DISCRETE LOGARITHM PROBLEM IN FINITE GROUPS

A. ANAS CHENTOUF

Abstract. The discrete logarithm problem plays a central role in modern cryptography. In this paper, we introduce the discrete logarithm problem in finite groups. We then present a diverse list of algorithms that solve this problem and analyze the efficiency of those algorithms.

Contents

1. Introduction

Consider a group G written multiplicatively, and let $\alpha, \beta \in G$ be two elements. The discrete logarithm of β with base α, denoted as $\log_\alpha \beta$, is defined as the smallest possible positive integer x such that

$$\alpha^x := \underbrace{\alpha \cdot \ldots \alpha}_{x \text{ times}} = \alpha^x = \beta$$

if such x exists, and $+\infty$ otherwise. Note that $\log_\alpha \beta$ is finite if and only if β lies in the subgroup generated by α.

For example, if $G = (\mathbb{Z}/n\mathbb{Z}, +)$, then the discrete logarithm $\log_\alpha \beta$ is the smallest solution to the linear congruence $\alpha x \equiv \beta \pmod{n}$. When α is invertible $\pmod{n}$, there exists a unique solution to the congruence $\beta\alpha^{-1} \pmod{n}$. However, if that is not the case, then there may exist multiple solutions: we can determine α in general using the generalized Euclidean algorithm.

To illustrate that the discrete logarithm indeed shares some properties with the familiar logarithm, we present the following result. For both properties, we use the fact that in a finite group, $\log_\alpha(\alpha^x) = x \pmod{\operatorname{ord} \alpha}$, where the right-hand side is the nonnegative remainder of the division of x by the order of the element α in the group.

Claim 1.1. *Let α, β, γ be elements of a finite group G such that $\beta, \gamma \in \langle \alpha \rangle$. The following results holds.*

Date: December 13, 2022.
This expository paper was produced for the Fall 2022 offering of 18.704, MIT's *Seminar in Algebra*.

(1) $\log_\alpha(\beta\gamma) = \big(\log_\alpha(\beta) + \log_\alpha(\gamma)\big) \bmod (\operatorname{ord}\alpha)$.
(2) If $\beta \in \langle\gamma\rangle$, *then* $\log_\gamma(\beta) = \log_\alpha(\beta)\log_\gamma(\alpha) \bmod (\operatorname{ord}\gamma)$

Proof. For the first result, since $\beta, \gamma \in \langle\alpha\rangle$, we may set $x = \log_\alpha(\beta), y = \log_\alpha(\gamma)$. Then

$$\log_\alpha(\beta\gamma) = \log_\alpha(\alpha^{x+y}) = (x+y) \bmod (\operatorname{ord}\alpha) = \big(\log_\alpha(\beta) + \log_\alpha(\gamma)\big) \bmod (\operatorname{ord}\alpha).$$

For the second part, write $x = \log_\alpha(\beta), y = \log_\gamma(\alpha)$. Then

$$\log_\gamma(\beta) = \log_\gamma(\alpha^x) = \log_\gamma(\gamma^{xy}) = xy \bmod (\operatorname{ord}\gamma).$$

□

The discrete logarithm problem is the computational problem of computing $\log_\alpha \beta$, given an input α, β, as efficiently as possible. It is often abbreviated as DLP in literature. Note that we have not yet specified the means of representing the group has not been made clear, but it shall be visited soon. The DLP is, in some sense, the "inverse" operation of the exponentiation problem, which given $\alpha \in G, x \in \mathbb{N}$, aims to compute α^x as efficiently as possible. Before presenting algorithms for the DLP, let us first see a reason why we would care about it in the first place.

Definition 1.2 (Primitive roots)**.** Consider a prime p. We say that $g \in (\mathbb{Z}/p\mathbb{Z})^\times$ is a primitive root if and only if $\mathbb{F}_p^\times = (\mathbb{Z}/p\mathbb{Z})^\times = \langle g\rangle$.

Note that this definition makes sense due to the well-known fact that the multiplicative group of a finite field is cyclic.

1.1. Why the DLP?.

In general, many cryptographical schemes rely on the "fact" that a certain problem is "hard", often much more difficult than its inverse. The RSA cryptosystem exploits the fact that factoring a semiprime $n = pq$ into its prime factors p, q is a **much** more difficult task than computing n by multiplying out p, q. In a similar fashion, the DLP is computationally much more difficult than its inverse problem of exponentiation. Protocols such as the Diffie-Hellman key exchange exploit this fact.

