# Algorithms for Solving Discrete Logarithm Problem

**[1]ANU PIUS, [2]SENTHILKUMAR B**

[1]M.Phil Research Scholar, [2]Assistant Professor
PRIST University, Thanjavur

***Abstract*: According to public key cryptosystems on integers, Discrete Logarithm Problem (DLP) is hard to break was the important assumption. These computational assumptions were tested by developing improved algorithms to decipher them. DLP for elliptic curves defined over certain finite fields is trusted to be difficult. For these types of problems, Pollard rho is the most suitable algorithm. The most effective and efficient idea to attack DLP is index calculus algorithm, because it solves DLP for finite fields in sub-exponential time. It is essential to comprehend these algorithms**

***Keywords*: Pollard-Rho Algorithm, Baby step Giant step Algorithm, Index Calculus Algorithm, Pohlig-Hellman Algorithm**

1. **Introduction**: Secure cryptosystems built by using complex computational problems. The RSA and Diffie-Hellman key exchange were on assumption that integer factorization and discrete logarithm problems were complex to decipher. If one breaks the underlying assumption, then cryptosystem is not secure anymore.

**Definition 1.1** *(One-way functions) Let* $\boldsymbol{k}$ *be a security parameter and* $\boldsymbol{n}$ *be a function of* $\boldsymbol{k}$*. Let f be* $\boldsymbol{f} : \{\boldsymbol{0},\boldsymbol{1}\}^* \rightarrow \{\boldsymbol{0},\boldsymbol{1}\}^*$. *Then f is a one-way function if*

1. *f is easy to compute. For all* $\boldsymbol{n}$ *and* $\boldsymbol{x} \in \{\boldsymbol{0},\boldsymbol{1}\}^{\boldsymbol{n}}$*, there is a deterministic polynomial time algorithm* $\boldsymbol{f_\alpha}$ *such that* $\boldsymbol{f_\alpha}$ $(x)$ *= f(x).*
2. *f is hard to invert. For all probabilistic polynomial time algorithms* $\boldsymbol{A}$*,*

$$\Pr\{x \longleftarrow \{0,1\}^n, y = f(x), x^1 \longleftarrow A(1^n, y) | f(x^1) = y\} < \frac{1}{2^k},$$

In addition to the one-way property of the encryption function, we also require Charles, who possesses the symmetric key $\boldsymbol{s_{K}}$ *to* decrypt the message. So we allow the decryption to be possible using a trapdoor, secret information that allows to easily inverting the encryption function. Such functions are called one-way trapdoor functions.

These one-way trapdoor functions are the building blocks of modern cryptosystems based on computational number-theoretic assumptions such as the integer factorization and discrete logarithm problems.

## 1.2 The integer factorization problem and RSA cryptosystem

**Definition 1.2.1**. *(Integer Factorization Problem) Let* $\boldsymbol{W}$ *be a positive composite integer. The integer factorization problem is to find the prime factorization of W and write* $\boldsymbol{W}$ *as* $W = \prod_{i=1}^{k} \boldsymbol{q_i}^{\alpha_i}$

*Where* $\boldsymbol{\alpha_i} > \boldsymbol{1}$ *and the primes* $\boldsymbol{q_i}$ *are pairwise coprime.*

The integer factorization problem is a well-studied problem and it is one of the number theoretic computational problems believed to be a candidate for a one-way function. The RSA [8] asymmetric key cryptosystem is based on the intractability assumption of factoring a large composite integer $\boldsymbol{W} = \boldsymbol{pq}$, where $\boldsymbol{p}$ and $\boldsymbol{q}$ are two distinct primes. The integer $\boldsymbol{W}$ ***is*** called a modulus.

The RSA cryptosystem is composed of the following algorithmic functions (See Chapter 24 Section 1 page 486 of [4]).

