J. Korean Math. Soc. **52** (2015), No. 4, pp. 797–819
http://dx.doi.org/10.4134/JKMS.2015.52.4.797

ANALYSIS OF POSSIBLE PRE-COMPUTATION AIDED DLP SOLVING ALGORITHMS

Jin Hong and Hyeonmi Lee

Abstract. A trapdoor discrete logarithm group is a cryptographic primitive with many applications, and an algorithm that allows discrete logarithm problems to be solved faster using a pre-computed table increases the practicality of using this primitive. Currently, the distinguished point method and one extension to this algorithm are the only pre-computation aided discrete logarithm problem solving algorithms appearing in the related literature. This work investigates the possibility of adopting other pre-computation matrix structures that were originally designed for used with cryptanalytic time memory tradeoff algorithms to work as pre-computation aided discrete logarithm problem solving algorithms. We find that the classical Hellman matrix structure leads to an algorithm that has performance advantages over the two existing algorithms.

1. Introduction

Let $G = \langle g \rangle$ be a cyclic group of (prime) order p with a fixed encoding scheme for expressing its elements. Choose an $x \in \mathbb{Z}_p$ at random and let $h = g^x$. The discrete logarithm problem (DLP) asks one to find the original $x = \log_g h$, given g and h that are expressed in the encoding scheme for G. The security of numerous cryptographic schemes relies on the believed or assumed difficulty of solving DLPs on certain groups.

There are a small number of algorithms for solving DLPs that are generic in the sense that they can be applied to any cyclic group G. The Pohlig-Hellman algorithm [33] allows a DLP defined on a composite order group to be reduced to multiple DLPs on smaller prime order groups, so that only prime order groups are usually considered in dealing with DLPs. The baby-step giant-step (BSGS)

Received September 18, 2014.
2010 *Mathematics Subject Classification.* Primary 11T71, 94A60; Secondary 68R99.
Key words and phrases. discrete logarithm problem, pre-computation, distinguished point, time memory tradeoff.
The first author was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2012R1A1B4003379). The second author was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (NRF-2012R1A1A2008392).

algorithm [36], the Pollard's rho algorithm [34], and their many variants are based on a common idea. One generates many elements $g_i \in G$ that can be expressed in the form $g_i = g^{x_i} h^{y_i}$, with known exponents x_i and y_i. Then, any collision $g_i = g_j$ among these elements, where the equality is checked under the encoding scheme for G, leads to the DLP solution $\log_g h = (x_j - x_i)(y_i - y_j)^{-1} \pmod{p}$.

The BSGS method generates the elements g_i in a deterministic manner that ensures the existence of a collision, whereas the rho method and its variants take a probabilistic approach. In the rho method and its variants, a semi-algebraic iteration function $f : G \to G$ is fixed and elements are generated in a mostly sequential manner by setting $g_{i+1} = f(g_i)$, starting from a $g_0 = g^{x_0} h^{y_0}$, chosen by the algorithm operator. A separate mechanism is set up for collision detection. The many rho variants differ in their choices of the iteration function f and the collision detection mechanism, but each discovers a collision in $O(\sqrt{p})$ iterations. In fact, $\Theta(\sqrt{p})$ group operations is known [37] to be a lower bound for the complexity of generic DLP solving algorithms.

This work deals with the DLP solving algorithms that attain *online* complexities strictly lower than $\Theta(\sqrt{p})$ per DLP instance, after a one-time *pre-computation* effort of complexity higher than $\Theta(\sqrt{p})$. We clarify that, even though the cost of pre-computation can be amortized over multiple DLPs, the notion of cost per DLP, even if it includes the shared cost of pre-computation, is not the usual cryptanalytic complexity associated with the solving of a DLP. In the remainder of this paper, our use of the phrase *DLP algorithm* will always refer to a *generic pre-computation aided DLP solving algorithm.*

The most direct application of a DLP algorithm would be as a cryptanalytic tool, where the high cost of pre-computation can be amortized through multiple online DLP solving instances, but such situations are hard to find in practice. A more interesting application is a constructive use of these algorithms that concerns a cryptographic primitive known as the trapdoor discrete logarithm groups. The trapdoor discrete logarithm groups are algebraic structures on which the feasibility of solving DLPs depends on the possession of some secret (trapdoor) information, and there are many cryptographic schemes [14, 18, 27, 28, 32, 39] that rely on this primitive.

One approach to constructing trapdoor discrete logarithm groups is the RSA setting [26, 27, 32], where DLPs on the group $\langle \mathbb{Z}_n^*, \cdot \rangle$, for a composite integer n, are considered. Someone that knows the factorization of n can reduce a DLP instance on $\mathbb{Z}_n^*$ to multiple DLPs on certain smaller groups, but there are security issues related to Pollard's $p-1$ factoring algorithm that make it advisable to choose n in such a way that these smaller groups are still as large as possible, subject to the condition that solving DLPs on them be practical for the owner of the trapdoor information. A (pre-computation aided) DLP algorithm will transform DLPs of sizes that can barely be solved with large resources into ones that can comfortably be solved in real time with small resources, after a large one-time pre-computation investment, and hence make these trapdoor discrete logarithm groups much more practical to use. Note that the pre-computation cannot be carried out by someone that does not know the factorization of n, so that the same tool is not available to an attacker.

Currently, there are only two DLP algorithms that can be found in the related literature. The first of these has been studied and discussed [8, 12, 15, 22, 23] from a variety of different viewpoints and with slight variations, but they are essentially the same algorithm, and we will refer to this as the distinguished point (DP) method. The more recent second method [7], which we will refer to as the BL method, is an extension to the DP method. Its main idea is to be selective in recording the results of the pre-computation. Since each pre-computation table entry is more useful on average, the online efficiency is increased, but this also has the negative effect of requiring more pre-computation. We chose to treat the BL method and the original DP method separately in this work, since their performance characteristics can be very different, depending on what fraction of the pre-computed results are retained by the BL method.

Many cryptographers will be familiar with the cryptanalytic time memory tradeoff (TMTO) algorithm often referred to as the DP tradeoff. In fact, the structure of the pre-computation matrices used by the DP DLP algorithm and the DP TMTO algorithm are identical. Noting that there are many more TMTO algorithms, each with a different pre-computation matrix structure, this work investigates the possibility of utilizing these structures to create a DLP algorithm that outperforms the existing two DLP algorithms.

We consider the most natural adaptation of each TMTO algorithm into a DLP algorithm and analyzed its execution complexities. The performance of each such candidate algorithm can be summarized as two equations expressing the balance between pre-computation cost, storage size, and online time complexity, and the two equations can be used to compare the performances of different algorithms. After many failed attempts, we find that the adaptation of the classical Hellman TMTO method into a DLP algorithm results in an algorithm that is better than the two existing algorithms. We clarify that our comparisons are based on the average case complexities rather than the worst case complexities.

The rest of this paper is organized as follows. In Section 2, we recall the DP method and summarize its performance characteristics by combining a few existing results. Our attempts at creating a DLP algorithm based on the rainbow matrix structure and its variants are briefly reported in Section 3. None of these attempts resulted in a meaningful algorithm, and most of the details are given in the appendices. In Section 4, we provide some theoretical evidence that the classical Hellman matrix structure could be a reasonable alternative to the DP matrix. In Section 5, the performance of the Hellman method is compared with those of the existing DP and BL algorithms. Some further discussions are given in Section 6 and the article is summarized in Section 7.