The Diffie-Hellman key exchange protocol between Alice and Bob [1] operates as follows [5]:

(1) Alice and Bob jointly choose (g, p), where p is a cryptographic-size prime [2] and g is a primitive root modulo p.
(2) Alice chooses a secret integer a (which she does not share), and shares with Bob the value $g^a \bmod p$.
(3) Bob chooses a secret integer b (which he does not share), and communicates with Alice the value $g^b \bmod p$.
(4) The secret key is given by $g^{ab} \bmod p$, which they can both compute.

An eavesdroppper to this conversation would have access to the prime p as well $g, g^a, g^b \bmod p$, but would need to compute the secret key $g^{ab} \bmod p$. To do so, one way is to use the Discrete Logarithm Attach, which calculates b using the Discrete Logarithm and then raises $g^a \bmod p$ to the power of b.

[1] it is "convention" to use Alice and Bob as the two sides in a cryptographic scheme
[2] This usually ranges from 256 to 2048 bit prime

This exchange scheme works in the group $(\mathbb{Z}/p\mathbb{Z})^\times$. Of course, we can modify this scheme to use any finite group G, but somehow the choice of which group should be something that both Alice and Bob would know.

Example 1.3 (Layman Exponentiation)**.** Consider the exponentiation problem of computing $2^{41} \bmod 47$. A natural way would be to repeatedly square using the fact that

$$2^{n+1} \equiv 2(2^n \bmod 47) \pmod{47}.$$

We would therefore get the sequence of residues

$$2, 4, 8, 16, 32, 17, 34, 21, \ldots, \boxed{25}.$$

Note that in general, computing $a^n \bmod m$ here would involve n multiplications (group operations) and n reductions modulo m.

Example 1.4 (Double-and-Add)**.** Consider the same exponentiation problem as above. A more efficient way to conduct this computation would be to look at the binary expansion of 41, which is $\overline{101001}_2$. We could then use the following algorithm, which can compute a^n in any monoid.

Algorithm 1 Double-and-Add

```
procedure DOUBLE-AND-ADD($\alpha, n$)
    Compute the binary representation $\sum_{i=0}^{k} a_i 2^i$ of n
    h $\leftarrow$ 1
        For i from k-1 to 0
        $h$ $\leftarrow$h$^2$
         if $a_i = 1$ then:
           h $\leftarrow$ $hg$
             return $h$
```

Note that this algorithm only uses $O(\log_2 n)$ group operations, and only stores the binary representation of n, as well as one group element (which is updated) and one pointer. Obtaining the binary representation of n is elementary to do, and can be do in $O(\log_2 n)$ group operations.

1.2. **Nuances of the Problem.** Note that the DLP, *per se*, is a perfectly well-defined problem over an infinite group. Nevertheless, we assume that G is a finite group. The reasons for this assumption are threefold. Firstly, most groups that arise in applications (such as cryptography) tend to be finite. Secondly, there is an inherent difficulty associated with formally defining the problem for inputting arbitrary infinite groups into an algorithm. Thirdly, and more relevant to us, is that finite groups impose a uniform upper bound on the discrete logarithm: $\log_\alpha \beta \leq |G|$. More generally, we usually require that the discrete logarithm exists - otherwise, it will not be known whether a probabilistic algorithm does not succeed because we are simply unlucky, or because the logarithm odes not exist.

We also have to deal with the formal question of the representation of the group G. That is, what would the input of any algorithm that solves the DLP be. In the next paragraph, we shall provide a sketch of this representation, but a complete formalization would be out of the scope of the paper. The uninterested reader is welcome to completely ignore the remainder of this subsection.

The workaround is to represent the finite group using strings of bits. An *encoding* of G is an injective map $\sigma : G \to \{0,1\}^n$. This allows us to represent each element of G as a string of bits, something that Turing machines can process as an input. We also want our encoding σ to appropriately interact with the group structure through an oracle. We therefore require that the oracle model supports certain properties:

(1) Identity: we can output $\sigma(1_G)$.
(2) Inverse: given $\sigma(\alpha)$, we can output $\sigma(\alpha^{-1})$ for all $\alpha \in G$.
(3) Group Operation: given $\sigma(\alpha)$ and $\sigma(\beta)$, we can output $\sigma(\alpha\beta)$ for all $\alpha \in G$.
(4) Random element: we can output $\sigma(\alpha)$ for a uniformly randomly distributed element $\alpha \in G$.

