Analysis of Discrete Logarithm Algorithm using Pre-computation

Hyung Tae Lee, Jung Hee Cheon, and Jin Hong

ISaC-RIM and Department of Mathematical Sciences,
Seoul National University, Seoul, 151-747, Korea
{htsm1138, jhcheon, jinhong}@snu.ac.kr

Abstract. Given an element h in a cyclic group G with a generator g, the Discrete Logarithm (DL) problem is to compute an integer x such that $h = g^x$. The DL computation can be accelerated using some data that was pre-computed for the base group G, before the target element h was given. This pre-computation is useful for breaking a cryptosystem based on the DL when it uses special parameters for efficient computation or standard parameters such as NIST curves. In this paper, we give a precise analysis on the performance of the Pre-comptation DL Algorithm (PDLA) based on the parallelized Pollard rho algorithm [20]. We also show that the PDLA can be parallelized with speedup linear in the resource employed. We discuss about combining practical extensions of the Pollard rho algorithm, the tag tracing technique [1] and restricted random walks [21, **?**,**?**], with the PDLA. Finally, as an application, we give the estimation of the time required for the key extraction process of an IBE [9, **?**] based on a trapdoor DL group.

Keywords: Discrete Logarithm, Pre-computation, Pollard rho, Parallelization, r-adding Walk, Distinguished Point

1 Introduction

Let G be a cyclic group of order q with a generator g. Given the target element $h \in G$, the discrete logarithm problem (DLP) is to find the smallest nonnegative integer x such that $h = g^x$. [1] When the base group G and the generator g are given prior to the target element h, one may generate some data in advance. Thereafter one can use the data to accelerate the discrete logarithm (DL) computation of h. Such a method is called a *pre-computation DL algorithm* (PDLA) and the procedure of the algorithm before (resp. after) the target element h is given is called *the pre-computation (resp. online) phase.* This method is useful when a cryptosystem based on the DL uses standard parameters [10] or special curves [7, **?**] selected for fast computation or when numerous instances of the discrete logarithm are to be solved.

We may consider the baby-step-giant-step algorithm [17] as a PDLA. If one is allowed to compute P operations in the pre-computation phase, then he can obtain the DL of the target element on a group of order q with $O(q/P)$ operations in the online phase. However it has a drawback that it requires $\Omega(P)$ memory for storing pre-computed data.

The parallelized Pollard rho algorithm [20] can be modified to give a PDLA which requires less memory to store pre-computed data. When a group G and a generator g are given, one generates target elements by himself, so called *fake DLPs*, and solves fake DLPs sequentially with the parallelized Pollard rho algorithm on a single processor. In this procedure one generates some chains which end with elements satisfying special properties, so called *distinguished points (DPs)*, and

[1] One may define the discrete logarithm problem is to find an integer x such that $h_2 = h_1^x$ for given the target elements $h_1, h_2 \in G$. To solve this problem using the pre-computation, one has to obtain DLs to the base g of target elements h_1 and h_2 when g is a generator used for pre-computation table generation. The online complexity T is increased by a factor of two.

stores created DPs and their DLs to the generator g in a table. Thereafter, when the target element h is given, one generates chains which start from a point depended on the target element and end with some DP. If the created DP is same with a stored one in the pre-computed table, one can obtain the DL of the target element h. Hitchcock et al. [3] estimated the expected online time of the above described PDLA with respect to the amount of pre-computation. However their analysis is not accurate; in particular, it ignores collisions occurring during the pre-computation phase.

In this paper, we describe the pre-computation algorithm for DL in more detail and give a precise analysis on the PDLA. According to our analysis, the expected pre-computation time P and the expected online time T have the relations

$$P = mt, \quad T \approx \frac{\sqrt{1+2\alpha}}{\sqrt{1+2\alpha}-1}\,t \quad \text{and} \quad PT \approx \frac{\alpha\sqrt{1+2\alpha}}{\sqrt{1+2\alpha}-1}\,q,$$

where m is the number of generated chains in the pre-computation phase, t^{-1} is the proportion of DPs in the group, and $mt^2 = \alpha q$. Some experimental results on small parameters related to our analysis are given.

We also analyze the parallelized version of the PDLA. According to our analysis, while the pre-computation phase of the PDLA can trivially be parallelized with speedup linear in the number of processors, the online phase of the PDLA can be parallelized with linear increments of storage for speedup linear in the number of processors.

We discuss about combining two extensions of the Pollard rho algorithm, the tag tracing technique [1] and restricted random walks [21, ?,?], with the PDLA. Through experiments we confirm that combining with the tag tracing technique yields the expected speedup. We also examine the speedup of combining the restricted random walks with the PDLA from a theoretical aspect.

As an application, we estimate the key extraction time of Maurer-Yacobi identity-based cryptosystem [9, ?]. For 2^{80} security, the parallelized Pollard rho algorithm [20] enhanced by the tag tracing technique requires about one hour with 100 processors and it is not practical for many applications. Our estimation shows, however, that the parallelized PDLA enhanced by the tag tracing technique requires about one minute and 46 seconds with 100 processors when 100 processors are used for pre-computation for about 50 days with 27 GB storage. Since the key extraction process is performed by the key generation center, 100 processors are quite practical.

2 Pre-computation DL Algorithm

In this section, we introduce two essential components of generalized Pollard rho algorithm, iterating function and collision detection method for presenting the PDLA. Thereafter, we describe a PDLA which is a modification of the parallelized Pollard rho algorithm [20].

Throughout this paper, $G = \langle g \rangle$ will be a finite cyclic group of prime order q, generated by the element g. The DLP target to be solved will always be given as $h \in G$, i.e., we are looking for the value $\log_g h$.

2.1 Preliminary

We shall briefly review the basic idea of the Pollard rho algorithm and its variants. Let $F : G \times \mathbf{Z}_q \times \mathbf{Z}_q \to G \times \mathbf{Z}_q \times \mathbf{Z}_q$ be a function defined by

$$F(g_i, a_i, b_i) = (g_{i+1}, a_{i+1}, b_{i+1})$$

where $g_i = g^{a_i}h^{b_i}$ for all $i > 0$.

