

Practically optimizing multi-dimensional discrete logarithm calculations: Implementations in subgroups of \mathbb{Z}_p^* relevant to electronic voting and cash schemes

Madhurima Mukhopadhyay

Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur
mukhopadhyaymadhurima@gmail.com

Abstract

Discrete logarithm problem(DLP) is the pillar of many cryptographical schemes. We propose an improvement to the Gaudry-Schoof algorithm, for multi-dimensional DLP. We have derived the cost estimates in general and specialized cases, which prove efficiency of our new method. We report the implementation of our algorithm, which confirms the theory. Both theory and experiments validate the fact that the advantage of our algorithm increases for large sizes, which helps in practical scenarios. Our method is applicable to speed-up electronic voting, cash schemes, along with other areas associated with multi-dimensional discrete logarithms (point-counting, speeding-up elliptic-curve arithmetic, group-actions, CSIDH etc.).

Keywords: Discrete logarithm problem (DLP), Multi-Dimensional discrete logarithm problem, Gaudry-Schoof algorithm, Electronic voting and cash schemes, Post-quantum cryptography, CSIDH
Mathematics Subject Classification (2020): 11T71 11Y16

1 Introduction

Adversaries could attack cryptographic methods used in the earlier half of the twentieth century because the exchange of keys occurred over an insecure channel. Diffie and Hellman [25] eradicated this defect. Their development resulted in asymmetric or public-key cryptography, which relies on the hardness of the discrete log problem.

The discrete logarithm problem (DLP) [36, 40, 37] 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 $h = g^a$. One can easily think of 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 $g_1, g_2, \dots, g_D, h \in G$ and $N_1, N_2, \dots, N_D \in \mathbb{N}$ for some positive integer D . The D -dimensional discrete logarithm problem is to find integers (if they exist) $a_1, a_2, \dots, a_D \in \mathbb{N}$ such that

$$h = \prod_{i=1}^D g_i^{a_i} \quad (1)$$

where $a_i \in [0, N_i) \forall i = 1, 2, \dots, D$.

We assume that the integers N_i are odd $\forall i, 1 \leq i \leq d$ and $N = N_1 \times N_2 \times \dots \times N_D$.

We note that an equivalent formulation of the problem for the additive abelian group allows negative exponents.¹

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

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 [23, 24, 33, 54]. Gaudry and Schost [24] 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. Thus, any enhancement in the Gaudry-Schost algorithm will provide a general development to point counting.

Gallant, Lambert, and Vanstone (GLV) proposed an approach [22, 21] to speed up elliptic curve arithmetic. Their method in [21] 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 [18, 20]. The same approach of translating the work of writing a multiple of a point to a multi-dimensional DLP [27] can happen in the case of Koblitz curves [30], 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 [3] and election scheme [13].

1.1.1 Recent post-quantum cryptography

Several schemes of present public-key cryptography are based on the difficulty of problems² that are not quantum-safe³, meaning the availability of quantum-computing systems has the capability to break the hardness of these problems. 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*. Evaluation of group actions is important from the perspective of isogeny-based cryptography. Couveignes [12] used group actions to design an isogeny-based cryptosystem in 1997, which was later rediscovered by Rostovtsev and Stolbunov [48]. The latest application occurred in 2018 in the CSIDH-schemes [7]. CSIDH, it's many [2, 4, 6, 10, 35, 43, 8, 15, 14] variants and protocols are an important component of isogeny-based cryptography. They have not been affected by the recent attacks [5, 32, 47], which utilized the presence of some auxiliary information

¹Using same notations as in Definition 1, let G be an additive abelian group. The problem would be to find $a_1, a_2, \dots, a_D \in \mathbb{Z}$ 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.

²Integer-factorisation problem or discrete logarithm problem in finite field or elliptic curves.

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

that broke another isogeny-based cryptography scheme called SIDH [26]. 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, [28]]. 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 [33] adopted the baby-step-giant-step algorithm [51]. The time and space complexity of this algorithm is $O(\sqrt{N})$. One option to reduce the space requirement was to utilize pseudo-random walks that had earlier been used [45, 44, 16] in the case of 1-dimensional DLP. Gaudry and Schost [24] 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 multi-dimensional situations for any dimension.

Gaudry and Schost used a deterministic pseudo-random walk so that the elements of the walk are the 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 [19, 56, 11]. There are some other works [17, 55], where the authors suggest generic improvements by considering special choice of tame and wild sets.

Good design of tame and wild sets will lessen the number of iterations required to get a collision. However, it does not reduce the costs associated with a single iteration. So the question that arises in this context is:

- Can we somehow reduce the work done per iteration?