A *generic group algorithm* on a group G is an algorithm that implements a certain task on the group through solely interacting with an oracle equipped with *any* encoding. As such, we search for a *generic group algorithm* which, on input $\sigma(\alpha)$ and $\sigma(\beta)$, uses the above oracle to compute $\log_\alpha(\beta)$.

Remark 1.5. Note that the fourth point allows generic group algorithms to be probabilistic, i.e. nondeterminstic, algorithms that use randomness as part of their procedure. Note also that this computational model allows us to compare the equality of two elements using their encodings thanks to the injectivity of σ:

$$\alpha = \beta \iff \alpha\beta^{-1} = 1 \iff \sigma(\alpha\beta^{-1}) = 1,$$

and the latter can be computed using $\sigma(\alpha)$ and $\sigma(\beta)$.

1.3. **Asymptotic Analysis.** In this paper, we will study algorithms pertaining to the discrete logarithm problem. To quantify this study, we use asymptotic notation to measure the space/time complexity of the algorithm.

Throughout this section, let f, g be functions whose domain is either $\mathbb{R}$ or $\mathbb{N}$ and whose codomain is $\mathbb{R}$. We impose the additional constraint that g is nonzero for sufficiently large inputs.

Definition 1.6 (Big-O Notation)**.** We say that $f(x) \in O(g(x))$ if

$$\limsup_{x\to\infty} \left(\left| \frac{f(x)}{g(x)} \right| \right) < \infty.$$

Definition 1.7 (Big-Omega Notation)**.** We say that $f(x) \in \Omega(g(x))$ if

$$\liminf_{x\to\infty} \left(\left| \frac{f(x)}{g(x)} \right| \right) < \infty.$$

Definition 1.8 (Big-Theta Notation)**.** We say that $\Theta(g)$ to be the intersection of $O(g)$ and $\Omega(g)$.

Definition 1.9 (Small-o Notation)**.** We say that $f(x) \in o(g(x))$ if

$$\lim_{x\to\infty} \left(\left| \frac{f(x)}{g(x)} \right| \right) = 0.$$

Remark 1.10. Note that $O(g(x))$ is technically a class of functions, hence the inclusion notation $f(x) \in O(g(x))$. However, we shall follow the commonly-committed abuse of notation in saying $f(x) = O(g(x))$.

Example 1.11. Note that $p(x) = o(e^x)$ since exponentials grow faster than any polynomial. Similarly, $x^2 + x = \theta(x)$.

The following theorem, whose proof is beyond the scope of this paper, presents a lower bound that we seek to achieve. Henceforth, our model for time complexity will measure the number of group operations needed, since that is what matters most from a group-theoretic perspective.

Theorem 1.12 (Shoup's Theorem)**.** *The runtime of a generic group algorithm for the discrete logarithm problem is at least $\Omega(\sqrt{|G|})$ group operations.*

2. Elementary Algorithms

There are many elementary algorithms that can be used to solve the discrete logarithm problem in a finite group G.

2.1. **Linear Search.** Consider the following algorithm.

```
procedure LINEAR SEARCH DLP(G, $\alpha, \beta$)
    $x \leftarrow 0$
    $g \leftarrow 1$
    while x $\leq |G|$ do:
        if $g == \beta$ then: return $x$
        else x+=1, $g = g \cdot \alpha$
```

The algorithm still relies on looping over possible powers of α until we find β. The worst case runtime can use at most $|G|$ group operations.

2.2. **Baby-Steps Giant-Steps.** There exists also an algorithm known as Baby-Steps Giant-Steps which relies on the "meet me in the middle" principal to calculate the exponent.

```
procedure BABY-STEPS GIANT STEPS(G,$\alpha, \beta$)
    Find integers $r, s \geq 1$ such that $rs \geq |G|$.
    Map $i \to \alpha^i$ for $i \in 0, \dots, r-1$ into a hash function.
    Set $x = \beta$.
    for $j \in 0, \dots, s-1$ do
        if collision occurs with $i$ then return $i + rj$
        else: x $\leftarrow x\alpha^{-r}$
```

To see why this algorithm works, we need to show that any nonnegative integer which is less than $|G|$ can be written in the form $i + rj$, where $0 \leq i \leq r-1$ and $0 \leq j \leq s-1$. This is true viewing it as Euclidean division. The condition $rs \geq |G|$ is needed to cover all remainders.