In the Pollard rho algorithm and its variances, one computes a function F iteratively from one or more random starting points. Since the number of group elements is finite, a point generated after some steps coincides with a certain previous generated element and thereafter the first components of outputs of F follow the track which was already created. If one finds a pair (g_i, a_i, b_i) and (g_j, a_j, b_j) such that $g_i = g_j$, then one can look for the discrete logarithm from the relation $a_i + b_i x \equiv a_j + b_j x \bmod q$ unless $b_i - b_j$ is not relatively prime to q, which occurs infrequently.

Iterating Function The pre-computation phase of the PDLA is started before a target element is given. Hence the iterating function of the original Pollard rho algorithm is not available, but the r-adding walk iterating function, suggested in [16], is suitable for the PDLA.

We briefly introduce the r-adding walk iterating function. Partition G into r roughly same sized subsets $G_1, \cdots, G_r$ so that $G = G_1 \cup \cdots \cup G_r$. The index function $s : G \to \{1, 2, \cdots, r\}$ is defined to be almost pre-image uniform and efficiently computable. Then choose r pairs $(u_i, v_i) \in \mathbf{Z}_q \times \mathbf{Z}_q$ and set r multipliers M_i to $g^{u_i}h^{v_i}$. (In [3], the authors suggested the use of multipliers which are independent of the target element.)

Define r-adding walk iterating function $F_r : G \times \mathbf{Z}_q \times \mathbf{Z}_q \to G \times \mathbf{Z}_q \times \mathbf{Z}_q$ by

$$F_r(y, a, b) = (y \cdot M_{s(y)}, a + u_{s(y)}, b + v_{s(y)})$$

where $y = g^a h^b$. Throughout this paper, F_r will denote an r-adding walk iterating function.

It was shown [15] that the expected number of iterations for finding a collision in a walk generated by r-adding walk for $r \geq 8$ is $O(\sqrt{q})$ and tests [19] over elliptic curve groups show that the expected number of iterations required to find a collision with 20-adding walk is very close to $\sqrt{\frac{\pi}{2}q}$, which is that with a random function.

Collision Detection Let us introduce the distinguished point (DP) technique [14], which was originally used in time memory tradeoff techniques. One sets the distinguishing property that is easy to check and define a DP by a point satisfying the distinguishing property in G. For example, one may define the distinguishing property that a certain number of the most significant are all zeros under a fixed encoding of G. He starts with an empty table, and the walk is computed iteratively until the walk encounters a DP. Then he searches for the same point with the occurred DP in the table. If it is not in the table, he stores it and generates another walk.

The DP collision detected method is required about t additional iterations for noticing a collision after a collision occurs in a walk when t^{-1} is the proportion of DPs in G.

It is straightforward to apply the DP method to multiple walks, so the DP method has an advantage that admits n-times speedup with an n-processor parallelization [20].

There are also other collision detection proposals [6, **?**,**?**,**?**,**?**], and these methods are efficient for finding a collision in a single walk. However the collision checking time of them depend on the iteration number of a walk, not a certain property of the encoding of generated elements, so they are not efficient for finding a collision in multiple walks. Since it requires to find a collision between generated walks in the pre-computation phase and the online phase, the DP method is the most suitable for the PDLA.

2.2 Pre-computation DL Algorithm (PDLA)

We describe the PDLA, which is a modification of the parallelized Pollard rho algorithm [20]. Our description is modified in three points compared with the PDLA described shortly in Section 1 and provided in [3]. First, in the pre-computation phase one generates chains which are started from a random starting point and are ended at a DP, not generating fake DL instances and solving fake DLPs. Second, multipliers of an r-adding walk iterating function are given by a special form. Third, one does not store created DPs in the online phase because its advantage is almost negligible.

Choose positive integers m, t such that $mt^2 = \alpha q$ where m is the number of generated chains in the pre-computation phase, t^{-1} is the proportion of DPs in the group, and α is not a large constant. The parameters are assumed not to be extreme, in the sense that $1 \ll m, t \ll q$ and a typical value is $\alpha \approx 1$. We shall later determine the proper size of parameters m, t, α. Fix an r-adding walk iterating function F_r so that the multipliers have the form $g^u h^0$ for a random $u \in \mathbf{Z}_q$. Since the pre-computation phase starts before a target element is given, the multipliers of r-adding walk iterating function F_r may have the above form. Determine the distinguishing property so that a proportion of distinguished points in G is t^{-1}.

In the pre-computation phase, one chooses m random starting points $\mathbf{g}_{i,0} = g^{a_{i,0}} h^0 \in G$ for $1 \le i \le m$ and iteratively compute $\mathbf{g}_{i,j+1} = F_r(\mathbf{g}_{i,j})$. Each chain is terminated at its first encounter with a DP. We call by a *DP chain* a chain ended at a DP. Then one stores the occurred DPs and the exponents corresponding to each DP in a DP table `DT`.

The average iterations to generate a DP chain is t, but some of chains may fall into a loop that never reaches a DP. In order to detect a chain falling into an infinite loop, we set a chain length bound $\hat{t}$. Any chain longer than this bound is discarded and one can choose to regenerate a chain from a different starting point. Note that the probability for a chain not to reach a DP until its $\hat{t}$-th iteration is $(1 - \frac{1}{t})^{\hat{t}+1} \approx \exp(-\frac{\hat{t}+1}{t})$. This shows that setting $\hat{t}$ to a reasonable multiple of t will suffice in removing the effect of any such discarded chains for any practical purpose.

After the pre-computation phase, one obtains the following matrix.

$$\begin{array}{l}
\mathbf{g}_{1,0} \xrightarrow{F_r} \mathbf{g}_{1,1} \xrightarrow{F_r} \cdots\cdots \xrightarrow{F_r} \mathbf{g}_{1,t_1} \\
\mathbf{g}_{2,0} \xrightarrow{F_r} \mathbf{g}_{2,1} \xrightarrow{F_r} \cdots \xrightarrow{F_r} \mathbf{g}_{2,t_2} \\
\vdots \\
\mathbf{g}_{m,0} \xrightarrow{F_r} \mathbf{g}_{m,1} \xrightarrow{F_r} \cdots\cdots\cdots \xrightarrow{F_r} \mathbf{g}_{m,t_m}
\end{array}$$

Such matrix consisting of m DP chains is called the *DP matrix*.

