Accelerating Pollard's Rho Algorithm on Finite Fields

Jung Hee Cheon · Jin Hong · Minkyu Kim

the date of receipt and acceptance should be inserted later

Abstract Most generic and memory-efficient algorithms for solving the discrete logarithm problem construct a certain random graph consisting of group element nodes and return the solution when a collision is found among the graph nodes.

In this work, we develop a technique for traveling through the random graph without fully computing each node and also provide an extension to the distinguished point collision detection method that is suitable for this new situation. Concrete constructions of this technique for multiplicative subgroups of the finite fields are given. Our implementations confirm that the proposed technique provides practical speedup over existing algorithms.

Keywords discrete logarithm problem · Pollard's rho · r-adding walk · distinguished point · finite field

1 Introduction

Let G be a finite cyclic group of order q generated by an element $\mathbf{g}$. Given any $\mathbf{h} \in G$, we know that it is possible to write $\mathbf{h} = \mathbf{g}^x$. The smallest such non-negative integer x is said to be the discrete logarithm of $\mathbf{h}$ to the base $\mathbf{g}$ and is denoted by $\log_{\mathbf{g}} \mathbf{h}$. Ever since the introduction of public key cryptography [6], the discrete logarithm problem (DLP) over various constructions of the cyclic group has been one of the most important mathematical primitives in the subject area. The security of various current cryptosystems, such as the Diffie-Hellman key agreement protocol [6], the ElGamal cryptosystem [8], and some signature schemes [8, 18], rely on the hardness of DLPs.

The first non-trivial algorithm for solving the DLP was the baby-step giant-step (BSGS) method [29]. The BSGS algorithm requires $O(\sqrt{q})$ operations and $O(\sqrt{q})$ storage when

An earlier version of this work appeared in [4].

Corresponding author: Jin Hong

This work was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (JHC,MK:2010-0000218,2010-0001655; JH:2010-0004561).

ISaC and Department of Mathematical Sciences, Seoul National University, Seoul 151-747, Korea
E-mail: {jhcheon,jinhong,minkyu97}@snu.ac.kr

applied to a cyclic group of order q. Pollard [22, 23] proposed two probabilistic algorithms called Pollard's rho method and the kangaroo method. Pollard's rho method has the same time complexity as BSGS but requires almost no storage. The kangaroo method is well suited for the case where $\log_{\mathbf{g}} \mathbf{h}$ is known to belong to an interval of length smaller than the group size. Its complexity in solving DLPs is $O(\sqrt{\omega})$ when the interval length is ω.

If the considered group has a composite order, the Pohlig-Hellman algorithm [21] can be used to reduce the initial DLP to DLPs in its prime order subgroups. Hence, it is sufficient to consider DLPs in prime order groups. The index calculus method [1] is a DLP solving algorithm that works on multiplicative subgroups of the finite field $\mathbf{F}_{p^n}$. The complexity of this algorithm is sub-exponential in the size of the base field, and the algorithm delivers the same performance on any subgroup of a fixed $\mathbf{F}_{p^n}^{\times}$. Nevertheless, in many situations, including the multiplicative subgroups of sufficiently large finite fields, variants of Pollard's rho method are the best known algorithms for solving the DLPs. Unlike the index calculus method, these algorithms are generic in the sense that they are not specific to the presentation of the group. These algorithms can be applied even to generic groups, where the group operation and equality testing are given as black-box algorithms. Other developments concerning DLP solving include parallelization [20] of the various algorithms and Pollard's rho variants specific to certain groups with fast endomorphisms [7, 10, 12, 34].

Pollard's rho algorithm and its variants generate a sequence of elements from the group, referred to as the *random walk*. More precisely, a specific function $F : G \to G$ is fixed and the random walk is generated by iteratively applying F, starting from a random group element. The function F is constructed in such a way that the solution to the DLP is obtained when the random walk revisits an element it has already passed over. Each variant of Pollard's rho algorithm provides a means of detecting this *collision* within the walk.

A popular design for the iteration function is the *r-adding*, in which each iteration is a product by one of r-many preset elements of the group. Let us focus our attention on the DLP solving algorithm that combines the r-adding style of generating the random walk with the *distinguished point* collision detection method. This combination may be considered the most practical among generic DLP solving algorithms. We bring two properties of this combination to attention. The first is that the choice of which element is to be multiplied at each iteration of the r-adding walk is essentially determined from a very small amount of information concerning the current point. The second is that the distinguished point method processes only a very small fraction of the random walk nodes in full.

In this paper, we present *tag tracing*, a technique that takes advantage of the above mentioned two properties. Although the random walk is iterative in nature, we provide a framework within which one may quickly pass through a certain length of iterations without fully computing each node on the walk. If the r-adding walk iteration function can be designed to meet certain criteria, one can use a pre-computed table and trace through each iteration of the random walk after computing only the aforementioned very small amount of information that determines the subsequent step. We also provide an extension to the distinguished point method that is suitable as a collision detection method for the modified r-adding walk. Tag tracing fully retains the parallelizability of the original algorithm.

Concrete constructions of tag tracing for subgroups of the finite fields are provided. Both the prime fields and the finite fields of large extension degrees are considered. Under a fixed encoding for the group, the core of these constructions amounts to being capable of computing just a few bits of the product of two group elements, where one element is random and the other is fixed, with a lesser effort than is required for computing the full product. These partial product computation methods may be seen as another contribution of this work.

In the case of multiplicative subgroups of $\mathbf{F}_{p^n}^{\times}$, as n is increased and the prime p is kept fixed, the tag tracing construction allows each iteration of the r-adding walk to be computed with time complexity that is asymptotically linear in n. For subgroups of fixed size, this implies that the DLP solving time is linear in n, or the bit size of the base field. In the case of multiplicative subgroups of the prime field $\mathbf{F}_p$, the time complexity achieved by tag tracing depends on the method of modulo product computation used. When the classical method of product computation is used, tag tracing iterations run at an asymptotic time complexity of $O(\log p \log\log p)$ as p is changed, and if the Fast Fourier Transformation method is used for product computations, the complexity becomes $O(\log p \log\log\log p)$.

In practice, our rudimentary implementation runs more than 10, 20, and 30 times faster than the original algorithms in solving DLPs on subgroups of $\mathbf{F}_p^{\times}$, when the bit length of p is 1024, 2048, and 3072, respectively. Our implementation on subgroups of $\mathbf{F}_{2^n}^{\times}$ performs approximately 13, 17, and 21 times faster when n is 1024, 2048, and 3072, respectively. Tag tracing slightly increases the total number of iterations, but the speedups are achieved by computing each iteration faster with the help of large pre-computed tables. The stated speedups do not include the time spent on table pre-computation, however this time is negligible compared to the online DLP solving time when working with large groups.

Organization. The rest of this paper is organized as follows. Section 2 includes a review of Pollard's rho algorithm and its variants. In Section 3, we present an extension of the distinguished point collision detection method. The main contribution of this paper, tag tracing, is presented in Section 4. The subsequent two sections show concrete constructions of tag tracing on multiplicative subgroups of the finite fields. Section 5 discusses the case of finite fields with large extension degrees, and Section 6 discusses the prime field case. Both of these sections also include implementation results. The final section summarizes the results of the present study and discusses future research.

2 Pollard's Rho Algorithm

There are many variants of Pollard's rho algorithm. In this section, we briefly review some of these algorithms that are used to solve DLPs. Readers should consult the original cited papers for further details. Throughout this paper $G = \langle \mathbf{g} \rangle$ denotes a finite cyclic group of prime order q generated by an element $\mathbf{g}$, and the notation $\mathbf{h} \in G$ is used for the DLP target. In other words, we are always looking for the value $\log_{\mathbf{g}} \mathbf{h}$. Hereinafter, Pollard's rho algorithm is referred to simply as the *rho algorithm*.

2.1 Overview

Each variant of the rho algorithm generates a sequence of elements belonging to the group under consideration and produces the solution to the DLP when a collision among these elements is found.

Given any function $F : G \to G$ and an initial element $\mathbf{g}_0 \in G$, we can iteratively define

$$\mathbf{g}_{i+1} = F(\mathbf{g}_i) \quad (i \geq 0)$$

to create a sequence $(\mathbf{g}_i)_{i \geq 0}$. Since G is a finite set, the resulting sequence must eventually return to an element that has previously been visited, after which the sequence repeats the path already traveled. The smallest integers $\mu \geq 0$ and $\lambda \geq 1$ that satisfy $\mathbf{g}_{\lambda+\mu} = \mathbf{g}_\mu$ are said to be the pre-period and period, respectively, of the *random walk* $(\mathbf{g}_i)_{i\geq 0}$. When the function F is chosen uniformly at random from the set of all functions operating on G, the rho length $\lambda + \mu$ is expected [9, 16] to be $\sqrt{\frac{\pi}{2}q} \approx 1.253\sqrt{q}$.

Let us say that the *iteration function* $F : G \to G$ is *exponent traceable* with respect to **g** and **h**, if it can be expressed in the form

$$F(\mathbf{g}^a\mathbf{h}^b) = \mathbf{g}^{F_\mathbf{g}(a,b)}\mathbf{h}^{F_\mathbf{h}(a,b)}$$

for some functions $F_\mathbf{g}$ and $F_\mathbf{h}$ of the input exponents that are easy to compute. In other words, if one knows how to express the input to F as a power of **g** and **h**, we require the same to be possible for its output. This does not necessarily imply that F can only be computed through the exponent information. For example, if F is the squaring operator on G, we can take $F_\mathbf{g}(a,b) = 2a$ and $F_\mathbf{h}(a,b) = 2b$, but F can be computed without access to the $\mathbf{g}^a\mathbf{h}^b$ form of the input.

Each variant of the rho algorithm employs an iteration function that is exponent traceable. Thus, starting from $\mathbf{g}_0 = \mathbf{g}^{a_0}\mathbf{h}^{b_0}$, with randomly chosen, but known, (a_0, b_0), one can always keep track of the exponents (a_i, b_i) that satisfy $\mathbf{g}_i = \mathbf{g}^{a_i}\mathbf{h}^{b_i}$. The cost of updating the exponents is usually much smaller than the cost of one function iteration and is disregarded during complexity analysis.

When a collision $\mathbf{g}_i = \mathbf{g}_j$ within the random walk is detected for some $i \neq j$, setting $x = \log_\mathbf{g} \mathbf{h}$, we know $\mathbf{g}^{a_i}(\mathbf{g}^x)^{b_i} = \mathbf{g}^{a_j}(\mathbf{g}^x)^{b_j}$, so that we can use

$$a_i + x\, b_i \equiv a_j + x\, b_j \pmod{q}$$

to solve for x. This will fail to bring out x when $b_i \equiv b_j \pmod{q}$, but such an event is rare for any large q and reasonable F.

Below, we describe the exponent traceable iteration functions that are used in variants of the rho algorithm and we also provide a number of *collision detection* mechanisms. These two components of DLP solving algorithms are rather independent, and any combination of the two parts that are given in this section can be used.

2.2 Iteration Functions

An iteration function is considered to be well designed if it is easy to compute and the number of iterations it takes to reach a collision is close to $\sqrt{\frac{\pi}{2}q}$, the value expected of a random function.

Pollard. Pollard [22] originally targeted his algorithm to solve DLPs on $\mathbf{F}_p^\times$, but his iteration function, which we denote by $F_P : G \to G$, can be modified for use on any cyclic group. Let $G = G_0 \cup G_1 \cup G_2$ be a partition of G into three nearly equal sized subsets. It is assumed that the encoding for G and the partition are compatible, in the sense that it is easy to identify which subset G_i a given group element belongs to. With the partition fixed, the iteration function is defined as follows.

$$F_P(\mathbf{y}) = \begin{cases} \mathbf{g}\mathbf{y}, & \text{if } \mathbf{y} \in G_0, \\ \mathbf{y}^2, & \text{if } \mathbf{y} \in G_1, \\ \mathbf{h}\mathbf{y}, & \text{if } \mathbf{y} \in G_2. \end{cases}$$

It is clear that this function is exponent traceable. For example, when $\mathbf{g}^a\mathbf{h}^b \in G_0$, we have $(F_P)_\mathbf{g}(a,b) = a+1$ and $(F_P)_\mathbf{h}(a,b) = b$. Although the expected rho length of F_P has recently been shown [13] to be $O(\sqrt{q})$, tests indicate that it takes more than $\sqrt{\frac{\pi}{2}q}$ iterations for F_P to reach a collision [31, 32], so that F_P is not an optimal iteration function.

Adding walks. Let $3 \le r \le 100$ be a small positive integer. As before, we fix a partition $G = G_0 \cup \cdots \cup G_{r-1}$ of G into r-many subsets of roughly the same size. The index function $s : G \to \{0,1,\ldots,r-1\}$ is defined by setting $s(\mathbf{y}) = i$ for $\mathbf{y} \in G_i$ and is assumed to be easy to compute. For each $i = 0,\ldots,r-1$, randomly choose integers $\alpha_i, \beta_i \in \mathbf{Z}/q\mathbf{Z}$ that are not both zero and set the multipliers $\mathbf{m}_i$ to $\mathbf{g}^{\alpha_i}\mathbf{h}^{\beta_i}$. Then the *r-adding* iteration function $F_r : G \to G$ is given by

$$F_r(\mathbf{y}) = \mathbf{y}\mathbf{m}_{s(\mathbf{y})}.$$

That is, one of the r-many randomly chosen, but pre-determined, elements $\mathbf{m}_i \in G$ is multiplied to the input, depending on which subset G_i it belongs to, so that the *r-adding walk* is given by

$$\mathbf{g}_{i+1} = \mathbf{g}_i\mathbf{m}_{s(\mathbf{g}_i)}.$$

This function is clearly exponent traceable, with the exponent functions $F_\mathbf{g}$ and $F_\mathbf{h}$ being additions by α_i and β_i, respectively. In this paper, the term *adding walk* is used, unless reference to the parameter r is required.

This method was introduced in [26] and the work [25] provides some support to the claim that a collision is expected after $O(\sqrt{q})$ iterations when $r \ge 8$. Testing [32] done on prime order elliptic curve groups show that 20-adding walks perform very close to a random function.

2.3 Collision Detection

The most straightforward approach to finding a collision within the random walk is to collect every point $\mathbf{g}_i$ that has been generated, in a sorted list until a collision is found. This method, however, provides no benefit over the BSGS algorithm, which is a deterministic DLP solving algorithm that requires $O(\sqrt{q})$ group operations and $O(\sqrt{q})$ storage. Thus, collision detection mechanisms with smaller storage requirements are desirable. In achieving this, most methods require additional function iterations to be computed even after the actual collision has occurred. It would be preferable for a collision detection mechanism to keep this additional cost at a minimum.

Below, we describe the three most cited collision detection methods. Though it would be difficult to devise a measure that would give a fair comparison of these methods, the distinguished point method, which will be described last, is usually regarded as being the most practical, and all of our further discussions will be based on the distinguished point method.

There are other methods that we do not explain in detail. The method in [28] proposes the storage of points that are equidistance from one another with occasional search of an entire random walk interval, whose length equals the mentioned distance, for collisions with the stored points. The distance between stored points is adaptively changed so that the storage size stays within a given bound. The proposal given by [17] is to maintain a stack of walk points in increasing order. All points in the stack that are smaller than the newest walk point are removed before the latter is added to the top of the stack. A collision is detected when the walk visits the smallest point on the cycle for the second time. This process is combined with a partitioning technique for performance enhancement.

Floyd. The idea given in [14] is to wait for a collision of the type $\mathbf{g}_i = \mathbf{g}_{2i}$ to occur. Three applications of the iteration function are needed at each iteration to update the two current states $\mathbf{g}_i$ and $\mathbf{g}_{2i}$ to $\mathbf{g}_{i+1}$ and $\mathbf{g}_{2(i+1)}$, respectively. This method detects a collision within $\lambda + \mu$ iterations, or equivalently, $3(\lambda + \mu)$ applications of the iteration function. Therefore, with a good iteration function, approximately $3\sqrt{\frac{\pi}{2}q}$ function applications are needed before a collision is detected, and the storage requirement is negligible.

Brent. We explain a method of Teske [31], which is an optimized version of the method proposed in [3, 26]. The eight most recent $\mathbf{g}_k$, for which the index k are powers of 3, are kept in storage. After each application of the iteration function, the current $\mathbf{g}_i$ is compared with the eight stored entries, and $\mathbf{g}_i$ replaces the oldest of the eight entries if the index i has reached a new power of 3. This approach terminates with a collision between the current $\mathbf{g}_i$ and some $\mathbf{g}_k$ from the storage in $1.25 \cdot \max(\lambda/2, \mu) + \lambda$ iterations. This is approximately $1.229\sqrt{\frac{\pi}{2}q}$ iteration when the iteration function is the random function.

Distinguished points. This approach was originally proposed for use with the time-memory tradeoff techniques and its use in DLP solving was suggested in [24]. A distinguished point is any element of G that satisfies a certain preset condition. This condition is usually set in such a way that it is easy to check. For example, under a fixed encoding for G, one may set distinguished points to be those elements of G with a certain number of most significant bits equal to zero.

We start with an empty table of distinguished points and the random walk is traveled until the current element $\mathbf{g}_i$ is found to be a distinguished point. Whenever a distinguished point is reached through the iteration, the element is searched for in the table of distinguished points and is added to the table if it is not found. The table is always kept in a sorted state, or a hash table technique is employed, in order to make later searches easy.

If $\frac{1}{\Delta}$ is the probability for a random element of G to be a distinguished point, then the distinguished point collision detection method will consume, on average, Δ extra applications of the iteration function after the initial collision within the walk, before terminating with a collision among the distinguished points. Thus, for a random iteration function, one would expect to compute $\sqrt{\frac{\pi}{2}q} + \Delta$ applications of the iteration function until collision detection.

To minimize the Δ extra iterations, one would like to define distinguished points in such a way that they occur frequently. This desire should be balanced with the need to keep the table of distinguished points within a manageable size. In addition to causing only a small amount of iteration overhead, the distinguished point method has the advantage of allowing parallelization [20] that achieves speedups that are linear in the number processors.

3 Extension to the Distinguished Point Collision Detection Method

There are two aspects of the distinguished point method that give it an advantage over other collision detection methods. The first is that only a small fraction of the points on the random walk needs to be processed for collision detection. The second is that the points that do go through the processing are already similar to each other. One may view these properties as having been obtained by sieving out the group elements with a test that is extremely easy to perform.

In the sections below, we encounter random walks $(\mathbf{g}_i)_{i\geq 0}$ for which only partial information about each $\mathbf{g}_i$ is available. The accessible information is typically of only a few bits, so that it is not possible to devise a test that sieves out a large fraction of the random walk points. We now present an adaptation of the distinguished point method that is suitable for this situation. While it is not possible to check for collisions reliably without full access to each $\mathbf{g}_i$, we shall show how the information present in the walk, rather than each point, can be used to construct a meaningful sieve.

