Improving Gaudry-Schost algorithm for multi-dimensional discrete logarithm calculations: Implementations relevant to electronic voting and cash schemes

Madhurima Mukhopadhyay

Improving Gaudry-Schost algorithm for multi-dimensional discrete logarithm calculations: Implementations relevant to electronic voting and cash schemes

Madhurima Mukhopadhyay

Department of Mathematics, Indian Institute of Technology Madras, Sardar Patel Road, Chennai 600036, India
ma24r008@smail.iitm.ac.in *

Abstract

We focus on improving the Gaudry-Schost algorithm, which solves multi-dimensional discrete logarithm problem. We have proposed a modified algorithm that reduces the cost of each iteration of Gaudry-Schost algorithm by a factor of $\frac{\frac{D-1}{\Delta}+\frac{1}{l}+\frac{C_{\overline{s}D}}{C_f}}{\frac{D-1}{\Delta}+1}$ where D is the dimension of the problem, l a specially designed quantity, C_g is the cost of computing any function g, $\overline{s}_D, f$ are functions and $(\frac{1}{l}+\frac{C_{\overline{s}D}}{C_f})<1$. The cost of our algorithm for subgroups modulo prime p, which arises in electronic voting and cash schemes diminishes by a factor of $\frac{\frac{||p||}{\Delta_{gs}}}{l\frac{\log(||p||)}{\Delta_{tt}}}$ where Δ's are the iterations to reach a certain type of points.

Our implementation of the algorithm confirms theoretical analysis. The reduction of cost per iteration sums to be much advantegeous when complete number of iterations have to be done to find the logarithms. Also, both theory and experiments confirm that the new algorithm reduces the dominant multiplication cost along with other additional costs, and the gain would be more as we increase the size of the group. We obtained about 12 times speed-up for groups of size 2076 bits.

Our algorithm will lead to a reduction of security of schemes based on multi-dimensional discrete logarithm problem or using multi-dimensional pseudo-random walk like electronic voting, cash schemes point-counting, speeding-up elliptic-curve arithmetic, group-actions, CSIDH etc.

Keywords: Discrete logarithm problem (DLP), Multi-Dimensional discrete logarithm problem, Gaudry-Schost algorithm, Electronic voting and cash schemes, Post-quantum cryptography, CSIDH

Mathematics Subject Classification (2020): 11T71 11Y16

1 Introduction

The hardness of discrete logarithm problem [17] is the basis of public-key cryptography. The discrete logarithm problem (DLP) [23, 26, 24] is: 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 $h = g^a$. This problem

*Another email: mukhopadhyaymadhurima@gmail.com

is for a single generator and so we can think this to be for dimension one. 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 $N = N_1 \times N_2 \times \ldots \times N_D$.

This definition [1] does not assume any relation between N and integers N_i with order of the group or the elements g_i respectively for each $i = 1, 2, \ldots, D$.

1.1 Applications

The multi-dimensional discrete logarithm problem arises in several situations which include structures as curves or a group modulo some integer. We note that computation of DLP in an interval is one of the steps to count points on curves over finite fields [15, 16, 22, 32]. Gaudry and Schost [16] developed their algorithm for the case of curves of genus 2. After using the Schoof-type algorithm, the remaining problem reduces to a multi-dimensional DLP.
GLV method to speed up elliptic curve arithmetic [14], requires expressing n times addition of a point P *i.e.*, $[n]P$ as $[n_1]P + [n_2]\psi(P)$ for some integers n_1, n_2 where ψ is an endomorphism. The bound on n_1, n_2 is that $|n_1|, |n_2| \approx \sqrt{n}$. This is an example of 2-dimensional DLP[2]. The same approach of expressing multiple of a point to a multi-dimensional DLP [19] can happen in Koblitz curves [21] 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 [2] and election scheme [7].

1.1.1 Recent post-quantum cryptography

Present public-key cryptography utilises hardness of integer-factorisation or discrete logarithm problem in finite field or elliptic curves that are not quantum-safe[3], meaning the availability of quantum-computing systems leads to insecurity. This has led to spiking interest in computational problems that cannot be efficiently solved by quantum computers. This field of study is known as *Post-Quantum Cryptography*, with a recent suggestion being *isogeny-based cryptography*. Group actions are used in

[1] There are equivalent formulation of the problem for additive abelian groups. Let G be an additive abelian group. The problem would be to find $a_1, a_2, \ldots, a_D \in \mathbb{Z}$ 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.

[2]. For dimensions that exceed 2, other examples of such work are present in [11, 13]

[3] Shor's algorithm, Grover's algorithm can lead to such problems being tractable.

CSIDH-schemes [4, 1, 8]. They have not been affected by the recent attacks [3, 30], which broke another isogeny-based scheme called SIDH [18]. The security of CSIDH is based on the intractability of the corresponding *group-action inverse problem* which can be related to the multi-dimensional discrete logarithm computations [Section 3, [20]]. Studying the hardness of the multi-dimensional discrete logarithm problem thus remains vital in post-quantum context.

1.2 Previous works

To solve the multi-dimensional discrete logarithm problem, Matsuo et al [22] adopted the baby-step -giant -step algorithm [31]. The time and space complexity of this algorithm is $O(\sqrt{N})$. Gaudry and Schost [16] proposed a low-memory algorithm by applying pseudo-random walks (with exponents of g_i's belonging to two types of sets called *tame* or *wild* sets). This algorithm can be modified for solving 2-dimensional DLP and can also be generalized for any multi-dimensional situation.

Gaudry and Schost used a deterministic pseudo-random walk where the elements of the walk are powers of the bases g_i. As they have presented such walks in a finite group, elements of the walk will certainly collide. We can find the multi-dimensional discrete logarithms by tracking the exponents of g_i's. Several improvements to the Gaudry-Schost algorithm have been proposed assuming a specialized structure of the group [12, 34, 6] or special choice of tame and wild sets [9, 33].
There are exactly two major ways to reduce the cost of the algorithm. The first is to lessen the number of iterations required by the pseudo-random walk to get a collision, and the second is to cut down the cost per iteration. The first strategy can be acheived by designing suitable tame and wild sets. Galbraith and Holmes [10] showed that, using two types of sets like tame and wild, the minimal number of iterations required is $\sqrt{\pi N}$. The recent choice of tame and wild sets in [33], has lowered the expected averge case complexity to $1.0171^D\sqrt{\pi N}$, which is already very close to the predicted bound of $\sqrt{\pi N}$. This implies that given the present state-of-the-art, to improve this algorithm we shoould focus on the second possiblity.

1.3 Importance of our work

We resort to the second method of the above paragraph to reduce the cost of the algorithm. The major work while performing each step of the walk in the Gaudry-Schost algorithm is to multiply two elements. The other jobs are minor as they comprise finding some index and keeping track of the exponents. The cost of multiplication becomes all the more pronounceable as the size of the underlying group increases. Cheon *et. al.*, [5] applied tag-tracing in pseudo-random walks of the Pollard Rho algorithm to reduce the number of multiplications in the original group and perform necessary multiplications in smaller subsets of the group. Multiplication in subsets was less costly than it's counterpart in the larger group. In this paper, we try to investigate whether tag-tracing can be used to improve the Gaudry-Schost algorithm. The reason this is necessary is primarily due to the differences between the earlier application scenario and the present one:

1. The previous application of tag-tracing was done for dimension one, whereas the Gaudry-Schost algorithm is for dimensions that exceed one.

2. Unlike the Pollard Rho algorithm, the Gaudry-Schost algorithm initiates another fresh walk on arriving at a special type of points called distinguished point.

3. For Gaudry-Schost algorithm walks are of two types(tame and wild)[4], while for Pollard Rho algorithm walks are of a single type.

These differences are quite significant, and hence to declare that tag-tracing is an effective method to reduce the cost, even when optimal iterations are reached, we need to study the behavior of Gaudry-Schost algorithm with and without tag-tracing. No sort of experimentation of tag-tracing has been done before for the pseudo-random walks of the Gaudry-Schost algorithm type. This also makes it necessary for us to do a implementation of the Gaudry-Schost algorithm when combined with tag-tracing.
We have described the entire procedure to incorporate tag tracing into the Gaudry-Schost algorithm in Section 3. We have derived the precise estimations of cost in both generalized versions and subgroups modulo primes. We have applied our algorithm to compute multi-dimensional discrete logarithms that arises in the set-up of electronic cash scheme [2] and election scheme [7]. We observe a significant speed-up, and our experimental verifications agree with the theoretical estimates of costs that we have derived.

