory of factorization and genera*. In Proc. Symp. Pure Math. 20, pages 415--440. AMS, Providence, R.I., 1971.
- [B&M] Birkhoff, G., MacLane, S., *A Survey of Modern Algebra*. The Macmillan Company, 1954.
- [SHP97] Shoup, V. *Lower bounds for discrete logarithms and related problems*, in Proc. Eurocrypt , 1997, pp. 256-266,
- [TSKE] Teske, E. *Square-root algorithms for the discrete logarithm problem (a survey)*. in Public Key Cryptography and Computational Number Theory, Walter de Gruyter, 2001, pp. 283-301. 12
- [KSLR] Kessler, G. C., *An Overview of Cryptography*. Gary Kessler.Net, 2006. Retrieved from: <http://www.garykessler.net/library/crypto.html>
- [HSSR] Haeusser, M ., et al., *Interactive Cryptography Scripts*. University of Tübingen, 2005, retrieved from <http://www-fs.informatik.uni-tuebingen.de/~reinhard/krypto/index.html>
- [RSA78] R. Rivest, A. Shamir, L. Adleman. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Communications of the ACM, Vol. 21 (2), pp.120–126. 1978