Practical Improvement to Gaudry-Schost Algorithm on Subgroups of $\mathbb{Z}_p^*$

Madhurima Mukhopadhyay

Indian Institute of Technology, Kanpur
mukhopadhyaymadhurima@gmail.com

February 9, 2023

Abstract

The discrete logarithm problem forms the basis of security of many practical schemes. When the number of bases is more than one, the problem of finding out the exponents is known as the multi-dimensional discrete logarithm problem. It arises in several circumstances both in groups modulo some integer or elliptic curve groups. Gaudry and Schost proposed a low-memory algorithm for this problem which performs a pseudo-random walk in the group. Tag tracing is a technique to practically speed-up a pseudo-random walk. We have incorporated this technique into the Gaudry-Schost algorithm to observe the results of practical differences in time. We implemented the new algorithm in subgroups of $\mathbb{Z}_p^*$. Such subgroups are cryptographically relevant in the context of electronic voting and cash schemes. Our algorithm showed a substantial decrease in run-time with a gain in time by some integral multiple. For example, on a single core of Intel Xeon E7-8890 @ 2.50 GHz we showed our algorithm required 12 times less time than the Gaudry-Schost algorithm. This leads to better practical behaviour of the algorithm over such groups. We also point out a few more optimizations that can be adopted along with future applications for other scenarios.

1 Introduction

Cryptographic methods used in the earlier half of the twentieth century were susceptible to attacks by adversaries as the exchange of keys happened over an insecure channel. This defect was eradicated by Diffie and Hellman [dif]. Their development led to asymmetric or public-key cryptography based on the hardness of the discrete log problem.
The discrete logarithm problem (DLP) is defined as: Given a cyclic group G, its generator g of order N for some $N \in \mathbb{N}$ and an arbitrary element $h \in G$, find the exponent $a \in [0, N)$ such that $h = g^a$. One can easily think this definition to be for dimension one as a single generator is sufficient to generate G. This definition can also be extended for higher dimensions as follows.

Definition 1. *Multi-Dimensional DLP: Let G be an abelian group. Suppose $g_1, g_2, \ldots, g_D, h \in G$ and $N_1, N_2, \ldots, N_D \in \mathbb{N}$ for some positive integer D. The D-dimensional discrete logarithm problem is to*

find integers (if they exist) $a_1, a_2, \ldots, a_D \in \mathbb{N}$ *such that*

$$h = \prod_{i=1}^{D} {g_i}^{a_i} \tag{1}$$

where $a_i \in [0, N_i)\ \forall\ i = 1, 2, \ldots, D$.
We assume that the integers N_i *are odd* $\forall i, 1 \leq i \leq d$ *and* $N = N_1 \times N_2 \times \ldots \times N_D$.

Note, that an equivalent formulation of the problem for additive abelian group allows negative exponents. [1]

Applications: The multi-dimensional discrete logarithm problem arises in several situations which include structures as curves or a group modulo some integer. Computation of DLP in an interval is one of the steps when counting points on curves over finite fields [GH00, GS04, MCT02, Wen06]. Gaudry and Schost [GS04] developed their algorithm for the case of curves of genus 1. After using Schoof-type algorithm, the remaining problem is a multi-dimensional DLP for some $D \geq 3$. Thus, any enhancement in the Gaudry-Schost algorithm would also provide a general development to point counting.
Gallant, Lambert, and Vanstone (GLV) proposed an approach [GLV01, GLV00] to speed up elliptic curve arithmetic. Their method in [GLV00] requires expressing addition of a point P for some integer n, *i.e.*, $[n]P = [n_1]P + [n_2]\psi(P)$ for some integers n_1, n_2 where ψ is an endomorphism. The bound on n_1, n_2 is that $|n_1|, |n_2| \approx \sqrt{n}$. Clearly, this is an example of 2-dimensional DLP. For dimensions that exceed 2, other examples of such work are present in [GLS09, GS08]. The same approach of translating the work of writing a multiple of a point to a multi-dimensional DLP [JD08] can happen in the case of Koblitz curves [Kob91], which are ordinary elliptic curves over $\mathbb{F}_2$ (*i.e.*, the group $E(\mathbb{F}_{2^m})$)) leading to a 2-dimensional DLP. Similar things happen in curves of genus 2 over $\mathbb{F}_2$ resulting in a 4-dimensional DLP. The multi-dimensional DLP also arises in the situation when constructing an electronic cash scheme [Bra93] and election scheme [CGS97].

To solve the multi-dimensional discrete logarithm problem, Matsuo *et. al.* [MCT02] adopted the baby-step -giant -step algorithm [Sha71]. The time and space complexity of this algorithm were $O(\sqrt{N})$. Pseudo-random walks had earlier been used [Pol78, PM99, GPR13] in case of 1-dimensional DLP, which reduced the space requirement. Gaudry and Schost [GS04] proposed a low-memory algorithm using pseudo-random walks (with exponents of g_i's belonging to two types of sets called *tame* or *wild* sets) which can be adapted for solving for 2-dimensional DLP. This algorithm can be generalized for the multi-dimensional situation for any dimension.

The central idea in the Gaudry-Schost algorithm is to do a pseudo-random walk such that it is deterministic. The elements of the walk are the powers of the bases g_i. Depending on the bound on the

[1] Using same notations as in Definition 1, let G be an additive abelian group. The problem would be to find $a_1, a_2, \ldots, a_D \in \mathbb{N}$ such that

$$h = \sum_{i=1}^{D} [a_i] g_i \tag{2}$$

where $[a_i]g_i$ means g_i (or $-g_i$) added absolute value of a_i times if a_i is positive (or negative) and $a_i \in [-n_i, n_i]$. The two definitions are equivalent when assuming $2n_i + 1 = N_i$ depending upon the case at hand.

exponents of g_i's such walks are called tame or wild walks. As such walks happen in a finite group, they are bound to collide. If the exponents of g_i's are tracked when such walks are performed, then one can easily find out the multi-dimensional discrete logarithms. Several improvements to the Gaudry-Schost algorithm have been proposed assuming a specialized structure of the group [GR10, ZZY+19]. Generic progress has also been suggested [GR09, WZ21] which considers special choice of tame and wild sets. Good options of tame and wild sets reduce the number of iterations that may be actually required to get a collision. But it does not reduce the costs associated with a single iteration. So the question that arises in this condition is:

- Can we somehow reduce the work done per iteration?

The major work while performing each step of the walk is to multiply two elements. The other jobs are minor as they comprise finding some index and keeping track of the exponents. As the size of the group increases, the cost of multiplication becomes all the more pronounceable.

The principle of tag tracing that has previously been applied in pseudo-random walks of the Pollard Rho algorithm reduces multiplications in the entire group and performs necessary multiplications in some smaller subsets of the group. We try to understand how this will result if applied in the case of the Gaudry-Schost algorithm for greater dimensions.
We have described the entire procedure to incorporate tag tracing into the Gaudry-Schost algorithm in Section 5. An important scenario of application of multi-dimensional discrete logarithms arises in the set-up of electronic cash scheme [Bra93] and election scheme [CGS97]. We have implemented our algorithm in the case of such groups. We observe that there is a significant speed-up. In particular, even with limited resources we observed about 12 times acceleration in time for large-sized groups. Improvements are bound to be more for better choices of parameters, which has been suggested. This reduction in the time required should give good amount of gain in real-world applications.

1.1 Our Contributions

We focus on the multi-dimensional discrete logarithm problem for subgroups of $\mathbb{Z}_p^*$. Our main results are as follows:

1. We suggest an algorithm (Algorithm 5) that would help in reducing the practical cost when Gaudry-Schost algorithm is used to solve multi-dimensional discrete logarithm problem. We aim to reduce the work per iteration. Tag tracing is a method to speed up the pseudo-random walk by reducing the number of multiplications and it has previously been applied while traveling the walk of Pollard Rho algorithm [CHK12]. We extend tag tracing for multi-dimensional cases, and in this work, we apply tag tracing to the Gaudry-Schost algorithm. The theoretical analysis shows a reduction in time, with no increase in memory requirement.

2. We present the results of our implementation (in Section 9) of Algorithm 5 for subgroups of $\mathbb{Z}_p^*$. Such groups are used in the construction of electronic cash and voting schemes. We observe that there is a huge gain in time with the small computational resource. Specifically, for large groups, we obtained about 12 times speed-up.

3. We have suggested certain future research directions [Section 10] that are sure to boost the present status.

1.2 Paper Organization

The rest of the paper is organized as follows: We describe the Gaudry-Schost algorithm in Section 2. We next try to understand the motivation behind applying tag tracing to the Gaudry-Schost algorithm in Section 3. Sections 4 and 5 notes the entire strategy of inclusion of tag tracing into the Gaudry-Schost algorithm. Specifically, Algorithm 5 provides the entire process. The corresponding theoretical difference in the cost is given in Section 6. In Section 7, a procedure to remove costly modulo p reductions has been noted. Section 8 gives the complete description of the functions and the values of the parameters that would be required for tag tracing in subgroups of $\mathbb{Z}_p^*$. The results obtained after the implementation of these strategies are presented in Section 9. Some future research directions are pointed out in Section 10. In Section 11 we summarize the conclusions that can be drawn from our study.