2. DP matrix

Let us recall the distinguished point (DP) pre-computation aided DLP solving algorithm. Certain decisions need to be made to setup the algorithm. One chooses integer parameters m and t such that $mt^2 \approx p$, where p is the order of the group on which the DLP is to be solved. Let $f : G \to G$ be any function that is exponent traceable in the sense that $f(g^a h^b)$ can again be written in the form $g^{a'} h^{b'}$ whenever a and b are known and which can be described independently of the DLP instance h. The r-adding walk iteration function [35, 38], with multipliers set to the form g^{α_i}, rather than $g^{\alpha_i} h^{\beta_i}$, is a good example. The iteration function f will always be the same in the remainder of this work, regardless of the DLP algorithm, and our computational complexities will always refer to the expected, rather than the worst case, number of iteration function applications. Fix a property for elements of G which is satisfied by a random element of G with probability $\frac{1}{t}$ and which may easily be checked under the encoding scheme for G. A typical example would be to require $\log t$ least significant bits to be zero. Any element of G that satisfies this distinguishing property is defined to be a DP.

To start the pre-computation phase, one first chooses m starting points $x_{i,0} = g^{\gamma_i}$ and, for each $1 \leq i \leq m$, iteratively computes $x_{i,j} = f(x_{i,j-1})$ until an $x_{i,j}$ is found to be a DP. The exponent traceable property of the iteration function allows for one to keep track of the discrete logarithm value of each $x_{i,j}$. The m DPs x_{i,j_i} that terminated the pre-computation chains are collected, together with their corresponding discrete logarithm values, and stored as the pre-computed *table*, after being sorted. Any duplicate ending point DPs are removed to reduce storage space. Below, we will refer to the gathering of all pre-computation chain elements $x_{i,j}$, collected over the indices $1 \leq i \leq m$ and the full range of $0 \leq j \leq j_i$ for each i, mentally visualized as a directed graph with arrows indicating the iteration function operations, as the *DP matrix*. The term *matrix* will also be used with other DLP algorithms to refer to similar structures.

The online phase starts when the DLP instance h is given. One randomizes h to a starting point $y_0 = g^{a_0} h^{b_0}$ ($b_0 \neq 0$) and computes the online chain by iteratively setting $y_j = f(y_{j-1})$, while keeping track of the exponent expressions $y_j = g^{a_i} h^{b_i}$. When the online chain reaches a DP, it is searched for in the pre-computation table. Any match will allow $\log_g h$ to be computed from a linear equation. If a collision is not found, the online chain generation is repeated from another randomized starting point.

Let us review an extremely rough analysis of the DP method. The expected length of a DP chain, such as any one of the pre-computation chains or online chains, is t. When the iteration function f is assumed to be a random function, the relation $mt \cdot t \approx p$ and the birthday paradox imply that the DP matrix will contain close to mt distinct elements. The same argument also implies that the online chain of expected length t has a reasonable chance of merging into the DP matrix. Hence, we can claim the online time complexity for the DP method to be approximately t iterations of f.

In summary, the pre-computation complexity $P = mt$, the storage complexity $M \approx m$, and the online time complexity $T \approx t$ satisfy the relations

$$PT \approx p \quad \text{and} \quad T^2 M \approx p. \tag{1}$$

These relations hold only approximately, but the hidden multiplicative factors are neither very large nor close to zero. Once again, we clarify that the complexities P, T, and M used here and later in this work are always the expected or *average case* complexities rather than the worst case complexities.

A more careful analysis was carried out by [23], which stated through its Eq. (6) that $T = \frac{\sqrt{1+2c}}{\sqrt{1+2c}-1}t$, where $c = \frac{mt^2}{p}$. In fact, this claim is an easy consequence of Proposition 10 from [17], and the long arguments given in [23] were unnecessarily complex. The work [23] combined this claim with $M \approx m$, but, as recently noted by [7], this is inappropriate when one requires accuracy. The correct value is stated by Lemma 2 of [24] to be $M = \frac{\sqrt{1+2c}-1}{c}m$, where $c = \frac{mt^2}{p}$ is as before. Recalling $P = mt$, the precise tradeoff curves may be stated as

$$PT = \frac{c\sqrt{1+2c}}{\sqrt{1+2c}-1}\, p \tag{2}$$

and

$$T^2 M = \frac{1+2c}{\sqrt{1+2c}-1}\, p, \tag{3}$$

where $c = \frac{mt^2}{p}$.

We wish to emphasize that these are not some arbitrary relations satisfied by the three complexities, but true tradeoff curves in the following sense. Suppose parameters m and t achieve performance complexities P, M, and T. Given any $\alpha > 0$, we can choose to execute the DP method at performance complexities $P' = \alpha P$, $M' = \alpha^2 M$, and $T' = \frac{T}{\alpha}$, by setting parameters to $m' = \alpha^2 m$ and $t' = \frac{t}{\alpha}$. In other words, we can increase or decrease online time T to suit our needs as long as it is balanced out by the change in pre-computation cost P and storage size M, as specified by the tradeoff curves (2) and (3) with the right-hand side held constant.

3. Rainbow matrix and its variants

In this section, we discuss four DLP algorithms that incorporate the rainbow matrix structure into its pre-computation matrix in some manner and conclude that none of these perform better than the DP method.

The DP matrix structure is more widely known among cryptographers in relation to the cryptanalytic time memory tradeoff (TMTO) technique [13] for inverting one-way functions than in relation to solving DLPs. Since the rainbow TMTO algorithm [31] is widely accepted as being more efficient than the DP

TMTO algorithm, it is only natural to ask if replacing the DP matrix structure with the rainbow matrix structure will have a positive effect on DLP solving.

An analysis of the most natural adaptation of the rainbow TMTO algorithm into a DLP algorithm is given in Appendix A. There it is shown that the pre-computation complexity P, storage complexity M, and online time complexity T of the resulting algorithm satisfy the relations

$$P\sqrt{T} \approx p \quad \text{and} \quad TM \approx p. \tag{4}$$

Comparing these two curve with the tradeoff curves (1) for the DP method, one can see that, to attain the same level of online time complexity, a rainbow method user must invest larger amount of pre-computation and utilize a larger storage space than a DP method user. It is clear that the use of the rainbow matrix structure in DLP solving results in an algorithm that is much worse than the DP method.

Cryptographers familiar with the details of the TMTO techniques may recall that the rainbow matrix was designed so as to make it harder for pre-computation chains to merge into each other. Since any variant of the Pollard's rho method requires a merge of chains to be successful, the inadequacy of the rainbow matrix approach to DLP solving may not be surprising. However, a closer look at the details given in Appendix A reveals that this view of the situation is not very accurate. The effect of the rainbow matrix structure in averting chain merges is fully realized when the size of the matrix is roughly the size of the space the one-way function is acting on, but the number of points covered by the rainbow matrix that may naturally be associated with DLP solving is much smaller than p. In other words, the advantage of the rainbow matrix structure is not leveraged by its use in DLP solving.

The intention of the rainbow TMTO in reducing chain merges was to be able to create a much larger pre-computation matrix, so that a large number of DP TMTO matrices could be combined into a single rainbow matrix. In fact, it is often argued that a DP matrix corresponds naturally to a single column of a rainbow matrix. Since the DP DLP algorithm requires only a single matrix, the possible advantage of the rainbow matrix structure, which allows for the gathering of multiple matrices into one, could not be put to use, when it was applied to DLP solving.

An observation that is very closely related to the above discussion, which essentially notes the rainbow TMTO's lack of freedom in choosing the number of its tables, is the difference in the time memory *data* tradeoff (TMDTO) curves for the multi-target versions of the two TMTO methods. Assuming a search space of size N, the tradeoff curve for the DP TMDTO method is $TM^2D^2 \approx N^2$ [10], where as that of the rainbow TMDTO method is $TM^2D \approx N^2$ [9]. Given the same number of targets D, among which only one needs to be inverted, the rainbow TMDTO method requires larger time and storage resources than the DP TMDTO method.