In the online phase, when a target element h of DLP is given, one starts to generate a DP chain from an element h^r for a random $1 \le r < q$. When a DP occurs in a chain, one compares it with the stored DPs in the table `DT`. If a collision is not found, one generates another DP chain from a distinct starting point $h^{r'}$ for a random $1 \le r' \ne r < q$. Otherwise, he can get the DL of h from the relation $a_i \equiv x b_j + a_j \bmod q$ where x is the DL of h, a pair (g_i, a_i, b_i) and (g_j, a_j, b_j) is the collision, (g_i, a_i, b_i) is the stored point in the table `DT`, and (g_j, a_j, b_j) is the created DP in the online phase.

In the modification of the PDLA described in [3], created DPs in the online phase are also added to the DP table `DT`. However, unless α is not extremely small, the number of newly created DPs in online phase is just one or two. It does not give a big help of accelerating DL computation of a present target element and hence we do not consider saving created DPs in the online phase.

The analysis of the PDLA can be inferred from DL algorithm [8] for multiple instances, which is a modification of the parallel Pollard rho algorithm. According to the analysis, the expected number of group operations to solve k DLPs sequentially with their algorithm is about $\sqrt{2kq}$. From these results, we simply guess that the expected number of group operations to solve the $(k+1)$-th

DLP is about $\sqrt{2q}/(\sqrt{k+1}+\sqrt{k})$ when $\sqrt{2kq}$ group operations was done in the pre-computation phase.

Moreover, Hitchcock et al. [3] provided the expected online time complexity of the PDLA associated with the amount of pre-computation. However, their analysis has two missing points:

1. Although collisions may occur in the pre-computation phase, it does not consider this fact. Hence they regard that the number of distinct elements in a DP matrix is the same with the number of iterations in the pre-computation phase.
2. It does not consider the additional iterations to reach a DP for collision detection after a collision occurs in the online phase.

3 Analysis

3.1 Complexity Analysis of PDLA

Let us fix the expected online phase complexity of the PDLA. We shall exploit the analysis technique of time memory tradeoff. In [4], the authors gave the expected number of distinct entries in a DP matrix approximately for various $\hat{t}$. However, it does not give an error bound of the approximation value. In the following lemma, we give the precise limit value of expected number of distinct entries in a DP matrix when $\hat{t}$ is sufficiently large and provide the difference between the limit value and the expected number when we use a specific $\hat{t}$. This lemma will make up for the first missing point of the analysis provided by Hitchcock et al.

Lemma 1. *Consider a DP matrix created with parameters satisfying $mt^2 = \alpha\, q$. When the iterating function is taken to be the random function and $\hat{t}$ is sufficiently large, we can expect the DP matrix to contain*

$$\frac{\sqrt{1+2\alpha+O(\frac{1}{t})}-1}{\alpha}m(t-1). \tag{1}$$

distinct entries. Moreover, the difference between the expected distinct entries contained in the DP matrix and the above limit value is bounded by $m(\hat{t}+t)\exp(-\hat{t}/t)$.

Proof. We follow the proof of Proposition 18 provided in [4]. Consider a DP matrix generated in the pre-computation phase. Let m_j be the number of new elements added by j-th column of a DP matrix. Now we assume that chains not reaching a DP until $\hat{t}$ steps remain on a DP matrix for an analysis and however we will give the error bound considering that these chains are discarded. Then the recurrence relation

$$\frac{m_j}{q} = \left(1-\exp\left(\frac{-m_{j-1}}{q}\right)\right)\left(1-\frac{1}{t}\right)\left(1-\frac{\sum_{i=0}^{j-1} m_j}{q(1-1/t)}\right) \quad \text{with} \quad m_0 = m$$

is satisfied from Lemma 14 in [4]. Differently from the analysis of [4], the initial value of the sequence $(m_i)_{i=0}^{\infty}$ is m since our DP table are generated from the exact m starting points without making up for the discarded chains to store the exact m DPs in a table.

Let $\mu_i = \dfrac{m_i}{q(1-1/t)}$ and $\sigma_i = \sum_{i=0}^{j-1}\mu_i$. Then the recurrence relation

$$\sigma_{j+1}-\sigma_j = \frac{m_0}{q} - \frac{1}{t}\sigma_j - \frac{1}{2}\sigma_j^2 + O\left(\frac{1}{t^3}\right) \quad \text{with} \quad \sigma_0 = 0$$

is also satisfied from Lemma 15 in [4]. The sequence $(\sigma_j)_{j=0}^{\infty}$ is monotone increasing from the definition of the sequence $(\sigma_j)_{j=0}^{\infty}$ and all σ_i's are bounded since the sum $\sum_{i=0}^{\infty} m_j$ does not exceed the group order q. Hence the sequence $(\sigma_j)_{j=0}^{\infty}$ converges and the limit S of this sequence is

$$-\frac{1}{t} + \sqrt{\left(\frac{1}{t}\right)^2 + 2\left(\frac{m}{q} + O\left(\frac{1}{t^3}\right)\right)}$$

and all σ_i's are less than S. Therefore, the expected number of distinct entries in a DP table is

$$\sum_{i=1}^{\infty} m_i = q\,(1-1/t)\left(-\frac{1}{t} + \sqrt{\left(\frac{1}{t}\right)^2 + 2\left(\frac{m}{q} + O\left(\frac{1}{t^3}\right)\right)}\right) \tag{2}$$

$$= \frac{\sqrt{1+2\alpha+O(\frac{1}{t})}-1}{\alpha}\, m(t-1). \tag{3}$$

Let E be the expected number of distinct elements in a DP table that chains whose length exceeds $\hat{t}$ are discarded. Then,

$$\begin{aligned} E &\le \sum_{i=1}^{\hat{t}} m_j \le q\,(1-1/t) \lim_{j\to\infty} \sigma_j \\ &= q\,(1-1/t)\, S \le \frac{\sqrt{1+2\alpha}-1}{\alpha}\, m(t-1). \end{aligned}$$