2 Gaudry-Schost Algorithm Adapted to Multi-Dimensional Discrete Logarithms

Gaudry and Schost [GS04] suggested a low-memory, parallelizable algorithm using a 2-dimensional pseudo-random walk. The proposed method is efficient for solving 2-dimensional discrete logarithm problem and hence can be extended to the multi-dimensional case.
The basic idea of the algorithm is to perform a *deterministic, pseudo-random walk* with the help of two specially designed sets called *tame* and *wild* sets. A point $\prod_{i=1}^{D} {g_i}^{x_i}$ is called a *tame point* or a *wild point* depending upon whether the exponents $(x_1, x_2, \ldots, x_D)$ lie in tame or wild set respectively. Generally, a wild point is perceived to be of the form $h\prod_{i=1}^{D} {g_i}^{y_i}$ where $(a_1 + y_1, a_2 + y_2, \ldots, a_D + y_D)$ is a member of the wild set.

The algorithm proposed by Gaudry and Schost [GS04] consists of offline and online phase. In the initial step of the offline phase, two positive integers n_s, M_{gs} are chosen such that

$$n_s > \log(\max(N_1, \ldots, N_D))$$

and

$$M_{gs} \approx \frac{N}{1000 \log_2(N)} \tag{3}$$

The purpose of choosing n_s is to define a selection function

$$S : G \to \{0, 1, \ldots, n_s - 1\} \tag{4}$$

for partitioning the group into n_s components. Given any element of the group, the selection function outputs some integer, which is the partition to which that element belongs. The function S should be such that the distribution of the image of elements of G in the domain set $\{0, 1, \ldots, n_s - 1\}$ is almost uniform. It may be a hash function with good statistical properties.
Next, n_s elements are pre-computed to perform a walk. These elements are stored in a pre-computed table P as:

$$P := \{w_j = \prod_{i=1}^{D} {g_i}^{e_{i,j}} | - M_{gs} < e_{i,j} < M_{gs}, j = 0, 1, \ldots, n_s - 1\} \tag{5}$$

The pseudo-random walk is defined as below:

Let $v_k = \prod_{i=1}^{D} {g_i}^{x_{i,k}}$ denote the $k-$th element of the walk. Then the $(k+1)$-th element v_{k+1} can be calculated as

$$v_{k+1} = v_k \cdot w_{S(v_k)} \tag{6}$$

and its exponents

$$(x_{1,k+1}, x_{2,k+1}, \ldots, x_{D,k+1}) = (x_{1,k} + e_{1,S(v_k)}, x_{2,k} + e_{2,S(v_k)}, \ldots, x_{D,k} + e_{D,S(v_k)}) \tag{7}$$

Equations 6 and 7 is combined and expressed by the walk equation

$$walk(v_k, x_{1,k}, x_{2,k}, \ldots, x_{D,k}) = (v_{k+1}, x_{1,k} + e_{1,S(v_k)}, x_{2,k} + e_{2,S(v_k)}, \ldots, x_{D,k} + e_{D,S(v_k)}) \tag{8}$$

The function *walk* is so constructed with the help of S so that it is a deterministic pseudo-random function. In the online phase, the pseudo-random walks are distributed parallely [VOW94, PM99] over some processors. These walks continue until at least one element is found which is a product of g_i's as well as the product of h, g_i's with known exponents belonging to tame set and wild set respectively. Such a scenario is called finding a match between the tame and wild sets.

2.1 Run-time Analysis

The complexity analysis of the Gaudry-Schost algorithm is done by using birthday paradox [SEL95, NS90]. Three heuristic assumptions done in the process are: The pseudo-random function exhibits similar behavior as that of a random function; distinguished point probability is high enough; for any specific type of walk (tame or wild), the bounds and the *walk* function are so chosen that child of tame (or wild) point is also tame (or wild respectively) for a sufficient number of steps. The last condition is important in practice, as better choices of tame and wild sets minimize the number of group operations required.

2.2 Collision Detection

The process of finding a match between tame and wild walks requires the storage of the previously generated elements of the walk. There are several strategies [SSY82, Niv04] to keep this storage size optimal. The common collision detection algorithms of one-dimensional discrete logarithm problem like Floyd's cycle finding [Knu14], and Brent's algorithm [Tes98] also tries to optimize the space constraints.

However, the most efficient of all these methods is the method of distinguished points [Qui95]. A *distinguished point* is any element of the group G that satisfies certain conditions. These conditions are defined in such a way that they are easy to check. For example, given a fixed encoding of the group, one can define distinguished points as those elements of the group that have a certain number of most (or least) significant bits equal to zero. Technically, it may be said that a point is distinguished if its image by a second carefully chosen hash function is zero. This hash function should be designed so that it is not too hard to compute and is independent of any arithmetic property of the element. Let ρ_D denote the probability that a point is distinguished. As we try to find a match between distinguished points instead of any points arising in the walk, ρ_D extra iteration of the *walk* function will be required after a collision is already found. The goal here is to balance the cost of extra iterations with the cost of storing elements so that the storage table is of a manageable size and the extra iterations are close to

optimal. In practice, $\rho_D = \frac{A}{\sqrt{N}}$ for some constant A so that the storage space requirement is constant. The distinguished point method can be parallelized [PM99] and leads to speed-ups that are linear in the number of processors. To optimize the space overhead cost, the distinguished points are stored in an easily searchable structure such as a binary tree.
As complete information regarding each point is available, one can choose some positive integer δ and define z_i to be distinguished if δ least significant digits are zero. This is the definition that we shall use in our implementations. This can be easily tested by checking the equation below:

$$z_i \text{ is distinguished if } \ 2^{\delta} | z_i \tag{9}$$

The number of iterations to reach a distinguished point is 2^{δ} in this case. The choice of δ should be optimized for the efficiency of the algorithm.

Each pseudo-random walk continues until a distinguished point is reached. The distinguished points that occur in the tame and wild walks are stored in two different tables. These tables are kept sorted to ease the process of finding a common element.

2.2.1 Abandoning a walk

There is a possibility that a walk may never reach a distinguished point. To deal with such a situation, Van Oorschot and Wiener [PM99] defined a bound on the number of steps after which it can be abandoned. They set this length as $L = \frac{20}{\rho_D}$. Then the proportion of walks that exceed this length is bounded by

$$(1 - \rho_D)^{\frac{20}{\rho_D}} \leq (\exp(-\rho_D))^{\frac{20}{\rho_D}} = \exp(-20)$$

Any abandoned trail is 20 times longer than average, the proportion of walks that is abandoned is about

$$20\exp^{-20} < 5 \times 10^{-8}$$

The quantity on the right side is negligible and so such long fruitless walks may be abandoned at any time.

2.3 Obtaining the Multi-Dimensional Discrete Logarithm

Let z be a point that arises as a distinguished point both in tame and wild walks. Then for some $(x'_1, x'_2, \ldots, x'_D) \in \mathcal{T}_{GS}$ and $(y'_1, y'_2, \ldots, y'_D) \in \mathcal{W}_{GS}$, z is of the form

$$z = \prod_{i=1}^{D} {g_i}^{x'_i} = h \prod_{i=1}^{D} {g_i}^{y'_i}$$

Then, $h = \prod_{i=1}^{D} {g_i}^{x'_i - y'_i}$. The multi-dimensional discrete logarithm is obtained as

$$a_i = x'_i - y'_i \ \forall \ 1 \leq i \leq D \tag{10}$$

2.4 Algorithm

Let us now discuss the technical details related to the Gaudry-Schost algorithm. Here we discuss both phases of the algorithm along with the complexities.

2.4.1 Offline Phase

The pseudo-random walk (equations 6 and 8) requires the knowledge of the pre-computed table P as in 5. In the offline phase (Algorithm 1), the aim is to compute such a table consisting of n_s random elements $\prod_{i=1}^{D} {g_i}^{e_i}$, $|e_i| < M_{gs}\ \forall\ 1 \leq i \leq D$.

Algorithm 1: Precomputation in Gaudry-Schost algorithm.

Input: $D, g_1, g_2, \ldots, g_D, n_s, M_{gs}$.
Output: Table P consisting of n_s products of elements g_i's for $i = 1, 2, \ldots, D$.

```
Set P as an empty set
for i := 1 to n_s do
    Choose D integers e_1, e_2, ..., e_D randomly from (-M_gs, M_gs) //Choosing random
     exponents.
    Compute the product as prod ← ∏_{i=1}^{D} g_i^{e_i} //Computing product with chosen exponents.
    Append the product prod to P
Return P
```

2.4.2 Online phase

We now present the algorithms for the server side and the tame and wild processes. We will assume that distinguished points are stored in an easily searchable structure. Due to this, searching and storing times are polynomial and so can be neglected.
The algorithm for the server side, as given in Algorithm 2 receives points from the tame and wild servers respectively. On receiving a point, either from the tame or wild side, it checks whether it has been previously obtained from the different side also. The multi-dimensional discrete logarithm is solved if it is found. Else, it just appends the point to the table of distinguished points allotted for that side.
The algorithm for the tame and wild processor is described in Algorithm 3. It initiates the walk from a random tame (or wild) point and continues until it hits a distinguished point. It is ensured that the length of the walk is bounded so that it does not fall into the trap of obtaining each point as non-distinguished.