This brings us to the fuzzy rainbow TMDTO method [5, 6], which was created with the specific goal of obtaining a rainbow TMDTO variant that has $TM^2D^2 \approx N^2$ as its time memory data tradeoff curve. The algorithm is a combination of the DP and rainbow methods and was the central tool used in the successful demonstrations [29, 30] of defeating GSM phone security. Even though both the DP and rainbow matrix structures can be recovered from the fuzzy rainbow matrix through extreme parameter choices, it was recently shown [20, 21] that the single-target restriction of the fuzzy rainbow algorithm greatly outperforms both the DP and rainbow TMTO algorithms.

An analysis of the most natural adaptation of the fuzzy rainbow TMDTO algorithm into a DLP algorithm is given in Appendix B. The pre-computation complexity P, storage complexity M, and online time complexity T of the fuzzy rainbow DLP algorithm satisfy the relations

$$PT \approx \frac{s+1}{2}p \quad \text{and} \quad T^2M \approx \left(\frac{s+1}{2}\right)^2 p, \tag{5}$$

where s is an integer parameter to be chosen for the algorithm.

The choice of $s = 1$ reduces the fuzzy rainbow DLP algorithm into the DP method, and the above two curves also reduce to the tradeoff curves (1) for the DP method. For all other cases, i.e., for $s > 1$, it is clear that the DP method performs better than the DLP algorithm that utilizes fuzzy rainbow matrices. As with the rainbow case, a review of the details given in Appendix B reveals that the parameters that are most suitable for the fuzzy rainbow TMDTO method could not be used with its DLP solving adaptation.

The fuzzy rainbow TMDTO method was introduced together with two auxiliary TMDTO methods that are referred to as the thin rainbow and thick rainbow methods [5, 6]. Even though there is no reason to expect these to perform any better than the fuzzy rainbow method, for completeness, very brief discussions of the DLP solving versions of these two algorithms are given in Appendix C. Neither of these hold any advantage over the DP method.

4. Classical Hellman matrix

We have witnessed that the DLP algorithm is not likely to benefit from the introduction of the reduction functions or colored iteration functions that appear in the rainbow matrix structure. Having considered all other widely known TMTO pre-computation matrix structures, we now turn to the most basic pre-computation matrix structure, which was used by the classical Hellman TMTO algorithm [13].

A straightforward adaptation of the classical Hellman TMTO method into a DLP algorithm requires parameters m and t such that $mt^2 \approx p$. Each pre-computation chain is generated to be of the form

$$\mathrm{SP}_i \xrightarrow{f} \circ \xrightarrow{f} \circ \ \cdots \ \circ \xrightarrow{f} \mathrm{EP}_i,$$

with the chain length fixed to t. After sorting, the m pre-computation chain ending points EP_i are stored, together with their corresponding discrete logarithm values, as the pre-computation table. Unlike the DP matrix, duplicates among the ending points of a Hellman matrix are quite rare.

The online phase consists of discrete steps that are continued until the DLP is solved. The online chain starts from a randomization $y_0 = g^a h^b$ of the given DLP instance h. On the k-th step, the online chain

$$y_0 \xrightarrow{f} y_1 \xrightarrow{f} \circ \;\cdots\cdots\; \circ \xrightarrow{f} y_{k-1}$$

is extended by a single f-iteration, after which the current ending point y_k is searched for in the pre-computation table for matches. Any match leads to the solution of the DLP. The Hellman online chain need not be restarted from a new randomized starting point, even when its length reaches t.

As with the DP method, one can argue that the Hellman matrix version of the DLP algorithm presents expected complexities $P = mt$, $M = m$, and $T \approx t$, which leads to relations identical to (1). Hence, we can claim that the performances of the DP method and the Hellman method are comparable, but the constant factors hidden behind the approximations prevent us from announcing either of the two algorithms as being superior over the other.

We wish to find tradeoff curves for the Hellman method analogous to (2) and (3). It is stated in [17] that each Hellman matrix of $m \times t$ dimensions contains $\left(\frac{\sqrt{2}}{\sqrt{c}}\frac{e^{\sqrt{2c}}-1}{e^{\sqrt{2c}}+1}\right)mt$ distinct entries, where $c = \frac{mt^2}{p}$. This is a direct consequence of claims found in [11, 25]. Assuming f to be a random function, each iteration of the online chain has probability

$$\left(\frac{\sqrt{2}}{\sqrt{c}}\frac{e^{\sqrt{2c}}-1}{e^{\sqrt{2c}}+1}\right)\frac{mt}{p} = \frac{\sqrt{2c}(e^{\sqrt{2c}}-1)}{e^{\sqrt{2c}}+1}\frac{1}{t}$$

of merging into the Hellman matrix, and one can expect to observe $\frac{e^{\sqrt{2c}}+1}{\sqrt{2c}(e^{\sqrt{2c}}-1)}t$ iterations of f, until the online chain merges into the Hellman matrix.

Once the online chain merges into the Hellman matrix, its further iterations must follow the iterations set forth by the pre-computation chain. If all mt entries of the Hellman matrix were distinct, we could expect $\frac{t}{2}$ further iterations to be performed until the online chain reaches the ending point and is detected through a match in the pre-computation table. Hence $\left(\frac{1}{2} + \frac{e^{\sqrt{2c}}+1}{\sqrt{2c}(e^{\sqrt{2c}}-1)}\right)t$ is an upper bound on the expected number of f iterations. We already know that only $\left(\frac{\sqrt{2}}{\sqrt{c}}\frac{e^{\sqrt{2c}}-1}{e^{\sqrt{2c}}+1}\right)mt$ of the Hellman matrix entries are distinct. The best possible situation is for this many entries closest to the ending points of the Hellman matrix to be distinct and for all points that are further away to be duplicates of these entries. In such a case, we can expect the merge of the online chain into the pre-computation Hellman matrix to be discovered after $\frac{1}{2}\left(\frac{\sqrt{2}}{\sqrt{c}}\frac{e^{\sqrt{2c}}-1}{e^{\sqrt{2c}}+1}\right)t$ further iterations.

The online time complexity T can now be bounded by

$$(6)\qquad \frac{1}{\sqrt{2c}}\left\{\frac{e^{\sqrt{2c}}-1}{e^{\sqrt{2c}}+1}+\frac{e^{\sqrt{2c}}+1}{e^{\sqrt{2c}}-1}\right\}t \le T \le \frac{1}{\sqrt{2c}}\left\{\frac{\sqrt{c}}{\sqrt{2}}+\frac{e^{\sqrt{2c}}+1}{e^{\sqrt{2c}}-1}\right\}t,$$

and the other two complexities $P = mt$ and $M = m$ are easy to state. The time complexity upper bound $\hat{T}$ satisfies the relations

$$(7)\qquad P\hat{T} = \frac{\sqrt{c}}{\sqrt{2}}\left\{\frac{\sqrt{c}}{\sqrt{2}}+\frac{e^{\sqrt{2c}}+1}{e^{\sqrt{2c}}-1}\right\}p,$$

$$(8)\qquad \hat{T}^2M = \frac{1}{2}\left\{\frac{\sqrt{c}}{\sqrt{2}}+\frac{e^{\sqrt{2c}}+1}{e^{\sqrt{2c}}-1}\right\}^2 p,$$

and the time complexity lower bound $\check{T}$ satisfies the relations

$$(9)\qquad P\check{T} = \frac{\sqrt{c}}{\sqrt{2}}\left\{\frac{e^{\sqrt{2c}}-1}{e^{\sqrt{2c}}+1}+\frac{e^{\sqrt{2c}}+1}{e^{\sqrt{2c}}-1}\right\}p,$$

$$(10)\qquad \check{T}^2M = \frac{1}{2}\left\{\frac{e^{\sqrt{2c}}-1}{e^{\sqrt{2c}}+1}+\frac{e^{\sqrt{2c}}+1}{e^{\sqrt{2c}}-1}\right\}^2 p.$$

The correct tradeoff curves will be somewhere in between these two sets of curves.