**Key formation**: On input, a security parameter $\boldsymbol{k}$, the probabilistic polynomial time key formation algorithm, Key formation formulates two distinct primes $\boldsymbol{p}$ and $\boldsymbol{q}$ of size approximately $K/\boldsymbol{2}$ bits each and sets $\boldsymbol{W} = \boldsymbol{pq}$. It chooses a random integer $\boldsymbol{\alpha}$ co-prime to $\boldsymbol{p} - \boldsymbol{1}$ and $\boldsymbol{q} - \boldsymbol{1}$ such that $\boldsymbol{p},\boldsymbol{q} \neq \boldsymbol{1}$ (mod $\boldsymbol{\alpha}$), and computes $\boldsymbol{d} = \boldsymbol{\alpha}^{-\boldsymbol{1}}\,(\boldsymbol{mod}\,\boldsymbol{\vartheta}(\boldsymbol{W})), \boldsymbol{where}\ \boldsymbol{\vartheta}(\boldsymbol{W}) = \boldsymbol{gcd}(\boldsymbol{p} - \boldsymbol{1}, \boldsymbol{q} - \boldsymbol{1})$ is the Carmichael function. The key formation algorithm then outputs $\boldsymbol{s_K} = (\boldsymbol{W},\boldsymbol{d})$ as the symmetric key and $\boldsymbol{p_K} = (\boldsymbol{W},\boldsymbol{\alpha})$ as the asymmetric key. Note that $\boldsymbol{gcd}$ stands for the greatest common divisor.

**Encryption**: Let **T** = $\mathbb{C}$ = $F^*{}_W$ be the plaintext and ciphertext spaces. The polynomial time encryption algorithm, Encryption takes the asymmetric key $\boldsymbol{p_K}$ and $t \in$ **T** as input and outputs $\boldsymbol{c} = \boldsymbol{t}^{\boldsymbol{\alpha}}\,(\boldsymbol{mod}\,\boldsymbol{W})$, where **c** $\in \mathbb{C}$ is the encryption of the message **t .**

***Decryption***: The deterministic polynomial time decryption algorithm, Decryption takes the symmetric key $\boldsymbol{s_K}$ and the ciphertext **c** as input and outputs ***t = c^d* (mod *W*).** It is required that Decrypt (Encrypt ($\boldsymbol{p_K}$, *t*), $\boldsymbol{s_K}$*) = t.*

**Sign(t, $s_K$):** On input the symmetric key parameter $s_K$ and a message ***t***. The probabilistic polynomial time signing algorithm Sign(t, $s_K$) outputs $s = t^d \ (mod\ W)$, a signature of the message ***t***. Note that there attacks such as a chosen-plaintext and a chosen-ciphertext against this plain signature scheme. An actual signature algorithm uses padding schemes and hash functions.
**Verify (t, s, $p_K$)** : On input, the asymmetric key parameter $p_K$, the signature ***s***, and the message t, the deterministic polynomial time verification algorithm Verify(t, s, $p_K$)computes $\hat{t} = s^\alpha$ **(mod *w*)** and outputs valid if $\hat{t}$ = ***t*** otherwise returns invalid.

For the encryption algorithm to be fast, it is tempting to take the asymmetric key $\alpha$ to be small such as $\alpha \in \{2^4 + 1, 2^{16} + 1\}$. The Rabin cryptosystem is a special type of RSA cryptosystem with $\alpha = 2$ and the primes $p$ ***and*** $q$ are selected to satisfy $p = q = 3 \ (mod\ 4)$ to simplify computations. We refer to Chapter 24 Section 2 page 491 of [4] for details.

To encrypt a message ***t*** given the asymmetric parameters $p_K = (W, e)$., we compute $c = t^\alpha$ **(mod *W*).** The corresponding symmetric key $s_K$ = ***(W, d)*** acts as a trapdoor. Whereas in the signature scheme, the holder of the symmetric key parameters signs a message or a document using the symmetric key parameters. One can then verify that indeed the message or document is signed by who claim to be the legitimate signer.

**Definition 1.2.2.** *(RSA Problem) Let* $c = t^\alpha$ **(mod *W*)**, *where* ***W= p q*** *is a product of two distinct primes. The RSA problem is to recover* ***t*** *given* **c** *and the asymmetric key parameters* $p_K = (W, e)$. *In other words, the RSA problem is computing the* $\alpha$ *root modulo W*

If factoring is easy, clearly breaking the RSA cryptosystem is easy too. We factor ***W*** to get its two prime factors ***p*** and ***q***, and we compute $\vartheta(W) = gcd(p - 1, q - 1)$. Finally, we recover ***d*** by computing $\alpha^{-1}$ (mod $\vartheta(W)$) using the extended Euclidean algorithm. So in order for the RSA cryptosystem to be secure, $p$ ***and*** $q$ should be large primes such that it is computationally infeasible to factor ***W*** with current methods.