1.4 Our contributions

Designing the algorithm to use tag-tracing with Gaudry-Schost algorithm: We have presented the application of tag-tracing to the Gaudry-Schost algorithm in Algorithm 5 to compute multi-dimensional discrete logarithms. Such an application is new and differs(Section 1.3) from it's previous applications .
Calculation of the decrease in cost: We have derived that, for any given group, while the cost of Gaudry-Schost algorithm is $(\overline{N}+\Delta)(\frac{C_D}{\Delta}+C_f)$, the corresponding cost of tag-tracing is $\left(\overline{N}+\Delta\right)\left(\frac{C_D}{\Delta}+(Pr(\overline{s} \text{ fails})+\frac{1}{l})C_f+C_{\overline{s}}\right)$ where $\overline{N},\Delta$ are iterations to reach collisions and distinguished points respectively, while $C_D, C_f, C_{\overline{s}}$ are the costs of D multiplications, single multiplication and computing $\overline{s}$ respectively and l is a fixed parameter. In particular, we show that the cost of Gaudry-Schost algorithm always decreases when we use tag-tracing, at the trade-off of a negligible increase in pre-computation cost(Lemma 3). We have derived the exact ratio of this decrease as $\frac{\frac{D-1}{\Delta}+1}{\frac{D-1}{\Delta}+\frac{1}{l}+\frac{C_{\overline{s}D}}{C_f}}$ which by design of functions, increases with the increase of l, D.
Tag-tracing and cost estimates in subgroups modulo primes: We next concentrate on subgroups modulo primes, p, which have applications in electronic voting and cash schemes. We derived that the cost of tag-tracing here is $\left\{\frac{D-1}{\Delta}\text{Mul}(||p||+\left(\frac{1}{l}+\frac{1}{\bar{r}}\right)\text{Mul}(||p||))+d\ \text{Mul}(||w||)\right\}(1.0171^D\sqrt{\pi N}+\Delta)$, for some appropriately chosen integers $\bar{r}, w, d$. We have further simplified this cost in Lemma 10 and Theorem 11. In Theorem 12, we prove the cost of Gaudry-Schost algorithm is greater than tag-tracing for a fixed D by the ratio $\frac{\frac{||p||}{\Delta_{gs}}}{l^{\frac{\log(||p||)}{\Delta_{tt}}}}$, where Δ_{gs}, Δ_{tt} are the iterations to reach distinguished points for Gaudry-Schost algorithm or the tag-tracing counterpart respectively.
Space-complexity: We have shown that the space required by tag-tracing is a polynomial function of l, D and the bit-size of the group in general case. This size is the bits in the prime for subgroups modulo primes.
Implementation confirming theory: We implemented the new algorithm in subgroups of $\mathbb{Z}_p^*$, relevant in the context of electronic voting and cash schemes. Our code is available at:

[4] 3 or 4 sets can also be used in other designs.

https://github.com/Madhurima11/Multi-Dimensional-Discrete-Logarithm.

This experimentation ensured that our theoretical predictions of decrease in cost were valid(Section 7.3). This implementation also certified that, tag-tracing reduces the dominant multiplication cost along with other additional costs and the gain would be more as we increase the size of the group. We obtained about 12 times speed-up with 2076 bit-sized groups.
Pointing out future research avenues: We have also noted some future research directions to gain from increase of the dimension of the problem.

1.5 Paper organization

In Section 1, we define the multi-dimensional discrete logarithm problem and note some applications. We describe the previous research works and point out the specific region where we would like to improve upon. We next state our contributions in this paper. In Section 2, we describe the Gaudry-Schost algorithm, which is used to compute multi-dimensional discrete logarithms. In Section 3, we give a detailed account of the method of inclusion of tag tracing into the Gaudry-Schost algorithm. Specifically, we describe the entire process in Algorithm 5. In Section 4, we theoretically derive the costs of both scenarios of Gaudry-Schost algorithm, with and without tag-tracing. We next deduce the theoretical estimate of the factor by which tag-tracing improves the algorithm. We concretely note some cryptographically relevant schemes in Section 5, where our improvements are applicable. After this part, we end the generalized view and fix our attention towards subgroups of $\mathbb{Z}_p^*$. We describe the associated functions and the values of the parameters required for tag tracing in subgroups of $\mathbb{Z}_p^*$ in Section 6. We also derive the time and space complexities in this case. We present the results of implementation of Gaudry-Schost algorithm with and without tag-tracing in Section 7 along with the comparison with our predicted estimates. We point out some future research directions in Section 8. Lastly, we summarize the conclusions that can be drawn from our study in Section 9.

2 Gaudry-Schost algorithm adapted to multi-dimensional discrete logarithm computation

In this section, we explore the adaptation of the Gaudry-Schost [16] algorithm to multi-dimensional discrete logarithm computation. 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*. [5]

2.1 Offline phase

Gaudry and Schost [16] chose two positive integers $n_s(> \log(\max(N_1, \ldots, N_D)))$ and M_{gs} to define a selection function $S : G \to \{0, 1, \ldots, n_s - 1\}$ for partitioning the group into n_s components, almost

[5] 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.

uniformly.[6]
Next, they pre-computed n_s elements in a table P as:

$$P := \{w_j = \prod_{i=1}^{D} g_i^{e_{i,j}} | -M_{gs} < e_{i,j} < M_{gs}, j = 0, 1, \ldots, n_s - 1\} \tag{3}$$

The exponent bound $M_{gs} \approx \frac{N}{\Gamma \log_2(N)}$ where the choice of Γ can be made so that all elements of P are distinct[7]. For practical purposes, it is simply chosen as some suitable power of 10.
They defined the $(k+1)$-th element of the deterministic pseudo-random walk as:

$$v_{k+1} = v_k \cdot w_{S(v_k)} \tag{4}$$

where $v_k = \prod_{i=1}^{D} g_i^{x_{i,k}}$ is the $k-$th element and its exponents

$$(x_{1,k+1}, x_{2,k+1}, \ldots, x_{D,k+1}) = (x_{1,k} + e_{1,S(v_k)}, x_{2,k} + e_{2,S(v_k)}, \ldots, x_{D,k} + e_{D,S(v_k)}) \tag{5}$$

The algorithm for the online phase is as below:

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
```

2.2 Collision detection

The most efficient method to detect a match between tame and wild walks is the method of distinguished points [29]. A *distinguished point* is any element of the group G that satisfies certain conditions, with probability p_D.[8] About $\rho_D (= \frac{1}{p_D})$ extra iterations are necesary to reach a distinguished point, the exact estimate of ρ_D will depend on it's trade-off with the cost to store distinguished points.[9] The distinguished point method can be parallelized [27] and leads to speed-ups that are linear in the number of processors.

[6] It may be a hash function with good statistical properties.

[7] An implicit assumption here is that the values of N_i are quite close to each other. If this is not the case in some situations, then we can construct the bound for each e_i so that it depends on the corresponding N_i in the same manner as above. Our aim would be to ensure that when we perform the walk, the exponents of g_i do not exceed the associated N_i.

[8] We can define these conditions so that they are easy to check. For example, given a fixed encoding of the group, we can denote distinguished points as those elements of the group that have a certain number of most (or least) significant bits equal to zero.

[9] To optimize the space overhead cost, the distinguished points are stored in an easily searchable structure such as a hash table.

2.2.1 Abandoning a walk

There is a possibility that we can encounter a walk that does not reach a distinguished point after many iterations. Van Oorschot and Wiener [27] defined a bound on the number of steps after which it can be abandoned as $L = \frac{20}{\rho_D}$. The proportion of walks that exceed this length is bounded by $(1-\rho_D)^{\frac{20}{\rho_D}} \leq (\exp(-\rho_D))^{\frac{20}{\rho_D}} = \exp(-20)$. This means that any abandoned trail is 20 times longer than the average length. Thus proportion of walks that are abandoned is about $20\exp^{-20} < 5 \times 10^{-8}$, which is negligible.

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, $h = \prod_{i=1}^{D} {g_i}^{x'_i - y'_i}$. We can obtain the multi-dimensional discrete logarithm as $a_i = x'_i - y'_i \; \forall \; 1 \leq i \leq D$.

2.4 Online phase

Algorithm 2 for the server will receive points both from tame and wild sides. The multi-dimensional discrete logarithm problem will be solved if a common point is found between both sides. Otherwise, it just appends the point to the table of distinguished points allotted for that side. Algorithm 3 is for the tame and wild processors. It initiates the walk from a random tame (or wild) point and continues until it hits a distinguished point.

2.5 Different choices of tame and wild sets

Gaudry and Schost [16] proposed their algorithm taking the tame and wild set as orthotopes in $\mathbb{Z}^d$. Later, Galbraith and Ruprai [9], proposed constant size of overlap, which lead to same expected running time for best, average, and worst cases. Wu and Zhuang [33] constructed another pair of tame and wild sets, with the second framework being asymptotically optimal.

We note that for the 1-dimensional case, the problem of finding $0 \leq a < N$ such that $h = g^a$ where $g, h \in G$ and $N \in \mathbb{N}$, can also be framed as finding x, $0 \leq x < 1$ such that $h = g^{xN}$.

In the second variant, Wu and Zhuang [33] defined the tame and wild sets for dimension 1 as:

$$\mathcal{T} = \left[0, N\right], \tag{6}$$

$$\mathcal{W} = \left[xN - \frac{\alpha N}{2}, xN + \frac{\alpha N}{2}\right] \tag{7}$$

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.

$$\mathcal{T} = \left[0, N\right]^D, \tag{8}$$

$$\mathcal{W} = \left[xN - \frac{\alpha N}{2}, xN + \frac{\alpha N}{2}\right]^D \tag{9}$$

where N is same as in Definition 1.

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 $\Big($*no collision between tame and wild points has been found*$\Big)$ **do**
  Receive a point $(z, b_1, \ldots, b_D)$ from the tame or wild processor
  if $\Big($ *z is received from a tame processor*$\Big)$ **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 $\Big((z, x_1, \ldots, x_D) \in D_T$ *for some* $x_1, \ldots, x_D\Big)$ **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

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

Theorem 1. *(Section 4.2, [33]) The expected average case complexity with the tame and wild set choice as provided in equations 8 and 9 is* $1.0171^D\sqrt{\pi N}$. *The expected worst case complexity is* $2^{\frac{D}{2}}\sqrt{\pi N}$. *The values of* N, D *can be set from Definition 1.*

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

3 Application of tag tracing in Gaudry-Schost algorithm

In Gaudry-Schost algorithm, the two main factors contributing to the cost are the number of iterations and complexity of obtaining the next point at each iteration. Average number of iterations as in Theorem 1, has already acheived a close approximation to $\sqrt{\pi N}$. Then, to reduce the complexity, we focus on reducing cost of each iteration. We note that any point in the pseudo-random walk is required to be known completely only if it is a distinguished point. Thus multiplication at each step of the walk is unnecessary if we can somehow extract the information required to test the condition. The principle of tag tracing [5], originally introduced in the context of the Pollard Rho algorithm [28], can traverse through a random walk without fully computing each point. As there are notable differences (Section 1.3) between the Gaudry-Schost walk and the Pollard Rho type of walk: we desire to investigate

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** $\left($*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** $\left(z$ *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 the 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.

whether tag-tracing is effective in the Gaudry-Schost algorithm. We note that the tag-tracing concept has not been used before for the multi-dimensional scenario.

3.1 Offline phase

The concept of tag tracing requires a larger pre-computed table. Initially, it computes a table $\mathcal{M}$ where

$$\mathcal{M} := \{w_j = \prod_{i=1}^{D} {g_i}^{e_{i,j}} | -M_{tt} < e_{i,j} < M_{tt}; j = 0, 1, \ldots, r-1\} \tag{10}$$

where r[10], $M_{tt}(\approx \frac{N}{\Gamma \log_2(N)})$[11] are positive integers for size and bound on exponents. We set $\mathcal{M}^{(0)} = \{1_G\}$, $\mathcal{M}^{(1)} = \mathcal{M}$ and for the next $l(> 0)$ steps compute:

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

Remark 1. *We can parallelize computation of each* $\mathcal{M}^{(k+1)}$ *by multiplying each element of* $\mathcal{M}$ *to every of* $\mathcal{M}^{(k)}$

[10] A possible choice of r that we can take is n_s.

[11] We have to define Γ suitably so that the table $\mathcal{M}$ contains all distinct elements.

. We set $\mathcal{M}_l = \mathcal{M}^{(0)} \cup \mathcal{M}^{(1)} \cup \mathcal{M}^{(2)} \cup \ldots \cup \mathcal{M}^{(l)} = \{w_{j_1} w_{j_2} \ldots w_{j_k}, 1_G | 0 \leq j_k \leq (r-1) \forall k = 1, 2, , \ldots, l\}$. We outline the entire procedure in Algorithm 4.

Lemma 2. *The total number of elements in $\mathcal{M}_l$ is $\binom{r+l}{r}$.*

Proof. To construct $\mathcal{M}_l$, we are choosing the first r elements randomly. At the next phase, we choose k elements at a time, for $2 \leq k \leq l$. Each of these k elements is either the identity of the group or some element that we chose randomly at the first step. We have used Lex ordering of indices in Algorithm 4 to avoid the same element from appearing more than once.
Also, once we choose the first r elements randomly, the multi-dimensionality of the problem will play no role. This is equivalent to choosing any l elements from a set of r elements or a special element(which is the identity here), where the combinations can have repetitions. The number of such elements is then $\binom{r+l}{r}$. ■

3.1.1 Storing the table $\mathcal{M}_l$:

We store each entry $\prod_{t=1}^{k} w_{j_t} = \prod_{i=1}^{D} g_i^{\sum_{t=1}^{k} e_{i,j_t}}$ of table $\mathcal{M}_l$ as $[(j_1, j_2, \ldots, j_k), \prod_{t=1}^{k} w_{j_t}, (\sum_{t=1}^{k} e_{1,j_t}, \sum_{t=1}^{k} e_{2,j_t}, \ldots, \sum_{t=1}^{k} e_{D,j_t})]$. We sort according to multiplier combination information, or a hash table technique.

Lemma 3. *The number of group operations to construct $\mathcal{M}_l$ is a polynomial of $D, \binom{r+l}{r}$.*

Proof. We are constructing the first r elements by multiplying the D elements. After we initially choose r elements, the dimension will not arise, and we need to multiply a single element at each step. This makes the number of group operations a polynomial function of the dimension and the size of the table. ■

3.2 The pseudo-random walk and the functions required

We define an auxiliary index function $\overline{s} : G \times \mathcal{M}_l \to \{0, 1, \ldots, n_s - 1\}$ with the help of s as:

$$\overline{s}(y, m) = s(ym) \ \forall y \in G, m \in \mathcal{M}_l \tag{11}$$

By the above equation, when we already know the auxiliary index of some point g_i of the walk, we can compute the next l auxiliary indices without any multiplication.

$$\begin{cases} g_{i+1} = g_i m_{s(g_i)} \implies s(g_{i+1}) = \overline{s}(g_i, m_{s(g_i)}) \\ g_{i+2} = g_{i+1} m_{s(g_{i+1})} = g_i m_{s(g_i)} m_{s(g_{i+1})} \implies s(g_{i+2}) = \overline{s}(g_i, m_{s(g_i)} m_{s(g_{i+1})}) \\ \vdots \\ g_{i+l} = g_{i+l-1} m_{s(g_{i+l-1})} = g_i m_{s(g_i)} m_{s(g_{i+1})} \ldots m_{s(g_{i+l-1})} \implies s(g_{i+l}) = \overline{s}(g_i, m_{s(g_i)} m_{s(g_{i+1})} \ldots m_{s(g_{i+l-1})}) \end{cases} \tag{12}$$

where $m_{s(g)}$ denotes the element of $\mathcal{M}_l$ corresponding to the element g of the group.
We can access the product $\prod_j m_{s(g_{i+j})}$ from $\mathcal{M}_l$. We design $\overline{s}$ so that we can compute $\overline{s}(y, m)$ easier than $y \cdot m$.