5. Comparisons of DLP algorithms

This section presents a comparison made between the DP method, the BL method [7], and the Hellman method for pre-computation aided DLP solving. We also provide some experimental evidence supporting our theoretical analysis of the Hellman method that was given in the previous section.

The method of comparison needs to be explained first. Since each DLP algorithm allows tradeoffs to be performed between storage size M and online time T, comparison of the T's for the three methods must only be done under a common M value. However, let us explain that the comparison can be made for all M values at once, i.e., without restricting ourselves to a few specific M values. We already know that both the DP and Hellman methods satisfy tradeoff curves of the form (1). Likewise, the paper [7] provided experimental evidence that the BL method also satisfies (1). Hence a direct comparison of the possible $\frac{T^2M}{p}$ values afforded by the three algorithms reveals how each algorithm fares concerning their requirement on the online resources T and M. The method with a smaller value of $\frac{T^2M}{p}$ requires a smaller storage space M in achieving the same online time T. Similarly, the value of $\frac{PT}{p}$ quantifies how effective an algorithm is in transforming the pre-computation effort P into shorter online

TABLE 1. Experimental online time complexities for the Hellman method. Group size is $p = 88629945038963$. Online chain length T has been averaged over 100 tables and 5000 DLP instances per table.

m	t	$\frac{mt^2}{p}$	$\frac{T}{t}$	failures	$\left(\frac{PT}{p}, \frac{T^2M}{p}\right)$
61910	2^{15}	0.75	1.991	16	$(1.493, 2.973)$
82550	2^{15}	1.00	1.643	6	$(1.643, 2.699)$
25790	2^{16}	1.25	1.435	26	$(1.793, 2.572)$
30950	2^{16}	1.50	1.291	9	$(1.936, 2.500)$
144450	2^{15}	1.75	1.187	4	$(2.077, 2.466)$
41280	2^{16}	2.00	1.109	13	$(2.217, 2.458)$
46430	2^{16}	2.25	1.046	6	$(2.354, 2.462)$
51590	2^{16}	2.50	0.995	11	$(2.487, 2.474)$

DLP solving time T, with a smaller value implying better efficiency. In summary, the two numerical values $\frac{PT}{p}$ and $\frac{T^2M}{p}$ for a DLP algorithm concisely express the algorithm's pre-computation and online efficiencies, respectively.

Let us now take a closer look at these two numerical quantities, focusing on the DP method. It is clear from the right-hand sides of (2) and (3) that these values are not constant and change with the choice of parameters. Furthermore, the two right-hand side values are not independent of each other, and it is reasonable to expect a $c = \frac{mt^2}{p}$ value that decreases one of the two to increase the other. That is, there will be an *upper level tradeoff* between the pre-computation and online efficiencies. Since everyone will place different relative values on pre-computation efficiency and online efficiency, and since these subjective appraisals will also be affected by the pre-computation and online resources available to the implementer, no single balance between these two can be objectively claimed to be optimal. We come to the conclusion that our comparison must be based on the whole range of $\left(\frac{PT}{p}, \frac{T^2M}{p}\right)$ pairs made possible by the DLP solving methods.

The upper level tradeoffs for DP, BL, and Hellman methods are illustrated in Figure 1. The thick solid line presents the $\left(\frac{PT}{p}, \frac{T^2M}{p}\right)$ pairs for the DP method, plotted as a curve parameterized by $c = \frac{mt^2}{p}$, using (2) and (3).

The two dashed lines present the $\left(\frac{P\hat{T}}{p}, \frac{\hat{T}^2M}{p}\right)$ pairs and $\left(\frac{P\check{T}}{p}, \frac{\check{T}^2M}{p}\right)$ pairs for the Hellman method, as given by the formulas (7) through (10). The series of eight filled dots are from the final column of Table 1, which contain the test results from our Hellman method implementation. These dots are where the true $\left(\frac{PT}{p}, \frac{T^2M}{p}\right)$ pair curve for the Hellman method should be. The dots appear between the two dashed lines and confirm the correctness of our theoretical bounds. Our experiments were carried out on a multiplicative subgroup

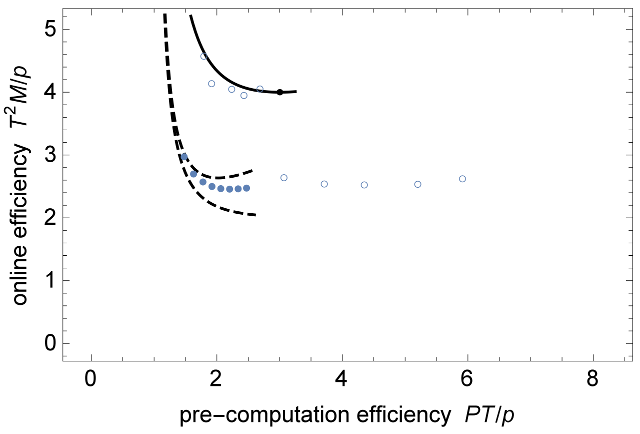

FIGURE 1. Performances of pre-computation aided DLP solving algorithms. Solid line: DP theoretical; Filled dots: Hellman experimental; Dashed lines: Hellman theoretical upper/lower bounds; Empty circles: BL experimental.

of the prime field $\mathbb{F}_q$, where $q = 177259890077927$ is a 48-bit prime. The cyclic group G was generated from $g = 2 \in \mathbb{F}_q$, and the group was of 47-bit prime order $p = 88629945038963$. The $\frac{T}{t}$ value in each row of Table 1 is an average taken over 100 Hellman pre-computation tables and 5000 DLP solving attempts per table. For some of the DLP instances, the Hellman method failed to return the answer until the online chain length reached $20t$, and we counted these instances as failures. Most of these are likely to be the result of the online chain entering a loop and a practical implementation would have solved these DLPs from the self-collision information. The final column of Table 1 contains $\left(\frac{PT}{p}, \frac{T^2M}{p}\right)$ coordinates calculated from $P = mt$, $M = m$, and the experimentally obtained T.

The ten empty circles in Figure 1 correspond to the experimental data for the BL method found in [7]. As was briefly mentioned before, although the article [7] did not present the exact tradeoff curves, it did provide some arguments and experimental data to support the possibility of tradeoffs of the form $PT \approx p$ and $T^2M \approx p$. The positions of the plotted empty circles were computed from the experimental P, T, and M values given in Table 4.1 of [7]. The table contained three sets of data that correspond to certain sets of parameter choices for the BL method referred to as $\mathsf{T} = \mathsf{N}$, $\mathsf{T} = 2\mathsf{N}$, and $\mathsf{T} = 8\mathsf{N}$ cases. The $\mathsf{T} = k\mathsf{N}$ case refers to when only $\frac{1}{k}$ of the pre-computed DPs are selectively retained with their discrete logarithm information in the final pre-computation table. The parameter choice of $\mathsf{T} = \mathsf{N}$ reduces the BL method to the DP method, and the set of five empty circles appearing near the thick line DP curve corresponds to this case. The other five empty circles appearing in

Figure 1 correspond to the $\mathsf{T} = 2\mathsf{N}$ case BL method. The empty circles for the third $\mathsf{T} = 8\mathsf{N}$ case data set are not visible in Figure 1, being much further away to the right, but they should be placed at the coordinate positions $(21.7, 1.91)$, $(29.1, 1.98)$, $(37.4, 2.04)$, $(45.8, 2.15)$, and $(55.3, 2.24)$.