Now, we consider the lower bound of E. Note that the probability that a chain reaches a DP at j steps is $\left(1-\frac{1}{t}\right)^{j-1}\frac{1}{t}$. Let E_1 be the expected number of entries belonging to the discarded chains. Then the relation $E \ge q\,(1-1/t)\,S - E_1$ is satisfied.

$$E_1 = m\sum_{i=\hat{t}}^{\infty}(1-\frac{1}{t})^{i-1}\frac{1}{t}i = \frac{m}{t}\sum_{i=\hat{t}}^{\infty}(1-\frac{1}{t})^{i-1}i \tag{4}$$

$$= m\left(\hat{t}\left(1-\frac{1}{t}\right)^{\hat{t}-1} + \left(1-\frac{1}{t}\right)^{\hat{t}} t\right) \tag{5}$$

from $\dfrac{x^{\hat{t}}}{1-x} = \sum_{i=\hat{t}}^{\infty} x^n$ and its derivation and

$$\begin{aligned} (5) &\le \left(1-\frac{1}{t}\right)^{\hat{t}}\left(m\hat{t} - m\frac{\hat{t}}{t} + mt\right) \\ &\le m(\hat{t}+t)\exp(-\hat{t}/t) \end{aligned}$$

since $f(x) = (1-\frac{1}{x})^x$ is an increasing function and $\lim_{x\to\infty} f(x) = \exp(-1)$. Hence

$$E_1 \ge \frac{\sqrt{1+2\alpha}-1}{\alpha}\, m(t-1) - m(\hat{t}+t)\exp(-\hat{t}/t).$$

Therefore, the difference between the limit value and the number of distinct entries on practical parameters is bounded by $m(\hat{t}+t)\exp(-\hat{t}/t)$.

Considering the fact that the authors ignored $(1-\frac{1}{t})$ in the proof of Proposition 18 in [4], Lemma 1 shows that the exact limit value of the expected number of distinct entries in a DP matrix is same with the approximation value presented in [4]. Also it shows that the difference between the limit value and the expected number on practical parameters is negligible since $m(\hat{t}+t)\exp(-\hat{t}/t)$ is sufficiently smaller than $\frac{\sqrt{1+2\alpha}-1}{\alpha}m(t-1)$ when $\hat{t}$ is sufficiently larger than t.

We are now ready to discuss about the probability for solving a DLP with generating a single DP chain in the online phase.

Theorem 1. *Fix parameters satisfying $mt^2 = \alpha\, q$ and generate a DP matrix. Then generate another chain from a random starting point, terminating at its first DP occurrence. When the iterating function F is taken to be the random function, we can expect the ending DP of the new chain to be equal to one of the ending DPs for the pre-generated chains with probability*

$$1-\frac{1}{\sqrt{1+2\alpha}},$$

and the error term is bounded by $\frac{7\alpha}{t}+\frac{\alpha}{\sqrt{1+2\alpha}}(c+1)\exp(-c)$ when $\hat{t}=ct$ for some constant c.

Proof. Let us write DP for the set of all DPs in G, DM (DP matrix) for the set of all elements belonging to the pre-generated DP chains, and DT (DP table) for the set of ending points of the pre-generated DP chains.

Once the DP matrix DM is ready, one is told to create a new chain

$$\mathbf{h}_0 \xrightarrow{F} \mathbf{h}_1 \xrightarrow{F} \cdots \xrightarrow{F} \mathbf{h}_j \xrightarrow{F} \cdots$$

from a random starting point $\mathbf{h}_0$. At each iteration, as a new $\mathbf{h}_j$ is created, one of the following three events may occur.

E1. $\mathbf{h}_j \in \mathtt{DM}$: The new chain has merged with a pre-generated chain. The rest of the chain will automatically follow the pre-generated chain and terminate with a DP belonging to DT.

E2. $\mathbf{h}_j \in \mathtt{DP}\setminus\mathtt{DM} = \mathtt{DP}\setminus\mathtt{DT}$: The new chain has terminated with a DP without merging with a pre-generated chain. The chain cannot reach a point belonging to DT.

E3. $\mathbf{h}_j \notin \mathtt{DM}\cup\mathtt{DP}$: The new chain has neither merged with one of the pre-generated chains nor reached a DP. One needs to continue onto the next iteration.

Hence the new chain terminates with an element from DT if and only if each iteration of the chain results in event E3, before finally sinking into event E1.

Let $\delta = \dfrac{\sqrt{1+2\alpha}-1}{\alpha}m + m(\hat{t}+t)\exp(-\hat{t}/t) + \frac{mO(1)}{\alpha(\sqrt{1+2\alpha}+\sqrt{1+2\alpha+O(\frac{1}{t})})}$. According to Lemma 1, for a random function, we can expect event E1 to happen, at each iteration, with probability

$$\Pr[\text{E1}] = \frac{\sqrt{1+2\alpha}-1}{\alpha}\frac{mt}{q}+\frac{\delta}{q} = \frac{\sqrt{1+2\alpha}-1}{t}+\frac{\delta}{q},$$

where the equality follows from a use of $mt^2 = \alpha\, q$. We can also state

$$\begin{aligned}
\Pr[\mathrm{E3}] &= 1 - \frac{\#(\mathtt{DM} \cup \mathtt{DP})}{q} = 1 - \frac{\#\mathtt{DM} + \#\mathtt{DP} - \#(\mathtt{DM} \cap \mathtt{DP})}{q} \\
&= 1 - \frac{\frac{q}{t} + \frac{\sqrt{1+2\alpha}-1}{\alpha} mt}{q} + \epsilon \\
&= 1 - \frac{\sqrt{1+2\alpha}}{t} + \epsilon
\end{aligned}$$

as the probability for event E3's occurrence when ϵ is $\frac{\#(\mathtt{DM}\cap\mathtt{DP})}{q} - \frac{\delta}{q^2} \le \frac{m}{q}$.