## 1.3 The discrete logarithm problem and Elgamal cryptosystem

As with the integer factorization problem, the discrete logarithm problem over finite fields is believed to be a hard computational problem.

**Definition 1.3.1.** ***(DLP)*** *Let* **H** *be a multiplicative group. Let* $g$, **h** $\in$ **H**. *Define* $\{< g >\}$ ***to be*** $\{< g >\} = \{ g^i / 0 \le i \le d - 1\}$,

*where* ***d*** *is the order of* $g$ *Given* $h \in < g >$, *the DLP is to find the unique integer* $\alpha, 0 < \alpha < d - 1$, *such that* $h = g^\alpha$. *It is denoted as* $\log_g h$

There are many cryptosystems that make use of the hardness of the discrete logarithm problem. A popular example is the ElGamal cryptosystem [3](see also Chapter 6 Section 1 of [10] and Chapter 20 Section 3 of [4]). As in the design motivation of the RSA and Rabin cryptosystems, the principle behind the design of DLP based cryptosystems is based on the one-way property of the exponentiation function. Given $\alpha$ we can compute $g^\alpha$ easily using the square-and-multiply algorithm, whereas computing the inverse ($\log_g h$) is a difficult computational problem for appropriate size parameters. Currently, a key length of **224 bits** and a group size of **2048 bi**ts are recommended [5].
Let $H = F_q{}^*$, where ***q*** is a prime. The ElGamal cryptosystem is composed of the following algorithms.

**Key formation**: Let **K** be a security parameter. On input **K**, the probabilistic polynomial time key formation algorithm, Key formation formulates a **K** bit prime ***p*** and an element $g \in F_q{}^*$, It then chooses a random integer $1 < \alpha < p - 1$ and sets $h = g^\alpha$ **(mod *q*).** The key formation algorithm finally outputs $s_K$ = **(p,** $g$, $\alpha$**)** as the symmetric key and $p_K = (p, g, h)$ as the asymmetric key.

**Encryption**: Let $T = F_q{}^*$, and $\mathcal{C} = F_q{}^* \times F_q{}^*$, be the plaintext and ciphertext spaces respectively. The probabilistic polynomial time encryption algorithm, Encryption key $(p_K, t)$ takes the asymmetric key $p_K$ and a message $t \in T$ as input. It chooses a random integer $1 < \beta < q - 1$ and sets $c_1 = g^\beta$ (mod $q$). It then outputs the ciphertext pairs $C = (c_1, c_2) \in \mathcal{C}$ as the encryption of the message $t$, where $c_2 = t\, h^\beta$ **(mod *q*).**

**Decryption**: The deterministic polynomial time decryption algorithm Decryption key $(C, s_K)$ takes the symmetric key $s_K$ and the ciphertext $C = (c_1, c_2)$ as input and outputs $t = c_2 c_1{}^{-\alpha} (mod\ q)$.

Given a ciphertext $C = (c_1, c_2)$the Elgamal decryption correctly decrypts. Indeed

$$t = c_2 c_1{}^{-\alpha} \quad (mod\ q).$$

$$= t\, h^\beta g^{-\alpha\beta} (mod q)$$

$$= \quad tg^{\alpha\beta}g^{-\alpha\beta}(\text{mod } q)$$
$$= \quad t \quad (\text{mod } q).$$

If we are able to solve the discrete logarithm problem, then the ElGamal cryptosystem is not secure. For example, given a ciphertext $C = (c_1, c_2)$, if we can

Compute $\log_g c_1$ to recover $\beta$, then $c_2 h^{-\beta}$ gives the message $t$. We can also recover the symmetric key $\alpha$ by computing $\log_g h$ from the asymmetric key parameter $h$ to finally recover the message $t$. by computing $t. = c_2 c_1{}^{-\alpha}$ $(mod\ q)$.