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 I, 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 $\Big(\{I \neq [(r-1), (r-1), \ldots, (r-1)]\}\Big)$ **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 I 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 $\mathcal{M}_l$

We define additional functions: Tag function $\tau : G \to \mathcal{T}$, Auxiliary tag function $\overline{\tau} : G \times \mathcal{M}_l \to \mathcal{T}$, Projection function $\sigma : \mathcal{T} \to \mathcal{S}$, Auxiliary projection function $\overline{\sigma} : \mathcal{T} \to \mathcal{S} \cup \{\text{Fail}\}$, where $\mathcal{T}(\supset \{0, 1, \ldots, n_s - 1\})$, so that indexing functions $s, \overline{s}$ are easily computable, surjective, pre-image uniform and $\tau, \overline{\tau}$ on most occasions, can produce output $\tau(\prod_j m_{s(g_{i+j})})$ or $\overline{\tau}(\prod_j m_{s(g_{i+j})})$, with just the knowledge of divisors $m_{s(g_{i+j})}$, without computing the product. When this fails, we have to compute $\prod_j m_{s(g_{i+j})}$. We define $s = \sigma \circ \tau : G \to \mathcal{S}$ and $\overline{s} = \overline{\sigma} \circ \overline{\tau} : G \times \mathcal{M}_l \to \mathcal{S} \cup \{\text{fail}\}$ so that equation 11 holds. We define a point z_i, $i \geq (\delta - 1)$ (where $\delta \geq 1$) as distinguished if the index of itself and each of it's $\delta - 1$ ancestors are zero:

$$z_i \text{ is } \textit{distinguished} \text{ if } s(z_{i-\delta+1}) = \ldots = s(z_{i-1}) = s(z_i) = 0 \tag{13}$$

The theorem below, summarizes the additional iterations required, by taking into consideration that, the probability of the image of a point under s to be zero is $\frac{1}{r}$.

Theorem 4. *[Theorem 1, [5]]If distinguished points are defined by a condition of δ consecutive points satisfying a condition with probability $\frac{1}{r}$, then for a random iteration function, the expected number of iterations to arrive at a distinguished point is $\Delta = \frac{r}{r-1}(r^\delta - 1)$*

Equations 11 and 13 imply a point z_i is a distinguished point if

$$\overline{s}(z_{i-\delta+1}, w_{i-\delta+1}) = \overline{s}(z_{i-\delta+2}, w_{i-\delta+2}) = \ldots = \overline{s}(z_{i-1}, w_{i-1}) = 0 \tag{14}$$

Apparently, this definition seems questionable at points that occur within δ iterations. But, we can neglect this ambiguity as δ is of logarithmic order in Δ, which means $O(\Delta)$ extra iterations will be required after the point of collision.

3.3 Online phase

Algorithm 5 is the online phase of Gaudry-Schost algorithm, modified with tag-tracing. The initial steps of choosing random points in tame(wild) sets remain the same. In the next iterative steps, instead of performing complete multiplication, we get the multiplier w from $\mathcal{M}_l$, and can check whether the product of z and w is a distinguished point. We use our definition of distinguished point using auxiliary functions which performs it roughly after l steps, instead of every step.

Choosing l: We have the advantage of doing lesser full product computations by selecting l large enough. But increasing l will also lead to a larger table $\mathcal{M}_l$. We should consider this trade-off along with the approximation in $l.M_{tt} \approx \frac{N}{\Gamma \log_2(N)}$ when we fix l.

4 Cost comparison for original Gaudry-Schost algorithm and modification using tag-tracing

4.1 Offline phase

Theorem 5. *The time required for offline phase is negligible whether tag-tracing is applied or not.*

Proof. Follows from considerations on size of n_s, Lemma 2 and Remark 1. ■

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

4.2 Online phase

Theorem 6. *Let $C_D, C_f, C_{\overline{s}}$ denote the costs for computing a random point with the help of $g_1, g_2, \ldots, g_D$, the cost of fully computing products of elements in G, and the cost computing $\overline{s}$ respectively. Also let $\overline{N}, \Delta$ denote the number of iterations required for finding collisions and distinguished points respectively. The costs of both the algorithms are as follows:*[12]
The total complexity of online phase of original Gaudry-Schost algorithm(Algorithm 3) is

$$(\overline{N} + \Delta)(\frac{C_D}{\Delta} + C_f) \tag{15}$$

The corresponding complexity of the online phase of the modifcation with tag-tracing(Algorithm 5) is

$$\big(\overline{N} + \Delta\big)\big(\frac{C_D}{\Delta} + (Pr(\overline{s}\ fails) + \frac{1}{l})C_f + C_{\overline{s}}\big) \tag{16}$$

Proof. Both Algorithm 3 and Algorithm 5 search for collision among distinguished points of G. Hence, total iterations is the sum $\big(\overline{N} + \Delta\big)$. Also, for both the algorithms, the dominant cost is the cost of multiplication among group elements.
Algorithm 3 requires multiplication of D elements only(line 3) when it reaches a distinguished point after Δ iterations, and initiates a new walk. All $\big(\overline{N} + \Delta\big)$ cases will need multiplication between two elements of G. This explains the expression in equation 15.
In Algorithm 5, the number of multiplications of D elements is unchanged from Algorithm 3. $\overline{s}$ computations are required in all(Lines 9 and 16) $\big(\overline{N} + \Delta\big)$ cases, and they are sufficient except when they fail or the walk completes a set of l iterations from a random point. These $\big(\overline{N} + \Delta\big)(Pr(\overline{s}\text{ fails}) + \frac{1}{l})$ cases need fully computing the product (line 19). This proves equation 16. ■

Theorem 7. *On designing $\overline{s}$ such that it can be computed easily with high probability on almost every point of the G, the online phase of modified version of Gaudry-Schost algorithm with tag-tracing, has lesser cost than online phase of original Gaudry-Schost algorithm.*
Considering $Pr(\overline{s}\ fails) \to 0$, and the cost of initially performing multiplications between D elements can be expressed in terms of a single multiplication, the advantage of applying tag-tracing increases with the increase of l.

Proof. The conditions mean that $C_{\overline{s}} << C_f$ and $Pr(\overline{s}\text{ fails})$ is negligible.
More formally, let us assume,

$$C_{\overline{s}} < \big(1 - (Pr(\overline{s}\text{ fails}) + \frac{1}{l})\big)C_f \tag{17}$$

Above implies,

$$\big(C_{\overline{s}} + (Pr(\overline{s}\text{ fails} + \frac{1}{l})C_f\big) < C_f$$

$$\implies \big(\overline{N} + \Delta\big)(\frac{C_D}{\Delta} + (Pr(\overline{s}\text{ fails}) + \frac{1}{l})C_f + C_{\overline{s}}\big) < (\overline{N} + \Delta)(\frac{C_D}{\Delta} + C_f)$$

This proves that the online phase of tag-tracing has lesser cost.

[12] We do not take into consideration the length L for abandoning the walk, as for nearly all instances the length of the pseudo-random walk, when it reaches a distinguished point would be much less than L.

For the next part, we note that

$$\frac{\text{Cost of Gaudry-Schost Algorithm}}{\text{Cost of modification with tag-tracing}} = \frac{(\frac{C_D}{\Delta} + C_f)}{(\frac{C_D}{\Delta} + \frac{1}{l}C_f + C_{\overline{s}})} \quad \text{(Assuming } Pr(\overline{s} \text{ fails}) \to 0\text{)}$$

$$= \frac{\frac{D-1}{\Delta}C_f + C_f}{\frac{D-1}{\Delta}C_f + \frac{1}{l}C_f + C_{\overline{s}}} \quad \text{(Using } C_D = (D-1)C_f\text{)}$$

$$= \frac{\frac{D-1}{\Delta} + 1}{\frac{D-1}{\Delta} + \frac{1}{l} + \frac{C_{\overline{s}}}{C_f}} \tag{18}$$

Clearly, as l increases, the value of above expression increases. ■

Remark 2. *It follows from the proof of the above theorem that a condition on the failure probability of* $\overline{s}$ *is* $Pr(\overline{s}$ *fails*$) < \left(1 - \frac{1}{l}\right)$

Remark 3. *Tag-tracing is more advantegeous as* $Pr(\overline{s}$ *fails*$) \to 0$.

The above theorems prove that the concept of tag tracing is effective when used with the Gaudry-Schost algorithm to compute multi-dimensional discrete logarithms. It reduces the work per iteration, by reducing the number of multiplications with a larger pre-computed table and some easily computable functions, to detect distinguished points. To increase the potency of tag-tracing in multi-dimensional scenario, the associated functions must be designed obeying certain conditions.

Theorem 8. *The effectiveness of tag-tracing increases with dimension of the multi-dimensional discrete logarithm problem, if the function* $\overline{s}$ *is designed so that it depends on the dimension, has high success probability and the cost of computing it satisfies* $\frac{C_{\overline{s}_{D_2}}}{C_f} < \frac{C_{\overline{s}_{D_1}}}{C_f} - \frac{(D_2 - D_1)}{\Delta}$ *whenever* $D_2 > D_1$*, where* D_2, D_1 *are the dimensions of the multi-dimensional discrete logarithm problem, and* Δ *is the number of iterations to reach a distinguished point.*

Proof. Let us denote the ratio of costs from equation 18 by R_D. It is enough to then show that under the given condition, $R_D = \frac{\frac{D-1}{\Delta}+1}{\frac{D-1}{\Delta}+\frac{1}{l}+\frac{C_{\overline{s}}}{C_f}}$ will increase with the value of D. Let $D_2 > D_1$ and the probabilites of failure of the functions $\overline{s}_{D_1}$ and $\overline{s}_{D_2}$ for dimensions D_1 and D_2 be negligible.

$$\text{Now, } \frac{C_{\overline{s}_{D_2}}}{C_f} < \frac{C_{\overline{s}_{D_1}}}{C_f} - \frac{(D_2 - D_1)}{\Delta}$$

$$\implies \frac{D_2}{\Delta} + \frac{C_{\overline{s}_{D_2}}}{C_f} < \frac{C_{\overline{s}_{D_1}}}{C_f} + \frac{D_1}{\Delta}$$

$$\text{Adding } (\frac{1}{l} - \frac{1}{\Delta}) \text{ to both sides, } \frac{D_2 - 1}{\Delta} + \frac{1}{l} + \frac{C_{\overline{s}_{D_2}}}{C_f} < \frac{D_1 - 1}{\Delta} + \frac{1}{l} + \frac{C_{\overline{s}_{D_1}}}{C_f}$$

$$\implies \frac{1}{\frac{D_2-1}{\Delta} + \frac{1}{l} + \frac{C_{\overline{s}_{D_2}}}{C_f}} > \frac{1}{\frac{D_1-1}{\Delta} + \frac{1}{l} + \frac{C_{\overline{s}_{D_1}}}{C_f}} \tag{19}$$