3.1 Distinguished Path Segment

Let us be given a random walk $(\mathbf{g}_i)_{i\geq 0}$, constructed with a certain iteration function. Suppose that it is possible to check each $\mathbf{g}_i$ for a property that is satisfied with probability $\frac{1}{r}$, for some small r. We will not be concerned with how to compute the walk and will completely ignore any information about $\mathbf{g}_i$ that comes from its encoding. The typical situation we have in mind is the r-adding walk with the property of the index being zero, except that someone else is doing the function iterations for us. More formally, we consider an index function $s : G \to \{0,1\}$ that maps $\frac{1}{r}$ of the elements of G to 0 and sends all other elements of G to 1, and we suppose that we are given access to only $(s(\mathbf{g}_i))_{i\geq 0}$ and not to each $\mathbf{g}_i$ or even to the iteration function.

Since r is small, the property $s(\mathbf{g}_i) = 0$ cannot serve as a reasonable distinguishing property. To overcome this problem, we fix a small positive integer δ and define a point $\mathbf{g}_i$ to be a distinguished point if and only if

$$s(\mathbf{g}_i) = s(\mathbf{g}_{i+1}) = \cdots = s(\mathbf{g}_{i+\delta-1}) = 0. \tag{1}$$

This may not seem to be a property of the point $\mathbf{g}_i$, as properties of its future points are being used. However, as long as the iteration function is fixed to one deterministic function, since future points are determined by $\mathbf{g}_i$, all future index values $s(\mathbf{g}_{i+j})$ are, in fact, completely determined by $\mathbf{g}_i$. Thus, this is a well-defined distinguishing property for the point $\mathbf{g}_i$.

A closer look at this distinguishing property reveals that it depends both on the group element $\mathbf{g}_i$ and on the iteration function. Hence, the term distinguished point may be slightly misleading. It would be more appropriate to say that the element $\mathbf{g}_i$, which satisfies (1), is the starting point of a distinguished path segment. Still, as long as we deal with a single fixed iteration function, this type of distinguishing property can be used as a reasonable sieve in the situation where access to information concerning each $\mathbf{g}_i$ is very limited.

One inconvenience experienced when sieving with condition (1), which utilizes future points, is that the distinguishing property of a point is confirmed only after one has passed $\delta - 1$ iterations beyond the point. It would be better if data concerning previous points are utilized to define a distinguishing property. Hence, let us consider specifying the last point of a distinguished path segment as a distinguished point. That is, we fix a small positive integer δ and define a point $\mathbf{g}_i$ to be a distinguished point if and only if

$$s(\mathbf{g}_{i-\delta+1}) = \cdots = s(\mathbf{g}_{i-1}) = s(\mathbf{g}_i) = 0. \tag{2}$$

Unlike the distinguishing property discussed before, whether or not a point $\mathbf{g}_i \in G$ satisfies this property is not completely determined by the point, even under a fixed iteration function. It may happen that $\mathbf{g}_i = \mathbf{g}_j$, as elements of the group G, but that the above property is satisfied by the points leading to $\mathbf{g}_i$ and not by the points leading to $\mathbf{g}_j$. Thus, condition (2) does not present a well-defined distinguishing property for the group element $\mathbf{g}_i$.

Nevertheless, as long as our aim is to sieve points before checking for collisions, the above mentioned definition is sufficiently useful. Let us recall the notation for the pre-period and period of an iterated walk. With the second approach, the points that appear between $\mathbf{g}_{\lambda+\mu} = \mathbf{g}_\mu$ and $\mathbf{g}_{\lambda+\mu+\delta-2} = \mathbf{g}_{\mu+\delta-2}$, both inclusive, may be sieved differently, depending on the path segments that lead to them. However, the property is well-defined for all points appearing from $\mathbf{g}_{\lambda+\mu+\delta-1} = \mathbf{g}_{\mu+\delta-1}$ onwards. The situation with the parallelized versions of the DLP solving algorithms may seem slightly different, as they do not exhibit a single loop, but the above distinguishing property is well-defined for all the points that appear at least $\delta - 1$ iterations away, in the forward direction, from each point of collision. Furthermore, we show below that the average distance between distinguished path segments is of exponential order in δ, so that the existence of such short segments for which the definition of the distinguishing property is not very accurate will have little effect on the performance of collision detection.

In the aforementioned two approaches for defining a distinguished point, the same definition of a distinguished path segment was used. The only difference between these two approaches was in whether the first or last entry of the distinguished path segment is designated as the distinguished point. It is clear that when adopting the *distinguished path segment* approach in defining a distinguished point, any position within the distinguished path segment may be set as the distinguished point, as long as this relative position is fixed throughout the DLP solving process. In the rest of this paper, when using the distinguished path segment approach, we use condition (2) and take the final point to be the distinguished point.

We state that for collision detection, it is not sufficient to define a distinguished path segment without specifying the position of the distinguished point. Partial information about the elements belonging to path segments cannot provide reliable collision detection, unless the length of the observed path segment is set to a value that is considerably larger than that required for sieving.

Before moving on to some technical details, let us briefly discuss how the idea of the distinguished path segment approach may be combined with any collision detection method. Such a combination may be interesting in relation to the recent method [17] that uses stacks and appears to be efficient.

Collision detection methods are usually described in terms of the nodes among which we look for collisions, but it is sufficient to have access to any *identification information* that uniquely identifies each node, and this identification information need not be the full encoding of each node. For example, in the case of DLP solving, we expect to process approximately $\sqrt{q}$ nodes. Hence, in view of the birthday paradox, if we have a little more than $\log q$ bits of information concerning each node, we can safely distinguish between the nodes. Using a sufficiently large δ, one may safely distinguish between the nodes using the $s(\mathbf{g}_k)$ values leading to each node.

3.2 Distance Between Distinguished Path Segments

In the previous subsection, we saw that it was possible to define a distinguishing property on the basis of the information gathered from multiple points. When applying this new method of defining distinguished points, one should be able to predict and control the average iteration count between occurrences of distinguished points, so that the distinguished point table is maintained at a reasonable size. This average distance is also the number of additional function iterations required between the occurrence and detection of a node collision. Hence, we have another reason to analyze the average distance between distinguished points.

Let us now refine our objective slightly. Note that if a certain node $\mathbf{g}_i$ is a distinguished point in the path segment sense, then the next node $\mathbf{g}_{i+1}$ is more likely to be a distinguished point than a randomly chosen point within the walk would be. This shows that the distinguished points defined through the path segment approach are not uniformly distributed within the random walk and that they tend to group together. On the other hand, for collision detection, it is sufficient to choose and store just one node from each string of distinguished points, as long as the choice is made in a consistent manner throughout the DLP solving process. Hence, in relation to the distinguished point table size, we must compute the average distance between consecutive strings or groups of distinguished points, rather than the average distance between all individual distinguished points. In fact, retaining the notations r and δ from the previous subsection, since the probability of a random point appearing within the random walk being a distinguished point is $\frac{1}{r^\delta}$, we can say that the average distance between all distinguished points is r^δ. However, this is not the information we need.

We need to compute the average number of iterations required to reach the first of the next string of distinguished points. For this purpose, whenever $\mathbf{g}_i$ is found to be a distinguished point, we ignore the possibility of $\mathbf{g}_{i+1}$ being a distinguished point, by simply forgetting all the $s(\mathbf{g}_k)$ values associated with the distinguished point $\mathbf{g}_i$ and starting afresh. This interpretation of average distance is also suitable in relation to the number of extra iterations spent on collision detection. In summary, our goal is to find the length expectation up to the first appearance of a δ-run of zeros in a random sequence of elements from $\{0,1,\ldots,r-1\}$.

Let p_i denote the probability that a random sequence of length i contains no δ-run of zeros and ends with a nonzero entry. Consider any sequence of length longer than δ that contains no δ-run of zeros and ends with a nonzero entry. We focus our attention on the last two nonzero entries. There are less than δ zeros between these two nonzero entries, with the possibility of there being no zeros. Note that the second nonzero entry from the end will mark the end of a subsequence that contains no δ-run of zeros and ends with a nonzero entry. These considerations allow us to state that

$$p_{i+1} = \frac{r-1}{r}\cdot p_i + \frac{r-1}{r^2}\cdot p_{i-1} + \cdots + \frac{r-1}{r^\delta}\cdot p_{i-\delta+1},$$

which can be equivalently stated in matrix form as

$$\begin{pmatrix} p_{i+1} \\ p_i \\ p_{i-1} \\ \vdots \\ p_{i-\delta+3} \\ p_{i-\delta+2} \end{pmatrix} = \begin{pmatrix} \frac{r-1}{r} & \frac{r-1}{r^2} & & & & \frac{r-1}{r^\delta} \\ 1 & 0 & & & & 0 \\ 0 & 1 & 0 & & & \\ & & \ddots & \ddots & & \\ & & 0 & 1 & 0 & \\ & & & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_i \\ p_{i-1} \\ p_{i-2} \\ \vdots \\ p_{i-\delta+2} \\ p_{i-\delta+1} \end{pmatrix}.$$

After adopting the notation

$$\mathfrak{p}_i = \begin{pmatrix} p_{i+\delta-1} \\ p_{i+\delta-2} \\ \vdots \\ \vdots \\ p_{i+1} \\ p_i \end{pmatrix} \quad \text{and} \quad \mathbf{A} = \begin{pmatrix} \frac{r-1}{r} & \frac{r-1}{r^2} & & & & \frac{r-1}{r^\delta} \\ 1 & 0 & & & & 0 \\ 0 & 1 & 0 & & & \\ & & \ddots & \ddots & & \\ & & 0 & 1 & 0 & \\ & & & 0 & 1 & 0 \end{pmatrix},$$

and writing $\mathfrak{e}_j$ for the j-th coordinate unit vector, in row vector form, one can give the general term as

$$\mathfrak{p}_i = \mathbf{A}^i\,\mathfrak{p}_0 \quad \text{and} \quad p_i = \mathfrak{e}_\delta\,\mathbf{A}^i\,\mathfrak{p}_0.$$

It is easy to argue that the initial values are $p_0 = 1$ and $p_1 = \cdots = p_{\delta-1} = \frac{r-1}{r}$.

Now, note that a sequence reaches its first distinguished point at length i if and only if the last δ terms are zeros and these zeros are preceded by a subsequence of length $i-\delta$ that contains no δ-run of zeros and ends with a nonzero entry. Hence, the probability for a random sequence to reach its first distinguished point at length i is

$$q_i := p_{i-\delta} \cdot \frac{1}{r^\delta}.$$

Our goal of finding the expected number of iterations up to the first distinguished point comes down to computing

$$\sum_{i=\delta}^{\infty} i\,q_i = \frac{1}{r^\delta}\sum_{i=\delta}^{\infty} i\,p_{i-\delta} = \frac{1}{r^\delta}\sum_{i=0}^{\infty}(i+\delta)\,p_i = \frac{1}{r^\delta}\sum_{i=0}^{\infty}\mathfrak{e}_\delta\,(i+\delta)\,\mathbf{A}^i\,\mathfrak{p}_0. \tag{3}$$

Before continuing, we provide a technical lemma concerning the convergence of a certain sequence.

Lemma 1 *The sequence of vectors $(i\mathbf{A}^i\,\mathfrak{p}_0)_{i\geq 0}$ converges to the zero vector.*

Proof The probability for a finite sequence of length δ to be a distinguished point is $\frac{1}{r^\delta}$. Since the probabilities for non-overlapping segments of length δ to be distinguished points are independent, the expected length up to the appearance of the first distinguished point is bounded above by $\delta \cdot r^\delta$. That is, the sum of positive values $\sum_{i=\delta}^{\infty} i\,q_i$ is bounded above and we are assured of the convergence of this infinite sum. This implies, in particular, that $\lim_{i\to\infty} i\,q_i = 0$.

Now, for any $0 \leq j < \delta$, we have

$$\mathfrak{e}_{\delta-j}\lim_{i\to\infty} i\mathbf{A}^i\,\mathfrak{p}_0 = \lim_{i\to\infty} i\,\mathfrak{e}_{\delta-j}\,\mathfrak{p}_i = \lim_{i\to\infty} i\,p_{i+j} = r^\delta \lim_{i\to\infty} i\,\frac{p_{i+j}}{r^\delta} = r^\delta \lim_{i\to\infty} i\,q_{i+j+\delta} = 0.$$

This shows that the $(\delta-j)$-th component of $\lim_{i\to\infty} i\mathbf{A}^i\,\mathfrak{p}_0$ must be zero. Since the coverage of $0 \leq j < \delta$ shows all components to be zero, the proof is complete. □

The following theorem shows that the average distance between strings of distinguished points is expected to be slightly larger than r^δ.

Table 1 Number of random sequence of elements from $\{0,1,\ldots,r-1\}$ until the first δ-run of zeros is observed

r	δ	$\frac{r}{r-1}(r^\delta-1)$	experiment result
4	6	5.460×10^3	5.468×10^3
	7	2.184×10^4	2.181×10^4
	8	8.738×10^4	8.734×10^4
	9	3.495×10^5	3.494×10^5
	10	1.398×10^6	1.398×10^6
8	4	4.680×10^3	4.678×10^3
	5	3.744×10^4	3.751×10^4
	6	2.995×10^5	3.002×10^5
	7	2.396×10^6	2.376×10^6
	8	1.917×10^7	1.956×10^7
20	4	1.684×10^5	1.685×10^5
	5	3.368×10^6	3.365×10^6

Theorem 1 *Suppose one defines distinguished points by checking for whether δ consecutive points on the random walk all satisfy a condition of $\frac{1}{r}$ probability. When the iteration function is taken to be a random function, one can expect to compute $\frac{r}{r-1}(r^\delta-1)$ function iterations, until the appearance of the first distinguished point.*

Proof One can check by direct expansion that

$$(\mathbf{I}-\mathbf{A})^2\sum_{i=0}^{k-1}(i+\delta)\mathbf{A}^i=\mathbf{A}+\delta(\mathbf{I}-\mathbf{A})-(k+\delta)\mathbf{A}^k+(k+\delta-1)\mathbf{A}^{k+1}.$$

Hence the average distance given by (3) is equal to

$$\begin{aligned}&\frac{1}{r^\delta}\,\mathfrak{e}_\delta\,(\mathbf{I}-\mathbf{A})^{-2}\lim_{k\to\infty}\big(\mathbf{A}+\delta(\mathbf{I}-\mathbf{A})-(k+\delta)\mathbf{A}^k+(k+\delta-1)\mathbf{A}^{k+1}\big)\,\mathfrak{p}_0\\&\qquad=\frac{1}{r^\delta}\,\mathfrak{e}_\delta\,(\mathbf{I}-\mathbf{A})^{-2}\big(\mathbf{A}+\delta(\mathbf{I}-\mathbf{A})\big)\,\mathfrak{p}_0,\end{aligned}$$

where the equality follows from Lemma 1. It now suffices to substitute

$$(\mathbf{I}-\mathbf{A})^{-1}=\begin{pmatrix}r^\delta&\cdots&r^2&r\\r^\delta&\cdots&r^2&r\\\vdots&&\vdots&\vdots\\r^\delta&\cdots&r^2&r\end{pmatrix}-\begin{pmatrix}0&1&\cdots&1\\&\ddots&&\vdots\\&&0&1\\&&&0\end{pmatrix}\quad\text{and}\quad\mathfrak{p}_0=\begin{pmatrix}\frac{r-1}{r}\\\vdots\\\frac{r-1}{r}\\1\end{pmatrix}$$

into the equation and simplify to arrive at our claim. □

A test of this theorem is summarized in Table 1. Using the random number generator provided by the NTL library [30], a sequence of random elements from $\{0,\ldots,r-1\}$ was generated until a δ-run of zeros was reached. Each experiment entry in the table is an average of 2^{20} resulting sequence lengths.

4 Tag Tracing

In the DLP solving algorithm that combines the adding walk with the distinguished point collision detection method, the process of determining the index value $s_i = s(\mathbf{g}_i)$ and then computing $\mathbf{g}_{i+1} = \mathbf{g}_i\mathbf{m}_{s_i} \in G$ is continuously repeated, while the product $\mathbf{g}_i\mathbf{m}_{s_i}$ is only occasionally searched for and placed in a table of small size. Focusing on the fact that the storing operation does not take place very frequently, we question whether the full computation of the product $\mathbf{g}_i\mathbf{m}_{s_i}$ is absolutely necessary at every iteration.

The iterative nature of the random walk generation requires each current position $\mathbf{g}_i$ to be fully available, but it might be possible to bypass this need through a suitable use of a pre-computed table containing products of $\mathbf{m}_j$'s. It would then suffice to compute just the index $s_i = s(\mathbf{g}_i)$ at most of the iterations. We explore this line of reasoning in this section.

The general approach is explained in this section and more concrete constructions appropriate for subgroups of finite fields are given in later sections.

4.1 Overview

Tag tracing is a DLP solving algorithm that generates a random walk with the adding iteration function and performs collision detection with the distinguished point method. Neither of these components are new and what tag tracing provides is a method of executing each iteration of the random walk at a faster pace. This is achieved by utilizing a pre-computed table and an auxiliary function that computes each index $s_i = s(\mathbf{g}_i)$ without full access to $\mathbf{g}_i$.

Suppose we are given an index function $s : G \to \mathscr{S} = \{0, 1, \dots, r-1\}$ and a multiplier set $\mathscr{M} = \{\mathbf{m}_i\}_{i\in\mathscr{S}}$, suitable for an adding walk. Fix a small positive number ℓ and suppose that we have pre-computed all elements in the product set $\mathscr{M}_\ell = (\mathscr{M} \cup \{\mathbf{id}\})^\ell$, where $\mathbf{id}$ is the identity element of G. The set $\mathscr{M}_\ell$ consists of all products of at most ℓ-many multipliers. Let us further assume that we have an auxiliary index function $\bar{s} : G \times \mathscr{M}_\ell \to \mathscr{S}$ such that

$$\bar{s}(\mathbf{y}, \mathbf{m}) = s(\mathbf{y}\,\mathbf{m}),$$

for every $\mathbf{y} \in G$ and $\mathbf{m} \in \mathscr{M}_\ell$.

Then, given $\mathbf{g}_i \in G$, we may obtain $s_{i+1} = s(\mathbf{g}_{i+1}) = \bar{s}(\mathbf{g}_i, \mathbf{m}_{s_i})$ without computing the product $\mathbf{g}_{i+1} = \mathbf{g}_i\mathbf{m}_{s_i}$. Once the index value s_{i+1} is available, since $\mathbf{m}_{s_i}\mathbf{m}_{s_{i+1}} \in \mathscr{M}_\ell$ has already been pre-computed, we can also obtain $s_{i+2} = s(\mathbf{g}_{i+2}) = \bar{s}(\mathbf{g}_i, \mathbf{m}_{s_i}\mathbf{m}_{s_{i+1}})$ without computing the product $\mathbf{g}_{i+2} = \mathbf{g}_i\mathbf{m}_{s_i}\mathbf{m}_{s_{i+1}}$. We can continue this process for up to ℓ iterations before we are forced to compute $\mathbf{g}_{i+\ell} = \mathbf{g}_i\mathbf{m}_{s_i}\cdots\mathbf{m}_{s_{i+\ell-1}}$ in order to proceed any further. Note that this last computation is a single product between $\mathbf{g}_i$ and the pre-computed element $\mathbf{m}_{s_i}\cdots\mathbf{m}_{s_{i+\ell-1}} \in \mathscr{M}_\ell$.