A position in the box of Figure 1 that is lower corresponds to parameters that would require less resources during the online phase. A position that is closer to the left edge of the box frame corresponds to more efficient use of pre-computation resources. Hence, one would prefer to use parameters corresponding to points that are lower and more to the left. For example, the lowest point on the DP curve is marked by the single dot on the solid line, and there is no reason to use parameters corresponding to points on the DP curve that are located to the right of this point, since they correspond to less efficient usage of pre-computation at worse online efficiency than the lowest point.

The curves clearly indicate that the Hellman method is very advantageous over the DP method. For example, we can compare the lowest point on the DP curve of coordinates $(3, 4)$ with the point approximately at $(1.2, 4)$ on the Hellman curve to state that the Hellman method can be made to operate with the same online storage and time complexities that are optimal for the DP method, but with the Hellman method calling for only $\frac{1.2}{3} = 40\%$ of the pre-computation effort required by the DP method. Similarly, making a very rough comparison in the vertical direction at $\frac{PT}{p} \approx 2$, we can claim that the two algorithms can be made to execute with the same online time complexity after the same investment in pre-computation effort, but with the Hellman table containing only half the number of entries required by the DP method.

A comparison between the BL method at $\mathsf{T} = 2\mathsf{N}$ and the Hellman method can likewise be made. The two methods have similar optimal online efficiencies, but the Hellman method can utilize pre-computation much more efficiently than the BL method in achieving the common online efficiency. As for the BL method at $\mathsf{T} = 8\mathsf{N}$, the explicit coordinates we had listed above show smaller $\frac{T^2M}{p}$ values than the Hellman method figures. Hence, the online phase of the BL method can be more efficient than the Hellman method. However, the small advantage in online efficiency must be paid for with much larger pre-computation, and this disadvantage seems too great to make the use of the BL method at $\mathsf{T} = 8\mathsf{N}$ reasonable.

In short, we can claim that the Hellman method outperforms both the DP and BL methods, when pre-computation computational complexity, storage complexity, and online computational complexity are taken into account.

6. Further discussions

In this section, we provide a rough high-level argument as to why the Hellman method turns out to be superior to the DP method. We also briefly discuss two practical issues, namely, the physical storage size and table lookup frequency issues, which could somewhat weaken the claimed superiority of the Hellman method over the DP and BL methods.

6.1. Core difference between DP and Hellman methods

This subsection aims to provide the reader with a rough understanding of what is happening behind the scene and contains arguments that are not necessarily accurate.

The Hellman method utilizes a pre-computation matrix of $m \times t$ dimensions, and, since the average length of randomly created DP chains is t, one could also view the matrix used by the DP method as being roughly of $m \times t$ dimensions. The online phases of both algorithms aim to uncover a merge of the online chain into the pre-computation matrix. Hence, on the surface, the DP and Hellman methods appear to be very similar to each other, and there seems to be no reason to expect the large performance difference we saw in the previous section.

In short, this plausible view fails because the DP chain of average length is not the typical or most commonly observed DP chain. The probabilities of encountering DP chains of various lengths form a geometric distribution and the *shape* of the DP matrix, if such a notion could be defined, is very different from the shape of the Hellman matrix, which most would take as being rectangular. More specifically, one must expect a DP matrix to consist of a small number of long chains that are often merging into each other and a large number of mostly independent short chains.

Consider a DP matrix and a Hellman matrix that were created from the same number of starting points m and with the same t. As may be confirmed through the *coverage rate* formulas found in [17], the non-uniform nature of the DP matrix and the property of chain merges being more common among longer chains results in the DP matrix containing a smaller number of distinct points than the Hellman matrix. Consequently, we must expect an online chain for the DP method to take longer in merging into the DP matrix than the same for the Hellman method. Furthermore, since the longer DP chains in a DP matrix are more likely to attract the online chain than the shorter ones, it takes longer for an online chain that has merged into a DP matrix to reach the ending point of the corresponding pre-computation chain, than for the same to occur with the Hellman matrix. For a common set of parameters m and t, which implies a common pre-computation cost, the DP matrix will require more one-way function iterations than the Hellman matrix for the online chain to merge into the matrix and also for the merge to be detected at the ending of the pre-computation chain.

Another difference between the DP and Hellman methods is the length of the online chain. With the DP method, when the online chain reaches a DP without merging into the pre-computation matrix, the online chain is regenerated from a newly randomized version of the DLP instance. Hence the DP method is limited in taking advantage of the effect of a longer (online) chain being more likely to merge into other (pre-computation) chains. Note that this disruption at the occurrence of a DP on the online chain cannot be resolved by making the second chain start from the first chain's ending DP, since we already know that no part of the first chain merges into the DP matrix. In contrast to the DP method, with the Hellman method, the online chain is iterated until the DLP is solved, regardless of the chain length. Even though making the length of the online chain longer has no additional merit once the chain length reaches t, one is not forced to discard the acquired higher merging probability advantage of a long online chain.

The chain merges of the DP matrix provides the DP method with the advantage of reduced storage size that is not available to the Hellman method. However, the analyses of the previous sections have shown that this cannot compensate for the multiple disadvantages of the DP method in the online time we have discussed above.

6.2. Physical storage size

Recall that the storage complexity M used in computing $\frac{T^2M}{p}$ presented the pre-computation table size in terms of the number of table entries. However, the numeric figure of practical interest would be the table size expressed in number of bits. The issue we have overlooked is the number of physical bits that need to be allocated to each table entry.

A naively structured pre-computation table would hold the full encoding of each group element together with the full discrete logarithm value of this group element. Each full discrete logarithm record would require $\log p$ bits of storage and the number of bits required to record each group element would depend on the encoding scheme for G. Now, note that it is not absolutely necessary to record each group element in full. One can use partial information concerning each group element entry to identify *probable* matches, and after computing the candidate $\log_g h$ value from the assumed match, the correctness of the obtained answer can easily be check through a single exponentiation. This and many other techniques for reducing the number of bits allocated to each table entry were discussed in [7].

Let us only state that all the storage reduction techniques applicable to the DP and BL methods can also be applied to the Hellman method, as long as at most $\log t$ additional bits are allocated to each Hellman table entry. In short, this is because the $\log t$ additional bits will allow only one of the t times more frequent Hellman table lookups to be falsely identified as matching.

Depending on the parameter choices and implemented reduction techniques, the additional $\log t$ bits may or may not be a significant portion of bits allocated to each table entry. For example, a reasonable implementation would allocate slightly more than $\log m$ bits to group element information and full $\log p$ bits to discrete logarithm information, and a further addition of $\log t$ bits will only be a small change, even though it would be noticeable. In theory, each table entry for the DP or BL method could be made much smaller than $\log m + \log p$ bits. However, the storage reduction techniques are quite complex and delicate, and even further complicated by byte boundary issues, so that it seems unlikely that we will see them implemented very aggressively in practice. In fact, similar storage reduction techniques are known for the cryptanalytic time memory tradeoff algorithms, but the pre-computed tables that are publicly available [1, 2, 3, 4] incorporate only the simplest of the theoretically available techniques.