$$\therefore R_{D_2} = \frac{\frac{D_2-1}{\Delta}+1}{\frac{D_2-1}{\Delta}+\frac{1}{l}+\frac{C_{\overline{s}D_2}}{C_f}} > \frac{\frac{D_1-1}{\Delta}+1}{\frac{D_2-1}{\Delta}+\frac{1}{l}+\frac{C_{\overline{s}D_2}}{C_f}} \text{ [As } D_2 > D_1\text{]}$$
$$> \frac{\frac{D_1-1}{\Delta}+1}{\frac{D_1-1}{\Delta}+\frac{1}{l}+\frac{C_{\overline{s}D_1}}{C_f}} \text{ [By inequality 19]} = R_{D_1}$$

■

4.3 Other additional costs

In the unmodified Gaudry-Schost algorithm, exponents have to be updated everytime (Line 9 of Algorithm 3), whereas in tag-tracing they can be updated after l iterations generally (unless $\overline{s}$ fails or distinguished point arises) (Line 22 of Algorithm 5). Thus, tag-tracing leads to D additions after a certain number of steps, whereas the original algorithm needs it after each step. This leads to additional communication cost as well.

4.4 Improvement in groups of prime order

Let G be a prime-ordered subgroup of some group containing $(p-1)$ elements for a prime p. When we apply tag-tracing to the Gaudry-Schost algorithm to compute multi-dimensional discrete logarithms in such groups, we can incorporate Montgomery multiplication into tag-tracing, as in[25] to further reduce costs.

5 Cryptographically relevant groups

Electronic cash and election schemes: An interesting case of application of our proposal is in the electronic cash scheme [2] and election scheme [7]. 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 $q|(p-1)$. For practical purposes, we can take the bit-size of p, q to be at least $2048, 256$ respectively, and the dimension d as $2, 3$. Higher values of d also work. We shall discuss the adoption of tag tracing in such groups in Section 6.

Other applications Our proposal can also be used in other applications[Section 1.1]. For example, Kim mentions in [20] that the security of CSIDH schemes can be reduced to 68 bits instead of 128. Employing our modification should lead to a further decrease in this security.

6 Designing suitable functions and resulting complexity for subgroups of $\mathbb{Z}_p^*$

We can adapt the functions defined by Cheon, et. al., [5], for appliying tag-tracing to Pollard Rho algorithm for prime ordered subgroups of $\mathbb{Z}_p^*$, in our context of multi-dimensional discrete logarithms.

Tag-function: This is used to detect distinguished points, which are easily defined in terms of some most or least significant bits. Analogously here we use some suitable powers of two, and obtain the terms by division. Let r, w be some suitable powers of two along with the terms $u = \sqrt{w}, d = \lceil \log_u(p-1) \rceil$, $\overline{t} = 2^{\lceil \log_2 du \rceil}$, $t = \frac{w}{\overline{t}}$, $\overline{r} = \frac{t}{r}, \bar{w} = \lceil \frac{p}{w} \rceil$. Let $\tau : G \to \mathcal{T} = \{0, 1, \ldots, (t-1)\}$ $\tau(z) = \lfloor \frac{z \mod p}{\overline{t}\overline{w}} \rfloor$.

Index function: Let $\sigma : \mathcal{T} \to \mathcal{S} = \{0, 1, \ldots, (r-1)\}$ such that $\sigma(z) = \lfloor \frac{z}{\overline{r}} \rfloor$. We can define the index function as $s = \sigma \circ \tau$. This function is roughly pre-image uniform[Proposition 1, [5]].

Auxiliary tag function: We can represent each element $z \in \mathbb{Z}_p^*$ in base u as $z = z_0 + z_1 u + \ldots + z_{d-1} u^{(d-1)}$. For every $m \in \mathcal{M}_l$, let $\hat{m}_i = \lfloor (u^i m \bmod p)/\overline{w} \rfloor$. We can pre-compute these values as a part of the offline work and store them in $\mathcal{M}_l$ itself. Based on this, the auxiliary tag function is:

$$\overline{\tau} : G \times \mathcal{M}_l \to \mathcal{T} \quad \overline{\tau}(y, m) = \left\lfloor \frac{\left(\sum_{i=0}^{d-1} y_i \hat{m}_i \right) \bmod w}{\overline{t}} \right\rfloor \tag{20}$$

This satisfies[Lemma 4, [5]] $\tau(zm) = \overline{\tau}(z, m)$ or $\tau(zm) = \overline{\tau}(z, m) \mod t$.

Auxiliary projection function:

$$\overline{\sigma} = \begin{cases} \text{fail} & \text{if } x \equiv -1 \bmod \overline{r}, \\ \lfloor x/\overline{r} \rfloor & \text{otherwise.} \end{cases} \tag{21}$$

This almost equalises the values of tag functions to get equation 11. Whenever $\overline{s}(= \overline{\sigma} \circ \overline{\tau})$ does not fail, it is equal to $s(= \sigma \circ \tau)$[Proposition 2, [5]]. We can define distinguished points as in equation 14, by choosing δ suitably so that so that distinguished point probability is sufficient.

The cost computing these functions is less than that of a full product computation $(size(p)^2$ using Schoolbook multiplication or $size(p)^{1.5}$ for Karatsuba method where $size(p)$ the bit-size of $p)$. As r is small(generally $4, 8$ for up to 3072-bit primes), in computing $\overline{s}$ the main time we spend is in computing $\overline{\tau}$ which is multiplication between w bit and u bit integers, where $size(w) << size(p)$ and $size(u) << size(p)$. Also if we choose w, u as powers of two appropriately, then we can easily do the operations that tag-tracing requires.

We can further accelerate the process by combining the method that we have mentioned in Section 4.4 along with tag tracing.

Let us now focus on the cost of applying tag-tracing to the Gaudry-Schost algorithm for subgroups modulo p. The essential difference between our problem and the considerations on the analyses done previously are in the D-dimensionality of the problem, different kinds of walks and abortion on reaching distinguished point.

Let us perceive the elements of the group G_q to be integers upper bounded by p. Further, let $Mul(k)$ and $||n||$ denote the cost of multiplication of a k-bit quantity and the bit-length of an integer n respectively.

6.1 Time complexity

Theorem 9. *The total time complexity when online phase of Gaudry-Schost algorithm is modified with tag-tracing for subgroups modulo p is*

$$\left\{\frac{D-1}{\Delta}Mul(||p||+\left(\frac{1}{l}+\frac{1}{\bar{r}}\right)Mul(||p||))+d\ Mul(||w||)\right\}(1.0171^{D}\sqrt{\pi N}+\Delta) \tag{22}$$

where Δ is the number of extra iterations to reach distinguished points.

Proof. We approximate the costs of functions used for tag-tracing.
Using definitions of $\overline{\sigma}$(equation 21), $\overline{\tau}$(equation 20) and $\bar{s}(=\overline{\sigma}\circ\overline{\tau})$, cost of computing $\bar{s}$ is: d multiplications modulo w, $(d-1)$ additions modulo w, and integer divisions. The dominant cost can then be approximnated as d Mul($||w||$).
As $\overline{\tau}$ does not fail, $\bar{s}$ fails fails only due to failure of $\overline{\sigma}$, and the probability of this from equation 21 is $Pr(\overline{s}\text{ fails})=\frac{1}{\bar{r}}$.
When $\bar{s}$ fails or when l iterations are completed we do a single multiplication. The cost here is $\left(\frac{1}{l}+\frac{1}{\bar{r}}\right)$Mul($||p||$).
We do $(D-1)$ multiplications when we reach a distinguished point, having cost $\frac{D-1}{\Delta}$Mul($||p||$).
The expression in equation 22 follows from expression 16 on substituting the number of iterations from Theorem 1

■

We can further modify the first term in the above expression. The multiplication that actually happens in this case is a multiplication between a $||w||$ bit integer and a $||u||$ bit integer where $u << w$. A specialisation in this case is, if we take w as a power of two, then we do not need to compute the significant bits of the product. When we compare this to multiplications modulo p, the gain is evident.

6.1.1 Asymptotic complexity for fixed group size and large prime modulo

We note that the multiplications in the group take place modulo p both for the original Gaudry-Schost algorithm and the new modified version with tag-tracing. Let us discuss the asymptotic complexity of the algorithms for a fixed group size q, when p is increased.

Lemma 10. *If the parameters of tag-tracing are chosen such that*

1. $l\approx\bar{r}$.
2. $d=O(\bar{r})$
3. $\frac{1.0171^{D}\sqrt{\pi N}}{\Delta}=O(1)$

then the time-complexity of tag-tracing to solve multi-dimensional DLP on subgroups of the group modulo p is $O(d)Mul(||w||)(1.0171^{D}\sqrt{\pi N})+DO(1)Mul(||p||)$.

Proof. Let us refer to Equation 22 for the cost of tag-tracing. We observe that under the condition $\frac{1.0171^{D}\sqrt{\pi N}}{\Delta}=O(1)$, we can simplify the terms as:

1. $\frac{D-1}{\Delta}\text{Mul}(||p||)(1.0171^D\sqrt{\pi N}+\Delta) = (D-1)O(1)\text{Mul}(||p||)$.
2. $(1.0171^D\sqrt{\pi N}+\Delta) \leq 2(1.0171^D\sqrt{\pi N})$.

Then, Equation 22 can be bounded as

$$\left\{d\ \text{Mul}(||w||) + \left(\frac{1}{l}+\frac{1}{\bar{r}}\right)\text{Mul}(||p||) + \frac{D-1}{\Delta}\text{Mul}(||p||)\right\}(1.0171^D\sqrt{\pi N}+\Delta)$$
$$\leq 2\left\{d\ \text{Mul}(||w||) + \left(\frac{1}{l}+\frac{1}{\bar{r}}\right)\text{Mul}(||p||)\right\}(1.0171^D\sqrt{\pi N}) + (D-1)O(1)\text{Mul}(||p||)$$
$$\leq 2\left\{d\ \text{Mul}(||w||) + \frac{O(1)}{\bar{r}}\text{Mul}(||p||)\right\}$$

$$(1.0171^D\sqrt{\pi N}) + (D-1)O(1)\text{Mul}(||p||)\ \text{[By using } l \approx \bar{r}\text{]}. \tag{23}$$

We can further simplify the above by using the definition of d and condition 2. We focus on the second term $\frac{O(1)}{\bar{r}}\text{Mul}(||p||)$. Multiplication can be at most quadratic in it's input length. Also here we perform multiplication between w and u bit integers where $||u|| < ||w||$. Then,

$$||p|| = \frac{||p||}{||w||}||w|| \leq \frac{||p||}{||u||}||w|| \approx d||w||\ \text{[By using } d = \lceil \log_u(p-1)\rceil\text{]}.$$

By the above inequality, $\frac{\text{Mul}(||p||)}{\bar{r}} \leq \frac{\text{Mul}(d||w||)}{\bar{r}} \leq \frac{d^2}{\bar{r}}\text{Mul}(||w||) = d\text{Mul}(||w||)$ [As $d = O(\bar{r})$]

Then, $\frac{O(1)}{\bar{r}}\text{Mul}(||p||) = O(d)\text{Mul}(||w||)$. Using this in inequality 23, the requisite cost is obtained. ■

Let us now derive the costs for various methods of multiplication used in tag-tracing.