The Diffie-Hellman key exchange protocol [2]is another popular cryptosystem that makes use of the hardness of the DLP over $H$. Similar to the ElGamal cryptosystem, we will have common asymmetric key parameters $p$ and $g$. Assume Eve and Charles want to exchange a key Ƙ over an insecure channel. Eve generates a random integer $1 < \alpha < q - 1$ and sends $h_\alpha = g^\alpha$ (mod $q$) to Charles. And Charles does the same, he chooses a random integer $1 < \beta < q - 1$ and sends $h_\beta = g^\alpha$ (mod $q$) to Eve. Then Eve and Charles compute the same key Ƙ $= h_\beta^\alpha$ (mod $q$) and Ƙ $= h_\beta^\alpha$(mod $q$) respectively. The exchanged key Ƙ can be compromised if the Computational Diffie-Hellman (CDH) problem can be solved.

**Definition 1.3.2.** *(CDH) Let $H = F_q{}^*$, be a multiplicative group. Then the CDH problem is given the triple $(g, g^\alpha, g^\beta)$ elements of $H$ to compute.*

As in the ElGamal cryptosystem, if we are able to solve the DLP, then clearly the CDH problem can be solved.

## 1.4 Algorithms for solving the discrete logarithm problem

*Let* $H$= $F_q$* *such that* $H = \langle g \rangle$,where $q$ is a prime. Let $h \in \langle g \rangle$. The DLP is to find a unique integer $\alpha < q$ such that $= g^\alpha$. The problem definition can also be extended to the case when $H \subset$ $F_q$* having a generator $g$ with order $d$, where $r$ is a prime power and $d = \#H$. We list some algorithms to solve the DLP problem.

### 1.4.1 The baby-step-giant-step algorithm

Let $\beta = \lceil\sqrt{d}\,\rceil$ Write $\alpha$ in base-$\beta$ representation as $\alpha = i\beta + j$ $for$ $0 \le i, j \le \beta$ The baby-step-giant-step algorithm [9] is based on the observation $g^j = g^a(g^{-m})^i = h(g^{-\beta})^i$ The algorithm involves two steps.

**The baby-step phase:** For $0 < j < \beta$, $g^j$ is computed and the values $(j, g^j)$ are stored in a list.

**The giant-step phase**: For $0 < i < \beta$, $h(g^{-\beta})^i$ is computed and checked for a match in the second entry of the baby-step stored list.

If a match is found then $\alpha = i\beta + j$ is a solution to the DLP. The baby-step-giant-step is a typical example of a time/memory tradeoff algorithm. It requires $O[\sqrt{d}\,]$storage and $O$ $(\sqrt{d}\,)$ arithmetic operations in $H$.

### 1.4.2 The Pohlig-Hellman Algorithm

Let the order of $g$ be given as $d = \prod_{i=1}^{k} p_i{}^{e_i}$ the idea of the Pohlig-Hellman algorithm [6] is based on the observation of the group homomorphism

$\varphi_{p_i}(g) = g^{d/p_i{}^{e_i}}$ from $< g >$,to a cyclic subgroup of order $p^e$.

Under the homomorphism, we have

$\varphi_{p_i}(h) = \varphi_{p_i}(g)^\alpha (mod\ p_i{}^{e_i}.)$

Let $g_0$ have order $p^e$ **for some** $e > 1$ and let $h_0 = g_0{}^\alpha$ (mod $d$). Observe that $\alpha$ can be written as *a* $\alpha = \alpha_0 + \alpha_1 p + \cdots + \alpha_{e-1}p^{e-1}$, where $0 \le \alpha_i < p$. To compute $\alpha$ modulo $p^e$, it is enough to recover $(\alpha_0, \alpha_1, ::::::, \alpha_{e-1})$.

Let $g_1 = g_0{}^{p^{e-1}}$ Then $h_0{}^{p^{e-1}} = g_1{}^{\alpha_0}$ . So we can recover $\alpha_0$ by trying all possibilities (assuming $p$ is not large). We can also use other methods such as the baby-step-giant-step algorithm.