The above process is advantageous over the original adding walk if the auxiliary index function $\bar{s}(\mathbf{y}, \mathbf{m})$ is easier to compute than the product $\mathbf{ym}$. In the subsections below, we explain the iteration procedure in more detail, discuss how the distinguished point collision detection method can still be applied even without access to individual $\mathbf{g}_i$'s, and provide a complexity analysis.

4.2 Iteration Function

Let us discuss the resources required for the creation and storage of the table $\mathscr{M}_\ell$, before describing the tag tracing formalism.

4.2.1 Table of multiplier products

The tag tracing approach to DLP solving requires pre-computation of the set $\mathscr{M}_\ell$, before the adding walks can be traveled. The following two lemmas allow us to find the range of r and ℓ, specifying $\mathscr{S}$ and $\mathscr{M}_\ell$, respectively, that can be used with the available resources.

Lemma 2 *The size of the set $\mathscr{M}_\ell$ is $\binom{\ell+r}{r}$ when duplicates that are trivially evident from the commutativity of the multipliers are removed.*

Proof The size of $\mathscr{M}_\ell$, taking into account the fact that multipliers commute with each other, is exactly the number of combinations with repetitions, where one chooses ℓ times from the set $\mathscr{M} \cup \{\mathbf{id}\}$ of size $r+1$. This count is $\binom{\ell+r}{\ell} = \binom{\ell+r}{r}$. □

Some example figures for this size are $\binom{84}{20} \sim 2^{63.2}$ for 20-adding walks with $\ell = 64$, and $\binom{72}{8} \sim 2^{33.4}$ for 8-adding walks with $\ell = 64$. This size is very sensitive to the r value.

Lemma 3 *The set $\mathscr{M}_\ell$ can be constructed using $\binom{\ell+r}{r} - r - 1$ group operations.*

Proof Consider the following arrangement of elements from $\mathscr{M}_\ell$ in an upright triangle. Place the identity element of G at the top vertex. Arrange all multipliers $\mathbf{m}_i$ in a row below the identity element. Next, place all products of two multipliers $\mathbf{m}_i\mathbf{m}_j$ on the third row, and so on. The bottom row consists of $\binom{r+\ell-1}{\ell}$-many ℓ-products of the multipliers and the complete triangle contains all $\binom{r+\ell}{\ell}$-many entries from $\mathscr{M}_\ell$.

Note that each element on any row can be computed from an element on its upper row using a single group operation. Hence, the number of products required to create the complete triangle is equal to the number of points on the triangle, less the elements of the top two rows, which do not require any computation. □

Our use of the set $\mathscr{M}_\ell$ in tag tracing will be restricted to parameters r and ℓ such that $\binom{\ell+r}{r} \ll \sqrt{q}$. Deferring careful complexity analysis to a later subsection, this implies that the cost of preparing this set is negligible in comparison to the main function iteration cost, which is of $\sqrt{q}$ order. Accordingly, the time complexity associated with the creation of $\mathscr{M}_\ell$ will be ignored when we state the tag tracing complexities.

The above mentioned parameter range $\binom{\ell+r}{r} \ll \sqrt{q}$ also means that nontrivial collisions within the set $\mathscr{M}_\ell$, i.e., those that are unrelated to the commutativity of the multipliers, are rare. However, when there are such collisions, if multipliers of the form $\mathbf{g}^\alpha\mathbf{h}^\beta$ with $\beta \neq 0$ were in use, the collision would bring about a linear equation concerning $\log_\mathbf{g} \mathbf{h}$ that returns the solution to the given DLP with a high probability.

4.2.2 Tag function

Let us formalize the auxiliary index function that was introduced in Section 4.1. To allow for flexibility, we consider a slightly more complicated structure. Given the group G, we will fix the index set $\mathscr{S}$ and a somewhat larger set $\mathscr{T}$, together with an appropriate choice of the following four functions.

$$\begin{aligned}
\tau &: G \to \mathscr{T}. \\
\bar{\tau} &: G \times \mathscr{M}_\ell \to \mathscr{T}. \\
\sigma &: \mathscr{T} \to \mathscr{S}. \\
\bar{\sigma} &: \mathscr{T} \to \mathscr{S} \cup \{\text{fail}\}.
\end{aligned}$$

The set $\mathscr{T}$ is called the *tag set* and the two functions τ and $\bar{\tau}$ are named the *tag function* and the *auxiliary tag function*, respectively. The *projections* σ and $\bar{\sigma}$ are to agree with each other whenever $\bar{\sigma}$ does not signal failure. We define the index function to be

$$s = \sigma \circ \tau : G \to \mathscr{S},$$

and we also define the auxiliary index function as

$$\bar{s} = \bar{\sigma} \circ \bar{\tau} : G \times \mathscr{M}_\ell \to \mathscr{S} \cup \{\text{fail}\}.$$

The four functions should be chosen in such a way that they satisfy the following conditions.

1. The index function s is surjective and roughly pre-image uniform, i.e., grouping G according to its image points under s partitions G into subsets of roughly the same size.
2. When $\bar{s}(\mathbf{y},\mathbf{m}) \neq \text{fail}$, we have $\bar{s}(\mathbf{y},\mathbf{m}) = s(\mathbf{ym})$.

Thus, we are looking for a function τ that resembles a usual index function, but with a larger image set, and we also seek another way $\bar{\tau}$ to evaluate τ on a pair of group elements. The preliminary functions τ and $\bar{\tau}$ are allowed the possibility of not corresponding to each other perfectly and this discrepancy is to be filtered out by the projections σ and $\bar{\sigma}$. The resulting composition $\bar{s} = \bar{\sigma} \circ \bar{\tau}$ should output what can reliably be said to depend only on the product of group elements and should be equal to $s = \sigma \circ \tau$, except that it may occasionally fail in producing an output.

For our tag tracing to be meaningful, the computation of $\bar{s}$ should be easier than the computation of a full product in G. Concrete constructions that satisfy all given conditions will be given in later sections, but the intuition behind this approach is that it should take less effort to obtain some partial information about a product than the full product itself. For example, under a fixed encoding for a group, computation of the most significant k bits of a product of two elements should be easier than computation of the full product. Computing k bits out of the full n bits may require $\frac{k}{n}$ of the time needed for a full product computation and, should some of the product bits be easier to compute than others, the computation time could be even shorter.

4.2.3 Random walks with tag tracing

Let us now continue along the lines of argument presented in Section 4.1. It should be clear that if functions τ, $\bar{\tau}$, σ, and $\bar{\sigma}$, that satisfy the conditions we have imposed on them, can be constructed, we can create the random graph of a normal adding walk without computing the full product at each iteration.

In traversing the random walk with tag tracing, full product computations are required only when either ℓ iterations have been processed since the last full product, or when $\bar{s}$ signals failure. When the computation of $\bar{s}$ is assumed to be faster than a group operation, each iteration of tag tracing will be faster on average than a single unmodified adding walk iteration. On the other hand, since the random walk produced through tag tracing iterations is a normal adding walk, its expected rho length will be that of a normal adding walk. Since each iteration is faster, the tag tracing iterations will reach a collision earlier than the original adding walk.

As a final check on tag tracing as an iteration mechanism, we would like to discuss its exponent tracing capability. Since, the index value is available to tag tracing at every iteration, either through the auxiliary index function $\bar{s}$ or a full product computation, one can keep track of the exponent information for the current element in the $\mathbf{g}^{a_i}\mathbf{h}^{b_i}$ form, just as was done with the original adding walk. Hence tag tracing is exponent traceable.

Tag tracing even allows an alternative that is slightly more efficient. Note that we know how to write each element of $\mathscr{M}_\ell$ in the form $\mathbf{g}^\alpha\mathbf{h}^\beta$. We view the set $\mathscr{M}_\ell$ as a *table*, in which every entry consists of three[1] components. The first component is the information regarding the combination of multipliers used in creating the entry, the second is the resulting group element, and the third is the exponent information of the group element when expressed in the $\mathbf{g}^\alpha\mathbf{h}^\beta$ form. The table $\mathscr{M}_\ell$ is sorted according to the multiplier combination information, or a hash table technique is utilized, so that table lookups are easy. Then, the bookkeeping of exponent information, which was necessary at each iteration, can be deferred until a full product computation is required, at which time, the information may be updated from the table with just two modular q additions.

4.3 Collision Detection

To complete the description of tag tracing as a DLP solving method, we need to check if it is possible to detect collisions. We shall see that the distinguished point method is suitable for tag tracing.

To collect every distinguished point in a table, one must compute a full product at every occurrence of a distinguished point, even when the ℓ-th tag tracing iteration has not yet been reached. Still, we do not want the process of detecting distinguished points to induce any extra full product computation. Below, we present three approaches to distinguished point detection. Although the first approach will always work and is efficient, we provide two other methods as alternatives that may be applicable to certain situations. The use of these two additional approaches will become clearer when explicit constructions are given. We also provide comments on how even the occasional full product computation done at each distinguished point occurrence may be avoided.

4.3.1 Using consecutive index values

During the tag tracing iterations, one does not have full access to each group element $\mathbf{g}_i$ and this prevents the usual approach to distinguished point collision detection. Still, one does have access to partial information $s_i = s(\mathbf{g}_i)$ concerning each point in the random walk. This is exactly the situation in which the distinguished path segment approach, described in Section 3, can be used.

Depending on the maximum size of the distinguished point table one plans to incrementally collect, one decides on the desired average distance Δ between distinguished points. One then chooses a positive integer δ such that $\Delta \approx \frac{r}{r-1}(r^\delta - 1)$. If one defines the distinguishing property to be that consecutive δ-many indices s_i are zero, then Theorem 1 assures us that the resulting distinguished points will be Δ iterations apart, on average.

Recall that, since we are defining the latest point on a distinguished path segment as the distinguished point, there is some ambiguity regarding the distinguished point definition for points on the random walk that are within δ iterations of the true point of collision. As briefly mentioned in Section 3, since δ is of logarithmic order in the desired average distance Δ

[1] In actual implementations, the first component may serve as the address of each entry and could be absent. Later, when concrete constructions suitable for finite fields are given, we shall add a fourth component to the three components given here.

between distinguished points, whereas the distinguished point collision detection method requires $O(\Delta)$ extra iterations after the point of collision, the distinguished point definition ambiguity has a negligible effect on the performance of collision detection.

4.3.2 Using consecutive tag values

Under some constructions of the tag tracing setting, one might have $\bar{\tau}(\mathbf{y},\mathbf{m}) = \tau(\mathbf{ym})$ for every $\mathbf{y} \in G$ and $\mathbf{m} \in \mathscr{M}_\ell$. In such a fortunate case, one may use the distinguished path segment approach on the tag values, rather than on the index values. Since tags are larger than indices, this allows the length of the observed path segment to be shorter.

This approach can also be used when $\bar{\tau}(\mathbf{y},\mathbf{m}) = \tau(\mathbf{ym})$ is not always true, but when some reliable information about $\mathbf{ym}$ can always be deduced from $\bar{\tau}(\mathbf{y},\mathbf{m})$. Of course, the index is one such reliable information, when it does not fail, but if information larger than that given by the index can be obtained from $\bar{\tau}$ without failure, its use could be advantageous over the use of the indices.

4.3.3 Using unavoidable full product computations

Let us first use a rough example to explain this final approach to defining a distinguished point. We shall temporarily use the notation $\mathbf{g}_j = \mathbf{g}_i\mathbf{m}$, with $\mathbf{m} \in \mathscr{M}_\ell$, and suppose that the tag set $\mathscr{T}$ consists of some integers. Recall that τ and $\bar{\tau}$ are to be constructed in such a way that $\bar{\tau}(\mathbf{g}_i,\mathbf{m})$ is almost equal to $\tau(\mathbf{g}_j)$, but that we cannot always rely on them to be identical. Still, suppose that the event $\tau(\mathbf{g}_j) = 0$ at least reliably implies $\bar{\tau}(\mathbf{g}_i,\mathbf{m}) \in \{-1,0,1\}$. Furthermore, suppose that the projection $\bar{\sigma}$ returns failure on every element of $\{-1,0,1\}$.

Under the assumptions we have described, whenever we have $\tau(\mathbf{g}_j) = 0$, the auxiliary index function $\bar{s} = \bar{\sigma} \circ \bar{\tau}$ returns failure, and the full product $\mathbf{g}_j = \mathbf{g}_i\mathbf{m}$ is computed. Hence, it is reasonable to set the distinguished point condition to something similar to

"$\tau(\mathbf{g}_j) = 0$ and twenty MSBs of $\mathbf{g}_j$ are equal to zero."

Then, whenever the current point $\mathbf{g}_j$ has the possibility of being a distinguished point, we would already have full access to $\mathbf{g}_j$, and both the tag-condition and the MSB-condition could be checked directly from $\mathbf{g}_j$. This distinguishing property does not entail any extra full product computation beyond what is already required by tag tracing iterations.

The double condition for a distinguished point is different from requiring just the MSB-condition, since full access to $\mathbf{g}_j$ was a byproduct of the tag-condition being met. Also, we cannot replace the tag-condition with either "$\bar{\tau}(\mathbf{g}_i,\mathbf{m}) \in \{-1,0,1\}$" or "failure of $\bar{s}$," since neither outcome is a reliable function of the current point $\mathbf{g}_j$. Both $\bar{\tau}$ and $\bar{s}$ are functions of the pair $(\mathbf{g}_i,\mathbf{m})$, not of the product $\mathbf{g}_j = \mathbf{g}_i\mathbf{m}$.

Let us summarize the above example as a general method. Suppose that we have a construction of the tag tracing formalism under which $\bar{s}(\mathbf{g}_i,\mathbf{m}) = \text{fail}$ is a necessary condition for $\tau(\mathbf{g}_i\mathbf{m})$ to satisfy a certain "Condition-X." Then, one may set the distinguishing property for group element $\mathbf{g}_j$ as a combination of Condition-X on tag value $\tau(\mathbf{g}_j)$ and some extra condition on $\mathbf{g}_j$. The extra condition on $\mathbf{g}_j$ can be used to control the average distance between distinguished points. This distinguished point definition does not increase the frequency of necessary full product computations. A more concrete example of the current approach is given in Section 6.3.3.

4.3.4 Distinguished point storage and full product computation

Our description of tag tracing defaults to computing a full product at each distinguished point occurrence. In practice, distinguished point occurrences are much less frequent than the return of the ℓ-th iteration limit, hence their effect on the total running time is very small. Still, let us explain that even these full products are not absolutely necessary.

In using the distinguished point method with any DLP solving algorithm, the final distinguished point table will contain much less than $\sqrt{\frac{\pi}{2}q}$ entries. Hence, the birthday paradox allows us to argue that computing and storing just $\log q$ bits of information for each distinguished point will suffice to safely distinguish between the distinguished points. This partial information may come either from a partial product computation, or, as described at the end of Section 3.1, it may be a sequence of index values that is sufficiently long.

If the partial information approach is used, collisions discovered within the distinguished point table are quite probably, but not necessarily, true collisions, so one would need to run a final check through explicit exponentiation when a solution candidate is returned by the DLP solving algorithm.

4.4 Complexity Analysis

Let us provide rough comparisons of the time and storage complexities of the tag tracing method with those of the original adding walks. The tag tracing algorithm is compactly presented as a pseudo-code in Algorithm 1 for easy reference.

Algorithm 1 Tag tracing

```
1: compute multiplier product table $\mathscr{M}_\ell$
2: prepare an empty table of distinguished points
3: $\mathbf{y} \leftarrow$ random power of $\mathbf{g}$
4: initialize exponent information for $\mathbf{y}$
5: while {distinguished point table contains no collision} do
6:     $i \leftarrow 0$, $\mathbf{m} \leftarrow \mathbf{id}$
7:     while $\{(i < \ell)$ and $(\bar{\tau}(\mathbf{y},\mathbf{m}) \neq \text{fail})$ and $(\mathbf{y}\mathbf{m}$ is not a distinguished point$)\}$ do
8:         $s_i \leftarrow \bar{\tau}(\mathbf{y},\mathbf{m})$
9:         look up $\mathbf{m}\mathbf{m}_{s_i} = \mathbf{m}_{s_0}\cdots\mathbf{m}_{s_i}$ in table $\mathscr{M}_\ell$
10:        $\mathbf{m} \leftarrow \mathbf{m}\mathbf{m}_{s_i}$
11:        $i \leftarrow i+1$
12:    end while
13:    fully compute the product $\mathbf{ym}$
14:    $\mathbf{y} \leftarrow \mathbf{ym}$
15:    update exponent information for $\mathbf{y}$ using that of $\mathbf{m}$
16:    if {$\mathbf{y}$ is a distinguished point} then
17:        add $\mathbf{y}$ to the table of distinguished points, together with its exponent information
18:    end if
19: end while
20: solve for DLP solution from the exponent information of colliding elements
```

4.4.1 Time complexity

We assume that the various parameters for tag tracing have been chosen so that the time taken for preparation of $\mathscr{M}_\ell$, given by Lemma 3, is insignificant when compared to that taken by the main function iterations. We also ignore the cost of keeping track of the exponents that lead to the linear equation concerning the discrete logarithm. Since a group of size q is usually realized as a subset of a considerably larger or complicated structure, the exponents of order q require less resources to compute than the group operations.

We denote the expected time for a single r-adding walk iteration by $|F_r|$ and that of a single $\bar{s}(\mathbf{y},\mathbf{m})$ evaluation by $|\bar{s}|$. Note that the time taken for a full product computation in G is almost equal to $|F_r|$. Hence, the requirement that $\bar{s}$ be easier to compute than the group operation, can be stated as $\frac{|\bar{s}|}{|F_r|} < 1$.

The time taken for the occasional full product computation performed during tag tracing is also assumed to be $|F_r|$. Care must be taken when using this assumption, since the time for full product computation depends on the encoding for G, and explicit construction for tag tracing may restrict the choice of encoding to something different from what would be used with the unmodified r-adding walk.

Most iterations of tag tracing will require just one computation of $\bar{s}$ and full products are required only when either the ℓ-th iteration is reached, $\bar{s}$ returns failure, or a distinguished point is found. Hence, the expected time for a single iteration of tag tracing is bounded above by

$$|\bar{s}| + \Big(\frac{1}{\ell} + P_{\text{fail}} + \frac{1}{\Delta}\Big)|F_r|, \tag{4}$$

where P_{fail} is the probability that $\bar{s}$ will return a failure and Δ is the average number of iterations between distinguished points. Since tag tracing generates a normal r-adding walk, its total time complexity in solving a DLP is

$$\Big\{|\bar{s}| + \Big(\frac{1}{\ell} + P_{\text{fail}} + \frac{1}{\Delta}\Big)|F_r|\Big\}\Big\{\Big(\begin{matrix}\text{rho length of}\\ r\text{-adding walk}\end{matrix}\Big) + \Delta\Big\}. \tag{5}$$

Since the time complexity of the unmodified r-adding walk is

$$|F_r|\Big\{\Big(\begin{matrix}\text{rho length of}\\ r\text{-adding walk}\end{matrix}\Big) + \Delta\Big\},$$

we can claim

$$\frac{|\bar{s}|}{|F_r|} + \frac{1}{\ell} + P_{\text{fail}} + \frac{1}{\Delta} \tag{6}$$

as the ratio of DLP solving time between the two algorithms. If this ratio is less than 1, we have a reduction in the DLP solving time. As discussed at the end of Section 4.2.2, intuitively, it should be possible to define an index function that is easier to compute than the full product, and hence be able to construct τ and $\bar{\tau}$ such that $|\bar{s}|$ is much smaller than $|F_r|$.