Theorem 11. *The parameters for tag-tracing can be chosen abiding certain conditions so that the complexity of full multi-dimensional discrete logarithm computation on subgroups modulo p when the classical method of multiplication is employed, is*

$$O(||p||\log||p||)(1.0171^D\sqrt{\pi N}) + DO(1)Mul(||p||)$$

The improved cost of using the FFT method is

$$O(||p||\log\log||p||)(1.0171^D\sqrt{\pi N}) + DO(1)Mul(||p||)$$

The dominating complexity is $O(||p||\log||p||)(1.0171^D\sqrt{\pi N})$ or $O(||p||\log\log||p||)(1.0171^D\sqrt{\pi N})$ according to the method of multiplication, classical or FFT.

Proof. Let us consider the statement of Lemma 10. It is enough to prove that $O(d)\text{Mul}(||w||) = O(||p||\log||p||)$ (classical) or $O(||p||\log\log||p||)$(FFT).
By our choice of d, w(Section 6), $u = \sqrt{w}$ and $d = \lceil \log_u(p-1)\rceil$, the approximations below will hold.

$$d \approx \frac{||p||}{||u||}\ \text{and}\ \text{Mul}(||w||) = O(\text{Mul}(||u||)) \implies d\text{Mul}(||w||) = \frac{||p||}{||u||}O(\text{Mul}(||u||))$$

We note that the elements of the group(modulo p) are written in u-ary notation(Section 6). This relates $||u||$ to $\log ||p||$. Also, for any k bit integer,

$$\text{Mul}(k) \approx \begin{cases} k^2 & \text{if classical multiplication is adopted.} \\ k \log k & \text{FFT method.} \end{cases}$$

Then by the above approximations:

$$O(d)\text{Mul}(||w||) = \begin{cases} O(||p||.||u||) = O(|p|| \log ||p||) & \text{if classical multiplication is adopted.} \\ O(||p||. \log ||u||) = O(|p|| \log \log ||p||) & \text{FFT method.} \end{cases}$$

. This proves the first part.

To prove the second part, we note that the term $(D-1)O(1)Mul(||p||)$ arises from the consideration of distinguished points. Number of such points will be much smaller than the total number of iterations. This implies, it can be neglected when we consider dominant complexity. ■

In general, certain assumptions about parameters may not always hold (For e.g., l and $\overline{r}$ may not be close enough or iterations to reach distinguished points may vary, when distinguished points are defined in a different way for both algorithms). The theorem below compares the two algorithms, in terms of cost per iterations, when we use classical multiplication and walk over large number of points.

Theorem 12. *Considering a fixed l, $\overline{s}$ which always successfully produces an output and small value of D, a rough approximation of ratio of time per iteration, by the dominant costs required by the original and tag-tracing modified version of the Gaudry-Schost algorithm is $\frac{\frac{||p||}{\Delta_{gs}}}{l\frac{\log(||p||)}{\Delta_{tt}}}$ where Δ_{gs} and Δ_{tt} are the number of iterations to get a distinguished point for the definitions of distinguished point adopted for both the methods.*

Proof. The aim here is to just use the dominant costs of multiplication and ignore the rest.[13] When the number of iterations to reach distinguished points are different for both algorithms, the probability of obtaining a distinguished point is thus different.
The main difference between the two algorithm is that full multiplication may be required only once after l iterations in tag-tracing. This makes the cost $\frac{C_f}{l}.Pr_{gs}$ where Pr_{gs} is the probability to reach a distinguished point for the original Gaudry-Schost algorithm. The cost of computing full multiplication is replaced by $C_{\overline{s}}$, which has to be multiplied by Pr_{tt} where Pr_{tt} denotes the probability to reach a distinguished point for tag-tracing.
We approximate these probabilities by the inverse of the number of iterations. By Theorem 11, the costs C_f and $C_{\overline{s}}$ can be substituted by $||p||^2$ and $||p||log(||p||)$ respectively. The required cost is then

$$\frac{\frac{||p||^2}{l}Pr_{gs}}{||p|| \log(||p||)Pr_{tt}} = \frac{\frac{||p||}{\Delta_{gs}}}{l\frac{\log(||p||)}{\Delta_{tt}}}$$

■

[13] In Appendix A.1, we note that multiplications are indeed the dominant costs. Also for subgroups modulo p and large l, $p << p^l \implies \frac{||p||}{l} << \log p \implies \frac{||p||^2}{l} << ||p|| \log p$.

Remark 4. *It may seem that the previous theorem implies that increasing l will result tag-tracing to be less favourable. But, this is not true as the above theorem only considers dominant costs and the leading cost is $C_{\overline{s}}$ as on increasing l, the other cost $\frac{1}{l}C_f$ decreases. We must note that for very large D, the domimant costs will change as C_D can play a subtantial role.*

6.2 Space complexity

Theorem 13. *The storage requirement is $\binom{r+l}{r} \times O(D||p||)$.*

Proof. Each component of the table consists of three entries [Section 3.1.1]. The second entry will require $||p||$ bits as it is an element in the group modulo p. For each $m \in \mathcal{M}_l$, $\hat{m}_i = \lfloor (u^i m \bmod p)/\overline{w} \rfloor$ for $i = 0, 1, \ldots, (d-1)$.
By our choice of parameters $w\bar{w} = p$. So storage for each of these entries is $d||w||$ bits.
The cost for storing the second and the fourth component $\hat{m}_i$'s is then $(||p|| + d||w||)$.
The first component of the table consists of at most l indices, each of size at most r, which is a fixed integer. We thus ignore the space required here. The third components are D exponents, and the size cannot exceed the bound $O(||p||)$. The total storage is

$$\text{Total number of entries} \times \text{Space required by each entry} = \binom{r+l}{r} \times (||p|| + d||w|| + D||p||)$$

The results follows by approximating $(||p|| + d||w|| + D||p||) = O(D||p||)$, since, $d \approx \frac{||p||}{||u||}$ and $||w|| = O(||u||)$ ■

Corollary 14. *The storage in Theorem 13 can be approximated as $l^r O(D||p||)$.*

Proof.

$$\binom{r+l}{r} = \frac{\prod_{i=0}^{r-1}(r+l-i)}{r!} = \prod_{i=1}^{r}(1+\frac{l}{i}) \tag{24}$$

■

Corollary 15. *In terms of p, r, the storage is $O(D||p||^{r+1})$*

Proof. This follows from the remark just above as $l = O(||p||)$. ■

Remark 5. *Following the same line of proof, we can show that for the general case, the storage required by tag-tracing for the multi-dimensional discrete logarithm computation using Gaudry-Schost algorithm, is a polynomial in l, D and the bit-size of the group G.*

7 Results of implementation in subgroups of $\mathbb{Z}_p^*$

Set-up: We present the results of original Gaudry-Schost algorithm and tag-tracing modified version in 2−dimensional discrete logarithm problem in prime order subgroups G_q (of order q) of $\mathbb{Z}_p^*$, with bitsize q from 256 to 2076 bits, and p about 1024 bits. We used classical multiplication for both algorithsms[14].In

[14]We did not adopt the Montgomery multiplication as the purpose was to observe how much improvement sole tag tracing can lead to for the Gaudry-Schost algorithm.

Appendix A, we jot down the exact choice of primes, the definition of distinguished points, and the average time it took to walk past each point.
We used $l = 20, r = 4, w = 2^{32}$ along with $u = \sqrt{w}, d = \lceil \log_u p - 1 \rceil, \bar{t} = 2^{\lceil \log_u p - 1 \rceil}, t = \frac{w}{\bar{t}}, \bar{r} = \frac{t}{r}, \bar{w} = \lceil \frac{p}{w} \rceil$ [15]. For the Gaudry-Schost algorithm, we use the definition that z_i is distinguished if $2^{\delta} | z_i$, after choosing δ_1 optimally. For tag tracing, we chose a positive integer δ_2 and checked whether Equation 14 is satisfied or not for the corresponding multipliers. The corresponding theoretical estimates, which would form a significant role in determining the number of iterations to reach a distinguished point 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. It is difficult to quantify all the associated variables to such values so that these two iterations are numerically equal. For both types of experiments that we did, we chose δ_1, δ_2 such that the number of iterations required for getting a distinguished point was less in the case of the original Gaudry-Schost algorithm than when tag-tracing was applied [16]. As we wanted to compare the run-times to collect a fixed number of distinguished points[17], we computed the ratio of average time per iteration of the walk function.
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 we took the pre-calculated table size as a multiple of two for the Gaudry-Schost algorithm so that the modulo operation becomes easier. This is another advantage that the implementation of Gaudry-Schost algorithm enjoyed due to our choice. We calculated the child point for Algorithm 5 with the help of $\overline{s}$ that we defined previously.
Two kinds of experiments: We did pseudo-random walk to collect fixed number of distinguished points(Section 7.1), full discrete logarithm computation(Section 7.2). Any such comparison was not done using tag-tracing previously.
Codes: They can be accessed from: GitHub link.
Computational resource: We used Intel Xeon E7-8890 @ 2.50 GHz, Magma version V2.22-3.
Ignoring offline time We focus on online time requirement as pre-computation time is theoretically and practically negligible considering total time necessity.
Correctness: Both methods provide valid multi-dimensional discrete logarithms. For e.g., for the full 2-dimensional discrete logarithm computation on input g_1, g_2, h [18], we tested that the output (a_1, a_2) of the algorithms were valid: a_1 and a_2 were within the interval $[0, N_1)$ and $[0, N_2)$ respectively, h satisfied $h = {g_1}^{a_1} {g_2}^{a_2}$.

7.1 Observations for pseudo-random walk over a large number of points

Scheme: We report the observations, when we walk over a large number of points ranging from 2 lakhs to more than 4 million, to collect a designated number of distinguished points.
Various parameters: In Table 4, we have noted the exact values of the $256, 512, 1024, 2076$[19] bit primes q with $p = 2q + 1$. For each group, we randomly fixed five targets and generators. We used (δ_1, δ_2) as $(12, 6)$ (for q with bit-size $256, 2076$) and $(10, 5)$ (for q with bit-size $512, 1024$).