In summary, the issue of number of bits allocated to each table entry could weaken our claimed advantage of the Hellman method over the DP method, but a proper adjustment is unlikely to bring the performance of the DP method anywhere close to that of the Hellman method. Consideration of the same issue could imply that the performances of the BL method at $\mathsf{T} = 2\mathsf{N}$ and the Hellman method are somewhat comparable, but it seems unlikely that the preference would be overturned because of this issue.

6.3. Table lookup frequency

We will next discuss the table lookup frequency disadvantage of the Hellman method. Our analyses of the DLP algorithms were focused on the storage and computational complexities, but the practical performance of any algorithm that relies on a large table is also heavily dependent on its number of table lookups. Note that the DP method requires only a small number of searches to the pre-computation table, whereas the Hellman method requires table searches of t order. If the pre-computation table is small enough to be fully loaded into the online phase system's fast main memory, the table lookups will not cause noticeable reduction in execution speed, even in the Hellman matrix approach, and our comparisons of the previous section will be valid. On the other hand, if the pre-computation table is too large to fit within the main memory and frequent accesses to slow disk storage is inevitable throughout the online phase, the time required for the table lookups will dominate the online time of the Hellman method, so that our analysis of the Hellman method and the comparisons of the previous section will have little practical meaning.

To consider whether or not the pre-computation table will fit within the online memory, let us briefly work with specific but very rough estimates. A review of the available pre-computed tables [1, 2, 3, 4] for the cryptanalytic time memory tradeoff technique indicates that approximately $\sum_{i=1}^{9} 62^i \approx 2^{53.6}$ or $\sum_{i=1}^{8} 95^i \approx 2^{52.6}$ iterations of a password hash function is the upper limit on the amount of pre-computation that can be done within a reasonable time by a commercial entity or through distributed computing on donated CPU cycles. The time taken for one password hash function iteration is different from those taken by a multiplication in a large finite field or an addition on an elliptic curve, but we can claim that $P \approx 2^{50}$ is a reasonable rough upper bound on the pre-computation iterations that can be done now in practice, disregarding large secretive government organizations. When applications to trapdoor discrete logarithm groups are considered, the upper bound should actually be taken much lower than 2^{50}, since, unlike the cryptanalytic time memory tradeoff situation, the pre-computation must be done on different groups for each intended user of the system. As for the online computational complexity T, we could verify that even our six years old laptop was capable of computing 2^{20} iterations of the 32-adding walk on a finite field of 1024-bit size within 8 seconds, using only a high-level Mathematica code. Even though the range of reasonable online time will be severely dependent on the application, let us assume that one is allowed $T \approx 2^{20}$ iterations during the online phase.

Putting the two complexities together, we can claim that the DP, BL, and Hellman DLP algorithms will use storage size of roughly $M \approx m = \frac{mt}{t} \approx \frac{P}{T} \approx 2^{30}$ order. If very large pre-computation resources of 2^{50}-order group operations are not available, the resulting pre-computation table will fit comfortably within the main memory of modern PCs. On the other hand, if the intended online platform is slow in computation or the intended application requires the DLPs to be solved within a very short time period, a larger pre-computation table, which cannot be loaded into fast memory, will be required.

Let us summarize our discussion concerning the table lookup frequencies. The Hellman method is practical to use only when the pre-computation table size is small enough to fit within the online phase system's main memory. When otherwise, the time taken for table lookups makes the advantage of the Hellman method, theoretically argued in the previous section, meaningless in practice. There will be many applications where loading of pre-computation table into fast memory can easily be done, but there will also be the opposite situations.

7. Conclusion

This work presented a comprehensive overview of possible algorithms for solving DLPs with the aid of pre-computation and uncovered the Hellman method which performs better than existing algorithms in many situations.

We first reviewed the DP method and took note of the fact that the structure of the DP pre-computation matrix is identical to that used in the cryptanalytic time memory tradeoff algorithm of the same name. Observing that the rainbow time memory tradeoff algorithm is widely believed to be better than the DP time memory tradeoff, we studied the effects of replacing the DP pre-computation matrix structure with the rainbow matrix and concluded that the resulting DLP algorithm was inferior to the original DP method. We further investigated the possibility of using the pre-computation matrix structures found in three other variants of the rainbow time memory tradeoff and confirmed that all of these adaptations performed worse than the DP method.

The classical Hellman pre-computation table structure was considered last. After confirming through a rough argument that the DP and Hellman methods would perform comparably, we attempted a more accurate analysis of the Hellman method, and could obtain upper and lower bounds for the expected performance. Results of our theoretical analysis were in good agreement with data obtained from our implementation of the Hellman method. Based on the obtained knowledge of the Hellman method performance, we could argue that the Hellman method is advantageous over the DP method, at least in theory, when pre-computation cost, storage size, and online computational complexity are taken into account. We could similarly claim that the Hellman method was better than the BL method, which was announced recently. We clarified that the theoretical comparisons could be of limited practical value when the full pre-computation table cannot be loaded into the main memory of the online phase system, but could observe through rough estimates of reasonable complexities that situations where the table loading requirement could be accommodated would be common.

Our investigation of possible pre-computation aided DLP solving algorithms uncovered an algorithm that is better than the existing algorithms, although its practicality could be limited in some situations. However, we believe a greater value of this study lies in the lessons learned through the investigation, which is why we had included the descriptions of the many failed attempts in the appendices of this paper. We can make the following vague comments or suggestions concerning future attempts at creating a better DLP algorithm.

First, the use of reduction functions or colored iteration functions is not likely to be beneficial. In the rainbow time memory tradeoff algorithm, colors provided certain independence between columns of the rainbow matrix so that most of the search space could be covered by a single table, but the requirement for search space coverage by a DLP algorithm is of such small order that appropriate coverage can be achieved without engaging any special techniques.

Second, if one is not going to attempt a radically different approach, a good place to start would be tweaks of the DP and Hellman methods. One might even hope for some combination of the two methods to retain the computational complexity advantage of the Hellman method and the minimal table lookup characteristic of the DP method.

Third, a possible direction to try is to re-approach the Hellman method with new measures of performance that better reflect what is important in practice. That is, one should go outside the traditional cryptographic framework that emphasizes computational time complexities and somehow include the wall-clock time taken for disk accesses in the performance analyses of the DLP algorithms. After preparing a more practical definition for performance, one should reorder the operations of the Hellman method so that storage accesses are made somewhat local and/or sequential. For example, one could first generate the online chain up to a certain length, sort the generated online chain elements, and then make multiple table lookups. This could especially be of practical interest in the situation where multiple DLPs need to be solved in batches and also in relation to the multi-core CPU platforms that are now very common.

Appendix A. Rainbow matrix

This section provides a rudimentary analysis of the DLP algorithm that is based on the rainbow matrix structure.

Let us start by describing a natural adaptation of the rainbow TMTO algorithm into a DLP algorithm. The rainbow TMTO algorithm uses parameters m and t that satisfy the so called matrix stopping rule $mt \approx N$, where N is the size of the search space, but analogously requiring $mt \approx p$ would already imply an excessively large pre-computation cost of $P = mt \approx p$. Hence, let us treat m and t only as matrix size parameters and defer the determination of the relation that should replace $mt \approx p$.

As in the DP method, the iteration function f needs to be chosen to be independent of the DLP instance. One must next choose t reduction functions $r_i : G \to G$ that are easy to compute. The TMTO algorithm only requires these to be bijections, and very simple maps such as byte permutations or constant XORs could be used. However, for use in DLP solving, these need to be exponent traceable as with f. An option that is analogous to the constant XOR operation of the TMTO case is to multiply by a fixed power of g, with the power chosen differently for each i. Once the reduction functions are chosen, the associated colored iteration functions $f_i = r_i \circ f$ can be fixed. Each iteration of f_i could be twice as costly as that of f to compute, but let us ignore this and give the rainbow method an unfair advantage, since we will still announce the rainbow method as being inferior to the DP methods in the end. Note that having the r-adding walk multipliers pre-computed to account for the reduction functions could make the costs of f_i and f iterations identical, but such an approach will call for an uncomfortably large main memory space of rt order to hold the multipliers.