Finally, gathering the above information and argument, we can compute the probability for the new chain to terminate at an element of $\mathtt{DT}$ to be

$$\begin{aligned}
&\sum_{k=0}^{\infty} \Big(1 - \frac{\sqrt{1+2\alpha}}{t} + \epsilon\Big)^k \Big(\frac{\sqrt{1+2\alpha}-1}{t} + \frac{\delta}{q}\Big) \\
&= \ 1 - \frac{1 - t\epsilon}{\sqrt{1+2\alpha} + t\epsilon} + \frac{\alpha\sqrt{1+2\alpha} + \alpha\delta}{mt(\sqrt{1+2\alpha} + \epsilon)} \\
&\approx \ 1 - \frac{1}{\sqrt{1+2\alpha}}.
\end{aligned}$$

When $\hat{t} = ct$ for some constant c, the error term of the last approximation is bounded by $2t\epsilon + \frac{5\alpha}{t} + \frac{\alpha}{\sqrt{1+2\alpha}}(c+1)\exp(-c) < \frac{7\alpha}{t} + \frac{\alpha}{\sqrt{1+2\alpha}}(c+1)\exp(-c)$.

Those familiar with time memory tradeoff techniques can interpret this theorem as giving the probability of false alarms occurrence[2] during the processing of a single non-perfect DP table. This high probability is an annoyance in the time memory tradeoff. While every collision of the online DP chain with the pre-computed table will always bring about a solution to the DLP since the r-adding walk does not modify the exponent of h. Hence this high probability is a good thing for DLP solving.

Execution complexity As given by Theorem 1, the PDLA succeeds using a single online DP chain with probability $1 - \frac{1}{\sqrt{1+2\alpha}}$.[3] Taking the inverse of this value, we can state the expected number of online DP chain creations until successful discrete logarithm retrieval to be $\frac{\sqrt{1+2\alpha}}{\sqrt{1+2\alpha}-1}$. Since the average length of a DP chain is t, we can state the expected online time complexity T, storage complexity[4] M, and the pre-computation time complexity P as

$$T \approx \frac{\sqrt{1+2\alpha}}{\sqrt{1+2\alpha}-1}\, t, \quad M \approx m, \quad \text{and} \quad P = mt. \tag{6}$$

[2] One should consider whether allowing the starting point to be the ending point will produce any difference in the final result.

[3] Although we consider the online time complexity with the error term of probability in Theorem 1, that increases less than $\frac{7(\sqrt{1+2\alpha}+1)^3}{4\alpha} + \frac{(\sqrt{1+2\alpha}+1)^2}{2}(c+1)\exp(-c)t$, i.e., $T < \frac{\sqrt{1+2\alpha}}{\sqrt{1+2\alpha}-1}\, t + \frac{7(\sqrt{1+2\alpha}+1)^3}{4\alpha} + \frac{(\sqrt{1+2\alpha}+1)^2}{2}(c+1)\exp(-c)t$. This error comes from setting the chain length bound $\hat{t}$. However, since t is sufficiently large and $(c+1)\exp(-c)$ is sufficiently smaller than t, we ignore this error on our analysis.

[4] We are ignoring the fact that duplicate entries may be removed to reduce storage.

Tradeoff curve Using the equation $mt^2 = \alpha\, q$, we obtain

$$PT \approx \frac{\alpha\sqrt{1+2\alpha}}{\sqrt{1+2\alpha}-1}\, q. \tag{7}$$

This shows that any increase in pre-computation is awarded by a corresponding decrease in online time. Interpretation of the following equations give us more detail.

$$\sqrt{M}\, T \approx \frac{\sqrt{\alpha}\sqrt{1+2\alpha}}{\sqrt{1+2\alpha}-1}\sqrt{q} \quad \text{and} \quad P \approx \sqrt{\alpha}\,\sqrt{M}\sqrt{q}. \tag{8}$$

With minimal storage, the expected pre-computation and online time are both $O(\sqrt{q})$. By utilizing a storage of size M, we can reduce online time by a factor of $\sqrt{M}$ at the price of a $\sqrt{M}$ factor increase in pre-computation time.

Note that for any fixed q, the right-side of tradeoff curve between storage and online time is optimal when the constant $\frac{\sqrt{\alpha}\sqrt{1+2\alpha}}{\sqrt{1+2\alpha}-1}$ is at its minimum. Explicitly, one can show that the minimum [5] value of 2.35480 is attained at $\alpha = \frac{1+\sqrt{5}}{4} \approx 0.809017$.

3.2 Experiments

We have tested the analysis of the PDLA by running it with small parameters. This was implemented on a dual-core AMD Opteron 2.6GHz system and the NTL library [18] was used to provide the required finite field arithmetics. Throughout the test, the cyclic group $G = \langle g \rangle$ was taken to be a subgroup of $\mathbf{F}_p^*$, where p was taken to be a random 1024-bit prime. The 20-adding walk served as the iterating function. As is customary in time memory tradeoff tables, sequential points $g^1, g^2, \ldots, g^m$ were used in place of the m random points to simplify implementation. Likewise, the first online chain started from DLP target h and sequential powers of h were used thereafter. The chain length bound was set to $10t$ to detect random walks that fell into loops without reaching a DP. Chains were not regenerated to replace the discarded chain.

Table 1. Online phase success probability and complexity for various α (q: 42-bit prime, $t = 2^{14}$)

α		0.001	0.01	0.5	0.81	1	1.5	2	10
success probability at 1st chain (%)	experiment	0.10	1.30	28.10	37.50	38.10	48.10	54.10	76.90
	theory	0.10	0.99	29.29	38.22	42.26	50.00	55.28	78.18
success probability at 2nd chain (%)	experiment	0.00	0.70	21.50	24.10	25.80	24.80	23.80	17.40
	theory	0.10	0.98	20.71	23.61	24.40	25.00	24.72	17.06
average iterations until solution (unit:t)	experiment	1085.7	99.37	3.46	2.62	2.51	2.00	1.87	1.33
	theory	1001.5	101.50	3.41	2.62	2.37	2.00	1.81	1.28

In Table 1, we compare our theory against experiment results under variations of α. The success probability entries test the validity of Theorem 1 and the last two rows of the table test the T value as given by (6).

[5] An in-depth analysis of the storage reduction obtained through duplicate entry removal, which remains to be done, will result in a different optimal position, but the approach of minimizing this coefficient should remain valid.