The space complexity of this algorithm is $\Omega(r+s)$ group elements which is at least $\Omega(\sqrt{|G|})$ by the AM-GM inequality, and trivially at most $O(|G|)$. The time complexity of the algorithm can be easily seen to be

$$\sum_{k=1}^{r} \log_2 k + \log_2 r + O(s) = \frac{r \log r - r}{\ln(2)} + O(\log r) + O(s),$$

where the last equality holds by Stirling's approximation. To demonstrate the difference between the two algorithms, we ran the two previous algorithms on CoCalc, yielding the following runtimes.

Size of p (bits)	BSGS	Linear Search
15	0.00055 s	0.0088 s
20	0.0474 s	0.4593 s
30	0.211s	824 s

Table 1. Runtime of Baby-Steps Giant-Steps vs. Linear Search

3. Discrete Logarithms in Nilpotent Groups

Following Romankov [2], we specialize to the case of finite nilpotent groups. Note that a finite nilpotent group is the product of its Sylow p-subgroups, and so we only present an algorithm for the case where our group G is a p-group of order p^k.

3.1. **Algorithm.**

Since G is nilpotent, it possesses a normal series

$$G = G_0 > G_1 > \cdots > G_k = 1.$$

Suppose that we would like to solve the discrete logarithm problem $\alpha^x = \beta$ for some unknown x. The algorithm proposed by Roman'kov calculates x by determines its base p expansion.

Define B_i to be the quotient G_i/G_{i+1}, an abelian p-group of rank r_i. The image of an element $g \in G$ is $\overline{g} \in B_0$.

Consider the discrete logarithm problem $(\overline{\alpha}, \overline{\beta})$. Compute the exponent x_0 associated to this DLP in B_0, and set $\alpha_1 = \alpha^p, \beta = \alpha^{-x_0}\beta \in G_1$. This now reduces to the problem in G_1:

$$\alpha_1^{(x-x_0)/p} = \beta_1.$$

If both reductions are the identity, then they already live in G_1 and we could have started the problem there. Then continue with

$$\alpha^{x - x_{k-1}p^{k-1}} = \beta$$

in G_1.

If $\beta \in G_1$ and $\alpha \notin G_1$, we have $x_0 = 0$ and so we continue with $\alpha_1^{x/p} = \beta$. Repeatedly applying this process, we obtain a solution to the discrete logarithm problem.

3.2. **Lifting Back.** Since G is a finite nilpotent group, it is the product of Sylow p-subgroups:

$$G = \prod_{p \in \pi(G)} G_p.$$

Write an element g in the form $g = (g_p)_{p\in\pi(G)}$. Applying the algorithm in the previous subsection we obtain a tuple of exponents $(x_p)_{p\in\pi(G)}$. Then a solution to the initial DLP problem can be obtained by finding a solution to the system of congruences for $p \in \pi(G)$

$$x \equiv x_p \bmod p^{t_p},$$

where p^{t_p} is the order of g_p. Using the Chinese Remainder Theorem, we can solve this system of congruences and obtain the minimal solution x.

Providing a runtime analysis of Romankov's algorithm is much more complicated. The algorithm not only involves solving DLP one group, but it projects to quotients and so the analysis will vary from model to model, and this is beyond the focus of the paper.

3.3. **Cryptographic Impracticality of the Algorithm.** Although the algorithm presented by Romankov is theoretically correct, Kahrobaei et al. [1] showed that it is impractical for cryptogrpahic-size inputs.

Determining a_0 when p is large enough can be quite difficult. Assume that $p = 2q + 1$ for another prime q, known as a safe prime, and that $G = \langle g^2 \rangle$, where g is a primitive root modulo p. Then the nilpotency series becomes

$$G = G_0 > G_1 = \{1\},$$

so determining a_0 amounts to solving the DLP in G, a group of order q. This means that the problem does not get much easier, and so the practicality of the implementation is compromised. Since then, cryptographic schemes for DLP which are based on nilpotency have been developed.

4. Pollard rho Algorithm

One can solve the DLP more efficiently by expanding our scope to probabilistic algorithms. To do so, we view group operations as a path on the elements of G, giving rise to the Pollard rho algorithm [6].

Definition 4.1. Let S be a finite set. A random function $f : S \to S$ is defined by assigning to each $s \in S$, an image in S uniformly randomly.