To recover $\alpha_1$, first define $h_1$ to be $h_1 = h_0 g_0{}^{-\alpha_0} = g_0{}^{\alpha_1 p + \cdots + \alpha_{e-1}p^{e-1}}$ . Then $h_1{}^{p^{e-2}} = g_1{}^{\alpha_1}$. So we can recover $\alpha_1$ using the same technique we have recovered $\alpha_0$.

Similarly to recover $\alpha_2$, we define $h_2 = h_1 g_0{}^{-\alpha_1 p} = g_0{}^{\alpha_2 p^2 + \cdots + \alpha_{e-1}p^{e-1}}$ Then we observe that $h_2{}^{p^{e-3}} = g_1{}^{\alpha_2}$ and $\alpha_2$ can be recovered. We continue to recover all $\alpha_i$ this way. So now we know $\alpha$ modulo $p^e$. For each prime power $p_i{}^{e_i}$ in the factorization of $d$ compute $\alpha$ modulo $p_i{}^{e_i}$. Then using the Chinese remainder theorem, $\alpha$ can be recovered.

**Lemma 1.4.2.1** *Let $h \in H$ has order $d$. Let $\mathcal{S}$ be abounded where $d$ is $\mathcal{S}$*-smooth. Then the DLP can be solved using the Pohlig-Hellman algorithm in

$$O((\log d)^2 + \mathcal{B} \log d)$$

group arithmetic operations in $H$.

### 1.4.3 The Pollard Rho algorithm

To compute $\log_g h$, the idea of the Pollard rho algorithm [7] is based on finding collisions between two sequences. Specifically, if we can find $(b_i, c_i, b_j, c_j) \in Z/rZ$ such that $g^{b_i}h^{c_i} = g^{b_j}h^{c_j}$

Where $c_i \neq c_j$ (mod d), then $h = g^{(b_j-b_i)(c_i-c_j)^{-1}}$(**mod d**).

So $\alpha \equiv (b_j - b_i)(c_i - c_j)^{-1}$**(mod *d*)** is a solution to the DLP.

We start by partitioning the group $H$ into three parts$S_1, S_2, S_3$ such that they are almost equal size. Define the pseudorandom function $f : (< g >, Z/rZ, Z/rZ) \to (< g >, Z/rZ, Z/rZ)$ as follows

$$(Y_{i+1}, b_{i+1}, c_{i+1}) = f(Y_i, b_i, c_i) = \begin{cases} g(Y_i, b_i + 1, c_i) & if\ Y_i \in S_1 \\ (Y_i^2, 2b_i, 2c_i) & if\ Y_i \in S_2 \\ (hY_i, b_i, c_i + 1) & if\ Y_i \in S_3 \end{cases}$$

Let $Y_i = g^{b_i}h^{c_i}$ The starting sequence $(Y_i, b_i, c_i)$is defined to be **(1, 0, 0).** Then we generate the next sequence $(Y_{i+1}, b_{i+1}, c_{i+1})$ according to $f$. As $H$ is a finite set, by birthday paradox we expect a collision
$Y_i = Y_j$ for some $1 < i < j$. If this is the case $g^{b_i}h^{c_i} = g^{b_j}h^{c_j}$ It is then immediate to find a solution to the DLP.
The DLP in group $H$ can be solved heuristically using the Pollard rho algorithm in

$$O((\sqrt{d})(\log d)^2)$$

group arithmetic operations in $H$

### 1.4.4 The index calculus method

The index calculus method [1] is the most effective algorithm to solve the DLP in finite fields in subexponential running time. The concept is similar to the quadratic sieve factorization method. The index calculus method remains our motivation to provide an index calculus algorithm for solving discrete logarithm problem over binary curves.
Let $S$ be the smoothness bound. We define a factor base $\mathcal{A}$ to be the set of primes $q_i$ less than $S$. $\mathcal{A} = \{q_1, q_2, \bullet\bullet\bullet, q_k\} \subset H$.

The idea of the index calculus method is to find the discrete logarithm of each prime element in the factor base $\mathcal{A}$ with respect to $g$ to finally solve the discrete logarithm problem. It has three stages.