2.5 Different Choices of Tame and Wild Set

Efficient design of tame and wild sets will expedite the process of finding common points between such walks. Gaudry and Schost [GS04] proposed their algorithm taking the tame and wild set as orthotopes in $\mathbb{Z}^d$. The wild set was defined to be a translation of the tame set. Galbraith and Ruprai [GR09] proposed different choices for the tame and wild sets. The key idea behind their proposal was: Constant size of overlap will lead to constant expected running time for all instances of the problem. Thus their expected running time is the same for best, average, and worst cases. Wu and Zhuang [WZ21] constructed another pair of tame and wild sets, with the second framework being asymptotically optimal.
In the 1-dimensional case, the multi-dimensional discrete logarithm problem (Definition 1) is: Given $g, h \in G$ and $N \in \mathbb{N}$, finding $0 \leq a < N$ such that $h = g^a$. This can also be framed as: Finding x, $0 \leq x < 1$ such that $h = g^{xN}$.

Algorithm 2: The Gaudry-Schost algorithm: server side.

Input: $g_1, g_2, \ldots, g_D, h \in G$, $N_1, N_2, \ldots, N_D \in \mathbb{N}$.
Output: Integers $a_1, a_2, \ldots, a_D$ such that $h = \prod_{i=1}^{D} {g_i}^{a_i}$.

$D_T := [\,], D_W := [\,]$//Empty tables for storing distinguished points
while $\left(\textit{no collision between tame and wild points has been found}\right)$ **do**
  Receive a point $(z, b_1, \ldots, b_D)$ from the tame or wild processor
  if $\left(\ z \textit{ is received from a tame processor}\right)$ **then**
    if $(z, y_1, \ldots, y_D) \in D_W$ *for some* $y_1, \ldots, y_D$ **then**
      Send terminate signal to all client processors
      return $(b_1 - y_1, b_2 - y_2, \ldots, b_D - y_D)$
    else
      Append $(z, b_1, \ldots, b_D)$ to D_T
  else
    if $\left((z, x_1, \ldots, x_D) \in D_T \textit{ for some } x_1, \ldots, x_D\right)$ **then**
      Send terminate signal to all client processors
      return $(x_1 - b_1, x_2 - b_2, \ldots, x_D - b_D)$
    else
      Append $(z, b_1, \ldots, b_D)$ to D_W

In the second variant, Wu and Zhuang [WZ21] defined the tame and wild sets for dimension 1 as:

$$\mathcal{T} = \left[0, N\right], \tag{11}$$

$$\mathcal{W} = \left[xN - \frac{\alpha N}{2}, xN + \frac{\alpha N}{2}\right] \tag{12}$$

such that $0 \leq \alpha \leq 1$.
For $D > 1$, the tame and wild sets were just the products as the dimensions are independent of each other.

$$\mathcal{T} = \left[0, N\right]^D, \tag{13}$$

$$\mathcal{W} = \left[xN - \frac{\alpha N}{2}, xN + \frac{\alpha N}{2}\right]^D \tag{14}$$

The complexity analysis is based on non-uniform birthday problem as the size of overlap was not the same throughout.

Theorem 1. *(Section 4.2, [WZ21]) The expected average case complexity with the tame and wild set choice as provided in equations 13 and 14 is* $1.0171^D\sqrt{\pi N}$. *The expected worst case complexity is* $2^{\frac{D}{2}}\sqrt{\pi N}$.

The analysis of complexity has to be done considering: a pseudo-random walk can never behave as that of a random walk, so some correctional factor has to be included.

Algorithm 3: The Gaudry-Schost algorithm: tame or wild processor.

Input: $g_1, g_2, \ldots, g_D, h \in G$, $D, N_1, N_2, \ldots, N_D \in \mathbb{N}$, function *walk*, maximum length L of consecutive non-distinguished points.

Output: A distinguished point z along with exponents of g_i for it: $(z, x_1, \ldots, x_D)$ such that $z = \prod_{i=1}^{D} {g_i}^{x_i}$ if tame processor and $z := h \prod_{i=1}^{D} {g_i}^{x_i}$ if wild processor .

1 **while** $\left(\textit{no terminate signal received from server}\right)$ **do**

2 Choose $(x_1, x_2, \ldots, x_D)$ from the tame set $\mathcal{T}$ or wild set $\mathcal{W}$ depending upon the walk// Selecting a random tame or wild point for initiating the walk

3 Set $z := \prod_{i=1}^{D} {g_i}^{x_i}$ if tame processor or $z = h \prod_{i=1}^{D} {g_i}^{x_i}$ if wild processor

4 Set $Length := 0$

5 **while** $\left(z \textit{ is not a distinguished point and Length is less than } L\right)$ **do**

6 $j \leftarrow S(z)$//Compute the index $S(z)$ for the current point and store it

7 Find the entry w_j of the pre-computed table P corresponding to the index of current point//Finds a pre-computed random power of g_i's along with the exponents

8 $z \leftarrow z \cdot w_j$//single multiplication for getting the next element of the walk with help of pre-computed elements

9 $(x_1, x_2, \ldots, x_D) = (x_1, x_2, \ldots, x_D) + (e_{1,j}, e_{2,j}, \ldots, e_{D,j})$//Updating the current exponents by adding it with exponents from line 7

10 $Length \leftarrow Length + 1$

11 Supply $(z, x_1, \ldots, x_D)$ to the server.

3 Speeding-up Pseudo-Random Walks

The main crux of the Gaudry-Schost algorithm, applied to any dimension to solve the multi-dimensional discrete log problem is to perform a pseudo-random walk and search for collisions. To keep storage manageable, complete information of every point of the walk is not stored. A point is only required to be known completely only if it satisfies some condition, which is generally taken to be an easily testable distinguished point condition. Thus multiplication at each step of the walk is unnecessary if the information required to test the condition can be somehow extracted. This would be particularly efficient in large-sized groups where multiplication is costly. The principle of tag tracing [CHK12] can traverse through a random walk without fully computing each point. It is adapted to the distinguished point collision detection method.

Tag tracing was originally introduced in the context of the Pollard Rho algorithm [Pol78]. The Pollard Rho algorithm is a generic algorithm for solving the discrete logarithm problem. The basic idea remains the same as the Gaudry-Schost algorithm, which is to perform a pseudo-random walk until a collision occurs. The walk is performed with the help of a pre-computed table. For detecting collisions, the distinguished point approach is the most efficient method. The only notable difference between the Gaudry-Schost walk and the Pollard Rho type of walk is that while the former aborts the walk on arriving at a distinguished point, the latter continues. But this difference does not translate to any difference in tag tracing as the adaption of this method to pseudo-random walks requires the computation of a point entirely when a distinguished point is reached. Barring this, the other variation is that the Gaudry-Schost algorithm performs two types of walks referred to as tame and wild walks, while the

Pollard Rho algorithm performs a single kind of walk.
Practical implementations [CHK12] of tag tracing provided concrete speed-ups. Improvement in runtime was demonstrated for prime order subgroups of multiplicative groups of finite fields. Two types of fields were considered: Small characteristic, large extension degree fields, and prime order fields. Due to the similarity between the overall technique of finding logarithms using pseudo-random walk, it is quite likely that advantages obtained from tag tracing should be applicable in the case of the Gaudry-Schost algorithm as well.
Designing better tame and wild sets for the Gaudry-Schost algorithm reduces the number of iterations required. As a result, the asymptotic complexity is reduced. The concept of tag tracing reduces the work that is required to be done per iteration, by reducing the number of multiplications in the larger group. The consequence of this is some actual gain in time. As a result, the overall cost decreases. Here we desire to obtain such a practical advantage in the multi-dimensional discrete logarithm case also.

4 The Method of tag tracing

The tag tracing method reduces the number of field multiplications with the help of a larger precomputed table and some easily computable functions. The addition in cost in the offline phase to compute a larger table can be parallelized. In the steps where a point has not been computed entirely, the task of knowing whether a point is distinguished or not is achieved with the help of some efficiently designed functions. These functions are so constructed that they are easier to compute than a field multiplication. A full product computation is required after a certain number of steps.

5 Application of tag tracing in the Online Phase of Gaudry-Schost Algorithm

Let us discuss the adaption of this method for solving multi-dimensional discrete logarithms using Gaudry-Schost [GS04] algorithm. Like the original version of the Gaudry-Schost [GS04] algorithm, two phases are present.