The above equation has some implications on the choice of parameters. One would of course like to keep all four terms small, but if one of the terms cannot be reduced beyond a certain point, it is meaningless to reduce the other terms to a value considerably smaller than the dominating term. For example, if there is a $\frac{|\bar{s}|}{|F_r|}$ value that one will have to accept for a certain group G on a certain implementation platform, it does not make sense to increase ℓ beyond a certain point at the cost of larger storage use and longer table preparation time.

4.4.2 Storage complexity

Tag tracing clearly requires more storage than does the original adding walk. The extra storage size is given by Lemma 2, but since both methods require a distinguished point table, whether this extra amount is significant or not depends on the design choices.

Recall that the distinguished point collision detection method requires extra function iterations after the point of collision. Since a larger distinguished point table reduces these extra iterations, one would tend to use as large a distinguished point table as is manageable under the available resources. The choice of parameters for the table $\mathscr{M}_\ell$ will also be similar. Up to a certain point, a larger pre-computation table implies faster tag tracing, and the limit of manageable storage size becomes a bound for the size of $\mathscr{M}_\ell$.

Given the same resources, it would be fair to say that a person who is implementing tag tracing would allocate most of the storage to a large $\mathscr{M}_\ell$ and use a smaller distinguished point table than a person who is implementing the original adding walk, because the effect of reducing the distinguished point table is not critical to the overall performance, except when near the extremes. We thus have an argument, albeit quite weak, that with a reasonable choice of parameters, the storage use by tag tracing will be comparable to that of the original adding walk.

4.5 Use of Small r

In a practical small software implementation of tag tracing, the available online memory places a restriction on the size of table $\mathscr{M}_\ell$. A large table brings about slow table lookups and leads to a larger $|\bar{s}|$. This may not be the case with large scale implementations in which the hash table technique is used for constant time access to large storage, but let us consider the situation where the size of $\mathscr{M}_\ell$ is restricted by the available resources.

The ratio (6) between the original adding walk and tag tracing of the time taken by a single iteration shows that when a small $\frac{|\bar{s}|}{|F_r|}$ is achievable, one would wish to use a large ℓ. On the other hand, Lemma 2 states that $|\mathscr{M}_\ell| = \binom{\ell+r}{r}$, so that r has a critical effect upon the table size when a large ℓ is in use. Therefore, a bound on storage size may restrict our choice of r to small integers.

Testing done on elliptic curve groups [32] has shown that the 20-adding walk presents behavior that is quite close to what is optimal. Since it is known that smaller r values lead to longer r-adding walk rho lengths, for practical uses of tag tracing, we need to discuss how the choice of small r impacts the overall performance. To explore this area, we ran tests comparing the average rho lengths of r-adding walks under small r values against that of the 20-adding walk.

Let us explain the details of the experiment setting. First, primes p and q and a cyclic group generator $\mathbf{g} \in \mathbf{F}_p^\times$ of order q were randomly generated in the manner described for DSA [18]. Next, an element $\mathbf{h} \in \mathbf{F}_p^\times$ and the multiplier set were randomly selected. The r-adding walk iteration function was then iterated from a random starting point until the walk intersected itself in a rho. We took measures to identify the exact point of collision, rather than rely on distinguished points that would give approximate collision positions. The length of the rho was recorded and this process was repeated 20,000 times for each iteration function with a newly generated set of $\mathbf{g}$, $\mathbf{h}$, and the multiplier set. The size of p was set to 1024 bits throughout the test, but the group size q was varied. The index function $s_r : \mathbf{F}_p \to \{0, \ldots, r-1\}$ for the r-adding walk was defined to be $s_r(\mathbf{y}) = \lfloor r \cdot (\varphi \cdot \mathbf{y} \mod 1) \rfloor$, where φ

Table 2 Average rho lengths for various iteration functions, in units of $\sqrt{\frac{\pi}{2}q}$

$\lceil \log q \rceil$	10	15	20	25	30	35	40	45	average
F_P	1.27	1.29	1.28	1.30	1.29	1.29	1.30	1.29	1.29
F_3	1.60	1.85	2.04	2.22	2.40	2.54	2.68	2.85	-
F_4	1.30	1.34	1.36	1.36	1.35	1.35	1.34	1.35	1.34
F_8	1.08	1.09	1.09	1.08	1.09	1.08	1.08	1.08	1.08
F_{20}	1.03	1.04	1.03	1.03	1.02	1.03	1.03	1.03	1.03

is a rational approximation of the golden ratio $\frac{\sqrt{5}-1}{2}$. When φ is of sufficient precision, this is known to bring about a uniform looking distribution [15], even on non-uniform inputs. For our experiment, a precision of 1044 binary places for φ is sufficient.

The test results are summarized in Table 2. The top row shows the size of prime q in bits. Each row corresponds to an r-adding walk for a specific r, except for the top row, which corresponds to Pollard's original iteration function. The numeric values are rho lengths given in units of $\sqrt{\frac{\pi}{2}q}$, which is the rho length expected of a random function.

Let us use the notation ρ_P for the expected rho length of Pollard's original iteration function and write ρ_r for the rho length of the r-adding walk. The table clearly shows that the $\frac{\rho_r}{\sqrt{\pi q/2}}$ values are quite stable over a wide range of group orders for each $r \geq 4$. The case of the 3-adding walk is a clear exception to this stable behavior. In fact, this clear distinction between the 3-adding walk and the other r-adding walks was also noted in [25] and [32], in which the tests were performed on groups different from the multiplicative subgroups of the prime fields.

Supported by the stable figures of Table 2, the arguments presented in this paper assume that the complexity of r-adding walks is $O(\sqrt{q})$ for $r \geq 4$, even when q is significantly larger than the values we have tested, and that the constants hidden behind the big-O notation can be inferred from Table 2. In particular, we assume that the ratios

$$\frac{\rho_4}{\rho_P} \approx 1.04, \quad \frac{\rho_4}{\rho_{20}} \approx 1.31, \quad \text{and} \quad \frac{\rho_P}{\rho_{20}} \approx 1.25, \tag{7}$$

which may easily be computed from the average values appearing in Table 2, are roughly correct, even on very large prime order groups.

Even though our claim (7) is based on experiment data, some related theoretic studies may be found in [25, 32]. They provide partial support to the claim that the expected rho lengths of r-adding walks, for a certain range of r values, are asymptotically $O(\sqrt{q})$.

Note that any theoretical comparison of tag tracing with the unmodified r-adding walk, concerning their asymptotic behaviors as q is increased, is not affected by the uncertainty of the r-adding walk rho lengths. We may simply let the tag tracing use whatever r value the unmodified r-adding walk would have used. In such a case the pre-computed table size may be very large, but it remains constant as q is increased. Arguments that consider asymptotic performance of tag tracing as the finite field size is increased, but with a fixed group size q, are also not affected by the true rho lengths of various r-adding walks. In such a discussion, as can be seen from (5), the rho length only works as a large multiplicative constant. The issues discussed in this subsection need only be considered for practical applications of tag tracing.

4.6 Pollard's Kangaroo Algorithm and Tag Tracing

Suppose we know beforehand that $\log_{\mathbf{g}} \mathbf{h}$ lies in an interval $[L,U] \subset [0,q-1]$. Such a situation may occur, for example, when the DLP target $\mathbf{h} = \mathbf{g}^x$ is created from a small exponent x, chosen with the intension of accelerating exponentiation. Pollard's kangaroo algorithm is another variant of the rho algorithm that is well suited to this situation. The algorithm is sometimes called Pollard's lambda[2] algorithm, and it solves DLP with $O(\sqrt{\omega})$ complexity, where $\omega = U - L$. We refer readers to the original work [22] and related articles [20, 23, 33] for details.

The kangaroo method uses a special type of the r-adding walk iteration function, in which the multipliers are taken to be of the form $\mathbf{g}^{\alpha}$. Collisions are detected through the distinguished point method. Hence, application of the tag tracing approach to the kangaroo method is quite natural. Moreover, since the multipliers do not depend on $\mathbf{h}$, we may precompute the table $\mathscr{M}_\ell$, even before the DLP target $\mathbf{h}$ is given.

There are two suggestions in the literature concerning the choice of multiplier exponents. One is to take the exponents to be of the form $\{1, x, \ldots, x^{r-1}\}$ and the other is to use twenty random exponents. The latter method is more appropriate for the tag tracing version of the kangaroo method, since a small r is desirable, and the former method tends to require a large index set in order for it to be useable.

5 Application to Subgroups of Finite Fields with Large Extension Degrees

In this section, we construct an explicit tag tracing setting that is suitable for cyclic subgroups of the finite fields with large extension degrees. The elements of a finite field with a large extension degree are usually represented in either the polynomial basis or the normal basis form. In this paper, we only discuss the case of the polynomial basis. Tag tracing on the normal basis is addressed in the appendix of [12].

Let us fix the finite field to $\mathbf{F}_{p^n} = \mathbf{F}_p[x]/\langle \mathfrak{p}(x) \rangle$, where $\mathfrak{p}(x) \in \mathbf{F}_p[x]$ is an irreducible polynomial of degree n, and view elements of $\mathbf{F}_{p^n}$ as polynomials $\mathbf{y}(x)$ of degree less than n. Given an integer k and a positive integer r, we use the notation $\mathrm{mod}_r\, k$ for the non-negative integer less than r that is congruent to k modulo r. Extending the notation used with integers to polynomials, we write $\left\lfloor \frac{\mathbf{y}(x)}{\mathbf{f}(x)} \right\rfloor$ and $\mathrm{mod}_{\mathbf{f}(x)} \mathbf{y}(x)$ to denote the quotient and remainder, respectively, of the polynomial division of $\mathbf{y}(x)$ by $\mathbf{f}(x)$. Throughout this section, the term p-digit will refer to an element of $\mathbf{F}_p$. Some readers may find it easier to read this section with p fixed to 2 and with the word bit used instead of the term p-digit.

5.1 Tag Tracing Setting

Let us construct the four functions that need to be provided for tag tracing on subgroups of $\mathbf{F}_{p^n}^{\times}$. Although the actual functions we provide below may appear complicated, the core ideas are quite simple. In very rough terms, the tag will be the most significant few p-digits of the given element. We show that it is possible to compute a few most significant p-digits of a product without fully computing the product. This partial product computation can be done without failure, and hence our auxiliary tag function can always agree with the tag

[2] The term Pollard's lambda algorithm is sometimes associated with the parallelized versions of the rho variants.

function. This removes the need for the auxiliary projection function to fail, and the auxiliary index function will always agrees with the index function.

Let us now be more concrete. Choose positive integers r and $\mathtt{t}$ such that $r \leq p^{\mathtt{t}}$. The multiplier product length ℓ is chosen in such a way that the time and storage complexities given by Lemma 2 and Lemma 3 are manageable and are also insignificant in comparison to $\sqrt{q}$. The tag set is fixed to

$$\mathscr{T} = \{ y(x) \in \mathbf{F}_p[x] \,|\, \deg y(x) < \mathtt{t} \},$$

which can be identified with the set of all non-negative integers less than $p^{\mathtt{t}}$. The tag function $\tau : G \to \mathscr{T}$ is defined to be

$$\tau\big(\mathbf{y}(x)\big) = \left\lfloor \frac{\mathrm{mod}_{\mathfrak{p}(x)}\mathbf{y}(x)}{x^{n-\mathtt{t}}} \right\rfloor. \tag{8}$$

Since a small r value is desirable, $(r,\mathtt{t}) = (4,2)$ or $(8,3)$ are good choices for application to subgroups of $\mathbf{F}_{2^n}^{\times}$.

The tag value (8) is simply the $\mathtt{t}$-many highest degree p-digit coefficients of the input group element in its polynomial basis format. No division is involved in obtaining these coefficients if the input is already in its modulo $\mathfrak{p}(x)$ reduced form. Clearly, the tag function τ, as a function defined on all of $\mathbf{F}_{p^n}$, is surjective and exactly pre-image uniform. Hence, when the target group G is assumed to be randomly distributed within $\mathbf{F}_{p^n}$, this tag function partitions G into subsets of roughly the same size.

As for the projection $\sigma : \mathscr{T} \to \mathscr{S} = \{0,1,\dots,r-1\}$, we choose any surjective function that is roughly pre-image uniform and easy to compute. For example, when r is a divisor of $|\mathscr{T}| = p^{\mathtt{t}}$, one may use

$$\sigma(k) = \mathrm{mod}_r\, k,$$

where we are viewing the elements of $\mathscr{T}$ as non-negative integers less than $p^{\mathtt{t}}$. Then, the index function $s = \sigma \circ \tau$ is roughly pre-image uniform. When $|\mathscr{T}|$ is not a multiple of r, one may have to devise an ad hoc projection, such as something similar to the s_r index function, which appeared in Section 4.5.

We now discuss the auxiliary tag function $\bar{\tau} : G \times \mathscr{M}_\ell \to \mathscr{T}$. Given two polynomials $\mathbf{y}(x), \mathbf{m}(x) \in \mathbf{F}_{p^n}^{\times}$ we want to compute the $\mathtt{t}$-many highest degree coefficients of their product reduced modulo $\mathfrak{p}(x)$. Under the notation $\mathrm{mod}_{\mathfrak{p}(x)}\mathbf{y}(x) = \sum_{i=0}^{n-1} y_i x^i$, we can write

$$\mathbf{y}(x)\,\mathbf{m}(x) \equiv \sum_{i=0}^{n-1} y_i x^i \,\mathbf{m}(x) \equiv \sum_{i=0}^{n-1} y_i \left\{ \mathrm{mod}_{\mathfrak{p}(x)} \big(x^i\,\mathbf{m}(x)\big) \right\} \pmod{\mathfrak{p}(x)}.$$

Observe that the right end of this modular equivalence is a sum of polynomials of degree less than n, and hence that the resulting sum is also of degree less than n. This shows that we have

$$\mathrm{mod}_{\mathfrak{p}(x)} \{ \mathbf{y}(x)\,\mathbf{m}(x) \} = \sum_{i=0}^{n-1} y_i \left\{ \mathrm{mod}_{\mathfrak{p}(x)} \big(x^i\,\mathbf{m}(x)\big) \right\}, \tag{9}$$

which is not just a modular equivalence, but an equality of polynomials.

Now, focusing on the few highest degree terms from both sides of this equality, we can conclude that, in order to compute the highest degree part of $\mathrm{mod}_{\mathfrak{p}(x)}\{\mathbf{y}(x)\,\mathbf{m}(x)\}$, it suffices to compute a $\mathbf{F}_p$-linear combination of the highest degree parts of $\mathrm{mod}_{\mathfrak{p}(x)}\big(x^i\,\mathbf{m}(x)\big)$'s. Restating this in terms of tag values, we can write

$$\tau\big(\mathbf{y}(x)\,\mathbf{m}(x)\big) = \sum_{i=0}^{n-1} y_i\, \tau\big(x^i\,\mathbf{m}(x)\big). \tag{10}$$

There is now only one reasonable way to define the auxiliary tag function. Given group elements $\mathbf{y}(x) \in G$ and $\mathbf{m}(x) \in \mathscr{M}_\ell$, we define their auxiliary tag function $\bar{\tau} : G \times \mathscr{M}_\ell \to \mathscr{T}$ output as

$$\bar{\tau}\big(\mathbf{y}(x), \mathbf{m}(x)\big) = \sum_{i=0}^{n-1} y_i\, \tau\big(x^i\, \mathbf{m}(x)\big), \tag{11}$$

where the notation y_i is given by $\mathrm{mod}_{\mathtt{p}(x)}\mathbf{y}(x) = \sum_{i=0}^{n-1} y_i x^i$. Since each $\tau\big(x^i\, \mathbf{m}(x)\big)$ is of degree $\mathtt{t}-1$ at the most, the defined output value is also of degree at most $\mathtt{t}-1$ and it lies in $\mathscr{T}$. The discussion thus far presented and the definition of $\bar{\tau}$ make it clear that given any two element $\mathbf{y}(x)$ and $\mathbf{m}(x)$ of $\mathbf{F}_{p^n}$, we always have

$$\tau\big(\mathbf{y}(x)\, \mathbf{m}(x)\big) = \bar{\tau}\big(\mathbf{y}(x), \mathbf{m}(x)\big).$$

The only remaining component, the auxiliary projection function, is set to $\bar{\sigma} = \sigma$ and does not return failure signs. We now automatically have the index function $s = \sigma \circ \tau$ and the auxiliary index function $\bar{s} = \bar{\sigma} \circ \bar{\tau}$ in agreement on corresponding inputs, without $\bar{s}$ ever failing.

Now that we have described the tag tracing setup in full detail, we state that an analogous approach that takes $\mathtt{t}$-many lowest degree p-digits coefficients as tags is also possible. It suffices to replace (8) with

$$\tau\big(\mathbf{y}(x)\big) = \mathrm{mod}_{x^{\mathtt{t}}}\left\{\mathrm{mod}_{\mathtt{p}(x)}\mathbf{y}(x)\right\}$$

and retain (11) as the definition for the auxiliary tag. We have chosen to work with the highest degree part, as it has advantages in computing $\mathscr{M}_\ell$, when the second highest term of $\mathtt{p}(x)$ is of low degree.

5.2 Tag Tracing

In the previous subsection, we provided all the four functions required for tag tracing that were introduced in Section 4.2.2, and we also verified that they satisfy almost all the conditions that were imposed on them. One condition we did not discuss was whether the auxiliary index function was easier to compute than the group operation. This will be discussed later in this subsection.

We are now ready to begin tag tracing on subgroups of $\mathbf{F}_{p^n}^{\times}$. Using the proof of Lemma 3 that appeared in Section 4.2.1 as a construction guide, we build the table $\mathscr{M}_\ell$. Recall from Section 4.2.3 that each entry of the table consists of three components, i.e., the multiplier combination information, the group element in full form, and the exponent information. For each $\mathbf{m} \in \mathscr{M}_\ell$, we add the following fourth component to each entry of the table.

$$\big(\tau(x^0\mathbf{m}), \tau(x^1\mathbf{m}), \ldots, \tau(x^{n-1}\mathbf{m})\big)\,. \tag{12}$$

Note that these have appeared before, in the right-hand side of (11).

We can now follow the discussion of Section 4.2.3 to step through each iteration of tag tracing. The group elements are written in polynomial basis form so that coefficients $y_i \in \mathbf{F}_p$ are quickly accessible. Auxiliary tags are computed through (11) with table lookups to the fourth entry (12). Since the auxiliary index function never fails, full product computation between group elements is required only when ℓ tag tracing iterations are reached or a distinguished point is discovered. In fact, the index value will be read off the element through (8) only after a full product computation. The path segment approach of Section 4.3.1 is used to define distinguished points. When tag set size $p^{\mathtt{t}}$ is strictly larger than r, it is also possible to use the approach of Section 4.3.2.