Table 2. Online phase success probability and complexity for various m and t (q : 42 bits prime, $\alpha = 0.81$)

$\log t$		8	11	14	17	20	theory
m		46121039	586463	12564	207	2	
storage size (MB)		11806.986	150.135	3.216	0.053	0.001	
success	1st chain	36.90	33.80	37.50	37.60	24.50	38.22
probability (%)	2nd chain	23.00	25.50	24.10	22.30	16.40	23.61
ave. iterations until solution (unit: t)		2.70	2.65	2.62	2.78	15.30	2.62

Table 2 shows a similar comparison under a fixed α and varying m and t parameters. This table gives an indication of how well the relation (8) is observed, i.e., whether the increase in storage results in the predicted reduction of online time complexity.

Test results given above are average values done over multiple table creations and multiple DLP target solving per created table. Each table was created with a different random multiplier set for the 20-adding walk. All tests used 10 tables and 100 DLPs per table.

The implementation results support our theoretic analysis very well for a wide range of parameters. The only visible exception corresponds to when m is extremely small. In this case the expected DP chain length approaches the expected rho length of a random walk and thus too many chains are being discarded. Unlike the small m case, the results for large m follows the theory closely. This implies that the PDLA will work well even when extremely large storage is employed.

3.3 Parallelization

The pre-computation phase of the PDLA can trivially be parallelized. While pre-computation is certainly the computationally intensive part of the PDLA, there may be situations where the parallelization of the online phase is desired, possibly to reduce the wall-clock running time of the online algorithm further. Let us investigate this matter in this subsection.

Readers familiar with time memory tradeoff techniques will have read the previous material with $\alpha \approx 1$ in their minds, but one can confirm through a careful re-reading of all proofs that everything we wrote is true for even extremely small α.

With Theorem 1 confirmed for small α, let us consider the parallel use of n processors at the online phase. We take parameters m and t such that $mt^2 = \frac{\alpha}{n}\, q$, where $\alpha \approx 1$. Then, based on Theorem 1, one can state that the probability of success at the first run of the online phase, i.e., after the n DP chains have been produced, is $1 - \left(\frac{1}{1+2\alpha/n}\right)^{n/2}$. Inverting this, we can state

$$T_{1/n} \approx \left\{1 + \frac{1}{(1+\frac{2\alpha}{n})^{\frac{n}{2}} - 1}\right\} t$$

as the average time spent by each processor, until the DLP is solved. As before, we require $M = m$ storage and $P = mt$ pre-computation time.

Applying the equation $mt^2 = \frac{\alpha}{n}\, q$, we can write the analogue to equation (7) as follows.

$$P\,T_{1/n} \approx \frac{\alpha}{n}\left\{1 + \frac{1}{(1+\frac{2\alpha}{n})^{\frac{n}{2}} - 1}\right\} q. \qquad (9)$$

It is easy to check that $1 + \frac{1}{e^{\alpha}-1} \le 1 + \frac{1}{(1+2\alpha/n)^{n/2}-1} \le 1 + \frac{1}{\sqrt{1+2\alpha}-1}$, so that we can treat the term inclosed between the braces as an insignificant constant. Noting that $n\,T_{1/n}$ is the total online processing time, one can see that (9) is almost identical to (7). In other words, regardless of parallelization, with the same pre-computation effort, one needs the same total online processor time to solve a given DLP.

Analogue of (8) is as follows.

$$\sqrt{M}\,T_{1/n} \approx \sqrt{\frac{\alpha}{n}}\left\{1+\frac{1}{(1+\frac{2\alpha}{n})^{\frac{n}{2}}-1}\right\}\sqrt{q} \quad \text{and} \quad P \approx \sqrt{\frac{\alpha}{n}}\,\sqrt{M}\sqrt{q}. \tag{10}$$

This says that with n-processor parallelization, one can achieve $\sqrt{n}$ times online (wall-clock) time reduction and also an $\sqrt{n}$ factor reduction in pre-computation. A more practical view is to say that, by deciding to invest n-times more on the online processing power, one reduces the required cost of the pre-computation phase. Since the pre-computation is much more resource consuming than the online phase, in most situations, the less-than-linear speedup of the online phase will be justified by the reduction of pre-computation cost. In addition, if the parameters were such that the storage cost outweighed the processor cost, then one can surely justify the cost of additional processors.

This brings us to the subject of what can be done while maintaining the pre-computation cost. In that case, one can read from (9) that we achieve linear speedup of the online phase, but one must be slightly more careful before jumping to this conclusion. The P part of (10) shows that the storage must also be increased by a factor of n in addition to the number of processors. In other words, by increasing the storage and processor used during the online phase by a factor of n, one achieves n-times speedup of the online phase. If the parameters were such that the storage cost is much smaller than the processor cost, we may claim linear speedup through parallelization. If otherwise, this may not be what most would call linear speedup, but n-times investment on what is used during the online phase results in an n-times speedup of online phase.

4 Discussions

4.1 Practical Consideration

Tag Tracing Algorithm The PDLA consists of r-adding walk iterating function and DP collision detection method. Hence the tag tracing algorithm [1] can be applied to the PDLA for solving DLPs on a subgroup of a finite field. In this subsection, we certify speeding up from applying tag tracing technique to the PDLA with experiments.

Test environment and implementation method of the PDLA were same with Section 3.2. In the tag tracing algorithm, we stored all possible products of multipliers of 4-adding walk up to 40 steps using pre-computation with 54.9 MB and 36.951 seconds on average of all tests. Other parameters were set equal to experiments of Section 4.5 in [1] for fair comparison.

In Table 3, we give experimental results of applying 4-adding walk with tag tracing technique and 20 adding walk to the PDLA. All results are average values done over 10 table creations and 100 DLP targets solving per created table. Each table was produced with a different random multipliers for use of different r-adding walks.

We know that the use of 4-adding walk with the tag tracing technique is about 12 times faster than the use of 20-adding walk without the tag tracing technique from the result of [1] in our implementation environment. The last row in Table 3 shows that the tag tracing algorithm makes the PDLA 9.487 – 13.622 times faster. This shows that applying tag tracing technique to the PDLA works well.