5.1 Offline phase

Initially, a table structurally similar to P in equation 5 of the original Gaudry-Schost algorithm is computed. For constructing the table, two positive integers r for size and M_{tt} for the bound on exponents are decided. The table computed initially for applying tag tracing is of the form:

$$\mathcal{M} := \{w_j = \prod_{i=1}^{D} {g_i}^{e_{i,j}} | -M_{tt} < e_{i,j} < M_{tt}; j = 0, 1, \ldots, r-1\} \tag{15}$$

The next step aims to compute all those elements which are products of at most l elements from $\mathcal{M}$. This integer l is chosen after careful analysis. This can be thought of as computing the union of sets $\mathcal{M}^{(0)} \cup \mathcal{M}^{(1)} \cup \ldots \cup \mathcal{M}^{(l)}$, where each $\mathcal{M}^{(k)}$ is the product of exactly k elements from the set $\mathcal{M}$, $k = 0, 1, \ldots, l$. The set $\mathcal{M}^{(0)}$ consists of only the identity element of the group and the set $\mathcal{M}^{(1)}$ is the set $\mathcal{M}$. Apart from these two, each $\mathcal{M}^{(k+1)}$ can be computed from $\mathcal{M}^{(k)}$ by multiplying exactly

one element of $\mathcal{M}$ at a time to every element of $\mathcal{M}_k$. So whatever the size of r, l the computation of each $\mathcal{M}^{(k)}$ can be parallelized. We will denote this union as $\mathcal{M}_l$ henceforth, which is the multiplier set consisting of at most l products of elements of $\mathcal{M}$.

Theorem 2. *(Lemma2, [CHK12])) The total number of elements in $\mathcal{M}_l$ is $\binom{r+l}{r}$.*

The algorithm for the offline phase is given in Algorithm 4. Steps 2 to 5 compute the table $\mathcal{M}$ of random products. Steps 7 to 17 computes $\mathcal{M}^{(k)}$ for each $k = 2, \ldots, l$. To compute $\mathcal{M}^{(k)}$, product of any $(k-1)$ terms can be known from $\mathcal{M}^{(k-1)}$ and this product is multiplied with an element of $\mathcal{M}$. The cost of computing an element of $\mathcal{M}^{(k)}$ is just a single multiplication once the factor in $\mathcal{M}^{(k-1)}$ is accessed with the help of known indices.

Technically, the sets obtained from equation 15 are:

$$\begin{aligned}
\mathcal{M}^{(0)} &= \{1_G\} \\
\mathcal{M}^{(1)} &= \mathcal{M} \\
\mathcal{M}^{(2)} &= \{w_{j_1} w_{j_2} | 0 \leq j_k \leq (r-1) \ \forall \ k = 1, 2\} \\
&\vdots \\
\mathcal{M}^{(l-1)} &= \{w_{j_1} w_{j_2} \ldots w_{j_l} | 0 \leq j_k \leq (r-1) \ \forall \ k = 1, 2, \ldots, l\}
\end{aligned}$$

where 1_G denotes the identity of G and w_j's are listed in equation 15.

$$\mathcal{M}_l = \mathcal{M}^{(0)} \cup \mathcal{M}^{(1)} \cup \mathcal{M}^{(2)} \cup \ldots \cup \mathcal{M}^{(l)} \tag{16}$$

$$= \{\prod_{k=1}^{l} w_{j_1} w_{j_2} \ldots w_{j_k}, 1_G | 0 \leq j_k \leq (r-1) \forall k = 1, 2, , \ldots, l\} \tag{17}$$

From equation 15, each exponent is absolutely bounded by M_{tt}. From equation 17, the exponents of g_i's absolutely bounded above by $l.M_{tt}$. Due to this, M_{tt} has to be chosen so that

$$l.M_{tt} \approx M_{gs} \approx \frac{N}{1000 \log_2(N)} \tag{18}$$

Choosing l: From Theorem 2 we see that increasing l will lead to a larger table. As the table has to be accessed at each step during the online phase, it is necessary to keep this size optimal. On the other hand, the approximation 18 provide another bound for selecting l.

Storing the table $\mathcal{M}_l$: Each entry of table $\mathcal{M}_l$ comprises of three components. The product of elements corresponding to indices $j_1, j_2, \ldots, j_k$ of $\mathcal{M}$ (equation 15) is $\prod_{t=1}^{k} w_{j_t} = \prod_{i=1}^{D} g_i^{\sum_{t=1}^{k} e_{i,j_t}}$. The exponent information and this product is stored in $\mathcal{M}_l$ as:
$[(j_1, j_2, \ldots, j_k), \prod_{t=1}^{k} w_{j_t}, (\sum_{t=1}^{k} e_{1,j_t}, \sum_{t=1}^{k} e_{2,j_t}, \ldots, \sum_{t=1}^{k} e_{D,j_t})]$. For easy maintenance, the table $\mathcal{M}_l$ can be sorted according to multiplier combination information, or a hash table technique can be used.

Algorithm 4: Offline phase when tag tracing is applied to Gaudry-Schost algorithm.

Input: r, M_{tt} and $l \in \mathbb{N}$.

Output: Multiplier table $\mathcal{M}_l$ consisting of $\binom{r+l}{r}$ entries with products of elements g_i's for $i = 1, 2, \ldots, D$ where each entry consists of a set of indices, the product, and the exponent information.

Set $\mathcal{M} \leftarrow [[(0), 1, (0, 0, \ldots, 0)]]$
for $j := 0$ *to* $(r-1)$ **do**
    Choose D integers $e_{1,j}, e_{2,j}, \ldots, e_{D,j}$ randomly from $(-M_{tt}, M_{tt})$
    Compute the product as $w_j \leftarrow \prod_{i=1}^{D} {g_i}^{e_{i,j}}$
    Append $[(j+1), w_j, (e_{1,j}, e_{2,j}, \ldots, e_{D,j})]$ to $\mathcal{M}$
Set $\mathcal{M}_l = \mathcal{M}$// Next it will compute all possible l many products
for $k := 2$ *to* l **do**
    $I := [0, 0, \ldots, 0]$ // Set of k indices all 0's
    $e := [0, 0, \ldots, 0]$// To be used for getting the exponents
    while $\Big(\{Indices \neq [(r-1), (r-1), \ldots, (r-1)]\}\Big)$ **do**
        $E \leftarrow$ Entry of $\mathcal{M}_l$ corresponding to $(I[1], I[2], \ldots, I[k-1])$ as first component//Finds and stores the entry
        $tmp \leftarrow E[2]$//The product $w_{I[1]} w_{I[2]} \ldots w_{I[(k-1)]}$ corresponding to the chosen indices that occurs as a second component of the entry
        $exp \leftarrow E[3]$//The exponents occur as the third component
        $prod := tmp \times w_{I[k]}$ //Single multiplication provides the product of k elements from the initially chosen r random elements where $w_{I[k]}$ is obtained from $\mathcal{M}$.
        $e \leftarrow [e[1], e[2], \ldots, e[k]] + [exp[1], exp[2], \ldots, exp[k]]$//Getting the exponents
        Append the term $[(I[1], I[2], \ldots, I[k]), prod, e]$ as the next entry of $\mathcal{M}_l$
        Increase I according to Lex ordering
Return $\mathcal{M}_l$

5.2 The Pseudo-Random Walk and the Functions Required

The pseudo-random walk when tag tracing is applied will be of the same type as in equation 8. The only difference here is that the pre-computed table will be $\mathcal{M}$ (as in equation 15). This requires choosing $n_s = r$. We assume that we have an index function $s : G \to \{0, 1, \ldots, n_s - 1\}$ which partitions G.
An auxiliary index function $\overline{s}$ can be defined with the help of s as:

$$\overline{s} : G \times \mathcal{M}_l \to \{0, 1, \ldots, n_s - 1\}$$

$$\overline{s}(y, m) = s(ym) \ \forall y \in G, m \in \mathcal{M}_l \tag{19}$$

By the above equation, given the auxiliary index of some point g_i of the walk, the next l auxiliary indices can be computed without any multiplication.

$$\begin{cases} g_{i+1} = g_i m_{s(g_i)} \implies s(g_{i+1}) = \overline{s}(g_i, m_{s(g_i)}) \\ g_{i+2} = g_{i+1} m_{s(g_{i+1})} = g_i m_{s(g_i)} m_{s(g_{i+1})} \implies s(g_{i+2}) = \overline{s}(g_i, m_{s(g_i)} m_{s(g_{i+1})}) \\ \vdots \\ g_{i+l} = g_{i+l-1} m_{s(g_{i+l-1})} = g_i m_{s(g_i)} m_{s(g_{i+1})} \ldots m_{s(g_{i+l-1})} \implies s(g_{i+l}) = \overline{s}(g_i, m_{s(g_i)} m_{s(g_{i+1})} \ldots m_{s(g_{i+l-1})}) \end{cases} \tag{20}$$