Definition 4.2. Let S be a finite set, and let $f : S \to S$ be a function on S. Let $s = s_1 \in S$ be the initial value, and consider the recurrence given by $s_{i+1} = f(s_i)$ for $i \geq 1$. We define the value $\rho_s(f)$ to be the smallest ρ such that there exists $\lambda < \rho$ such that $s_\lambda < s_\rho$, and $\lambda_s(f)$ to be the associated λ.

Note that by finiteness of S, we know that $\rho_s(f)$ is finite. In fact, it is bounded above by $|S| + 1$ using the pigeonhole principle.

Lemma 4.3 (Expected Values for $\boldsymbol{\rho}$ in Random Walks)**.** *Let S be a finite set. For any $s \in S$ and a random function $f : S \to S$, the asymptotic relation*

$$\mathbb{E}[\rho_s(f)] \to \sqrt{\frac{\pi|S|}{2}}$$

holds as $|S| \to \infty$. Moreover,

$$\mathbb{E}[\lambda_s(f)] \to \sqrt{\frac{\pi|S|}{8}}.$$

Proof. We only prove the first result, as the second has a similar proof. Using basic counting, one can see that for any $n \geq 0$

$$\Pr[\rho_s(f) > n] = \prod_{i=0}^{n} \left(1 - \frac{i}{|S|}\right).$$

Now

$$\mathbb{E}[\rho_s(f)] = \sum_{n=1}^{|S|-1} n \Pr[\rho_s(f) = n] = \sum_{n=1}^{|S|-1} n\Big(\Pr[\rho_s(f) > n-1] - \Pr[\rho_s(f) > n]\Big),$$

which telescopes to

$$\sum_{i=1}^{|S|-1} \Pr[\rho_s(f) > n] - (|S|-1) \Pr\big[\rho_s(f) > |S|-1\big].$$

Note that

$$\Pr[\rho_s(f) > n] = \exp\left(\sum_{i=0}^{n} \log\left(1 - \frac{i}{|S|}\right)\right) < \exp\left(-\frac{1}{|S|}\sum_{i=0}^{|S|} i\right) < \exp\left(\frac{-n^2}{2|S|}\right).$$

Similarly using bounds that following from the Taylor series,

$$\Pr[\rho_s(f) > n] = \exp\left(\sum_{i=0}^{n} \log\left(1 - \frac{i}{|S|}\right)\right) > \exp\left(-\sum_{i=1}^{n}\Big(\frac{i}{|S|} + \frac{i}{|S|^2}\Big)\right).$$

Assuming that $n < |S|^{0.6}$, we get that

$$\sum_{i=1}^{n}\Big(\frac{i}{|S|} + \frac{i}{|S|^2}\Big) < \sum_{i=1}^{n}\Big(\frac{i}{|S|} + |S|^{-0.8}\Big) < \frac{n^2+n}{2|S|} + |S|^{-0.2} < \frac{n^2}{2|S|} + 2|S|^{-0.2}.$$

Since the latter term can be bounded by a constant, this shows that

$$\Pr[\rho_s(f) > n] > C \exp\frac{-n^2}{2|S|}$$

for some constant $C \in 1 + o(1)$.

Now combining this to the telescoping formula we proved earlier

$$\sum_{n=0}^{\lfloor M \rfloor} \Pr[\rho_s(f) > n] + \sum_{n=\lceil M \rceil}^{|S|-1} \Pr[\rho_s(f) > n] + o(1),$$

where $o(1)$ arises from the term $n \Pr[\rho_s(f) > n] < n \exp(-n^2/2|S|)$. We also note that the second term is negligible (that is, $o(1)$) since it is bounded above by

$$|S| \exp(-M^2/2|S| = |S| \exp\left(-|S|^{-0.2}/2\right)) = o(1).$$

Note

$$\sum_{n=0}^{\lfloor M \rfloor} \Pr[\rho_s(f) > n] = \sum_{n=0}^{\lfloor M \rfloor} \Big(1 + o(1)\Big) \exp\Big(\frac{n^2}{2|S|}\Big).$$

which, up to a difference of $O(1)$, is equal to

$$\Big(1 + o(1)\Big) \int_0^\infty e^{-\frac{x^2}{2|S|}}.$$

Integrating this yields $\Big(1 + o(1)\Big)\sqrt{\frac{\pi|S|}{2}}$, and taking the limits proves the result. $\square$