It still remains to be seen if each iteration of this tag tracing realization will be faster than a group operation. The computation of auxiliary tag function (11) requires $n\mathtt{t}$ products and less than $n\mathtt{t}$ additions in the small field $\mathbf{F}_p$. Disregarding the cost of projection $\bar{\sigma}$, we may state that the computation of $\bar{s}$ requires less than $2n\mathtt{t}$ operations in $\mathbf{F}_p$. Let us write $\mathrm{Mul}_p(n)$ to denote the number of operations in $\mathbf{F}_p$ that are required to multiply two elements of $\mathbf{F}_{p^n}$. Recalling (6), since $\bar{s}$ never returns failure, we can claim

$$\frac{2n\mathtt{t}}{\mathrm{Mul}_p(n)} + \frac{1}{\ell} + \frac{1}{\Delta} \tag{13}$$

as the ratio of the DLP solving time of tag tracing and that of the unmodified adding walk. As discussed in Section 4.5, this ratio may have to be multiplied by $\frac{\rho_4}{\rho_{20}} \approx 1.31$ when practical storage issues force the use of $r = 4$ with tag tracing. In any case, when n is large compared to $\mathtt{t}$, which is natural to expect for finite fields of large extension degree, this ratio is likely to be considerably smaller than 1, which implies that tag tracing will be advantageous over the unmodified adding walks.

5.3 Asymptotic Complexity

The time complexity of adding walks in solving DLPs is usually said to be of $O(\sqrt{q})$ and in Section 4.5 we provided experimental evidence that the same may be said of tag tracing when $r \geq 4$ is used. Note that the unit used in this complexity claim is the time taken for a single application of the iteration function. For subgroups of the finite fields, this unit of time may be rather independent of q, but it is intimately correlated to the size of the finite field containing the group. In this subsection, we consider the asymptotic complexity of tag tracing on prime order q subgroups of $\mathbf{F}_{p^n}^{\times}$ as n is increased, while p and q are kept fixed. In other words, we study the asymptotic behavior of the constant hiding behind the big-O notation of $O(\sqrt{q})$ as the base field size is increased.

We clearly state once more that this section considers the asymptotic behavior of tag tracing when the size of the finite field containing the group, rather than the size of the group, is increased.

Reviewing the arguments leading up to (13) and recalling (5), the complexity of tag tracing may be written as

$$\Big\{2n\mathtt{t} + \Big(\frac{1}{\ell} + \frac{1}{\Delta}\Big)\mathrm{Mul}_p(n)\Big\}\Big\{\rho_r + \Delta\Big\}, \tag{14}$$

where ρ_r is the expected rho length of an r-adding walk. We take $\mathtt{t}$ and hence also $r = p^{\mathtt{t}}$ to be small constants and choose to use $\ell = \Theta(n)$. The average distance Δ between distinguished points is fixed in such a way that the number of entries $\frac{\rho_r}{\Delta}$ in the final distinguished point table becomes a manageable constant $O(1)$. The first term $2n\mathtt{t}(\rho_r + \Delta)$ is then trivially $O(n)\rho_r$, and since $\mathrm{Mul}_p(n)$ is at most quadratic in n, the second term $\frac{1}{\ell}\mathrm{Mul}_p(n)(\rho_r + \Delta)$ is also $O(n)\rho_r$. As for the third term, we can claim

$$\frac{1}{\Delta}\mathrm{Mul}_p(n)\big(\rho_r + \Delta\big) = \mathrm{Mul}_p(n)\big(O(1) + 1\big) = O(1)\,\mathrm{Mul}_p(n). \tag{15}$$

Combining the three terms, we can write the asymptotic complexity of tag tracing as

$$O(n)\rho_r + O(1)\,\mathrm{Mul}_p(n).$$

The $O(1)$ factor in front of $\mathrm{Mul}_p(n)$, which is roughly the final size of the distinguished point table, can only be much smaller than ρ_r. Hence, in practice, the $O(n)\rho_r$ term, rather than the $O(1)\,\mathrm{Mul}_p(n)$ term, is the dominating term. Furthermore, the smaller second term may be replaced by a single solution verification exponentiation if the approach of Section 4.3.4 is taken. In conclusion, if we exclude a finite number of products in $\mathbf{F}_{p^n}$, the complexity of tag tracing becomes linear in the extension degree n.

We have already mentioned that the auxiliary tag function $\bar{\tau}$ computes $\mathtt{t}$-many most significant p-digits for the product of the two given inputs. To obtain the full product of two elements from $\mathbf{F}_{p^n}$, given in their polynomial basis representation, one must compute all products between n^2-many pairs of $\mathbf{F}_p$ elements. In comparison, $\bar{\tau}$ computes $n\mathtt{t}$-many products of p-digit pairs. We have obtained a $\frac{\mathtt{t}}{n}$ factor reduction in computation, by requiring only $\mathtt{t}$-many p-digits out of the n-many p-digits of information.

The issue of storage complexity remains to be considered. To be able to compute (11) without delay, each entry of $\mathscr{M}_\ell$ requires $n\mathtt{t}$-many p-digit places to store the fourth component (12). Disregarding the first three smaller components, the storage complexity may be written as

$$\binom{\ell+r}{r} n\mathtt{t},$$

where the unit of storage is a p-digit. When $\ell = \Theta(n)$, this is of $O(n^{r+1})$ complexity.

In summary, as n is increased with p and q fixed, tag tracing runs at the linear $O(n)$ time complexity. The reduction that is seen when this is compared with the unmodified adding walk, which typically runs at $O(n^2)$ time complexity, is achieved at the cost of storage of polynomial complexity $O(n^{r+1})$. The use of seemingly larger storage is justifiable in practice, as it can be chosen to be much smaller than the effective time complexity of $O(n)\rho_r$.

Before ending this section, we remark that the ratio (13) shows that there is no advantage in increasing the tag set size $\mathtt{t}$ to anything larger than what is required for the index. Since the practical bound on the size of table $\mathscr{M}_\ell$ is likely to force the use of a small r, it is advisable to equate r with $p^{\mathtt{t}}$ and use a small $\mathtt{t}$.

5.4 Implementation of Tag Tracing on Binary Fields

We have tested tag tracing on prime order subgroups of the binary fields $\mathbf{F}_{2^n}$ with an implementation on a modern PC[3] and compared it with the original rho algorithm and the unmodified 20-adding walk. In implementing Pollard's original iteration function and the 20-adding walk, we used the binary field arithmetics functions from the NTL library [30], so as not to be biased in favor of tag tracing.

Note that [11], for each $n \leq 5000$, it is possible to choose the irreducible polynomial $\mathfrak{p}(x)$ in such a way that its second highest term is of degree that is at most $3 + \log_2 n$. Many implementations of the binary field take advantage of this situation, as having a second highest term of low degree allows for efficient field multiplication computations. We took the same approach in our implementations.

Throughout the test, we set the extension degree n to one of 1024, 2048, or 3072, so that they matched the sizes used in our prime field experiments, which will be presented later, but readers may want to recall [5] that the DLPs on $\mathbf{F}_{2^n}$ are easier than the DLPs on $\mathbf{F}_p$ when the two fields are of approximately the same size.

[3] Intel Core2Duo E6700 2.67GHz CPU, 4GB RAM. Only a single core was used in any timing measurements.

Table 3 Average time taken for 10^8 iterations of various iteration functions, measured on 206-bit prime order subgroups of $\mathbf{F}_{2^n}^{\times}$ (Unit: second; $F_{r,\ell}$ refers to tag tracing that uses r-adding walk and multiplier product length ℓ; Timings for pre-computation of $\mathscr{M}_\ell$ are given in parentheses)

n	1024	2048	3072
F_P	295.5	751.2	1390.1
F_{20}	401.8	1070.5	2014.0
$F_{4,10}$	46.7 (0.0091)	120.5 (0.018)	222.1 (0.028)
$F_{4,20}$	27.0 (0.070)	67.1 (0.15)	122.8 (0.28)
$F_{4,30}$	21.9 (0.28)	53.9 (0.68)	93.9 (1.16)
$F_{4,40}$	22.0 (0.87)	48.7 (1.93)	81.1 (3.45)
$F_{4,50}$	22.0 (1.90)	45.7 (4.43)	73.5 (7.97)
$F_{4,60}$	22.6 (3.89)	44.3 (8.97)	68.1 (15.8)
$F_{4,70}$	23.1 (7.17)	43.1 (16.7)	64.9 (28.8)
$F_{4,80}$	23.6 (11.7)	41.7 (27.5)	62.7 (49.1)
$F_{4,90}$	24.1 (18.5)	42.1 (43.8)	–
$F_{4,100}$	24.5 (28.3)	–	–
$F_{8,10}$	60.2 (0.34)	141.3 (0.71)	248.8 (1.24)
$F_{8,18}$	52.6 (11.8)	105.8 (25.1)	169.1 (44.3)
$F_{8,19}$	52.0 (16.7)	102.6 (35.3)	–
$F_{8,20}$	52.2 (24.0)	–	–

5.4.1 Speed comparisons on large groups

We wanted to compare the DLP solving algorithms under realistic parameters. However, since it is not possible to do any full DLP solving on such parameters, we measured the speed of the various iteration functions under large parameters. The comparison of full DLP solving time among different algorithms can be based on the measured function iteration speeds and the relative expected rho lengths given by (7).

We ran Pollard's original iteration function, the 20-adding walk, and the tag tracing iteration over multiple parameter sets, each for 10^8 iterations and measured the execution time. Each timing measurement started with a randomly generated 4-tuple of group generator $\mathbf{g}(x)$, DLP target $\mathbf{h}(x)$, multipliers $\mathbf{m}(x)_i$, and initial point. Each parameter set was run 100 times and the collected timings were averaged. The subgroup sizes were always taken to be 206-bit primes. For tag tracing, every combination of

$$r \in \{4,8\} \quad \text{and} \quad n \in \{1024, 2048, 3072\}$$

was used with tags of bit length

$$\mathtt{t} = \log_2 r.$$

All results are summarized in Table 3. Each row labeled as $F_{r,\ell}$ corresponds to tag tracing that is based on an r-adding walk with the multiplier product length set to ℓ. The values given in parentheses for each of the tag tracing timings are the time taken for preparation of the corresponding multiplier product table $\mathscr{M}_\ell$.

Notice that Pollard's original iteration function F_P is clearly faster than the unmodified 20-adding walk function F_{20}, regardless of the finite field size. This is because Pollard's iteration function uses squaring, which is faster than multiplication on binary fields. As indicated by (7), the rho length of Pollard's original iteration function is likely to be longer than that of the 20-adding walks by a factor of $\frac{\rho_P}{\rho_{20}} \approx 1.25$. Taking into account both this and the speed differences presented in Table 3, we can expect the rho algorithm to be faster than or at least comparable to the 20-adding walk in solving DLPs. Hence, on subgroups of the binary fields, it is reasonable to compare tag tracing with the rho algorithm rather than with the 20-adding walk.

Let us briefly discuss the data obtained for tag tracing. Going down the $n = 1024$ column and observing the data for tag tracing with $r = 4$, we see that the speed initially increases with an increase in ℓ, but slowly decreases after achieving its highest speed at $\ell = 30$. This can be understood from (4), the expected time for a single tag tracing iteration. When the effort $|\bar{\tau}|$ of computing tags matches $\frac{1}{\ell}$-th of the full product computation effort $|F_r|$, any further increase in ℓ has very little effect in reducing the average tag tracing iteration time. On the other hand, the increase in table size that accompanies an increase in ℓ slowly worsens the performance. Such an initial increase in speed followed by a decrease is also observable for $n = 2048$, although only barely. The size of online memory available to us and the primitive table structure in our implementation limited the range of ℓ that we could use, and this prevented us from observing similar behavior in the other cases, including all three $r = 8$ cases.

For the $n = 1024$ case, of all the parameters that we tried, tag tracing gave its best performance at $r = 4$ and $\ell = 30$, achieving a speed of 21.9 seconds per 10^8 iterations. In comparison, the speed of the rho algorithm for $n = 1024$ was 295.5 seconds per 10^8 iterations. Since we know from (7) that the rho length of a 4-adding walk is expected to be slightly longer than that of the rho algorithm by a factor of $\frac{\rho_4}{\rho_P} \approx 1.04$, the time ratio for full DLP solving by these two algorithms is expected to be $\frac{295.5}{21.92}\frac{1}{1.04} \approx 13.0$, with the tag tracing at an advantage. Corresponding figures that compare the rho algorithm with tag tracing on binary fields of $n = 2048$ and 3072 are 17.3 and 21.3, respectively. Thus, on subgroups of the binary field, tag tracing achieves approximately 13.0, 17.3, and 21.3 factor speedups in DLP solving compared to existing algorithms, when the field extension degree is 1024, 2048, and 3072, respectively. Note that, since our use of tag tracing was limited by the amount of available online memory, better results, especially in the $n = 3072$ case, can be anticipated with larger resources.

The issue of pre-computation time for the table $\mathscr{M}_\ell$ needs to be addressed. Our speed comparison thus far has completely ignored the pre-computation time required for tag tracing. This requires some explanation, since, for example, the entry "62.7(49.1)" in Table 3 for $F_{4,80}$ at $n = 3072$, shows table preparation time comparable to the tag tracing execution time. To this end, we should recall that the timings provided in Table 3 were for $10^8 \approx 2^{26.6}$ iterations and that we were testing on a 206-bit sized group. To solve a DLP on a group of 206-bit prime order, we would expect to compute $O(2^{103})$ iterations of tag tracing. Hence, compared to the expected full DLP solving time of $O(62.7 \times 2^{76.4})$ seconds, the 49.1 seconds of table preparation time is indeed negligible. Even if we had worked with an 80-bit sized group (of 40-bit security), any of the pre-computation times presented in Table 3 would be negligible in comparison to the total tag tracing iteration time, so the complete dismissal of table preparation time is justified.

5.4.2 DLP solving on small groups

In the previous subsection, we provided test results that compared the performance of tag tracing with existing algorithms on subgroups of the binary fields. These tests were done with large parameters, and we had to supplement the test results with certain arguments before being able to claim advantage in the total DLP solving time. In particular, the collision

Table 4 Time spent on random walks until collision detection and their ratios, measured on 51-bit order subgroups of $\mathbf{F}_{2^n}^{\times}$

n	1024	2048	3072
F_P	176.0 sec	432.5 sec	808.4 sec
F_{20}	170.3 sec	514.5 sec	978.7 sec
$F_{4,30}$, $F_{4,80}$, $F_{4,80}$	12.5 sec	23.4 sec	38.8 sec
$F_{8,19}$, $F_{8,19}$, $F_{8,18}$	24.4 sec	49.9 sec	80.6 sec
F_P / ($F_{4,30}$, $F_{4,80}$, $F_{4,80}$)	14.1	18.5	20.8
F_{20} / ($F_{4,30}$, $F_{4,80}$, $F_{4,80}$)	13.6	22.0	25.2
F_P / ($F_{8,19}$, $F_{8,19}$, $F_{8,18}$)	7.2	8.7	10.0
F_{20} / ($F_{8,19}$, $F_{8,19}$, $F_{8,18}$)	7.0	10.3	12.1

detection mechanism of tag tracing was not tested. To fully verify that tag tracing performs as expected, we ran complete DLP solving tests on small groups. Tag tracing was set to use the distinguished path segment approach and the other two algorithms were set to use the original distinguished point method.

As before, every combination of

$$(r,n) \in \{4,8\} \times \{1024, 2048, 3072\}$$

was tried with

$$\mathtt{t} = \log_2 r,$$

but the group size was now always set to 51-bit primes. The multiplier product length ℓ was fixed as follows, depending on r and n.

n	1024	2048	3072
$r = 4$ case	30	80	80
$r = 8$ case	19	19	18

As can be observed in Table 3, these are the optimal ℓ values among those within our resource constraints, for each of the six parameter setups.

We measured the time spent by each algorithm in traveling the random walk until the solution to the DLP was returned. These measurements were averaged over 100 randomly generated starting points and multiplier sets, and the results are summarized in Table 4. The test results agree with our previous argument that the rho algorithm is comparable to or faster than the 20-adding walk in solving DLPs on binary fields. One can also see that tag tracing with $r = 4$ is 13.6, 18.5, and 20.8 times faster than existing algorithms on binary fields of extension degrees 1024, 2048, and 3072, respectively. These figures are not too different from the 13.0, 17.3, and 21.3 factor advantages previously predicted in Section 5.4.1, for large groups.

We must admit that the figures in Table 4 for tag tracing do not contain the time spent on preparation of the table $\mathscr{M}_\ell$, and that we had intentionally stated the figures as the time spent on random walks rather than as the DLP solving time. In fact, the above mentioned choices for r and ℓ are such that $\binom{r+\ell}{r}$ is somewhat close to $\sqrt{q} \approx 2^{25.5}$, and the table preparation time is not negligible in comparison to the random walk time. Hence, the claimed ratio of the execution time is not the correct ratio of the full DLP solving time in the case of these small groups. That said, we must note that the time for the preparation of $\mathscr{M}_\ell$ under a fixed r and ℓ depends on the size of $\mathbf{F}_{p^n}$ rather than on the order q of the target group and hence should remain almost constant as q is increased. On the other hand, as discussed at the end of Section 5.4.1, the random walk time of $\sqrt{q}$ order quickly increases with q to the point where the table preparation time becomes negligible in comparison. Since the random walk time ratio should not change too much with q, the claimed performance ratio is correct as the full DLP solving time ratio when the group size q is very large.

6 Application to Subgroups of Prime Fields

In this section, we show how to apply the tag tracing approach to subgroups of the prime field and present some implementation results. Throughout this section, G is a prime order q subgroup of the prime field $\mathbf{F}_p$. While the ideas discussed in this section are simple, the details are somewhat complicated, and we suggest that our readers understand Section 5 thoroughly before trying to understand these details.

We continue to use the notation $\mathrm{mod}_x y$, introduced in the previous section, which signifies the unique integer between 0 and $x-1$ congruent to y modulo x.

6.1 Overview

We first advise readers that almost every equation in this subsection is either incorrect or correct only in a certain sense. They are to be understood as preliminary or approximate versions of the correct equations that will appear in later subsections. Some of the variables used in this subsection are fixed to explicit values, but these variables should be allowed to vary, beginning with the next subsection.

Let us try to extend the tag tracing approach used with finite fields of large extension degree to that on prime fields. To reduce the confusion between the two p's appearing in $\mathbf{F}_{p^n}$ and $\mathbf{F}_p$, in this subsection, we confine ourselves to the binary field $\mathbf{F}_{2^n}$ case whenever we refer to arguments in the previous section that dealt with general finite fields of large extension degrees.

We fix p to a prime of 1024-bit size. Suppose that we want a tag set of size $w = 2^{32}$. In the binary field case, the tag was simply an appropriate number of the most significant bits. One might try to apply the same approach to the prime field case by taking the most significant 32 bits of the given prime field element, written in its binary expansion form, as the tag, but a more careful analogue of (8) would be to define

$$\text{tag-proposal}(\mathbf{y}) = \left\lfloor \frac{\mathrm{mod}_p \mathbf{y}}{\bar{w}} \right\rfloor, \quad \text{where} \quad \bar{w} = \left\lceil \frac{p}{w} \right\rceil. \tag{16}$$

For now, this will serve as our preliminary tag function definition. This definition will be revised later in this subsection.