The product of multipliers on the right-hand side of the above equation can be accessed from the table $\mathcal{M}_l$. Tag tracing requires the design of the auxiliary index function $\overline{s}$ such that $\overline{s}(y, m)$ is easier to compute than multiplying y and m. This would make the pseudo-random walk for the next l steps easier once an element is computed fully by multiplying its components.
The function $\overline{s}$ is constructed with the help of four other functions and a larger set $\mathcal{T}$ (such that $\mathcal{S} = \{0, 1, \ldots, n_s - 1\} \subset \mathcal{T}$) called the tag-set. The functions are:

$$\begin{gathered} \text{Tag function } \tau : G \to \mathcal{T} \\ \text{Auxiliary tag function } \overline{\tau} : G \times \mathcal{M}_l \to \mathcal{T} \\ \text{Projection function } \sigma : \mathcal{T} \to \mathcal{S} \\ \text{Auxiliary projection function } \overline{\sigma} : \mathcal{T} \to \mathcal{S} \cup \{\text{Fail}\} \end{gathered}$$

The index function and the auxiliary index function are constructed by composing these so that s is surjective, pre-image uniform, and obeys equation 19.

$$\begin{gathered} s = \sigma \circ \tau : G \to \mathcal{S} \\ \overline{s} = \overline{\sigma} \circ \overline{\tau} : G \times \mathcal{M}_l \to \mathcal{S} \cup \{\text{fail}\} \end{gathered}$$

There is a certain probability of failure, on which a full product computation has to be done. The index function which is already being computed in each step can be used to define distinguished points. This ensures that deciding whether a point is distinguished or not does not incur any extra cost. A simple way of defining distinguished points is this set-up would be: A point z_i, $i \geq (\delta - 1)$ (where $\delta \geq 1$) is distinguished if the index of itself and each of its $\delta - 1$ ancestors are zero, *i.e.*,

$$z_i \text{ is } \textit{distinguished} \text{ if } s(z_{i-\delta+1}) = \ldots = s(z_{i-1}) = s(z_i) = 0$$

Assuming $z_{j+1} = z_j w_j$ for any index j, and using relation given by Equation 19 among $\overline{s}$ and s, a point z_i is a distinguished point if

$$\overline{s}(z_{i-\delta+1}, w_{i-\delta+1}) = \overline{s}(z_{i-\delta+2}, w_{i-\delta+2}) = \ldots = \overline{s}(z_{i-1}, w_{i-1}) = 0 \tag{21}$$

The probability of such a distinguished point is $\Delta = \frac{r}{r-1}(r^{\delta} - 1)$ [Theorem 1, [CHK12]]. The ambiguity for points that occur within δ iterations is negligible when compared to the total number of points required for a complete logarithm computation, as δ is of logarithmic order in Δ and the distinguished point method will require $O(\Delta)$ extra iterations after the point of collision. In case, some application requires checking whether distinguishing property is present in such points, it can easily use the output of $\overline{\tau}$, which is available for every point, and define distinguished points based on that. There are other efficient ways to define distinguished points [CHK12].
In order to lessen run-time when calculating s in terms of $\overline{s}$ (equation 19), $\frac{C_{\overline{s}}}{C_{full}} < 1$ where $C_{\overline{s}}$ and C_{full} are the cost in terms of time required to obtain $\overline{s}$ and the time to compute a full product respectively.

5.3 Online phase

The algorithm for the tame and wild sides in the online phase is given in Algorithm 5. The application of tag tracing into the Gaudry-Schost algorithm eliminates the necessity of multiplication at each step. Multiplication once done, can be skipped for l consecutive steps in general.

The tame and wild walks begin by selecting a random point from the tame and wild sets. In this case, one can continue with the best possible choice of tame and wild sets for the Gaudry-Schost algorithm. This computation of random points is the only place where D multiplications are required. After that, we find l multipliers w (line 13) from the previously generated table $\mathcal{M}_l$. For finding any such multiplier no multiplication is required. We check whether the product of z and the newly found multiplier w is a distinguished point (line 12). If any distinguished point is found then it is sent to the server. It is important to note [Equation 21] that distinguished points are defined in such a way that, complete multiplication of z and w is not required for knowing whether $z \cdot w$ is distinguished. It is determined easily from the partial knowledge of z and w.

The exponents of g_i's are known for every point of the walk. This exponent tracing capability is used for obtaining discrete logarithms.

6 Cost Comparison for the Online Phase

The major contributor to the run-time for both Algorithm 3 and Algorithm 5 is the cost of multiplication. The method of tag tracing applied to the Gaudry-Schost algorithm requires a single multiplication after every l steps. When this is compared to the original Gaudry-Schost algorithm, we see that there is some improvement due to the reduction in the number of multiplications modulo p which results in a real gain in time. This is particularly useful in situations where multiplications are costly.

Let $\overline{N}$ denote the number of iterations for a tame or wild walk. Also, let $\overline{L}$ denote the minimum number of iterations until a walk terminates, *i.e.*, either the walk reaches a distinguished point or it reaches the maximum limit up to which a walk can travel if it is devoid of distinguished points. In general, this number is equal to the number of iterations required for distinguished points.
Each set of $\overline{L}+1$ points of the pseudo-random walk is either an initial point of the walk or some element

Algorithm 5: Tag tracing applied to Gaudry-Schost algorithm: tame or wild processor.

Input: $g_1, g_2, \ldots, g_D, h \in G$, $d, N_1, N_2, \ldots, N_D \in \mathbb{N}$, function $walk$, maximum length L of consecutive non-distinguished points.

Output: A distinguished point z along with exponents of g_i for it: $(z, x_1, \ldots, x_D)$ such that $z = \prod_{i=1}^{D} {g_i}^{x_i}$ if tame processor and $z := h\prod_{i=1}^{D} {g_i}^{x_i}$ if wild processor .

while $\Big($ *no terminate signal received from server* $\Big)$ **do**
- Choose $(x_1, x_2, \ldots, x_D)$ from the tame set $\mathcal{T}$ or wild set $\mathcal{W}$ depending upon the walk//Set-up a random tame or wild walk
- Set $z := \prod_{i=1}^{D} {g_i}^{x_i}$ if tame processor or $z = h\prod_{i=1}^{D} {g_i}^{x_i}$ if wild processor
- Set $Length = 0$
- **while** $\Big(z$ *is not a distinguished point and Length is less than* $L\Big)$ **do**
  - $w \leftarrow 1_G$.
  - $k \leftarrow 0$.
  - $(e_{1,k}, e_{2,k}, \ldots, e_{D,k}) = (0, 0, \ldots, 0)$
  - $tmp_{\overline{s}} \leftarrow \overline{s}(z, w)$
  - **if** $tmp_{\overline{s}}$ *is not fail* **then**
    - $j_k \leftarrow tmp_{\overline{s}}$
  - **while** $\Big($*(*$k < l$*) and (*$\overline{s}$ *does not fail) and (*$z.w$ *is not a distinguished point)* $\Big)$ **do**
    - $E \leftarrow$ Entry of $\mathcal{M}_l$ corresponding to $(j_0, j_1, \ldots, j_k)$//This finds the entry with first component $(j_0, j_1, \ldots, j_k)$
    - $w \leftarrow E[2]$//The second component is the pre-computed product $w_{j_0} w_{j_1} \ldots w_{j_k}$
    - $k = k + 1$
    - $tmp_{\overline{s}} \leftarrow \overline{s}(z, w)$
    - **if** $tmp_{\overline{s}}$ *is not fail* **then**
      - $j_k \leftarrow tmp_{\overline{s}}$
  - $z \leftarrow z \cdot w$//single multiplication in G that does a full product computation and is generally needed after skipping all multiplications for the previous l steps
  - $Length \leftarrow Length + l$
  - Update the exponent information $(e_{1,k}, e_{2,k}, \ldots, e_{D,k})$ corresponding to $(j_0, j_1, \ldots, j_k)$//The third component of each entry in $\mathcal{M}_l$ contains exponent information.
  - $(x_1, x_2, \ldots, x_D) = (x_1, x_2, \ldots, x_D) + (e_{1,k}, e_{2,k}, \ldots, e_{D,k})$//Updating the exponent information of z using exponent information of w
- Send $(z, x_1, x_2, \ldots, x_D)$ to the server

obtained by applying the *walk* function to its previous element. The initial points are obtained by multiplication of D group elements while getting roughly the next points require a single multiplication in the full group and l multiplications roughly in smaller subset. Let C_D denote the cost of obtaining the initial element. The total cost, denoted by T_0, for each consecutive set of $\overline{L}+1$ points is then:

$$T_0 = (C_D + \overline{L}.C_{full}) \tag{22}$$

In the case of applying tag tracing, full product multiplication is required either when l iterations of successfully getting $\overline{s}$ is completed or when $\overline{s}$ fails in some intermediate step. Let $Prob[\overline{s}$ fails$]$ denote the probability of failure of $\overline{s}$. The total cost, denoted by T_1, for each consecutive set of $\overline{L}+1$ points when tag tracing is applied is:

$$T_1 = (C_D + \overline{L}(Prob[\overline{s} \text{ fails}] + \frac{1}{l})C_{full} + \overline{L}(1 - Prob[\overline{s} \text{ fails}] - \frac{1}{l})C_{\overline{s}}) \tag{23}$$

Roughly, the total cost estimates in terms of the multiplications required in the case of Algorithm 3 is then $\frac{\overline{N}}{(\overline{L}+1)}T_0$. The same estimate for Algorithm 5 is $\frac{\overline{N}}{(\overline{L}+1)}T_1$. The overall improvement comes from the fact that $C_{\overline{s}} < C_{full}$.
In the case of the original Gaudry-Schost algorithm, the exponents need to be updated every time. This cost is ignored in the complexity analysis as it is less costly than iterating the walk function. When applying tag tracing, the exponents can be updated after l iterations generally or in exceptional cases when we get a distinguished point or $\overline{s}$ fails. Thus the cost of D additions at every iteration is replaced by D additions until a full-product computation is required.