Instead of defining a random function, Pollard's algorithm will partition G into three disjoint subsets of almost equal size S_1, S_2, S_3, and will use the following function.

$$f(X) = \begin{cases} \alpha X & \text{if } X \in S_1 \\ X^2 & \text{if } X \in S_2 \\ \beta X & \text{if } X \in S_3 \end{cases}.$$

It will then run starting at 1_G, and will proceed by storing all previous iterants until a collision is detected. Any collision will give a result of the form $\alpha^X \beta^Y = \alpha^x \beta^y$, from which we can apply the extended Euclidean algorithm to determine $\log_\alpha \beta$. Note that here we need to determine the order of α in order to apply the discrete logarithm, but that can be done relatively easily using other algorithms.

Remark 4.4. One mnemonic about the algorithm explains why it is called ρ as follows. Tracing from the root of ρ (initial value), the algorithm traverses elements of the field, but it then cycles back to a point it had already passed through, completing the loop in ρ.

In this case, it is assumed that this function indeed behaves like a random function. Note that this implementation will indeed allow us to achieve Shoup's lower bound of $\Omega(\sqrt{|G|})$, as the theorem above indicates. However, a slight improvement could be made.

The current implementations stores all the previous iterants to check for collisions. Instead, we can convert this information into a graph-theoretic sense: with vertices being group elements, the random function corresponding to choice of outgoing vertices. Then a cycle-detection algorithm, such as Floyd's algorithm, is used to detect cycles without having to explicitly run and store all iterants. This will reduce the space complexity of the algorithm, and will add a "negligible" number of group operations associated with the graph-theoretic portion of the algorithm. Similar improvements which optimize the probabilistic model and the graph-theoretic component have been made to the Pollard-rho algorithm. These essentially seek to reduce the constant c in the runtime of $c\sqrt{|G|}$.

5. Conclusion

The DLP highlights a central property of computational group theory: the ability to use advanced results in pure mathematics for computational applications. Throughout, we have used arguments and tools from group theory, probability, and graph theory to develop efficient algorithms that achieve Shoup's minimal bound.

We conclude by presenting a summary of the algorithms we surveyed:

Algorithm	Deterministic	Generic	Runtime
Linear Exponentiation	✓	✓	Linear
Double-and-add	✓	✓	Linear
Baby-steps Giant-steps	✓	✓	$O(r \log r + s)$
Romankov	✓	X	Complicated
Pollard rho	X	✓	$O(\sqrt{\lvert G\rvert})$

Table 2. Summary of the algorithms we presented for the Discrete Logarithm Problem

The DLP is a very active subject of research - there are plenty of improvements to the algorithms we presented that we did not cover in detail, let alone completely new algorithms in literature. One cannot write a paper about the DLP without mentioning, at least in passing, the index method for the interested reader - it solves in the DLP in the multipliclative group of a finite field $\mathbb{F}_{p^k}$. The method exploits the fact that finite fields can be related, after lifting, to the integers, and so there is multiplicative but also additive structure. The Pohlig-Hellman algorithm is also a central topic in the field. The discrete logarithm problem continues to be at the forefront of computational group theory: both because of the rich theoretical background and the computational and cryptographic relevance of the problem.

Acknowledgements

I would like to thank Kent Vashaw for his valuable feedback and suggestions, and my classmates Mark Jabbour and Boris Velasevic for their valuable peer-review. I would also like to thank Professor Bjorn Poonen for the LaTeX template on which which this paper was created, and Andrew Sutherland for introducing me to this beautiful topic.

References

[1] Kahrobaei et al., A Closer Look at the Multilinear Cryptography using Nilpotent Groups, 2021, arxiv.org/pdf/2102.04120.pdf. 7

[2] V. A. Roman'kov, Discrete logarithm for nilpotent groups and cryptanalysis of polylinear cryptographic system, Prikl. Diskr. Mat. Suppl., 2019, Issue 12, 154–160. 6

[3] Granger et al, On the discrete logarithm problem in finite fields of fixed characteristic, American Mathematical Society, 2017, Issue 5, pp 3129–3145.

[4] A. Sutherland, MIT 18.783 Lecture Notes, 2022.

[5] E. Dummit, Discrete Logarithms in Cryptography. URL. 2

[6] J.M. Pollard, Monte Carlo methods for index computation (mod p), Mathematics of Computation 143 (1978), 918–924.

7

Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139-4307, USA

Email address: `chentouf@mit.edu`