Our next goal is to compute the tag value of the product $\mathbf{ym}$, where $\mathbf{y}$ and $\mathbf{m}$ are elements of the prime field. Following the lines of argument given for the binary field case, we fix a small positive integer $u = 2^{16}$ and write the element $\mathbf{y} = \sum_{i=0}^{63} y_i u^i$ in its u-ary expansion. Note that the range of index i in this sum is bound by $\log_u p \approx 64$. An (incorrect) analogue of (9) might then be written as

$$\mathrm{mod}_p(\mathbf{y}\,\mathbf{m}) = \sum_{0 \le i < 64} y_i\, \mathrm{mod}_p(u^i\, \mathbf{m}). \tag{17}$$

After noting $\mathrm{mod}_p(u^i\,\mathbf{m}) = \left\lfloor \frac{\mathrm{mod}_p(u^i\,\mathbf{m})}{\bar{w}} \right\rfloor \bar{w} + \mathrm{mod}_{\bar{w}}\big(\mathrm{mod}_p(u^i\,\mathbf{m})\big)$, the above may equivalently be stated as

$$\mathrm{mod}_p(\mathbf{y}\,\mathbf{m}) = \sum_{0\le i<64} y_i \left\lfloor \frac{\mathrm{mod}_p(u^i\,\mathbf{m})}{\bar{w}} \right\rfloor \bar{w} + \sum_{0\le i<64} y_i\,\mathrm{mod}_{\bar{w}}\big(\mathrm{mod}_p(u^i\,\mathbf{m})\big), \tag{18}$$

and we would like this to imply an analogue of (10), which might take the form

$$\left\lfloor \frac{\mathrm{mod}_p(\mathbf{y}\,\mathbf{m})}{\bar{w}} \right\rfloor = \sum_{0\le i<64} y_i \left\lfloor \frac{\mathrm{mod}_p(u^i\,\mathbf{m})}{\bar{w}} \right\rfloor. \tag{19}$$

If this were true, the right-hand side could be used to define an auxiliary tag function that is easy to compute. Unfortunately, none of the above three equations are correct.

There are two major problems. The first is that, contrary to the binary field case, it is quite likely that the right-hand side integer sum of (17) will be larger than the modulus p. The summation result must be reduced modulo p for the equality to hold. The second difficulty we face is somewhat similar to the first. The second summation term $\sum_i y_i\,\mathrm{mod}_{\bar{w}}\big(\mathrm{mod}_p(u^i\,\mathbf{m})\big)$ appearing in (18) is likely to be larger than $\bar{w}$, which implies that even if (17), which is equivalent to (18), was true, our naive wish (19) would still be false.

The first of these two difficulties can surely be removed by applying a $\mathrm{mod}_p(\;)$ operation to the entire right-hand side of (17) or (18), but such an operation would be costly to compute, unless p is of special form, and would also deter further development of arguments toward (19). To resolve this, we first note that, although the second summation term of (18) is greater than $\bar{w}$, it is considerably smaller than p. Explicitly, we have

$$\sum_{0\le i<64} y_i\,\mathrm{mod}_{\bar{w}}\big(\mathrm{mod}_p(u^i\,\mathbf{m})\big) < 64\,u\,\bar{w} = 2^{22}\,\bar{w} \approx 2^{22}\,\frac{p}{w} \approx \frac{p}{2^{10}}. \tag{20}$$

Since this is much smaller than the modulus p, the equality

$$\mathrm{mod}_p(\mathbf{y}\,\mathbf{m}) = \mathrm{mod}_p\left(\sum_{0\le i<64} y_i \left\lfloor \frac{\mathrm{mod}_p(u^i\,\mathbf{m})}{\bar{w}} \right\rfloor \bar{w} \right) + \sum_{0\le i<64} y_i\,\mathrm{mod}_{\bar{w}}\big(\mathrm{mod}_p(u^i\,\mathbf{m})\big) \tag{21}$$

is likely to hold with good probability. The next step is to note that the definition of $\bar{w}$ implies a very good approximation $p \approx w\bar{w}$, so that we have

$$\mathrm{mod}_p(k\,\bar{w}) \approx \mathrm{mod}_{w\bar{w}}(k\,\bar{w}) = \big\{\mathrm{mod}_w k\big\}\bar{w},$$

for any integer k. This observation allows us to modify (21) into

$$\mathrm{mod}_p(\mathbf{y}\,\mathbf{m}) \approx \left\{ \mathrm{mod}_w \left(\sum_{0\le i<64} y_i \left\lfloor \frac{\mathrm{mod}_p(u^i\,\mathbf{m})}{\bar{w}} \right\rfloor \right) \right\} \bar{w} + \sum_{0\le i<64} y_i\,\mathrm{mod}_{\bar{w}}\big(\mathrm{mod}_p(u^i\,\mathbf{m})\big). \tag{22}$$

We have arrived at a version of (18) that is likely to be correct, at the cost of a single reduction modulo $w = 2^{32}$, which is much easier to compute than a modulo p reduction.

To remedy the second of our two difficulties, we determine how far past the $\bar{w}$ boundary the second sum of (22) can extend and then divide it out from both sides of (19). In more explicit terms, after noting the $2^{22}\bar{w}$ bound that appeared in (20), we first divide both sides of (22) by $\bar{w}$ and then divide the quotient once more by $\bar{t} = 2^{22}$, and we can claim that

$$\left\lfloor \left\lfloor \frac{\mathrm{mod}_p(\mathbf{y}\,\mathbf{m})}{\bar{w}} \right\rfloor \frac{1}{\bar{t}} \right\rfloor \approx \left\lfloor \left\{ \mathrm{mod}_w \left(\sum_{0\le i<64} y_i \left\lfloor \frac{\mathrm{mod}_p(u^i\,\mathbf{m})}{\bar{w}} \right\rfloor \right) \right\} \frac{1}{\bar{t}} \right\rfloor \tag{23}$$

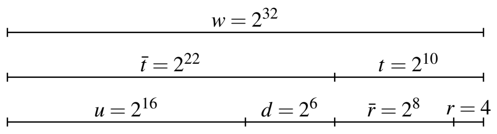

Fig. 1 Example tag tracing parameters for 1024-bit prime fields

is likely to hold true. Our definition of tag function is revised accordingly to

$$\text{tag-proposal}(\mathbf{y}) = \left\lfloor \left\lfloor \frac{\mathrm{mod}_p \mathbf{y}}{\bar{w}} \right\rfloor \frac{1}{\bar{t}} \right\rfloor . \tag{24}$$

For our specific parameter setting, this definition produces 10-bit tags.

Note that even if (22) was an exact identity, our solution to the second problem works only probabilistically, because carries may ripple upward to even the most significant parts when the two right-hand side terms of (22) are added.

The right-hand side of formula (23) is easy to compute when pre-computed $\left\lfloor \frac{\mathrm{mod}_p(u^i \mathbf{m})}{\bar{w}} \right\rfloor$ values are provided, since the integer products involved in the computation are of small numbers. The auxiliary tag function can now be defined based on the right-hand side of (23). Any error hidden in the approximation notation needs to be filtered out through the projection function.

6.2 Tag Tracing Setting

Let us now present the correct versions of the various functions that need to be provided for tag tracing on a prime field. We ask readers to mentally set aside all explicit formulas given in the previous subsection and to instead retain only the basic ideas.

6.2.1 Parameter setup

We fix the index set size r and the multiplier product length ℓ in such a way that the time and storage complexities given by Lemma 2 and Lemma 3 are manageable and are also insignificant in comparison to $\sqrt{q}$. The tag set $\mathscr{T} = \{0, 1, 2, \ldots, t-1\}$ is taken to be of size $t = r\bar{r}$, a multiple of r. Next, we take a positive integer u and set $d = \lceil \log_u(p-1) \rceil$. Finally, we choose an integer $\bar{t} > d(u-1)$. We fix the notation $w = t\bar{t}$ and assume that all parameters have been chosen in such a way that $w < p^{\frac{1}{3}}$.

The optimal choices for these parameters will depend on many factors, including the size of prime p, the resources available, and the speed of large integer multiplications. The parameter set given in Figure 1 may be appropriate for use on a modern PC when the prime p is of 1024-bit size. Readers may keep this parameter set example in mind to facilitate their understanding of the further material in this section.

Even though our description of the parameter set began with the smallest integer r and ended at the larger integer w, in practice, the choices are not so straightforward. The most computationally intensive operation for tag tracing on prime fields will be integer multiplication modulo w. Hence a natural choice for w is to match it to the word size of the intended implementation platform. The parameters r, $\bar{r}$, and u need to be chosen so that they all fit, together with $d \approx \log_u p$, inside w.

We have already mentioned in Section 4.5 that a small r is likely to be desirable, and hence $r = 4$ or $r = 8$ will be good choices. Recall that $\bar{\tau}$ is designed to be almost a function of the product of its two given inputs and that any existing fluctuations are to be filtered out by $\bar{\sigma}$ in creating the reliable index $\bar{s}$. The ratio $\bar{r}$ between tag size t and index size r is a measure of the buffer size that can distill the unreliability and is inversely correlated to the P_{fail} term of (6). The parameter d, which is determined by the choice of u, will be the number of the aforementioned modulo w multiplications that are required on each tag tracing iteration. Hence, it is desirable to make both $\bar{r}$ and u large, and since they need to fit within w, one must decide on a suitable balance between the two.

6.2.2 Tag function

Let us set $\bar{w} = \lceil \frac{p}{w} \rceil$ and summarize the notation given so far as $r\bar{r} = t$, $t\bar{t} = w$, $w\bar{w} \approx p$, $d \approx \log_u p$, and $\bar{t} \approx du$. Define the tag function $\tau : G \to \mathscr{T} = \{0,1,\ldots,t-1\}$ as

$$\tau(\mathbf{y}) = \left\lfloor \frac{\operatorname{mod}_p \mathbf{y}}{\bar{t}\,\bar{w}} \right\rfloor. \tag{25}$$

Since $0 \le w\bar{w} - p$ implies $\frac{p-1}{\bar{t}\bar{w}} < \frac{w}{\bar{t}} = t$, we know that the above quotient does lie in $\mathscr{T}$. For later use, we briefly note that the definition of $\bar{w}$ implies

$$w\bar{w} - p < w. \tag{26}$$

We also remark that our assumption of $w < p^{\frac{1}{3}}$, given in Section 6.2.1, implies $p^{\frac{2}{3}} < \bar{w}$, and thus

$$w^2 < \bar{w}. \tag{27}$$

The projection $\sigma : \mathscr{T} \to \mathscr{S} = \{0,1,\ldots,r-1\}$ is defined as

$$\sigma(x) = \left\lfloor \frac{x}{\bar{r}} \right\rfloor \tag{28}$$

and this fixes the index function $s = \sigma \circ \tau : G \to \mathscr{S}$. The following proposition shows that we can expect this to be roughly pre-image uniform.

Proposition 1 *Assuming that the elements of G are randomly distributed over $\mathbf{F}_p$, we can expect the index function s to be roughly pre-image uniform in the sense that*

$$\left| \begin{array}{l} \operatorname{Prob}[s(\mathbf{y}) = i \text{ for random } \mathbf{y} \in G] \\ \quad - \operatorname{Prob}[s(\mathbf{y}) = j \text{ for random } \mathbf{y} \in G] \end{array} \right| < \frac{w}{p} < \frac{1}{p^{\frac{2}{3}}}$$

for any $i, j \in \mathscr{S}$.

Proof Let us view τ and $s = \sigma \circ \tau$ as having been defined on all of $\mathbf{F}_p$, rather than on just $G \subset \mathbf{F}_p^{\times}$. Note that $\varepsilon := t \cdot \bar{t}\bar{w} - p = w\bar{w} - p < w = \bar{t}t < \bar{t}\bar{w}$. This implies that for each fixed $k = 0,\ldots,t-2$, there are exactly $\bar{t}\bar{w}$ elements $0 \le \mathbf{y} < p$ with $\tau(\mathbf{y}) = k$ and that there are $\bar{t}\bar{w} - \varepsilon$ elements satisfying $\tau(\mathbf{y}) = t-1$.

In terms of the index values, since $t = r\bar{r}$, this implies that for each fixed $i = 0,\ldots,r-2$, there are exactly $\bar{r}\bar{t}\bar{w}$ elements $0 \le \mathbf{y} < p$ with $s(\mathbf{y}) = i$ and that there are $\bar{r}\bar{t}\bar{w} - \varepsilon$ elements satisfying $s(\mathbf{y}) = r-1$. Thus, the maximal difference between the pre-image sizes of s is ε. The difference between the image probabilities of s, as a function defined on all of $\mathbf{F}_p$, can now be seen to be at most $\frac{\varepsilon}{p} < \frac{w}{p}$. Assuming that G is randomly distributed within $\mathbf{F}_p$, one may say the same for G. The final inequality follows from the assumption $w < p^{\frac{1}{3}}$. □

After reading the proof of this proposition, it should be clear that the index function $s = \sigma \circ \tau$ may directly be computed through

$$s(\mathbf{y}) = \left\lfloor \frac{\mathrm{mod}_p \mathbf{y}}{\bar{r}\bar{t}\bar{w}} \right\rfloor. \tag{29}$$

6.2.3 Auxiliary functions

We should now construct the auxiliary index function $\bar{s} : G \times \mathscr{M}_\ell \to \mathscr{T} \cup \{\text{fail}\}$, which is essentially equal to s.

Let us present a given $\mathbf{y} \in \mathbf{F}_p$ in its u-ary expansion. Specifically, we can always write

$$\mathrm{mod}_p\, \mathbf{y} = \sum_{i=0}^{d-1} y_i\, u^i$$

with each $0 \le y_i < u$. The definition of $d = \lceil \log_u(p-1) \rceil$ clearly implies that the range for the summation index should be as given here. Next, for each $0 \le i \le d-1$ and any $\mathbf{m} \in \mathbf{F}_p$, we can write

$$\mathrm{mod}_p\, u^i\mathbf{m} = \hat{m}_i \bar{w} + \check{m}_i \tag{30}$$

with $0 \le \hat{m}_i \le \frac{p-1}{\bar{w}} < w$ and $0 \le \check{m}_i < \bar{w}$. The right-hand side is the result of usual integer division with remainder of $\mathrm{mod}_p u^i\mathbf{m}$ by $\bar{w}$. Using this notation, we define the auxiliary tag function $\bar{\tau} : G \times \mathscr{M}_\ell \to \mathscr{T}$ to be

$$\bar{\tau}(\mathbf{y}, \mathbf{m}) = \left\lfloor \frac{\mathrm{mod}_w \sum_{i=0}^{d-1} y_i \hat{m}_i}{\bar{t}} \right\rfloor. \tag{31}$$

Readers may notice that this is the right-hand side of (23). Since $w = t\,\bar{t}$, it is clear that each computed quotient lies in $\mathscr{T}$.

Let us check how close $\bar{\tau}(\mathbf{y}, \mathbf{m})$ and $\tau(\mathbf{ym})$ are to each other.

Lemma 4 *For any $\mathbf{y}, \mathbf{m} \in \mathbf{F}_p$, the tag value $\tau(\mathbf{ym})$ for the product is equal to either $\bar{\tau}(\mathbf{y}, \mathbf{m})$ or $\mathrm{mod}_t(\bar{\tau}(\mathbf{y}, \mathbf{m}) + 1)$.*

Proof Noting that $w > \bar{t}$, given any integer k, we can perform integer division with remainder of k by w, and then divide the obtained remainder once more by $\bar{t}$ to write

$$k = k_2 w + k_3, \qquad \text{where } k_2 = \left\lfloor \frac{k}{w} \right\rfloor \text{ and } k_3 = k - \left\lfloor \frac{k}{w} \right\rfloor w < w, \text{ and}$$

$$k = k_2 w + k_1 \bar{t} + k_0, \quad \text{where } k_1 = \left\lfloor \frac{k_3}{\bar{t}} \right\rfloor \text{ and } k_0 = k_3 - \left\lfloor \frac{k_3}{\bar{t}} \right\rfloor \bar{t} < \bar{t}.$$

We use this observation to write

$$\sum_{i=0}^{d-1} y_i \hat{m}_i = c_2 w + c_1 \bar{t} + c_0,$$

where $c_1\bar{t} + c_0 < w$ and $c_0 < \bar{t}$. It should be clear from definition (31) that $\bar{\tau}(\mathbf{y}, \mathbf{m}) = c_1$.

In the notation used in defining $\bar{\tau}(\mathbf{y},\mathbf{m})$, we may write

$$\begin{aligned}\mathbf{ym} = \sum_{i=0}^{d-1} y_i u^i \mathbf{m} &\equiv \Big(\sum_{i=0}^{d-1} y_i \hat{m}_i\Big)\bar{w} + \sum_{i=0}^{d-1} y_i \check{m}_i \pmod{p}\\ &\equiv c_1 \bar{t}\bar{w} + c_0\bar{w} + c_2(w\bar{w}-p) + \sum_{i=0}^{d-1} y_i\check{m}_i \pmod{p}. \qquad (32)\end{aligned}$$

After noting

$$c_2 = \left\lfloor \frac{\sum_{i=0}^{d-1} y_i \hat{m}_i}{w} \right\rfloor \le \frac{d(u-1)(w-1)}{w} < d(u-1) \le \bar{t}-1$$

and

$$\sum_{i=0}^{d-1} y_i \check{m}_i < d(u-1)(\bar{w}-1) < (\bar{t}-1)\bar{w},$$

we can use the relation (26) to bound the smaller terms of (32) by

$$c_0\bar{w} + c_2(w\bar{w}-p) + \sum_{i=0}^{d-1} y_i\check{m}_i < (\bar{t}-1)\bar{w} + (\bar{t}-1)w + (\bar{t}-1)\bar{w} = (2\bar{t}\bar{w}-w) + (\bar{t}w - 2\bar{w}).$$

Since (27) shows $\bar{t}w < t\bar{t}w = w^2 < 2\bar{w}$, we may conclude

$$0 \le c_0\bar{w} + c_2(w\bar{w}-p) + \sum_{i=0}^{d-1} y_i\check{m}_i < 2\bar{t}\bar{w} - w. \qquad (33)$$

Recall that $c_1 = \bar{\tau}(\mathbf{y},\mathbf{m})$, as an element of the tag set, must be less than or equal to $t-1$. Let us temporarily restrict ourselves to the case in which c_1 is strictly less than $t-1$. In this case, we have

$$\begin{aligned}\mathbf{ym} &\equiv c_1\bar{t}\bar{w} + c_0\bar{w} + c_2(w\bar{w}-p) + \sum_{i=0}^{d-1} y_i\check{m}_i \pmod{p}\\ &< (t-2)\bar{t}\bar{w} + (2\bar{t}\bar{w}-w) = w\bar{w} - w < p,\end{aligned}$$

which implies

$$\operatorname{mod}_p \mathbf{ym} = c_1\bar{t}\bar{w} + \Big\{c_0\bar{w} + c_2(w\bar{w}-p) + \sum_{i=0}^{d-1} y_i\check{m}_i\Big\}. \qquad (34)$$

Recalling the definition of the tag function given by (25), let us note that $\tau(\mathbf{ym})$ is the quotient obtained by performing integer division of the above value by $\bar{t}\bar{w}$. Let us temporarily denote the sum of terms inside the braces of (34) by $\{\star\}$. Since (33) has shown the non-negative integer value $\{\star\}$ to be strictly less than $2\bar{t}\bar{w}$, when integer division by $\bar{t}\bar{w}$ is performed on the right-hand side of (34), the quotient will be either c_1 or c_1+1. Here, the outcome will depend on whether $0 \le \{\star\} < \bar{t}\bar{w}$ or $\bar{t}\bar{w} \le \{\star\} < 2\bar{t}\bar{w}$. We have shown that $\tau(\mathbf{ym})$ is either $\bar{\tau}(\mathbf{y},\mathbf{m})$ or $\bar{\tau}(\mathbf{y},\mathbf{m})+1$, when $\bar{\tau}(\mathbf{y},\mathbf{m}) < t-1$.