During the pre-computation phase, fixed-length chains of the form

$$\mathrm{SP}_i \xrightarrow{f_1} \circ \xrightarrow{f_2} \circ \ \cdots \ \circ \xrightarrow{f_t} \mathrm{EP}_i,$$

are generated from m starting points $\mathrm{SP}_i = g^{\gamma_i}$, and the ending points EP_i, together with their discrete logarithm values, are recorded as the pre-computation table. Duplicate ending points may be removed to reduce storage space, but collisions among pre-computation chain ending point will be rare for parameters to be described below.

The online phase consists of $t+1$ stages. In the k-th stage ($0 \le k \le t$), the online chain

$$y_{k,0} \xrightarrow{f_{t-k+1}} y_{k,1} \xrightarrow{f_{t-k+2}} \circ \ \cdots\cdots \ \circ \xrightarrow{f_t} y_{k,k}$$

of length k is generated, starting from a suitable randomization $y_{k,0} = g^{\alpha}h^{\beta}$ of the DLP instance h, which may be the same for all k. The ending point $y_{k,k}$ for the k-th online chain is searched for in the pre-computation table. Any match will lead to the answer of the DLP instance. If all $t+1$ stages fail to return the answer, the process is repeated with a different randomized starting point for the online chains.

One can infer from Lemma 6 of [16] that an online chain of length k will merge into a pre-computed rainbow chain with probability $\frac{k+1}{p}$. In fact, this claim is easy to verify through elementary random function arguments. The number of merges expected from the $t+1$ stages of the online phase, which requires $T = \frac{t(t+1)}{2}$ function iterations, is $\sum_{k=0}^{t} m\frac{k+1}{p} \approx \frac{mt^2}{2p}$. If the parameters m and t are chosen so that $\frac{mt^2}{2p} \approx 1$, then one can expect one of the k online chains to produce a match with reasonable probability. If $\frac{mt^2}{2p}$ is made sufficiently larger than 1, one is almost certain to see a match.

In summary, the rainbow matrix version of the DLP algorithm running with parameters m and t satisfying $mt^2 \approx p$ displays pre-computation complexity $P = mt$, storage complexity $M \approx m$, and online time complexity $T \approx t^2$, ignoring small multiplicative factors. It is easy to check that these complexities satisfy the tradeoff curve as given by (4).

Appendix B. Fuzzy rainbow matrix

This section provides a rudimentary analysis of the DLP algorithm that is based on the fuzzy rainbow matrix structure.

We start with the description of a natural pre-computation aided DLP solving algorithm adaptation of the fuzzy rainbow TMTO method. Positive integer parameters m, t, and s first need to be chosen. The fuzzy rainbow TMTO method sets the matrix stopping rule to $mt^2s \approx N$, but it will become evident below that $mt^2s^2 \approx p$ is more appropriate for our purposes. A distinguishing property is set so that $\frac{1}{t}$ of the search space elements are designated as DPs, and s-many reduction functions $r_i : G \to G$ are fixed so that the corresponding colored iteration functions $f_i = r_i \circ f$ are defined.

The pre-computation chains are first generated from m elements of G, where the DP method of Section 2 is used to determine the end of each chain, except that f_1 is used in place of f. The ending points of this DP matrix are taken as the starting points for a second DP matrix that is computed using f_2 as the iteration function. This is continued until s DP matrices of different colors have been generated. The ending point of the final DP matrix, together with the corresponding discrete logarithm values, are stored as the pre-computation table. One can infer from the arguments of [5] that our choice of $mt^2s^2 \approx p$ makes merges among pre-computation chains uncommon, so that the storage complexity M will be very close to m, even after removal of duplicate entries.

The online phase consists of s stages. In the 1-st stage, an online DP chain is created using the f_s iteration function, starting from the DLP instance h, and the terminal point is searched for in the pre-computation table. In the

2-nd stage, a DP chain is first created using f_{s-1} and its ending point is used to extend the first DP chain to a second DP chain that uses f_s as the iteration function. The terminal point of the second DP chain is searched for in the pre-computation table. The general case should now be clear. The completion of the t stages is expected to require approximately $\frac{s(s+1)}{2}t$ applications of the iteration function.

Let us next explain that we are assured of a reasonable probability of merge between one of the online chains and the fuzzy rainbow matrix, so that the given DLP instance is likely to be solved within the s stages. Consider a single DP chain segment contained in some DP sub-matrix of the fuzzy rainbow matrix created with a specific f_i. Since the expected length of a DP chain is t, assuming f_i to be a random function, an online DP chain segment created with f_i will merge into this DP chain segment with probability $1 - \left(1 - \frac{t}{p}\right)^t \approx \frac{t^2}{p}$. An extension of this argument implies that the online chain created at the k-th stage will merge into any one chain in the fuzzy rainbow matrix with probability $1 - \left(1 - \frac{t}{p}\right)^{kt} \approx \frac{kt^2}{p}$. (A similar argument appears within the proof to Lemma 3 of [19].) The full set of s stages is expected to create $\sum_{k=1}^{s} m\frac{kt^2}{p} = \frac{mt^2s(s+1)}{2p}$ merges with the fuzzy rainbow matrix. Hence, our choice of $mt^2s^2 \approx p$ ensures a non-negligible probability of merge between chains.

We can now summarize the performance of the fuzzy rainbow matrix version of the DLP algorithm. When realized with parameters m, t, and s such that $mt^2s^2 \approx p$, the algorithm displays pre-computation, storage, and online time complexities of $P = mts$, $M \approx m$, and $T \approx t\frac{s(s+1)}{2}$, respectively, ignoring small multiplicative factors. It is easy to check that these complexities satisfy the tradeoff curve as given by (5), for each fixed choice of s.

Appendix C. Thin and thick rainbow matrices

This section discusses the DLP algorithms that are based on the thin and thick fuzzy rainbow matrix structures. Results of our rudimentary analyses are provided below without any detailed arguments.

The thin rainbow TMTO method uses chains of the form

$$\xrightarrow{f_1}\xrightarrow{f_2}\cdots\xrightarrow{f_s}\ \xrightarrow{f_1}\xrightarrow{f_2}\cdots\xrightarrow{f_s}\ \cdots\cdots\ \xrightarrow{f_1}\xrightarrow{f_2}\cdots\xrightarrow{f_s},$$

where the full set of colors 1 through s are repeated t times. Its natural pre-computation DLP adaptation require parameters such that $mt^2s^2 \approx p$. This is expected to display performance $P = mts$, $M \approx m$, and $T \approx ts^2$, which satisfy the relations

$$PT \approx sp \quad \text{and} \quad T^2M \approx s^2p. \tag{11}$$

This is quite similar to the fuzzy rainbow matrix curve (5).

The thick rainbow TMTO method uses chains of the form

$$\xrightarrow{f_1}\xrightarrow{f_1}\cdots\xrightarrow{f_1}\ \xrightarrow{f_2}\xrightarrow{f_2}\cdots\xrightarrow{f_2}\ \cdots\cdots\ \xrightarrow{f_s}\xrightarrow{f_s}\cdots\xrightarrow{f_s},$$

where each colored iteration function f_i is iterated t times. Its natural pre-computation DLP adaptation requires parameters such that $mt^2s^2 \approx p$. This is expected to display performance $P = mts$, $M \approx m$, and $T \approx t^2s^2$, which satisfy the relations

$$P\sqrt{T} \approx p \quad \text{and} \quad TM \approx p. \tag{12}$$

This is identical to the rainbow matrix curve (4), except that the constant factors hidden behind the approximation symbols can be different.