Since tag tracing in no way interferes with the pseudo-randomness of the walk, the number of iterations $\overline{N}$ needed to get a collision will not change. As each iteration, will be accelerated, so the overall time required will be less.

7 Improvement in Groups of Prime Order

Let G be a prime-ordered subgroup of some group containing $(p-1)$ elements for a prime p. It has already been seen [CHK12] that tag tracing speeds up pseudo-random walks in such groups G.
A multiplication in G is a two-step operation: The first step is to do an integer multiplication which is followed by a reduction modulo p operation. For arbitrary primes without any structural specialty, reduction operation is costly. The strategy [MS22] of replacing these multiplications with Montgomery multiplications helps to completely get rid of this cost. Instead, the cost incurred is quite inexpensive as they are divisions by powers of two, which are done just by right shift operations.
In fact, the reduction operation is performed using division algorithm [Algorithm 14.20, [MVOV18]]. So in general cases, the cost of the reduction operation is the same as integer multiplication. Substituting this with cheap divisions by powers of two lessens the cost by about half. The technique of incorporating Montgomery multiplication into the method of tag tracing has been entirely described in [MS22]. This combination of tag tracing and Montgomery multiplications has been achieved without any trade-offs. It is noteworthy that the entire procedure can be replicated into the Gaudry-Schost method as well to find multi-dimensional discrete logarithms. This would eventually lead to more practical speed-ups.

8 Cryptographically Relevant Groups

An interesting case of application of our proposal is in the electronic cash scheme [Bra93] and election scheme [CGS97]. The group $G(= G_q$ for the sake of notation) considered in these schemes, is a prime order subgroup of $\mathbb{Z}_p^*$, the unit group of the set of integers modulo p. G_q has (known) order q, where p, q are large primes such that $p|(q-1)$. For practical purposes, the bit-size of p, q are taken to be at least $512, 140$ respectively while the dimension d may be taken as $2, 3$. Higher values of d also work. Let us discuss the adoption of tag tracing in such groups.

8.1 Functions Required for tag tracing

The values of l, M_{tt} are such that

$$l * M_{tt} \approx \frac{N}{1000 \log_2(N)} \tag{24}$$

The following choice of functions can be taken in the case of such groups. The definitions of the functions as in Section 5.2 depend on some pre-fixed quantities. Also, let r, w be fixed as some suitable powers of two. We consider $u = \sqrt{w}, d = \lceil \log_u(p-1) \rceil$, $\overline{t} = 2^{\lceil \log_2 du \rceil}$, $t = \frac{w}{\overline{t}}$, $\overline{r} = \frac{t}{r}$. The functions are defined as:

$$\tau : G \to \mathcal{T} = \{0, 1, \ldots, (t-1)\} \quad \tau(z) = \lfloor \frac{z \mod p}{\overline{t}\overline{w}} \rfloor$$

$$\sigma : \mathcal{T} \to \mathcal{S} = \{0, 1, \ldots, (r-1)\} \quad \sigma(z) = \lfloor \frac{z}{\overline{r}} \rfloor$$

Each element $z \in \mathbb{Z}_p^*$ can be represented in base u as $z = z_0 + z_1 u + \ldots + z_{d-1} u^{(d-1)}$. For every $m \in \mathcal{M}_l$, we define $\hat{m}_i = \lfloor (u^i m \mod p)/\overline{w} \rfloor$. These values can be pre-computed as a part of the offline work and stored in the table. Based on this, we define

$$\overline{\tau} : G \times \mathcal{M}_l \to \mathcal{T} \quad \overline{\tau}(y, m) = \left\lfloor \frac{(\sum_{i=0}^{d-1} y_i \hat{m}_i) \mod w}{\overline{t}} \right\rfloor$$

It can be verified that $\tau(zm) = \overline{\tau}(z, m)$ or $\tau(zm) = \overline{\tau}(z, m) \mod t$. This difference is filtered out by defining $\overline{\sigma}$ properly so that $s, \overline{s}$ are equal for the same components of the product term.
The other functions can be defined as

$$\overline{\sigma} = \begin{cases} \text{fail} & \text{if } x \equiv -1 \bmod \overline{r}, \\ \lfloor x/\overline{r} \rfloor & \text{otherwise.} \end{cases}$$

Clearly, whenever $\overline{s}(= \overline{\sigma} \circ \overline{\tau})$ does not fail, it is equal to $s(= \sigma \circ \tau)$.
The distinguished point definition can be taken as in Section 5.2 for some suitable choice of δ, where δ can be chosen according to our will so that the distinguished point probability is sufficient enough.
The cost of a full product computation, in this case, would be $size(p)^2$ where $size(p)$ denotes the bit-size of p, whereas, in computing $\overline{s}$ the main time is spent in $\overline{\tau}$ as r is taken to be a small positive integer (generally $4, 8$ for upto 3072-bit primes). $\overline{\tau}$ is essentially multiplication between w bit and u bit integers, where $size(w) << size(p)$ and $size(u) << size(p)$. Also if w, u is chosen as a power of two appropriately, then operations can be done quite easily.

The combination of the method mentioned in Section 7 along with tag tracing should help to gain further acceleration of the process.

9 Results of Implementation

In this section we present the results of implementation of Algorithm 3 and Algorithm 5 in prime order subgroups G_q (of order q) of $\mathbb{Z}_p^*$. The primes q chosen in our experiment range from 256 to 2076 bits. The primes p were *safe* primes. We aimed to compare the original Gaudry-Schost method with the latest choice of tame and wild sets [WZ21] and the tag tracing method applied to the Gaudry-Schost algorithm. Complete multi-dimensional discrete logarithm computation is not feasible with quite limited resources for the sizes of primes that we considered. We resort to the method of walking over a large number of tame and wild points to collect some distinguished points from each. This was done for random targets and generators using both algorithms. The time noted by observing the experiments included the full time to find the number of distinguished points we desired for each of the algorithms. The computations were run on a single core of server Intel Xeon E7-8890 @ 2.50 GHz. We implemented our algorithms using Magma version V2.22-3.
We observe that our experiments indicate that when tag tracing is applied to the Gaudry-Schost algorithm, there is a significant speed-up. For the primes that we have tested, this gain in time varies from 3 to 12 times. We discuss the details in the following sections. We mention here that we have used the same multiplication routine for both algorithms so that results do not become skewed. Our multiplication is the classical multiplication method. No Montgomery multiplication strategy is adopted for these as the purpose was to observe how much improvement sole tag tracing can lead to for the Gaudry-Schost algorithm. The exact choice of primes, the definition of distinguished points, and the average value of the time taken to walk past each point are given in the Appendix.

The time to compute the table of random points for both algorithms, may not be negligible when a large-sized group is considered for practical purposes, but this time can be ignored when compared with the corresponding total time for finding the solution to the multi-dimensional discrete logarithms. Thus, we focussed on comparing the online phase of both algorithms. Firstly, we have tested the correctness of both algorithms on groups of small sizes. Both the algorithms terminate and provide valid multi-dimensional discrete logarithms. We next initiate our experiments for larger groups. The exact values of the $256, 512, 1024, 2076$ bit primes chosen are given in Table 2. In all the experiments, we use $p = 2 \cdot q + 1$ and $r = 4, w = 2^{32}, u = \sqrt{w}$. For the values of the other quantities and choice of functions, we abide by the things mentioned in Section 8.1. The next work was to set an easily testable distinguishing property. For the Gaudry-Schost algorithm, we used Equation 9 to define distinguished points, *i.e.*, we choose some δ_1 optimally and check whether 2^{δ_1} divides the present walk point or not. For tag tracing, we chose some positive integer δ_2 and checked whether Equation 21 is satisfied or not for the corresponding multipliers. The corresponding theoretical estimates, which would form a significant role in determining the number of iterations required are 2^{δ_1} for Algorithm 3 and $\overline{r} + \frac{r}{r-1}(r^{\delta_2} - 1)$ (where $\overline{r}$ arises due to cases when $\overline{s}$ fails) for Algorithm 5 respectively. We chose δ_1, δ_2 such that the number of iterations required for getting a distinguished point was almost the same whether tag tracing was applied or not. We used (δ_1, δ_2) as $(12, 6)$ (for q with bit-size $256, 2076$) and $(10, 5)$ (for q with bit-size $512, 1024$). These choices were done with the intention so that results are unbiased towards tag tracing. It can be seen from Table 3 in Appendix that with such selections, the number of iterations required for tag tracing was more. To compute the next element of the walk for Algorithm 3, we obtained the index of the multiplier by doing a modulo operation with the pre-computed table size. Here pre-calculated table size is taken as a multiple of two so that the modulo operation is easier. The child point for Algorithm 5 is calculated with the help of $\overline{s}$ defined previously.