It remains to consider the case of $c_1 = t-1$. If the value (32) happens to be strictly less than p, then (34) holds, and since the term within the braces is non-negative with $t-1 = c_1 \le \tau(\mathbf{ym}) \le t-1$, we must have $\tau(\mathbf{ym}) = c_1$. Otherwise, we can use (33) and (26) to write

$$p \le \text{value given by (32)} < (t-1)\bar{t}\bar{w} + 2\bar{t}\bar{w} - w = w\bar{w} - w + \bar{t}\bar{w} < p + \bar{t}\bar{w}.$$

This shows that $\operatorname{mod}_p \mathbf{ym}$ is equal to the right-hand side of (34) minus p, which becomes strictly less than $\bar{t}\bar{w}$, and so $\tau(\mathbf{ym}) = 0 \equiv c_1 + 1 \pmod{t}$. ☐

We now define $\bar{\sigma} : \mathscr{T} \to \mathscr{S} \cup \{\text{fail}\}$ as follows.

$$\bar{\sigma}(x) = \begin{cases} \text{fail} & \text{if } x \equiv -1 \bmod \bar{r}, \\ \lfloor \frac{x}{\bar{r}} \rfloor & \text{otherwise.} \end{cases} \tag{35}$$

It is easy to show that $\bar{s} = \bar{\sigma} \circ \bar{\tau}$ produces $s = \sigma \circ \tau$ values, whenever it does not fail.

Proposition 2 *When* $\bar{s}(\mathbf{y},\mathbf{m}) \neq \text{fail}$*, we have* $\bar{s}(\mathbf{y},\mathbf{m}) = s(\mathbf{ym})$.

Proof It is clear that $\lfloor \frac{k}{\bar{r}} \rfloor = \lfloor \frac{\text{mod}_t(k+1)}{\bar{r}} \rfloor$ for any $k \in \mathscr{T}$, unless $k \equiv -1 \bmod \bar{r}$. Hence, for any $\mathbf{y},\mathbf{m} \in G$ such that $\bar{\tau}(\mathbf{y},\mathbf{m}) \not\equiv -1 \bmod \bar{r}$, we have

$$\bar{s}(\mathbf{y},\mathbf{m}) = \left\lfloor \frac{\bar{\tau}(\mathbf{y},\mathbf{m})}{\bar{r}} \right\rfloor = \left\lfloor \frac{\text{mod}_t(\bar{\tau}(\mathbf{y},\mathbf{m})+1)}{\bar{r}} \right\rfloor.$$

Since we know from Lemma 4 that either $\bar{\tau}(\mathbf{y},\mathbf{m})$ or $\text{mod}_t(\bar{\tau}(\mathbf{y},\mathbf{m})+1)$ is equal to $\tau(\mathbf{ym})$, the above equal values must also be identical to $\lfloor \frac{\tau(\mathbf{ym})}{\bar{r}} \rfloor = s(\mathbf{ym})$. □

The definitions of tag and auxiliary tag functions for the prime field case are more complicated than what we saw for the finite fields of large extension degrees. This is due to the presence of additive carries which does not appear with the polynomial basis representation for finite fields of large extension degrees.

6.3 Tag Tracing

We are now ready to begin tag tracing on subgroups of the prime field.

6.3.1 Tag tracing iterations

Using the proof of Lemma 3 appearing in Section 4.2.1 as a construction guide, we build the table $\mathscr{M}_\ell$. Recall from Section 4.2.3 that each table entry was to consist of three components. These are the multiplier combination information, the group element, and the exponent information. For each $\mathbf{m} \in \mathscr{M}_\ell$, we add the following fourth component to each table entry.

$$\left(\left\lfloor \frac{\text{mod}_p(u^0\mathbf{m})}{\bar{w}} \right\rfloor, \left\lfloor \frac{\text{mod}_p(u^1\mathbf{m})}{\bar{w}} \right\rfloor, \ldots, \left\lfloor \frac{\text{mod}_p(u^{d-1}\mathbf{m})}{\bar{w}} \right\rfloor \right) \tag{36}$$

Notice that these are the $\hat{m}_i$ appearing in (30) and that they are analogous to (12) used for finite fields of large extension degrees.

We can now follow the discussion in Section 4.2.3 to step through each iteration of tag tracing. The elements of G are written in u-ary representation so that we may quickly compute the index s, based on Proposition 2 and using equations (31) and (35). Note that as long as u is a power of 2, the u-ary representation will not slow down the occasional full product computations inordinately. In fact, for software implementations, if w corresponds to the natural word size of the implementation platform and the length of u matches a byte boundary, we can use type castings to almost remove the cost of conversions between w-ary and u-ary representations of any group element.

The distinguishing property for collision detection may be set as explained in Section 4.3.1. With the distinguished path segment approach, full products are required every time we reach ℓ iterations, or when we have an $\bar{s}$ calculation failure, or when the current point is found to be a distinguished point. The path segment approach is an effective collision detection method in the sense that it does not necessitate any extra full product computation.

6.3.2 Time complexity

Let us use the notation $\mathrm{Mul}(k)$ for the cost of multiplication modulo an integer of k-bit size and write $[\![x]\!]$ for the bit length of an integer x.

From equations (31) and (35), we can see that the cost of evaluating the auxiliary index function $\bar{s}$ is d multiplications modulo w, $d-1$ additions modulo w, and two integer divisions. Thus, ignoring the small fixed number of divisions and the relatively cheaper additions, we can say that the $\bar{s}$ evaluation cost is approximately $d\,\mathrm{Mul}([\![w]\!])$. Further, on the basis of definitions (31) and (35), we can reasonably claim that the probability of the auxiliary index function $\bar{s}$ returning failure is $\frac{1}{\bar{r}}$.

Gathering the above mentioned arguments and recalling (5), we can write the time complexity of tag tracing in solving a DLP on prime order subgroups of the prime field as

$$\Big\{d\,\mathrm{Mul}([\![w]\!]) + \Big(\frac{1}{\ell} + \frac{1}{\bar{r}} + \frac{1}{\Delta}\Big)\mathrm{Mul}([\![p]\!])\Big\}\Big\{\rho_r + \Delta\Big\}, \tag{37}$$

where ρ_r denotes the expected rho length of an r-adding walk. For example, if we take $[\![p]\!] = 1024$, together with the parameters given in Figure 1, and assume $\mathrm{Mul}(k) = k^2$, this complexity would be $\big\{2^{16} + \big(\frac{1}{\ell} + \frac{1}{2^8} + \frac{1}{\Delta}\big)2^{20}\big\}\big\{\rho_r + \Delta\big\}$.

One may want to be slightly more precise when considering dedicated hardware implementations. The multiplication that we argued as costing $\mathrm{Mul}([\![w]\!])$ is performed between a $[\![w]\!]$-bit integer and a $[\![u]\!]$-bit integer. This can be cheaper than an arbitrary integer multiplication between two $[\![w]\!]$-bit integers. Furthermore, since we only require the outcome reduced modulo w, if w is taken to be a power of 2, the significant bits of the integer product simply do not need to be computed. This is in contrast to modulo p reductions, which is much more complicated.

6.3.3 Alternative distinguished point definition

While the distinguished path segment approach to distinguished points given in Section 4.3.1 works efficiently for the current situation, we discuss application of the distinguished point approach given in Section 4.3.3 to provide an example of its usage.

We redefine $\bar{\sigma} : \mathscr{T} \to \mathscr{S} \cup \{\text{fail}\}$ of (35) to

$$\bar{\sigma}(x) = \begin{cases} \text{fail} & \text{if } x = 0, \\ \text{fail} & \text{if } x \equiv \text{-1} \bmod \bar{r}, \\ \lfloor \frac{x}{\bar{r}} \rfloor & \text{otherwise.} \end{cases}$$

It is evident that $\bar{s} = \bar{\sigma} \circ \bar{\tau}$ still agrees with the index s values, when it does not fail. A point $\mathbf{y} \in G$ is defined to be a distinguished point if $\tau(\mathbf{y}) = 0$ and if, in addition, a certain fixed number of its least significant bits are zero. We use the LSBs rather than the MSBs for the extra condition, since the MSBs could somehow be correlated to the tag values as given by (25), and hence the use of the LSBs will allow for easier control over the probability of distinguished point occurrences.

It can be seen from Proposition 2 that for the event $\tau(\mathbf{g}_i\mathbf{m}) = 0$ to take place, we must have either $\bar{\tau}(\mathbf{g}_i, \mathbf{m}) = t-1$ or $\bar{\tau}(\mathbf{g}_i, \mathbf{m}) = 0$. Hence $\mathbf{g}_j = \mathbf{g}_i\mathbf{m}$ can be a distinguished point only when $\bar{\tau}(\mathbf{g}_i, \mathbf{m})$ fails, and in such a case the full product $\mathbf{g}_j$ is available. This approach is slightly less efficient than the distinguished path segment approach, since we had to impair $\bar{s}$ into giving more failure returns.

6.4 Asymptotic Complexity

As in Section 5.3, let us consider the asymptotic complexity of tag tracing on subgroups of the prime field as the base field size p is increased, while the group size q is kept fixed. Based on Table 2, we may claim that the asymptotic behavior of tag tracing under an increase in group size q is likely to be $O(\sqrt{q})$ when $r \geq 4$ is used, but this subsection considers the asymptotic behavior of tag tracing when the size of the finite field containing the group, rather than the size of the group, is increased.

Let us first slightly simplify the complexity of tag tracing given by (37).

Lemma 5 *Suppose that the parameters for tag tracing that is based on r-adding walks are chosen in such a way that $\ell \approx \bar{r}$ and $d = O(\bar{r})$ and that the average distance Δ between distinguished points is set to satisfy $\frac{\rho_r}{\Delta} = O(1)$. Then the time complexity of tag tracing in solving a single DLP on a cyclic subgroup of $\mathbf{F}_p^{\times}$ is*

$$O(d)\,\mathrm{Mul}(\llbracket w \rrbracket)\rho_r + O(1)\,\mathrm{Mul}(\llbracket p \rrbracket).$$

Proof The computation that is expected to be consumed by tag tracing until the DLP solution is returned is given by (37). Under the $\frac{\rho_r}{\Delta} = O(1)$ condition, we can write

$$\frac{\mathrm{Mul}(\llbracket p \rrbracket)}{\Delta}\big(\rho_r + \Delta\big) = O(1)\,\mathrm{Mul}(\llbracket p \rrbracket)$$

concerning the final term. To deal with the other three terms, we note that $\rho_r + \Delta \leq 2\rho_r$, under the $\frac{\rho_r}{\Delta} = O(1)$ condition. The computational complexity must thus be at most

$$\Big\{ d\,\mathrm{Mul}(\llbracket w \rrbracket) + \frac{O(1)}{\bar{r}}\,\mathrm{Mul}(\llbracket p \rrbracket) \Big\} 2\rho_r + O(1)\,\mathrm{Mul}(\llbracket p \rrbracket), \tag{38}$$

where we have additionally applied $\ell \approx \bar{r}$.

We now focus on the $\frac{O(1)}{\bar{r}}\,\mathrm{Mul}(\llbracket p \rrbracket)$ term. Since we always have $\llbracket u \rrbracket \leq \llbracket w \rrbracket$ and since Mul( ) is at most quadratic in its input, we have

$$\frac{1}{\bar{r}}\,\mathrm{Mul}(\llbracket p \rrbracket) \leq \frac{1}{\bar{r}}\,\mathrm{Mul}\Big(\frac{\llbracket p \rrbracket}{\llbracket u \rrbracket} \cdot \llbracket w \rrbracket\Big) \approx \frac{1}{\bar{r}}\,\mathrm{Mul}(d\,\llbracket w \rrbracket) \leq \frac{d}{\bar{r}}\,d\,\mathrm{Mul}(\llbracket w \rrbracket).$$

Hence, under the condition $d = O(\bar{r})$, the term $\frac{O(1)}{\bar{r}}\,\mathrm{Mul}(\llbracket p \rrbracket)$ must be $O(d)\,\mathrm{Mul}(\llbracket w \rrbracket)$. Our claim is now evident from (38). $\square$

The complexity of a single r-adding iteration, in the original style, is typically quadratic in $\llbracket p \rrbracket$, because the complexity of a single multiplication in the prime field $\mathbf{F}_p$ is typically quadratic. Let us show that one can run a single iteration of tag tracing with $O(\llbracket p \rrbracket \log \llbracket p \rrbracket)$ complexity.[4]

Proposition 3 *The parameters for tag tracing can be chosen so that, on prime order q subgroups of $\mathbf{F}_p^{\times}$, the time complexity of tag tracing in solving a single DLP is*

$$O(\llbracket p \rrbracket \log \llbracket p \rrbracket)\rho_r + O(1)\,\mathrm{Mul}(\llbracket p \rrbracket).$$

When the FFT method of field multiplication, which is asymptotically faster than classical methods, is used, the above can be reduced to

$$O(\llbracket p \rrbracket \log\log \llbracket p \rrbracket)\rho_r + O(1)\,\mathrm{Mul}(\llbracket p \rrbracket).$$

[4] The linear complexity claim made in our earlier paper [4] was erroneous. In the notation of [4], we overlooked the fact that the size of ω_1 depended on p.

Proof We fix r to a small positive integer and let the other parameters depend on p through the relations $\ell = \bar{r} = u = [\![p]\!]$ and $\bar{t} = du$. It is then easy to check that $d \approx \frac{\log p}{\log u} \le \log p$, so that

$$w = r\,\bar{r}\,d\,u = O\big((\log p)^3\big).$$

This implies that the condition $w < p^{\frac{1}{3}}$, which needs to be satisfied by any parameter set, is met when p is sufficiently large.

Since the conditions $d = O(\bar{r})$ and $\ell = \bar{r}$ of Lemma 5 are trivially satisfied by our choice of parameters, after additionally setting $\frac{\rho_r}{\Delta} = O(1)$, we obtain

$$O(d)\,\mathrm{Mul}([\![w]\!])\rho_r + O(1)\,\mathrm{Mul}([\![p]\!])$$

as the computational complexity. To prove our claim, it suffices to make $d\,\mathrm{Mul}([\![w]\!])$ more specific.

It is easy to check that $[\![d]\!] \approx \log \frac{\log p}{\log u} \le \log[\![p]\!] \approx [\![u]\!]$, so that

$$[\![w]\!] \le [\![r]\!] + [\![\bar{r}]\!] + [\![d]\!] + [\![u]\!] = O([\![u]\!]),$$

and we can write

$$d\,\mathrm{Mul}([\![w]\!]) \approx \frac{[\![p]\!]}{[\![u]\!]}\,\mathrm{Mul}([\![w]\!]) = \frac{[\![p]\!]}{[\![u]\!]}\,O\big(\mathrm{Mul}([\![u]\!])\big).$$

Finally, since Mul( ) is at most quadratic in its input, we have

$$d\,\mathrm{Mul}([\![w]\!]) = O([\![p]\!]\,[\![u]\!]) = O([\![p]\!]\,\log[\![p]\!]),$$

which directly implies our first claim. Then, when the FFT method of product computation [27], which gives complexity $\mathrm{Mul}(k) = k\log k$, is used, we arrive at

$$d\,\mathrm{Mul}([\![w]\!]) = O([\![p]\!]\,\log[\![u]\!]) = O([\![p]\!]\,\log\log[\![p]\!]).$$

This brings about our second claim. □

The $O(1)$ factor in front of $\mathrm{Mul}([\![p]\!])$ is roughly the number of distinguished points one is willing to manage, and this can only be much smaller than ρ_r. Hence, in practice, the first term $O([\![p]\!]\,\log[\![p]\!])\rho_r$, rather than the $O(1)\,\mathrm{Mul}([\![p]\!])$ term, is the dominating term. Furthermore, the smaller second term may be replaced by a single solution verification exponentiation if the approach of Section 4.3.4 is taken. We may state that if we exclude a finite number of full group operations, the complexity of tag tracing is $O\big([\![p]\!]\,\log[\![p]\!]\big)$ on groups of fixed size. Unlike what we saw in Section 5.3, this is no longer linear in the field size $[\![p]\!]$, but still close to being linear.

The storage complexity remains to be evaluated. Of the four components that constitute each entry of the table $\mathscr{M}_\ell$, which was explained in Section 6.3.1, the full group element expression requires $[\![p]\!]$ bits and the fourth component requires $d\,[\![w]\!]$ bits. The other two components are smaller or do not depend on p. If the parameters taken for the proof of Proposition 3 are used, we already know that $[\![w]\!] = O([\![u]\!])$. Since $d \approx \frac{[\![p]\!]}{[\![u]\!]}$, we may state that the storage requirement in bits is bounded by

$$\left\{\begin{matrix}\text{(number of entries in } M_l)\\ \times\text{ (bit-size of each entry)}\end{matrix}\right\} = \binom{\ell+r}{r}\big([\![p]\!] + d[\![w]\!]\big) = \binom{\ell+r}{r}O([\![p]\!]) = O\big(\ell^r\,[\![p]\!]\big),$$

and this is $O([\![p]\!]^{r+1})$, when $\ell \approx [\![p]\!]$.

In summary, as p is increased and q is fixed, tag tracing runs at $O\big([\![p]\!] \log[\![p]\!]\big)$ time complexity. The reduction that is obtained as compared to the original r-adding walk, which typically runs at $O([\![p]\!]^2)$ time complexity, is achieved at the cost of storage of polynomial complexity $O([\![p]\!]^{r+1})$. The use of seemly larger storage is justifiable in practice, as it is much smaller than the effective time complexity of $O\big([\![p]\!] \log[\![p]\!]\big)\rho_r$.

6.5 Implementation of Tag Tracing on Prime Fields

Let us present the test results of running tag tracing on prime order subgroups of $\mathbf{F}_p^\times$ and compare them with Pollard's original iteration function and the 20-adding walk. The implementation platform that we used is identical to the one used in binary field experiments. As before, we used the distinguished path segment approach for collision detection with tag tracing and the original distinguished point method with the other two algorithms. The prime field arithmetics functions of the NTL library [30] were used for any full group operation, so that the results would not be biased toward tag tracing. Throughout the test, whenever random primes p and q, together with a cyclic group of order q, were needed, they were generated in the style specified for DSA [18].