**Stage 1**: This is called the relation generation stage. Pick a random value $k \in Z/rZ$ and compute $\mathcal{R} = g^k$.
If $\mathcal{R}$completely factors over the factor base $\#\mathcal{A}$, that is $\mathcal{R} = \prod_{i=1}^{k} p_i^{e_i}$ then we have a relation. Taking logs, each relation gives $k = \sum_{i=1}^{\#\mathcal{A}} e_i \log_g p_i$We store k as a column vector and $(e_1, e_2, \bullet\bullet\bullet, e_{\#\mathcal{A}})$ as a row in a matrix.

**Stage 2**: This is a linear algebra stage. In this stage, we compute the discrete logarithm of each factor base element with respect to $g$. If we collect $\#\mathcal{A}$ independent relations in stage **1**, then applying Gaussian elimination on the full rank matrix gives actual values of the discrete logarithm of the factor base elements with respect to $g$. Note that if $r$ is not prime, the Gaussian elimination (over the ring $\mathbf{Z_r}$) is not guaranteed to work.

**Stage 3:** In this stage, we find a relation that includes $h$. We continuously pick a random value $k^1 \in Z/rZ$ until $hg^{k^1}$ factors over the factor base $\mathcal{A}$. Assume $hg^{k^1}$ factors over the factor base $hg^{k^1} = \sum_{i=1}^{\#\mathcal{A}} p_i^{e_i 1}$ Let the discrete logs of each factor base elements found in stage **2** be given by $b_i = \log_g p_i$.Then the solution to the DLP is given by

$$\log_g h \equiv (\textstyle\sum_{i=1}^{\#\mathcal{A}} e_i^1 b_i) - k^1 \ (\text{mod } d).$$

**Lemma 1.4.2.1** *Let* $H = \mathbf{F_q}^*$ *such that* $H = <g>$*, where* $q$ *is a large prime. Let* $h = g^{\alpha}$ **(mod *q*).** *The index calculus algorithm solves* $\log_g h$ *in the subexponential time given by* $O(e^{c(\log q)^{1/2}(\log\log q)^{1/2}})$

*arithmetic group operations in* $H$ *for some constant* $c \in \mathcal{R}$ *greater than zero.*

***Proof: see pages 301-334 of[4]***

## CONCLUSION

The history of cryptography is filled with the back-and-forth between cryptographers creating "unbreakable" ciphers and cryptanalysts breaking the unbreakable. In this regard, many try to decipher the underlying complex computational problems of cryptosystems. If computational problem is intrinsically easy, one can provide algorithms to decipher the problem in polynomial time. We test computational assumptions by developing improved algorithms to solve them.in this paper we give a summary of the existing algorithms to solve the integer factorization and discrete logarithm problems.

**Bibliography**

[1] L. Adleman, J. DeMarrais. *A subexponential algorithm for discrete logarithms over all finite fields. In Advances in Cryptology*-CRYPTO'93, pages 147-158, 1994.

[2] W. Diffie, M. Hellman. *New directions in cryptography*. IEEE Trans. Information Theory, vol. IT-22(6), pages 644-654, 1976.

[3] T. ElGamal. *A public key cryptosystem and a signature scheme based on discrete logarithms. In Advances in Cryptology*, vol. 196 of LNCS, pages 10-18, 1985.

[4] S. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, Page 615, 2012.

[5] D.Giry. BlueKrypt| *Cryptographic Key Length Recommendation.* http://www.keylength. com/en/. BlueKrypt, vol. 29.2, 2015.

[6] S. Pohlig, M. Hellman. *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance*. IEEE Transactions on Information Theory, vol. IT-24 pages 106–110, 1978.

[7] J. Pollard. Monte Carlo *methods for index computation (mod p). Mathematical Computation*, vol. 32(143), pages 918–924, 1978.

[8] R. Rivest, A. Shamir, L. Adleman. *A method for obtaining digital signatures and asymmetric-key cryptosystems. Communications of the ACM*, vol. 21(2), pages 120–126, 1978.

[9] D. Shanks. *Class number, a theory of factorization, and general. Proc. Sympos. Pure Mat*h., vol. 20, pages 415–440, 1971.

[10] D. Stinson. *Cryptography: theory and practice*. CRC Press, 1995.