For each such group $G_q(\subseteq \mathbb{Z}_p^*)$ we set the dimension as two and randomly chose 5 targets h along with bases g_1, g_2 intending to collect the same number of distinguished tame and wild points for both the algorithms. As the aim was simply to compare the relative run-time, we have not focussed on fine-tuning the parameters for the algorithms. But care has been taken so that Algorithm 5 does not enjoy any extra advantage. For example, we defined distinguished points so that lesser iterations are needed to reach them in the case of the original Gaudry-Schost algorithm and the calculation of the next element was made easier by choosing the pre-calculated table size as a power of two .

In Table 1, we have noted the time along with the number of points covered for each group considering tame and wild walks for both Algorithms 3 and 5. The number of distinguished points extracted was the same for both algorithms. In columns 5 and 8, we note the average time taken to travel each point of the walk, whether tame or wild in the case of both the algorithms. This ratio is approximated by choosing to walk over a large number of points of the pseudo-random walk. In Tables 4 and 5 in the Appendix, we have compared these ratios for various sizes of q.
It is obvious that in the case of both algorithms, major time is spent in doing the multiplications whether complete or partial. To satisfy ourselves of the fact that the values in columns 5, and 8 of Table 1 are quite accurate, we have compared two experimentally observed values (of time to travel a single point) for the same algorithm and same walk type (tame or wild) such that in one case size of q is roughly twice the other. The experimentally observed ratio in time is compared with the theoretical estimate obtained from the cost of multiplication. From the statistics noted in Table 4 for the Gaudry-Schost algorithm and Table 5 when tag tracing is applied, we see that the experimental observations and theoretical estimates are close enough. We can also observe from Table 1 that the time per point is almost the same for time or wild considering each group, whether tag tracing is used or not.
From the arguments in Section A.1 of the Appendix, we see that the average times noted in columns 5 and 8 of Table 1 can be used for comparing the relative run-time. We also note that for both algorithms, the major time exhausted was on performing multiplications. This is tuned with our previous assumption. In the last column of Table 1, we compare the ratio of these two averages (*i.e.*, Average time taken to traverse a single point using Algorithm 3/ Average time taken to traverse a single point when Algorithm 5 is applied).

We see that this ratio varies between 3 to 12 for the groups we tested. Clearly, this shows that tag tracing requires much less time when it is applied to the Gaudry-Schost algorithm. This suggests that the gain in time for realistic parameters would be quite significant in the case of complete multidimensional discrete logarithm computation. It is also noteworthy that as the size of the group increases tag tracing becomes more and more efficient. This accelerates the computation of logarithms about 12 times for the largest group that we have considered.

Our sole target here was to compare the relative run-time of the two algorithms. Tag tracing can be further optimized by choosing better parameters. For example, the l value chosen by us is quite small. Larger l will lead to walking through more points without performing complete multiplications. Also, the design of better choices of tame and wild sets specialized to apply tag tracing and the associated quantities to define the functions should also lead to the advancement of the process. The strategies mentioned in Section 7 can also be included. These would be future work and it would lead to even better estimates.

(Bits in q,Number of distinguished points)	Walks	Gaudry-Schost			tag tracing			Ratio
		Time	Points traversed in pseudo-random walk	Average time taken to travel single point	Time	Points traversed in pseudo-random walk	Average time to travel single point	Ratio of time to traverse single point
$(256, 1000$	tame	551267	3912583	0.140	233983	5620981	0.041	3.414
	wild	589798	4151376	0.142	231168	5580770	0.041	3.463
$(512, 500)$	tame	766317	1367179	0.560	254391	2507186	0.101	5.544
	wild	627970	1145379	0.548	240485	2274981	0.105	5.219
$(1024, 100)$	tame	227953	102899	2.215	35121	150748	0.232	9.547
	wild	210711	97425	2.162	36523	148622	0.245	8.824
$(2076, 50)$	tame	1630114	193740	8.413	259295	382000	0.678	12.408
	wild	1872828	221984	8.436	279945	410358	0.682	12.369

Table 1: Factor by which average time requirement reduces when tag tracing is applied is shown for various groups. The total time and the total number of points walked through are also given.

10 Future Works

We see that tag tracing leads to remarkable gain in time when applied in the original Gaudry-Schost algorithm. A few other things done in addition to this inclusion should also lead to superior results.
Firstly, the design of tame and wild sets can be done such that they are more suited towards tag tracing. We have seen cases of non-constant [WZ21] and constant overlap [GR09]. Here the overlap can be varied according to whether complete multiplications are required or not. In particular, instead of defining the tame and wild sets solely on the values of N, α, the value of l and the quantities needed for getting suitable functions can be included as well. This duo of better tame and wild sets to get collision faster, along with tag tracing resulting in lesser work per iteration, should again help to reduce overall practical cost.
The functions can be designed in such a way that this overlap is hastened. Also, the definition of distinguished points can be more adapted for the multi-dimensional situations so that less work is required for testing it.
Also taking some optimal value of l should lead to further advancement. The increase in the size of the pre-computed table due to the surge in the value of l can be tackled by choosing a proper way to handle it, for example, exponents can be added later, and the online phase requires getting hold of only those information necessary to know whether a point is distinguished or not.
The combination of these things along with replacing modulo p reductions by divisions by powers of two, as in Section 7 also seems interesting.
Our implementation are in subgroups of $\mathbb{Z}_p^*$. The application of the tag tracing procedure into multi-dimensional discrete logarithm problem for elliptic curve groups, and also for performing better in other

situations where the multi-dimensional walk is required is another venue that also should be worth exploring.

11 Conclusion

We have proposed a modified form of the Gaudry-Schost algorithm by applying tag tracing. This has lead to practical improvement in time to compute multi-dimensional discrete logarithms in subgroups of $\mathbb{Z}_p^*$. Our algorithm [Algorithm 5] was implemented for groups with orders ranging between 256 to 2076 bits. With limited resources, the improvement obtained was quite impressive. We obtained about 12 times speed-up for groups of practical sizes. An open problem is to fine-tune the parameters and to specially design tame and wild sets more suited for the new algorithm. This should lead to a overall reduction in run-time by reducing the number of iterations. We have also noted some future research directions for overall improvement.

Bits in q	q	d	$\overline{r}$
256	57896044618658097711785492504343953926634992332820282019728792003956608167403	16	1024
512	6703903964971298549787012499102923063739682910296196688861780721860882015036773488400937149083451713845015929093243025426876941405973284973216824506305919	32	512
1024	8988465674311579538646525953945123668089884894711532863671504057886633790275048156635423866120376801056005693993569667882939488440720831124642371531973706218888394671243274263815110980062304705972654147604250288441907534117123144073695655527041361858167525534229314911997362296923985815241767816481211399942 9	64	256
2076	$(1846389521368) + (11^{600})$	130	64

Table 2: Group orders of various sizes along with corresponding values of $d, \overline{r}$.

Bits in q	2^{δ_1}	$\overline{r} + \frac{r}{r-1}(r^{\delta_2} - 1)$
256	4096	6484
512	1024	1876
1024	1024	1620
2076	4096	5524

Table 3: The values that play crucial role in determining the number of iterations required for both algorithms to reach a distinguished point. Numbers on the right side are always greater than the corresponding ones on left.

A Appendix

This appendix lists the numerical values of various associated quantities used in our experiments. The prime ordered group size and $d, \overline{r}$ needed for Algorithm 5 is noted on Table 2. The theoretical estimates which would play a crucial role in determining the number of extra iterations needed for obtaining a distinguished point are mentioned in Table 3. In Section A.1 we argue that the time required to walk past each point obtained by taking the average time taken to walk through a large number of nodes of the pseudo-random walk is an appropriate quantity to compare the relative runtimes.

A.1 Average Time Noted from Experiments and their Comparison With Theoretical Estimates

In the Gaudry-Schost algorithm, all multiplications are done modulo p whereas for the case of utilization of the concept of tag tracing, multiplication modulo p happens roughly after l steps, and the rest are multiplications modulo w, u. The numerical values of w, u can be chosen to be powers of two, so such multiplications are cheap.
In the first column of both Table 4 and Table 5, we have considered the size q_1, q_2 of the groups G_{q_1}, G_{q_2}.

$b(p)_1, b(p)_2$	Experimentally observed		Theoretical estimate $(\frac{b(p)_1{}^2}{b(p)_2{}^2})$
	Tame	Wild	
$513, 257$	$\frac{0.560}{0.140} = 4$	$\frac{0.548}{0.142} = 3.859$	3.984
$1025, 513$	$\frac{2.215}{0.560} = 3.955$	$\frac{2.162}{0.548} = 3.945$	3.992
$2077, 1025$	$\frac{8.413}{2.215} = 3.798$	$\frac{8.436}{2.162} = 3.901$	4.106

Table 4: Comparison of the average time required in Gaudry-Schost algorithm for various bit-sizes. Values match with the theoretical estimates given on the right side.