Table 3. Comparison of 20-adding walk and tag tracing ($\alpha = 0.81$)

q	phase	20-adding walk	tag tracing	20-add./tag.	storage
42bit ($t=2^{14}$)	pre-computation	1032.221 s	77.995 s	13.234	5 MB
	online	0.256 s	0.027 s	9.487	
48bit ($t=2^{16}$)	pre-computation	17005.030 s	1274.148 s	13.346	17 MB
	online	1.038 s	0.105 s	9.886	
54bit ($t=2^{18}$)	pre-computation	210312.400 s	15439.350 s	13.622	68 MB
	online	4.115 s	0.419 s	9.821	
80bit* ($t=2^{30}$)	pre-computation	137 y 104.360 d	10 y 136.282 d		268 MB
	online	4 h 39.620 m	29 m 29.472 s		
80bit† ($t=2^{30}$)	pre-computation	1 y 136.097 d	37 d 20.707 h		27 GB
	online	2 m 47.772 s	17.695 s		

estimated value (*with 1 processor, †parallel processing with 100 processors)

Moreover, we also estimate the time to solve a DLP on a group of 80-bit order from experimental results. In Table 3, we confirm that the pre-computation time and the online time are about $2^{2d/3}$ and $2^{d/3}$ times increased respectively, when d is the difference between group orders. From this fact, we give the estimation of solving a DLP on a 80-bit order group using the PAPM with 20-adding walk iterating function and 4-adding walk with the tag tracing algorithm in the eighth and ninth rows in Table 3. The last two rows in Table 3 also provide the estimation time to solve a DLP using the parallelized PAPM with 100 processors.

Restricted Random Walks There are some extensions [21, 2, ?] of the Pollard rho algorithm, which use iteration functions whose image sizes are smaller than the size of the group G. In [5], the authors proposed the iteration function whose image size reduces using the normal basis representation for solving DLP defined over a subgroup of $F^*_{p^\ell}$ where p is a prime and ℓ is an extension degree. According to their analysis, combining proposed iteration function with the Pollard rho algorithm is about $\frac{3p-3}{4p-3}\sqrt{\ell}$ times faster than the Pollard rho algorithm combined with the original Pollard iteration function.

In case of DLP defined over a subgroup of points on elliptic curves, there are some iteration functions which improve the original or parallelized Pollard rho algorithm. Wiener and Zuccherato [21] proposed the iteration function whose image restricts x-coordinates of points using the fact that x-coordinates of point and its negative over elliptic curves are same and speed up the original algorithm by a factor of $\sqrt{2}$. Moreover, they presented other iteration functions for solving DLPs defined over subfield curves, binary anomalous curves, or Koblitz curves. Gallant et al. [2] also proposed the iteration function using the Frobenius map, which the algorithm using this function speeds up the original algorithm by a factor of $\sqrt{2\ell}$ for solving a DLP defined over binary anomalous curves over F_{2^ℓ}.

As mentioned in Section 2, since the PDLA is started before the target element is given, the iteration function used in the PDLA has to be defined so that it is independent on a target element. Since proposed restricted random walks can satisfy this property and DP method can be well-combined with them, we expect that applying these iteration functions to the PDLA also works well.

4.2 Estimation of the Key Extraction Time of Maurer-Yacobi IBE

A cyclic group is called a tradpoor DL group if there is a trapdoor information such that the DLP on the group is hard without the trapdoor information, but becomes easy with the information. Maurer and Yacobi [9, 11] proposed an identity-based encryption (IBE) based on a trapdoor DL group. We estimate the key extraction time of Maurer and Yacobi's IBE, whose key extraction process is to solve a DLP.

Let us introduce on the above trapdoor DL groups [9, **?**], a maximal cyclic subgroup G of $\mathbf{Z}_n^*$ where $n = pq$ is a product of two roughly same sized primes and both $p-1$ and $q-1$ are B-smooth. The best known algorithm for the DLP on G is to use a factorization of n. If a factorization of n is known, one can solve the DLP on G in $O(\sqrt{B} \log n / \log B)$ time by solving it on each subgroup of order at most B using the Pohlig-Hellman method [12] and then applying the Chinese remainder theorem. Otherwise, one may try to factorize n. The best known factoring algorithm when n is large enough (say, 1024 bits for 2^{80} security) is the Pollard $p-1$ method [13] of complexity $O(B \log n / \log B)$.

For achieving 2^{80} security, B is taken to be 2^{80} and the DLP with trapdoor requires about 2^{42} modular multiplications in $\mathbf{F}_p$ or $\mathbf{F}_q$[6]. Under the assumption that one modular multiplication takes 1 μsec[7], this takes $2^{22.06}$ seconds or about 50 days. Even under a 1,000-processor parallelization [20] with the tag tracing technique [1], this would amount to about one hour. Hence the DLP solving computation is possible, but very uncomfortable.

However the use of the PDLA can reduce the key extraction time through pre-computation. We shall give the estimated key extraction time with experimental results applying the PDLA. Test environment was same with Section 3.2. Throughout the test, G was taken to be a maximal cyclic subgroup of $\mathbf{Z}_n^*$ where n was an 1024-bit integer which was a product of 512-bit primes p, q and two prime factors of $p-1$ and $q-1$ were $(\log B)$-bit and other prime factors were less than $(\log B)/2$-bit each. We used the PDLA enhanced by the tag tracing technique for solving DLPs on groups of four large prime factors and applied Pollard rho algorithm with 20-adding walk and DP collision detection method for solving DLPs on the other groups. Parameters for tag tracing technique were set equal to Section 4.1 and those for the PDLA were set to $t = 2^{(\log B)/3}$, $\alpha = 0.81$.

Table 4. Key extraction time for various B

security level	pre-comp. size	pre-comp. time	online time			storage
			small factors	large factors	total	
42	$2^{29.9}$	3 m 25 s	0.266 s	0.079 s	0.345 s	17 MB
48	$2^{33.9}$	54 m 45 s	0.495 s	0.310 s	0.805 s	68 MB
54	$2^{37.6}$	7 h 14 m	1.692 s	1.678 s	3.370 s	270 MB
80*	2^{51}	13 y 231 d	3 m 12.605 s	2 h 52.578 m	2 h 55.788 m	270 MB
80†	2^{51}	49 d 19 h	1.926 s	1 m 43.547 s	1 m 45.473 s	27 GB

estimated value (*with 1 processor, †parallel processing with 100 processors)

6 In order to keep the complexity of Pollard $p-1$ method on linear in B, each $p-1$ and $q-1$ has at least two prime factors of roughly same size with B. However large prime factors cause more DLP solving time. We set each $p-1$ and $q-1$ has exactly two prime factors of roughly same size with B and then the complexity of Pohlig-Hellman can be reduced to $O(\sqrt{B})$. However one who wants to factorize n can also reduce the complexity of Pollard $p-1$ method to $O(B)$.