Average time to travel each point: In Table 1, we have noted the time along with the total number

[15] As in Section 6.
[16] It has nothing to do with the tag-tracing algorithm and we can change it as we desire.
[17] This should be independent of the fact how we choose distinguished points, as otherwise results would also be biased towards the process which can gather these points easily.
[18] Conventions are from definition 1.
[19] The value 2076 seems to be odd at first glance. We chose it intending to assure the astute part of ourselves that the choice of powers of two for the size of the group has no effect on experimental results.

of points of each group through which we walk for both types of walk and both algorithms. Columns 5 and 8 are for average time to travel each point. We approximated this ratio by choosing to walk over a large number of points of the pseudo-random walk. It is obvious that in the case of both algorithms, the major time that we had to spend was to do the multiplications whether complete or partial. To satisfy ourselves with the fact that the values in columns 5, and 8 of Table 1 are quite accurate and can be used for contrasting the two algorithms, 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. We measured the experimentally observed ratio in time with the theoretical estimate obtained from the cost of multiplication. From the numerical values that we have noted in Table 6 for the Gaudry-Schost algorithm and Table 7 when we apply the tag-tracing algorithm, we see that the experimental observations and theoretical estimates are close enough. We can also observe from Table 1 that for each algorithm separately, the time per point traversal is almost the same for tame and wild for every group.

Multiplication leads to dominant cost: 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 are valid for use when comparing the relative run-times. We also note that for both algorithms, the major time exhausted was on performing multiplications. This is in tune 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).

Table 1: Comparison of the average time requirement to walk past each point using Algorithm 3 and Algorithm 5.

(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

7.2 Complete multi-dimensional discrete logarithm computation

Parameters: We performed complete multi-dimensional discrete logarithm computation in the group G_q using both algorithms for 1024 sized p and 20 bit q. The exact values are:

p:=89884656743115795386465259539451236680898848947115328636715040578866337902750481566354238661203768010560056939935696678829394884407208311246423715319737062188883946712432742638151109800623047059726541476042502884419075341171231440736956555270413618581675255342293149119973622969239858152417678164812132075497;

q:=1094833

Scheme: We did these experiments for 10 random targets and generators. The choice of more iterations for distinguished points reflected experimentally, with more variation due to pseudo-randomness instead of perfectly random walk. The first four columns of Table 2 contain the time and iterations that we required for all the random targets using both algorithms. As we had intentionally set the number of iterations to reach distinguished points more in the tag-tracing algorithm, we can appropriately compare the time free of this bias only if we consider the time to walk over a single point for both algorithms. The right-most column of Table 2 computes the ratio: $\frac{\text{Time per iteration using Gaudry-Schost algorithm (Algorith}}{\text{Time per iteration with the version modified with tag-tracing (A}}$

Table 2: Complete 2−dimensional discrete logarithm computation using the original Gaudry-Schost algorithm (Algorithm 3) and the modification with tag-tracing (Algorithm 5) for 1024 bit sized p and random targets.

Gaudry-Schost algorithm		Modification with tag-tracing		Ratio of time per iteration
Iterations(i_{gs})	Time(t_{gs})	Iterations(i_{tt})	Time(t_{tt})	$((\frac{t_{gs}}{i_{gs}})/(\frac{t_{tt}}{i_{tt}}))$
917721	2185556.670	2190849	601790.870	8.6726

7.3 Observations and comparision with theoretical estimates

Observations tallying and indicating improvement: The last columns of Table 1 and 2 indicate tag-tracing leads to speed-up. The acceleration in time for tag-tracing increases with the increase in p, q.

The rightmost column of Table 2 reveals to 8 to 9 times improvement which are almost equal to the values in the rightmost column for 1024 bit row in Table 1. This proximity indicates the experimental accuracy of our method.

Experimental and theoretical estimates close enough: In the last column of Table 3, we have computed the numerical estimate of improvement of tag-tracing per iteration that we proposed in Theorem 12. We see that this numerical estimate match closely, for all the experimental cases, as we have noted in last columns of Table 1 and 2. As in Section 4.3, actual values that we observe experimentally are even greater due to the cost of communication and, the cost of updating exponents and elements in the unmodified version of the Gaudry-Schost algorithm.

Table 3: Calculation of the theoretical estimate of acceleration in time from Theorem 12 when tag-tracing is used.

p	Δ_{gs}	Δ_{tt}	$\frac{\frac{\|p\|}{\Delta_{gs}}}{l\frac{\log(\|p\|)}{\Delta_{tt}}}$
1157920892373161954235709850086879078532699846656405640394575840079132163348 07	4096	6484	2.54
1340780792994259709957402499820584612747936582059239337772356144372176403007354697680187429816690342769003185818648605085375388281194656994643364901261183 9	1024	1876	5.22
1797693134862315907729305190789024733617976978942306572734300811577326758055009631327084773224075360211201138798713933576587897688144166224928474306394741243777678934248654852763022196012460941194530829520850057688381506823424628814739131105408272371633505106845862982399472459384797163048353563296242279988 59	1024	1620	8.11
1369749310834200254530679580036610468083484978076177222971836857920731605698838656277048288702390299730566841244193760525556900533822623598002454736939366317506679980480461792443638318605298389040200105821413170745906015163904067594097896876823197408358258656197650242822820496791368753021748617561787960747792470840940163665846235806711142535836303417724579506990306715192118107795544099539104962334782240954782566067632093156654840485002188891538596633452216970605559050440471818681649052611887604940188217793767686309661392074092468195634847717859952214989114926593779041718085219817208066928620539864382197467636257639473 9	4096	5524	12.71
8988465674311579538646525953945123668089884894711532863671504057886633790275048156635423866120376801056005693993569667882939488440720831124642371531973706218888394671243274263815110980062304705972654147604250288441907534117123144073695655527041361858167525534229314911997362296923985815241767816481213207549 7	4096	5716	7.15

7.4 Summary of the observations:

Our experiments confirm that tag-tracing indeed accelerates the computation of multi-dimensional logarithms when it is intertwined with the Gaudry-Schost algorithm. The theoretical estimatesof speed-up and the experimental observations aligns, and the speed-up increases as the size of the underlying prime grows. Besides cost of multiplication, other additional costs also lessen in tag-tracing.

8 Future works

We see that tag tracing leads to gain in time when applied to Gaudry-Schost algorithm. This gain can be further optimized by utilising the multi-dimensionality D of the problem.
The function $\overline{s}$ can be designed with the help of D and abiding the condition mentioned in Theorem 8. Distinguished points can also be defined using D. Both of these will increase the advantage of using tag-tracing as D increases. The tame and wild sets can also be constructed keeping the dimension and the tag-tracing criterias in mind.
A combination of the above aspects, and in general the design of other functons and parameters will lead to continual advantage of tag-tracing.

9 Conclusion

We focussed on the Gaudry-Schost algorithm, which is the state-of-the-art algorithm to compute multi-dimensional discrete logarithms. The two ways to improve this algorithm is either to find collisions faster in the pseudo-random walk that it performs, or to lessen the cost for each step of this walk. When using this algorithm with two special sets, called tame and wild, the lower bound [10] on the number of iterations to get a collision is $\sqrt{\pi N}$, a close approximation of which has already been reached [33]. The only way to get a pronounced development was to improve the cost per step. We applied the concept of tag-tracing [5], which was previously designed in the context of Pollard Rho's algorithm. The difference between the previous scenario there and our situation is the greater dimension of the problem, abortion on arriving at *distinguished* points and the fact that Pollard Rho algorithm was for walks using a single type of set, instead of two in Gaudry-Schost algorithm.
We proposed an algorithm that incorporates tag-tracing into the Gaudry-Schost algorithm. We derived the cost of this algorithm, which showed that tag-tracing always lessens the time requirement of Gaudry-Schost algorithm at each iteration and this factor $\frac{\frac{D-1}{\Delta}+1}{\frac{D-1}{\Delta}+\frac{1}{l}+\frac{C_{\overline{s}D}}{C_f}}$ [20], can be increased on increasing the dimension D and designing other quantities appropriately. An important context of application was in subgroups modulo primes, which have applications in electronic voting and cash schemes. We have deduced an approximate factor of $\frac{\frac{||p||}{\Delta_{gs}}}{l\frac{\log(||p||)}{\Delta_{tt}}}$ [21], by which the cost reduces. The reduction in each iteration is obtained at some negligible one-time cost. The storage complexity is also polynomial in the dimension and the bit-size of the group.