It can be seen that the size of q_1 noted in that column is approximately twice that of q_2. This leads to similar nature in the sizes of corresponding p's as well.
Let us first focus on the cost analysis in the case of the Gaudry-Schost algorithm. Theoretically, the complexity of multiplication modulo p when the schoolbook method is applied is about $O(\log_2(p)^2)$. So the ratio of the costs of multiplication, from theoretical estimates must be about 4 if one prime is about two times the other. This is also reflected in the experimental observations both for tame and wild in columns 2 and 3 of Table 4. This harmonization makes sure that the ratios observed in column 5 of Table 1 are good enough to use for comparison purposes.
In case when the tag tracing algorithm is used, complete multiplication is needed roughly after every l steps as the other case arises when $\overline{s}$ may fail to produce an output. From the definition of $\overline{s}$ as in Section 8.1 this can happen roughly after every $\overline{r}$ steps. From our choice of $\overline{r}$ (Table 2) for the various primes over here, it can be seen that it is greater than $l = 20$. The complexity of multiplication is then $O(\frac{\log_2(p)^2}{l} + d \cdot C_{wu})$ where C_{wu} is the complexity of multiplication between w bit and u bit integers. It can be seen from the definition of d in Section Section 8.1 that it depends on p. The values of d for our examples have been listed in Table 2. So for comparing the costs of multiplications for primes p_1 and p_2, we compute the numerical value of the quantity $\frac{\frac{b(p_1)^2}{20 \cdot 32 \cdot 16} + d(p_1)}{\frac{b(p_2)^2}{20 \cdot 32 \cdot 16} + d(p_2)}$. It is evident from Table 5 that the theoretical estimates and the practical ones are quite close.
This justifies that the observed average time from experiments, which is the time taken to walk through each point of the pseudo-random walk is appropriate to use when the two algorithms have to be compared. It also confirms our intuitive supposition, that while comparing the complexities of the two algorithms, it is sufficient to look at the corresponding costs of multiplication.

References

[Bra93] Stefan A Brands, *An efficient off-line electronic cash system based on the representation problem.*

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers, *A secure and optimally efficient multi-authority election scheme*, European transactions on Telecommunications **8** (1997), no. 5, 481–490.

$b(p)_1, b(p)_2$	Experimentally observed		Theoretical estimate $\left(\frac{\frac{b(p_1)}{20\cdot 32\cdot 16}+d(p_1)}{\frac{b(p_2)}{20\cdot 32\cdot 16}+d(p_2)}\right)$
	Tame	Wild	
$513, 257$	$\frac{0.101}{0.041} = 2.463$	$\frac{0.105}{0.041} = 2.560$	2.570
$1025, 513$	$\frac{0.232}{0.101} = 2.297$	$\frac{0.245}{0.105} = 2.333$	2.887
$2077, 1025$	$\frac{0.678}{0.232} = 2.922$	$\frac{0.682}{0.245} = 2.783$	3.309

Table 5: Comparison of the average time for various sizes of the group in case tag tracing is also applied. Values are close to the theoretical predictions.

[CHK12] Jung Hee Cheon, Jin Hong, and Minkyu Kim, *Accelerating Pollard's Rho algorithm on finite fields*, Journal of cryptology **25** (2012), no. 2, 195–242.

[dif] *New directions in cryptography ieee transactions on information theory, v. it-22, n. 6.*

[GH00] Pierrick Gaudry and Robert Harley, *Counting points on hyperelliptic curves over finite fields*, International Algorithmic Number Theory Symposium, Springer, 2000, pp. 313–332.

[GLS09] Steven D Galbraith, Xibin Lin, and Michael Scott, *Endomorphisms for faster elliptic curve cryptography on a large class of curves*, Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2009, pp. 518–535.

[GLV00] Robert Gallant, Robert Lambert, and Scott Vanstone, *Improving the parallelized Pollard lambda search on anomalous binary curves*, Mathematics of Computation **69** (2000), no. 232, 1699–1705.

[GLV01] Robert P Gallant, Robert J Lambert, and Scott A Vanstone, *Faster point multiplication on elliptic curves with efficient endomorphisms*, Annual International Cryptology Conference, Springer, 2001, pp. 190–200.

[GPR13] Steven Galbraith, John Pollard, and Raminder Ruprai, *Computing discrete logarithms in an interval*, Mathematics of Computation **82** (2013), no. 282, 1181–1195.

[GR09] Steven Galbraith and Raminder S Ruprai, *An improvement to the Gaudry-Schost algorithm for multidimensional discrete logarithm problems*, IMA International Conference on Cryptography and Coding, Springer, 2009, pp. 368–382.

[GR10] Steven D Galbraith and Raminder S Ruprai, *Using equivalence classes to accelerate solving the discrete logarithm problem in a short interval*, International Workshop on Public Key Cryptography, Springer, 2010, pp. 368–383.

[GS04] Pierrick Gaudry and Éric Schost, *A low-memory parallel version of Matsuo, Chao, and Tsujii's algorithm*, International Algorithmic Number Theory Symposium, Springer, 2004, pp. 208–222.

[GS08] Steven D Galbraith and Michael Scott, *Exponentiation in pairing-friendly groups using homomorphisms*, International Conference on Pairing-Based Cryptography, Springer, 2008, pp. 211–224.

[JD08] Benits Junior and Waldyr Dias, *Applications of Frobenius expansions in elliptic curve cryptography*, Ph.D. thesis, Royal Holloway, University of London, 2008.

[Knu14] Donald E Knuth, *Art of computer programming, volume 2: Seminumerical algorithms*, Addison-Wesley Professional, 2014.

[Kob91] Neal Koblitz, *CM-curves with good cryptographic properties*, Annual international cryptology conference, Springer, 1991, pp. 279–287.

[MCT02] Kazuto Matsuo, Jinhui Chao, and Shigeo Tsujii, *An improved baby step giant step algorithm for point counting of hyperelliptic curves over finite fields*, International Algorithmic Number Theory Symposium, Springer, 2002, pp. 461–474.

[MS22] Madhurima Mukhopadhyay and Palash Sarkar, *Combining montgomery multiplication with tag tracing for the pollard rho algorithm in prime order fields*, International Conference on Security, Privacy, and Applied Cryptography Engineering, Springer, 2022, pp. 138–146.

[MVOV18] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone, *Handbook of applied cryptography*, CRC press, 2018.

[Niv04] Gabriel Nivasch, *Cycle detection using a stack*, Information Processing Letters **90** (2004), no. 3, 135–140.

[NS90] Kazuo Nishimura and Masaaki Sibuya, *Probability to meet in the middle*, Journal of Cryptology **2** (1990), no. 1, 13–22.

[PM99] C Paul and J Wiener Michael, *Parallel collision search with cryptanalytic applications*, Journal of Cryptology **12** (1999), no. 1, 01–28.

[Pol78] John M Pollard, *Monte Carlo methods for index computation (* mod *p)*, Mathematics of computation **32** (1978), no. 143, 918–924.

[Qui95] Jean-Jacques Quisquater, *How easy is collision search.*, Advances in Cryptology-CRYPTO'89: Proceedings, vol. 435, Springer, 1995, p. 408.

[SEL95] Boris I SELIVANOV, *On waiting time in the scheme of random allocation of coloured particies.*

[Sha71] Daniel Shanks, *Class number, a theory of factorization, and genera*, Proc. of Symp. Math. Soc., 1971, vol. 20, 1971, pp. 41–440.

[SSY82] Robert Sedgewick, Thomas G Szymanski, and Andrew C Yao, *The complexity of finding cycles in periodic functions*, SIAM Journal on Computing **11** (1982), no. 2, 376–390.

[Tes98] Edlyn Teske, *Speeding up Pollard's rho method for computing discrete logarithms*, International Algorithmic Number Theory Symposium, Springer, 1998, pp. 541–554.

[VOW94] Paul C Van Oorschot and Michael J Wiener, *Parallel collision search with application to hash functions and discrete logarithms*, Proceedings of the 2nd ACM Conference on Computer and Communications Security, 1994, pp. 210–218.

[Wen06] Annegret Weng, *A low-memory algorithm for point counting on Picard curves*, Designs, Codes and Cryptography **38** (2006), no. 3, 383–393.

[WZ21] Haoxuan Wu and Jincheng Zhuang, *Improving the Gaudry–Schost algorithm for multidimensional discrete logarithms*, Designs, Codes and Cryptography (2021), 1–13.

[ZZY+19] Yuqing Zhu, Jincheng Zhuang, Hairong Yi, Chang Lv, and Dongdai Lin, *A variant of the Galbraith–Ruprai algorithm for discrete logarithms with improved complexity*, Designs, Codes and Cryptography **87** (2019), no. 5, 971–986.