6.5.1 Speed comparisons on large groups

We test the performance of tag tracing on subgroups of prime fields used today and compare it with the performance of other algorithms. Since fully solving even a single DLP on practical parameters is far from possible with our resources, we measured the average time required per function iteration by each algorithm and accounted for the differences in the expected rho lengths to arrive at a claim concerning the ratio of full DLP solving time.

Referencing the recommendations of [19], we can see that primes p and q of sizes such that

$$\big([\![p]\!], [\![q]\!]\big) \in \big\{(1024,160),\ (2048,224),\ (3072,256)\big\}$$

are of interest. Our tests were conducted on all three parameter sets. The parameters for tag tracing, set size r and modulus w, were set up by taking every combination of

$$(r,\ w) \in \big\{4,8\big\} \times \big\{2^{32}, 2^{64}\big\},$$

and fixing the other parameters to

$$u = \sqrt{w}, \quad \bar{t} = 2^{\lceil \log_2 du \rceil}, \quad t = \frac{w}{\bar{t}}, \quad \text{and} \quad \bar{r} = \frac{t}{r},$$

for each chosen combination of r and w. We also used different ℓ's for each of the described parameter sets. Our primitive data structuring for table $\mathscr{M}_\ell$ and the size of online memory available to us restricted the range of ℓ that we could use, especially for the $r = 8$ cases.

We ran Pollard's original iteration function, the unmodified 20-adding walk, and tag tracing over various parameter sets, each for 10^8 iterations, and measured the execution time. Each test started from a fresh set of randomly chosen primes p and q, group generator $\mathbf{g}$, DLP target $\mathbf{h}$, multipliers $\mathbf{m}_i$, and initial point $\mathbf{g}_0$. Each parameter set was run 100 times and the measured timings were averaged.

All timing results are given in Table 5. As in Section 5.4.1, each row labeled as $F_{r,\ell}$ corresponds to tag tracing that is based on an r-adding walk with the multiplier product

Table 5 Average timings for 10^8 iterations of various iteration functions, measured on prime q order subgroups of $\mathbf{F}_p^\times$ (Unit: second; Time taken for the pre-computation of $\mathscr{M}_\ell$ are given in parentheses)

$([\![p]\!],[\![q]\!])$	$(1024,160)$		$(2048,224)$		$(3072,256)$	
w for tag tracing	2^{32}	2^{64}	2^{32}	2^{64}	2^{32}	2^{64}
F_P	379.1		1237.4		2614.6	
F_{20}	403.7		1304.6		2731.0	
$F_{4,10}$	58.1 (0.21)	55.0 (0.11)	165.4 (1.15)	157.0 (0.60)	340.7 (3.40)	310.9 (1.73)
$F_{4,20}$	33.5 (2.33)	30.5 (1.18)	93.8 (12.2)	83.9 (6.36)	194.8 (36.1)	164.5 (18.5)
$F_{4,30}$	27.1 (9.96)	24.1 (5.14)	73.1 (53.4)	63.5 (27.7)	150.9 (158.3)	120.3 (80.6)
$F_{4,40}$	25.3 (29.1)	22.8 (15.0)	64.2 (156.4)	55.1 (81.3)	129.7 (461.0)	99.5 (235.7)
$F_{4,50}$	25.1 (68.3)	22.6 (35.0)	59.4 (364.3)	50.4 (189.0)	117.7 (1074.5)	87.6 (549.0)
$F_{4,60}$	25.2 (136.4)	23.0 (70.2)	56.4 (732.5)	48.0 (380.5)	110.3 (2165.9)	79.8 (1104.0)
$F_{4,70}$	25.7 (248.2)	23.4 (128.3)	54.7 (1327.2)	46.0 (688.8)	104.8 (3910.0)	74.7 (1999.2)
$F_{4,80}$	26.0 (416.3)	23.9 (215.6)	53.3 (2224.0)	44.7 (1153.6)	101.6 (6563.0)	70.5 (3349.8)
$F_{4,90}$	26.4 (654.5)	24.3 (335.5)	52.5 (3518.5)	43.9 (1833.5)	99.2 (10468.0)	67.4 (5270.8)
$F_{4,100}$	26.9 (987.8)	24.9 (509.5)	50.6 (5379.9)	43.0 (2751.8)	–	–
$F_{8,10}$	66.7 (9.5)	65.8 (4.9)	183.6 (50.7)	175.4 (26.1)	374.4 (148.8)	341.1 (76.7)
$F_{8,19}$	54.3 (473.7)	52.8 (243.2)	127.4 (2607.5)	116.5 (1313.9)	250.3 (7541)	207.4 (3880.8)
$F_{8,20}$	53.9 (668.0)	52.8 (346.3)	121.7 (3612.4)	111.9 (1864.9)	–	–

length set to ℓ, and the values in parentheses are the time taken for the preparation of the table $\mathscr{M}_\ell$.

The table shows that Pollard's iteration function F_P is faster than the 20-adding walk function F_{20}. However, as was stated in (7), the rho length of F_P is likely to be longer than that of F_{20} by a factor of $\frac{\rho_P}{\rho_{20}} \approx 1.25$. When this is taken into account, we can expect the 20-adding walk to solve DLPs faster than the rho method. Hence we choose to compare tag tracing with the unmodified 20-adding walk rather than with the rho algorithm.

Let us discuss the measurements obtained for tag tracing. Examining the data for the $([\![p]\!],[\![q]\!]) = (1024,160)$, $w = 2^{32}$, and $r = 4$ case, we see that the 10^8 iteration timing is smallest at $\ell = 50$. To understand this, we use the corresponding parameters $u = 2^{16}$, $d = 64$, $\bar{t} = 2^{22}$, $t = 2^{10}$, and $\bar{r} = 2^8$ to rewrite the DLP solving time complexity (37) as

$$\Big\{64\,\mathrm{Mul}(32) + \Big(\frac{1}{\ell} + \frac{1}{2^8} + \frac{1}{\Delta}\Big)\,\mathrm{Mul}(1024)\Big\}\Big\{\rho_4 + \Delta\Big\}.$$

Assuming $\mathrm{Mul}(k) = k^2$, the complexity of a single iteration may be written as

$$2^{16} + \Big(\frac{1}{\ell} + \frac{1}{2^8} + \frac{1}{\Delta}\Big)2^{20}$$

Thus, at very small ℓ, the $\frac{1}{\ell}2^{20}$ term dominates the 2^{16} term and increasing ℓ leads to a direct reduction of the execution time. At $\ell = 16$ the domination shifts to the 2^{16} term and the act of increasing ℓ begins to gradually lose its effectiveness. Still, the execution time reduction continues until the disadvantage of increased table lookup time outweighs the theoretical speedup.

For the $(\llbracket p \rrbracket, \llbracket q \rrbracket) = (1024, 160)$ DLP environment, tag tracing was at its best with $r = 4$, $w = 2^{64}$, and $\ell = 50$, achieving a speed of 22.6 seconds per 10^8 iterations. In comparison, the unmodified 20-adding walk required 403.7 seconds per 10^8 iterations. Recalling from (7) that the 4-adding walk is likely to yield a rho length that is approximately 1.31 times that of the 20-adding walk, we can state that a DLP is likely to be solved approximately $\frac{403.7}{22.6}\frac{1}{1.31} \approx 13.6$ times faster by tag tracing with $r = 4$ than by the original 20-adding walk. Corresponding speed ratios between 20-adding walks and tag tracing at $r = 4$, for the DLP environments $\llbracket p \rrbracket = 2048$ and $\llbracket p \rrbracket = 3072$, are 23.3 and 31.2, respectively. It seems safe to claim that tag tracing achieves over 10, 20, and 30 times speedup of DLP solving on subgroups of the prime field, when the bit size of the base field is 1024, 2048, and 3072, respectively.

Tag tracing results with $r = 8$ were less satisfactory than those with $r = 4$, which seems mostly due to our poor use of storage. Even in the $r = 4$ case, the available online memory was the main restriction on the choice of ℓ, so a higher speedup than what we achieved should be possible for the $\llbracket p \rrbracket = 2048$ and $\llbracket p \rrbracket = 3072$ cases with larger ℓ.

We have experienced through various tweaks that table lookups to $\mathscr{M}_\ell$ present a considerable fraction of the time taken by each tag tracing iteration. Unlike our primitive testing, large scale implementation of tag tracing would require the use of advanced hash table techniques that allow constant time table lookups.

As discussed at the end of Section 5.4.1, the time required for the preparation of table $\mathscr{M}_\ell$ can be safely ignored when discussing DLP solving time on large groups. Even though the pre-computation timings listed in Table 5 are not small, they will be negligible in comparison to the full DLP solving time if the 10^8 iteration timings are scaled to the number of required iterations expected for a large group.

6.5.2 DLP solving on small groups

To verify that tag tracing is fully operational, we ran full DLP solving tests on groups of small prime size q, where $\llbracket q \rrbracket = 50$. The DLP environment was fixed by choosing prime p that satisfy

$$\llbracket p \rrbracket \in \{1024, 2048, 3072\}.$$

For tag tracing, the two choices

$$r \in \{4, 8\}$$

were used. The other parameters were fixed to

$$w = 2^{64}, \quad u = 2^{32}, \quad \bar{t} = 2^{\lceil \log_2 du \rceil}, \quad t = w/\bar{t}, \quad \text{and} \quad \bar{r} = t/r.$$

The choice of ℓ was made as follows, according to the data collected in Table 5.

$\llbracket p \rrbracket$	1024	2048	3072
$r = 4$ case	50	100	90
$r = 8$ case	19	20	19

Table 6 Time spent on random walks until collision detection, measured on 50-bit order subgroups of $\mathbf{F}_p^\times$

$[\![p]\!]$	1024	2048	3072
F_P	178.36 sec	619.91 sec	1395.41 sec
F_{20}	152.43 sec	474.12 sec	1068.25 sec
$F_{4,50}, F_{4,100}, F_{4,90}$	11.93 sec	22.17 sec	33.51 sec
$F_{8,19}, F_{8,20}, F_{8,19}$	21.3 sec	47.2 sec	80.7 sec
F_P / ($F_{4,50}, F_{4,100}, F_{4,90}$)	14.95	27.96	41.64
F_{20} / ($F_{4,50}, F_{4,100}, F_{4,90}$)	12.78	21.39	31.88
F_P / ($F_{8,19}, F_{8,20}, F_{8,19}$)	8.4	13.1	17.3
F_{20} / ($F_{8,19}, F_{8,20}, F_{8,19}$)	7.2	10.0	13.2

For each prime field size and the choice of $r = 4$ or $r = 8$, the chosen ℓ is the optimal value among those we had tried for large groups.

We measured the time that each algorithm spent traveling the random walk until the solution to the DLP was returned. This was then averaged over 100 randomly generated starting points and multiplier sets, with a fixed but randomly selected pair of p and q. The results are summarized in Table 6. Tag tracing is faster than the existing algorithms by factors of approximately 12.8, 21.4, and 31.9 on prime fields of sizes 1024, 2048, and 3072, respectively. These factors lie within the same range as stated in Section 6.5.1 for large groups. We can safely claim that tag tracing is more than 10, 20, and 30 times faster than existing algorithms on prime fields of bit sizes 1024, 2048, and 3072, respectively.

As before, the figures in Table 6 for tag tracing do not contain the time spent on creation of the table $\mathscr{M}_\ell$. In fact, the above mentioned choices for r and ℓ are such that $\binom{r+\ell}{r}$ is somewhat close to $\sqrt{q} \approx 2^{25}$ and the table preparation time is not negligible with respect to the random walk time. Hence, the ratio of execution time that we claimed is not of the ratio of the full DLP solving time on these small groups. Still, as was discussed at the end of 5.4.1, the preparation time of $\mathscr{M}_\ell$ quickly becomes negligible in comparison to the full DLP solving time as the group size q is increased. Hence our time ratio claims are valid as full DLP solving time ratios when the group size q is large.

7 Conclusion

The main subject of this paper was the r-adding walk style of random walk generation and the distinguished point collision detection method used in DLP solving algorithms. We first noted that the essential decision made at each iteration of the r-adding walk depended on a very small amount of information. This allowed us to present tag tracing, a framework under which the group operations to be performed at each iteration could be postponed and gathered, to be processed later in a single step with the help of a pre-computed table. We also developed an extension of the distinguished point method for the tag tracing framework.

Concrete construction of tag tracing for prime order subgroups of finite fields were presented. We showed that it was possible to compute a small number of bits of the product of a random element and a preset element from a finite field in a considerably shorter time than that required for computing the full product. This translates to tag tracing being faster than the original r-adding walks. In the case of finite fields of large extension degrees, the speedup achieved was linear in the encoding length of the field elements, and in the case of prime fields, the speedup was almost linear. These complexity advantage claims are not just theoretical, as was confirmed by our implementations that showed practical improvements over the original algorithms.

The recent work [2] applies tag tracing to elliptic curve groups and reports a rather small speedup factor of 1.2. Their approach was to compute just the x-coordinates during the tag tracing steps. It would be interesting to see if approaches that yield better results can be found, even though the complicated structure of the group operation on elliptic curves makes an efficient tag tracing construction much harder to find than in the case of finite fields. Combining tag tracing with the technique used for groups with fast endomorphisms [7, 10, 12, 34] also needs to be considered, although this does not appear to be easy. Applying tag tracing to other structures where DLP is considered, such as ideal class groups and abelian varieties, would be another direction for further research.

References

1. L. Adleman, A subexponential algorithm for the discrete logarithm problem with applications to cryptography, in *Proc. of the IEEE 20th Annual Symposium on Foundations of Computer Science (FOCS)*, pp.55–60, 1979.
2. J. W. Bos, M. E. Kaihara, T. Kleinjung, A. K. Lenstra, P. L. Montgomery, Solving a 112-bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction. Private communication.
3. R. Brent, An improved Monte Carlo factorization algorithm, in *BIT*, Vol. 20, pp.176–184, 1980.
4. J. Cheon, J. Hong, M. Kim, Speeding up the Pollard rho method on prime fields, in *ASIACRYPT '08*, LNCS 5350, Springer-Verlag, pp. 471–488, 2008.
5. D. Coppersmith, Fast evaluation of logarithms in fields of characteristic two, in *IEEE Trans. Inform. Theory*, Vol. 30, pp.587–594, 1984.
6. W. Diffie, M. Hellman, New directions in cryptology, in *IEEE Trans. Inform. Theory*, Vol. 22, pp.644–654, 1976.
7. I. Duursma, P. Gaudry, F. Morain, Speeding up the discrete log computation on curves with automorphisms, in *ASIACRYPT '99*, LNCS 1716, Springer-Verlag, pp.103–121, 1999.
8. T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, in *IEEE Trans. Infrom. Theory*, Vol. 31, pp.469–472, 1985.
9. P. Flajolet, A. M. Odlyzko, Random mapping statistics, in *EUROCRYPT '89*, LNCS 434, Springer-Verlag, pp.329–354, 1990.
10. R. Gallant, R. Lambert, S. Vanstone, Improving the parallelized Pollard lambda search on binary anomalous curves, in *Math. Comp.*, Vol. 69, pp.1699–1705, 2000.
11. J. von zur Gathen, M. Nöcker, Polynomial and normal bases for finite fields, in *J. Cryptology*, Vol. 18, pp.337–355, 2005.
12. M. Kim, J. Cheon, J. Hong, Subset-restricted random walks for Pollard rho method on $\mathbf{F}_{p^m}$, in *PKC '09*, LNCS 5443, Springer-Verlag, pp.54–67, 2009.
13. J. H. Kim, R. Montenegro, Y. Peres, P. Tetali, A birthday paradox for markov chains, with an optimal bound for collision in the Pollard rho algorithm for discrete logarithm, in *ANTS-VIII 2008*, LNCS 5011, Springer-Verlag, pp.402–415, 2008.
14. D. Knuth, *The Art of Computer Programming, vol. II: Seminumerical Algorithms*, Addison-Wesley, 1969.
15. D. Knuth, *The Art of Computer Programming, vol. III: Sorting and Searching*, Addison-Wesley, 1973.
16. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography.* CRC Press, 1997.
17. G. Nivasch, Cycle detection using a stack, in *Information Processing Letters*, Vol. 90, pp.135–140, 2004.
18. NIST, Digital signature standard, NIST. U.S. Department of Commerce. Federal Information Processing Standards Publication (FIPS PUB) 186, May 1994.
19. NIST, Recommendation for key management-part 1: General (revisited), `http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-57Part1_3-8-07.pdf`.
20. P. van Oorschot, M. Wiener, Parallel collision search with cryptanalytic applications, in *J. Cryptology*, Vol. 12, pp.1–28, 1999.
21. S. Pohlig, M. Hellman, An improved algorithm for computing discrete logarithms over $GF(p)$ and its cryptographic significance, in *IEEE Trans. Inform. Theory*, Vol. 24, pp.106–110, 1978.
22. J. M. Pollard, A Monte Carlo method for index computation (mod$p$), in *Math. Comp*, Vol. 32(143), pp.918–924, 1978.

23. J. M. Pollard, Kangaroos, monopoly and discrete logarithms, in *J. Cryptology*, Vol. 13, pp.437–447, 2000.
24. J. Quisquater, J. Delescaille, How easy is collision search? Application to DES, in *EUROCRYPT '89*, Springer-Verlag, LNCS 434, pp.429–434, 1989.
25. J. Sattler, C. Schnorr, Generating random walks in groups, in *Ann.-Univ.-Sci.-Budapest.-Sect.-Comput.*, Vol. 6, pp.65–79, 1985.
26. C. Schnorr, H. Lenstra, Jr., A Monte Carlo factoring algorithm with linear storage, in *Math Comp.*, Vol. 43(167), pp.289–311, 1984
27. A. Schönhage and V. Strassen, Schnelle multiplikation grobner zahlen, in *Computing*, Vol. 7, pp.281–292, 1971.
28. R. Sedgewick, T. Szymanski, and A. Yao, The complexity of finding cycles in periodic functions, in *SIAM Journal on Computing*, Vol. 11, No. 2, pp.376–390, 1982.
29. D. Shanks, Class number, a theory of factorization and genera, in *Proc. Symp. Pure Math.*, Vol. 20, pp.415–440, 1971.
30. V. Shoup, NTL: a library for doing number theory, Ver 5.5.1, `http://shoup.net/ntl/`.
31. E. Teske, Speeding up Pollard's rho method for computing discrete logarithms, in *ANTS '98*, Springer, LNCS 1423, pp.541–554, 1998.
32. E. Teske, On random walks for Pollard's rho method, in *Math. Comp.*, Vol. 70, pp.809–825, 2001.
33. E. Teske, Computing discrete logarithms with the parallelized kangaroo method, in *Disrete Applied Mathematics*, Vol. 130, pp.61–82, 2003.
34. M. Wiener and R. Zuccherato, Fast attacks on elliptic curve cryptosystems, in *SAC '98*, LNCS 1556, Springer-Verlag, pp.190–200, 1999.