Acknowledgments. Part of this work was done while both authors were visiting Department of Mathematics, U. C. Davis. We are grateful to them for their hospitality.

References

[1] *Cryptohaze*, GPU Rainbow Cracker; https://www.cryptohaze.com

[2] *Free Rainbow Tables*, Distributed Rainbow Table Project; http://freerainbowtables.com

[3] *Ophcrack*, Windows Password Cracker; http://ophcrack.sourceforge.net

[4] *RainbowCrack Project*, http://project-rainbowcrack.com

[5] E. P. Barkan, *Cryptanalysis of ciphers and protocols*, Ph.D. Thesis, Technion—Israel Institute of Technology, March 2006.

[6] E. Barkan, E. Biham, and A. Shamir, *Rigorous bounds on cryptanalytic time/memory tradeoffs*, in: CRYPTO 2006, pp. 1–21, LNCS 4117, Springer, 2006.

[7] D. J. Bernstein and T. Lange, *Computing small discrete logarithms faster*, in: INDOCRYPT 2012, pp. 317–338, LNCS 7668, Springer, 2012.

[8] ———, *Non-uniform cracks in the concrete: the power of free precomputation*, in: ASIACRYPT 2013, pp. 321–340, LNCS 8270, Springer, 2013.

[9] A. Biryukov, S. Mukhopadhyay, and P. Sarkar, *Improved time-memory trade-offs with multiple data*, in: SAC 2005, pp. 110–127, LNCS 3897, Springer, 2006.

[10] A. Biryukov and A. Shamir, *Cryptanalytic time/memory/data tradeoffs for stream ciphers*, in: ASIACRYPT 2000, pp. 1–13, LNCS 1976, Springer, 2000.

[11] C. Calik, *How to invert one-way functions: time-memory trade-off method*, M.S. Thesis, Middle East Technical University, January 2007.

[12] A. E. Escott, J. C. Sager, A. P. L. Selkirk, and D. Tsapakidis, *Attacking elliptic curve cryptosystems using the parallel Pollard rho method*, CryptoBytes **4** (1999), 15–19.

[13] M. E. Hellman, *A cryptanalytic time-memory trade-off*, IEEE Trans. Inform. Theory **26** (1980), no. 4, 401–406.

[14] R. Henry, K. Henry, and I. Goldberg, *Making a nymbler Nymble using VERBS*, in: PETS 2010, pp. 111–129, LNCS 6205, Springer, 2010.

[15] 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*, Int. J. Inf. Secur. **3** (2004), 86–98.

[16] J. Hong, *The cost of false alarms in Hellman and rainbow tradeoffs*, Des. Codes Cryptogr. **57** (2010), no. 3, 293–327.

[17] J. Hong and S. Moon, *A comparison of cryptanalytic tradeoff algorithms*, J. Cryptology **26** (2013), no. 4, 559–637.

[18] D. Huhnlein, M. J. Jacobson Jr., and D. Weber, *Towards practical non-interactive public-key cryptosystems using non-maximal imaginary quadratic orders*, Des. Codes Cryptogr. **39** (2003), no. 3, 281–299.

[19] B.-I. Kim and J. Hong, *Analysis of the non-perfect table fuzzy rainbow tradeoff*, IACR Cryptology ePrint Archive, Report 2012/612, version 20121116:123317; http://eprint.iacr.org/2012/612.

[20] ———, *Analysis of the non-perfect table fuzzy rainbow tradeoff*, in: ACISP 2013, pp. 347–362, LNCS 7959, Springer, 2013.

[21] ———, *Analysis of the perfect table fuzzy rainbow tradeoff*, J. Appl. Math. **2014** (2014), Article ID 765394, 19 pages.

[22] F. Kuhn and R. Struik, *Random walks revisited: extensions of Pollard's rho algorithm for computing multiple discrete logarithms*, in: SAC 2001, pp. 212–229, LNCS 2259, Springer, 2001.

[23] H. T. Lee, J. H. Cheon, and J. Hong, *Accelerating ID-based encryption based on trapdoor DL using pre-computation*, IACR Cryptology ePrint Archive, Report 2011/187, version 20120112:021951; http://eprint.iacr.org/2011/187.

[24] G. W. Lee and J. Hong, *A comparison of perfect table cryptanalytic tradeoff algorithms*, IACR Cryptology ePrint Archive, Report 2012/540, version 20140622:150618; http://eprint.iacr.org/2012/540.

[25] D. Ma and J. Hong, *Success probability of the Hellman trade-off*, Inform. Process. Lett. **109** (2009), no. 7, 347–351.

[26] U. M. Maurer and Y. Yacobi, *Non-interactive public-key cryptography*, in: EUROCRYPT '91, pp. 498–507, LNCS 547, Springer, 1991.

[27] ———, *A non-interactive public-key distribution system*, Des. Codes Cryptogr. **9** (1996), no. 3, 305–316.

[28] Y. Murakami and M. Kasahara, *A discrete logarithm problem over composite modulus*, Electronics and Communications in Japan (Part III) **76** (1993), 37–46.

[29] K. Nohl, *Attacking phone privacy*, presented at Black Hat USA 2010, Las Vegas, July 2010.

[30] K. Nohl and C. Paget, *GSM-SRSLY?*, presented at 26th Chaos Communication Congress (26C3), Berlin, December 2009.

[31] P. Oechslin, *Making a faster cryptanalytic time-memory trade-off*, in: CRYPTO 2003, pp. 617–630, LNCS 2729, Springer, 2003.

[32] K. G. Paterson and S. Srinivasan, *On the relations between non-interactive key distribution, identity-based encryption and trapdoor discrete log groups*, Des. Codes Cryptogr. **52** (2009), no. 2, 219–241.

[33] S. C. Pohlig and M. E. Hellman, *An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance*, IEEE Trans. Inform. Theory **24** (1978), no. 1, 106–110.

[34] J. M. Pollard, *Monte Carlo methods for index computation* (mod p), Math. Comp. **32** (1978), no. 143, 918–924.

[35] C. P. Schnorr and H. W. Lenstra Jr., *A Monte Carlo factoring algorithm with linear storage*, Math. Comp. **43** (1984), no. 167, 289–311.

[36] D. Shanks, *Class number, a theory of factorization and genera*, 1969 Number Theory Institute (Proc. Sympos. Pure Math., Vol. XX, State Univ. New York, Stony Brook, N.Y., 1969), pp. 415–440. Amer. Math. Soc., Providence, R.I., 1971.

[37] V. Shoup, *Lower bounds for discrete logarithms and related problems*, in: EUROCRYPT '97, pp. 256–266, LNCS 1223, Springer, 1997.

[38] E. Teske, *Speeding up Pollard's rho method for computing discrete logarithms*, in: ANTS-III, pp. 541–554, LNCS 1423, Springer, 1998.

[39] ———, *An elliptic curve trapdoor system*, J. Cryptology **19** (2006), no. 1, 115–133.

Jin Hong
Department of Mathematical Sciences and ISaC
Seoul National University
Seoul 151-747, Korea
E-mail address: jinhong@snu.ac.kr

Hyeonmi Lee
Department of Mathematics and Research Institute for Natural Sciences
Hanyang University
Seoul 133-791, Korea
E-mail address: hyeonmi@hanyang.ac.kr