7 Our un-optimized test implementation described in the previous section showed 5.24 μsec performance.

In Table 4, the rows from third to fifth provide our experimental results of the key extraction time corresponding to B, which are average values done over 10 table generations and 100 DLP targets solving per a created table. From this, when the difference between $\log B$'s is d, we are sure that the pre-computation time and the online-time for large prime factors are about $2^{2d/3}$, $2^{d/3}$ times increased respectively and the online time for small prime factors is about $2^{d/4}$ times increased.

We estimate the pre-computation time and the online time for 2^{80} security. Assume that we perform $P = 2^{51}$ pre-computation for four large prime factors and store 2^{18} elements in each table. Then it takes 13 years and 231 days for pre-computation with 270 MB storage. Since the pre-computation can be parallelized, one can generate the table in 49 days and 19 hours using 100 processors and 270 MB storage[8], in addition to the factors of n, as the trapdoor information. When an instance of DLP is given, it can be solved in 2 hours and 56 minutes on one processor.

If, as described in Section 3.3, we decide to parallelize the online phase across 100 processors while keeping the pre-computation at 2^{51}, the online phase reduces to one minute and 46 seconds with the use of 27 GB storage. Since this process is to be done by the key generation center to handle just key extractions, the presented pre-computation and online time may both be quite practical.

5 Conclusion

In this paper, we provided an accurate analysis of the discrete logarithm algorithm with the aid of pre-computation, which is a modification of the parallelized Pollard rho algorithm [20]. We also analyzed the complexity of the parallelized version of the algorithm and considered combining with the extension of the Pollard rho algorithm, the tag tracing technique and restricted random walks. Finally, we gave the estimation of key extraction time of Maurer and Yacobi identity-based encryption, whose key extraction process is to solve a DLP. We expect that our analysis would enable ones who try to solve a DLP on large parameters to forecast the precise budget with respect to desired time.

Although the PDLA described in this paper requires less memory than the baby-step-giant-step algorithm, we do not confirm that this method requires the optimal memory. Constructing a PDLA using less memory than the described one or proving the optimality of this algorithm is one of research topics related to this work.

References

1. J. H. Cheon, J. Hong, and M. Kim. Speeding up the Pollard rho method on prime fields. In J. Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, LNCS 5350, pages 471–488. Springer, 2008.
2. R. Gallant, R. Lambert, and S. Vanstone. Improving the parallelized Pollard lambda search on anomalous binary curves. *Mathematics of Computation*, 69(232):1699–1705, 2000.
3. Y. Hitchcock, P. Montague, G. Carter, and E. Dawson. The efficiency of solving multiple discrete logarithm problems and the implications for the security of fixed elliptic curves. *International Journal of Information Security*, 3:86–98, 2004.
4. J. Hong and S. Moon. A comparison of cryptanalytic tradeoff algorithms. Cryptology ePrint Archive, Report 2010/176, 2010.
5. M. Kim, J. H. Cheon, and J. Hong. Subset-restricted random walks for Pollard rho method on F_{p^m}. In *Public Key Cryptography 2009*, volume 5443 of *LNCS*, pages 54–67. Springer, 2009.

[8] One can also consider keeping just the iterating function (multipliers) secret and leaving the table unprotected.

6. D. E. Knuth. *The art of computer programming - Seminumerical algorithms*, volume 2. Addison-wesley publishing company, second edition, 1981.
7. N. Koblitz. CM-curves with good cryptographic properties. In J. Feigenbaum, editor, *Advances in Cryptology—CRYPTO '91*, volume 576 of *LNCS*, pages 279–287. Springer, 1992.
8. F. Kuhn and R. Struik. Random walks revisited: extensions of Pollard's rho algorithm for computing multiple discrete logarithms. In S. Vaudenay and A. M. Youssef, editors, *Selected Areas in Cryptography 2001*, LNCS 2259, pages 212–229. Springer, 2001.
9. U. M. Maurer and Y. Yacobi. Non-interactive public-key cryptography. In D. W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91*, LNCS 547, pages 498–507. Springer, 1991.
10. NIST. Digital signature standard (DSS). FIPS PUB 186-3, 2009.
11. K. G. Paterson and S. Srinivasan. On the relations between non-interactive key distribution, identity-based encryption and trapdoor discrete log groups. *Designs, Codes and Cryptography*, 52(2):219–241, 2009.
12. S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 24:106–110, 1978.
13. J. M. Pollard. Theorems of factorization and primality testing. *Cambridge Philosophical Society*, 76:521–528, 1974.
14. J.-J. Quisquater and J.-P. Delescaille. How easy is collision search. New results and applications to DES. In G. Brassard, editor, *Advances in Cryptology - CRYPTO*, LNCS 435, pages 408–413. Springer, 1989.
15. J. Sattler and C.-P. Schnorr. Generating random walks in groups. *Ann. Univ. Sci. Budapest. Sect. Comput.*, 6:65–79, 1985.
16. C. P. Schnorr and J. H. W. Lenstra. A Monte Carlo factoring algorithm with linear storage. *Mathematics of Computation*, 43:289–311, 1984.
17. D. Shanks. Class number, a theory of factorization, and genera. In *Symposia in Pure Mathematics*, 20, pages 415–440, 1969.
18. V. Shoup. NTL: A library for doing number theory ver 5.5. http://www.shoup.net/ntl/, 2009.
19. E. Teske. On random walks for Pollard's rho method. *Mathematics of Computation*, 70(234):809–825, 2001.
20. P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12:1–28, 1999.
21. M. J. Wiener and R. J. Zuccherato. Faster attacks on elliptic curve cryptosystems. In *Selected Areas in Cryptography '98*, volume 1556 of *LNCS*, pages 190–200. Springer, 1999.