Our experiments in subgroups of $\mathbb{Z}_p^*$, confirmed that the theoretical estimations are accurrate as they matched with the practically observed values. It also showed that tag-tracing reduces the dominant multiplication cost along with other additional costs and the gain would be more as we increase the size

[20] We have mentioned the notations in the Section 1.4
[21] The notations are mentioned in Section 1.4

Table 4: Group orders of various sizes along with corresponding values of $d, \overline{r}$.

Bits in q	q	d	$\overline{r}$
256	57896044618658097711785492504343953926634992332820282019728792003956608167403	16	1024
512	67039039649712985497870124991029230637396829102961966888617807218608820150367734884009371490834517138450159290932430254268769414059732849732168245063059	32	512
1024	89884656743115795386465259539451236680898848947115328636715040578866337902750481566354238661203768010560056939935696678829394884407208311246423715319737062188883946712432742638151109800623047059726541476042502884419075341171231440736956555270413618581675255342293149119973622969239858152417678164812113999429	64	256
2076	$(1846389521368) + (11^{600})$	130	64

Table 5: Comparison of iteration determining factors $\Delta_{gs} = 2^{\delta_1}$ (for Gaudry-Schost algorithm) and $\Delta_{tt} = \overline{r} + \frac{r}{r-1}(r^{\delta_2} - 1)$ (for our modified algorithm).

Bits in q	Δ_{gs}	Δ_{tt}
256	4096	6484
512	1024	1876
1024	1024	1620
2076	4096	5524

of the group. We obtained about 12 times speed-up with 2076 bit-sized groups.
We point out some future research directions based on the theories that we derived, which should lead to further improvement. Multi-dimensional discrete logarithm computation is important both from computational number theoretic and cryptographic perspectives. Employment of our modification should lead to further lowering of the security of these schemes.

A Appendix

In this appendix, we list the numerical values of parameters we used in our experiments. In Table 4, we list the group size which is a prime and $d, \overline{r}$ that we needed for Algorithm 5. In Table 5, we mention the theoretical estimates of the number of extra iterations to obtain a distinguished point. We argue in Section A.1 that the time required to walk past each point obtained by taking the average time to walk through a large number of nodes of the pseudo-random walk is an appropriate quantity to compare the relative runtimes.

Table 6: Comparison (both theoretical and practical) of the average time required in Gaudry-Schost algorithm for various bit-sizes.

$b(q_1), b(q_2)$	Experimentally observed		Theoretical estimate $\frac{\|\|p_2\|\|^2}{\|\|p_1\|\|^2}$
	Tame	Wild	
$512, 256$	$\frac{0.560}{0.140} = 4$	$\frac{0.548}{0.142} = 3.859$	3.984
$1024, 512$	$\frac{2.215}{0.560} = 3.955$	$\frac{2.162}{0.548} = 3.945$	3.992
$2076, 1024$	$\frac{8.413}{2.215} = 3.798$	$\frac{8.436}{2.162} = 3.901$	4.106

A.1 Average time noted from experiments and their comparison with theoretical estimates

We derived in Section 7.3, that the costs of the original and modified Gaudry-Schost algorithm are $O(||p||^2)(1.0171^D\sqrt{\pi N})$ and $O(||p||\log||p||)(1.0171^D\sqrt{\pi N})$ respectively under some considerations. In this section, we argue that the dominant cost of multiplication indeed plays a vital role in the total time requirement.

We consider the bit-sizes of the group size of $G_q(\subset \mathbb{Z}_p^*)$ and the prime p. We mention the exact value of p and q in Table 3 and Table 4 respectively. Here, the size $||q||$ (hence $||p||$ where $p = 2q + 1$) doubles at each step.

Let us focus on cost analysis of Gaudry-Schost algorithm. As the asymptotic cost for a prime p is $O(||p||^2)(1.0171^D\sqrt{\pi N})$, it is sufficient to compute the ratio $\frac{||p_2||^2}{||p_1||^2}$ to compare the time-requirement when we apply algorithms in subgroups modulo p_2 and p_1 respectively. If p_2 is about $2p_1$, this estimate is approximately 4. We see that the actual value of $\frac{||p_2||^2}{||p_1||^2}$ for our specific choice of primes p_1, p_2 in each case, match closely with observations of average time both for tame and wild walk in columns 2 and 3 of Table 6. We have computed the average time from the ratios that we noted in column 5 of Table 1. This harmonization makes sure that the ratios in column 5 of Table 1 are good enough to use for comparison purposes.

We now consider the tag-tracing counterpart. The asymptotic cost of $O(||p||\log||p||)(1.0171^D\sqrt{\pi N})$, implies that the estimate of $\frac{||p_2||Log(||p_2||)}{||p_1||Log(||p_1||)}$ should play a pivotal role when we compare timings for primes modulo p_2 and p_1 respectively. Columns 2 and 3 of Table 7 are the observations for tame and wild walks while the last column is the actual values of $\frac{||p_2||Log(||p_2||)}{||p_1||Log(||p_1||)}$. Clearly, these values are quite close.

Our tabulations validate the following:

1. Our experimental observations of the time required match with the theoretical estimates for both the algorithms. This implies that these timings are appropriate to use when comparing the two algorithms.

2. It also confirms our intuitive supposition, that while comparing the complexities of these two algorithms, it is sufficient to look at the corresponding costs of multiplication.

Table 7: Comparison (both theoretical and practical) of the average time for various sizes of the group in case of the modified Gaudry-Schost algorithm

$b(q_1), b(q_2)$	Experimentally observed		Theoretical estimate $\frac{\|\|p_2\|\|Log(\|\|p_2\|\|)}{\|\|p_1\|\|Log(\|\|p_1\|\|)}$
	Tame	Wild	
$512, 256$	$\frac{0.101}{0.041} = 2.463$	$\frac{0.105}{0.041} = 2.560$	2.245
$1024, 512$	$\frac{0.232}{0.101} = 2.297$	$\frac{0.245}{0.105} = 2.333$	2.220
$2076, 1024$	$\frac{0.678}{0.232} = 2.922$	$\frac{0.682}{0.245} = 2.783$	2.233

References

[1] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. Csi-fish: efficient isogeny based signatures through class group computations. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 227–247. Springer, 2019.

[2] Stefan A Brands. An efficient off-line electronic cash system based on the representation problem. 1993.

[3] Wouter Castryck and Thomas Decru. An efficient key recovery attack on sidh. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 423–447. Springer, 2023.

[4] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. Csidh: an efficient post-quantum commutative group action. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III 24*, pages 395–427. Springer, 2018.

[5] Jung Hee Cheon, Jin Hong, and Minkyu Kim. Accelerating Pollard's Rho algorithm on finite fields. *Journal of cryptology*, 25(2):195–242, 2012.

[6] Craig Costello and Patrick Longa. Four: four-dimensional decompositions on a-curve over the mersenne prime. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 214–235. Springer, 2015.

[7] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European transactions on Telecommunications*, 8(5):481–490, 1997.

[8] Luca De Feo and Steven D Galbraith. Seasign: compact isogeny signatures from class group actions. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38*, pages 759–789. Springer, 2019.

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

[10] Steven D Galbraith and Mark Holmes. A non-uniform birthday problem with applications to discrete logarithms. *Discrete Applied Mathematics*, 160(10-11):1547–1560, 2012.

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

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

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

[14] Robert Gallant, Robert Lambert, and Scott Vanstone. Improving the parallelized Pollard lambda search on anomalous binary curves. *Mathematics of Computation*, 69(232):1699–1705, 2000.

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

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

[17] Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[18] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29–December 2, 2011. Proceedings 4*, pages 19–34. Springer, 2011.

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

[20] Taechan Kim. Security analysis of group action inverse problem with auxiliary inputs with application to csidh parameters. In *International Conference on Information Security and Cryptology*, pages 165–174. Springer, 2019.

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

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

[23] Madhurima Mukhopadhyay. *Aspects of Index Calculus Algorithms for Discrete Logarithm and Class Group Computations*. PhD thesis, Indian Statistical Institute, Kolkata, 2021.

[24] Madhurima Mukhopadhyay and Palash Sarkar. Faster initial splitting for small characteristic composite extension degree fields. *Finite Fields and Their Applications*, 62:101629, 2020.

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

[26] Madhurima Mukhopadhyay, Palash Sarkar, Shashank Singh, and Emmanuel Thomé. New discrete logarithm computation for the medium prime case using the function field sieve. *Cryptology ePrint Archive*, 2020.

[27] C Paul and J Wiener Michael. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):01–28, 1999.

[28] John M Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978.

[29] Jean-Jacques Quisquater. How easy is collision search. In *Advances in Cryptology-CRYPTO'89: Proceedings*, volume 435, page 408. Springer, 1995.

[30] Damien Robert. Breaking sidh in polynomial time. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 472–503. Springer, 2023.

[31] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. of Symp. Math. Soc., 1971*, volume 20, pages 41–440, 1971.

[32] Annegret Weng. A low-memory algorithm for point counting on Picard curves. *Designs, Codes and Cryptography*, 38(3):383–393, 2006.

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

[34] 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(5):971–986, 2019.