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. As the size of the underlying prime (modulo which group operations are done) increases, the cost of multiplication becomes all the more pronounceable.

Cheon *et. al.*, [9] 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 a smaller set was less costly than its counterpart in the original larger group. We try to understand how this will result if applied in the case of the Gaudry-Schost algorithm for dimensions greater than one.

We have described the entire procedure to incorporate tag tracing into the Gaudry-Schost algorithm in Section 4. An important scenario of application of multi-dimensional discrete logarithms arises in the set-up of electronic cash scheme [3] and election scheme [13]. We have implemented our algorithm in such groups, in which we observe a significant speed-up. Better choices of certain parameters suggested in the paper should also lead to further improvements. The combined effect should result gain in time in real-world applications.

1.3 Our contributions

1. We have presented an optimization to the Gaudry-Schost algorithm, which is the algorithm used for computing multi-dimensional discrete logarithms.
2. We have obtained the complexity of our method, which shows the gain that our modification leads to both in the general case and more precisely in subgroups modulo primes.
3. We implemented the new algorithm in subgroups of \mathbb{Z}_p^* . Our code is available at:

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

This experimentation ensured that our theoretical predictions were valid. Specifically, we obtained about 12 times speed-up with 2076 bit-sized groups. This implementation also certified that, the gain would be more as we increase the size of the group.

4. We have pointed out how our method improves the electronic voting and cash schemes. We have also noted how the outline of our strategy can be used in other applications both in classical and post quantum cryptography.

1.4 Paper organization

In Section 1, we define the multi-dimensional discrete logarithm problem, some applications, previous works, and our contributions to this work. In Section 2, we describe the Gaudry-Schost algorithm, which is used to compute multi-dimensional discrete logarithms. We next express the motivation behind applying tag tracing to the Gaudry-Schost algorithm in Section 3. In Section 4, 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 5, we calculate the corresponding theoretical difference in the cost that the algorithm can bring. We further note some cryptographically relevant schemes in Section 6. After this part, we end the generalized view and fix our attention towards subgroups of \mathbb{Z}_p^* . In Section 5.2, we have noted a procedure to remove costly modulo p reductions. We describe the associated functions and the values of the parameters required for tag tracing in subgroups of \mathbb{Z}_p^* in Section 7. In this section, we derive the exact estimates for subgroups modulo primes. We present the results of implementation of these strategies in Section 8 along with the comparison with our predicted estimates. We point out some future research directions in Section 9. Lastly, we summarize the conclusions that can be drawn from our study in Section 10.

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

In this section, we explore the adaptation of the Gaudry-Schost algorithm to multi-dimensional discrete logarithm computation. Gaudry and Schost [24] suggested a low-memory, parallelizable algorithm using a 2-dimensional pseudo-random walk. Their method is efficient for solving the 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, \dots, 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 $(a_1 + y_1, a_2 + y_2, \dots, a_D + y_D)$ is a member of the wild set.

The algorithm proposed by Gaudry and Schost [24] consists of offline and online phase. In the initial step of the offline phase, they chose two positive integers n_s and M_{gs} . They used

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

to define a selection function

$$S : G \rightarrow \{0, 1, \dots, n_s - 1\} \quad (3)$$

for partitioning the group into n_s components. The purpose of the selection function will be to output some integer for every element of the group. This output will indicate the partition to which the input group element will belong according to the partitioning convention adopted. The function S should be such that the distribution of the image of elements of G in the domain set $\{0, 1, \dots, n_s - 1\}$ is almost uniform. It may be a hash function with good statistical properties.

Next, they pre-computed n_s elements to perform a walk. For that, they choose an exponent bound

$$M_{gs} \approx \frac{N}{\Gamma \log_2(N)} \quad (4)$$

We shall specify Γ shortly. They store the pre-computed elements in a table P as:

$$P := \{w_j = \prod_{i=1}^D g_i^{e_{i,j}} \mid -M_{gs} < e_{i,j} < M_{gs}, j = 0, 1, \dots, n_s - 1\} \quad (5)$$

The choice of Γ can be made so that all elements of P are distinct⁴. For practical purposes, it is simply chosen as some suitable power of 10.

They defined the pseudo-random walk as below:

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

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

and its exponents

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

We can combine the equations 6 and 7 and express them by the walk equation

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

They constructed the function $walk$ with the help of S so that it is a deterministic pseudo-random function. In the online phase, the pseudo-random walks will be distributed parallelly [53, 44] over some

⁴An 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 .

processors. These walks will 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.

The complexity analysis of the Gaudry-Schost algorithm is done by using birthday paradox [50, 41]. Three heuristic assumptions vital in this process are: The pseudo-random function will exhibit similar behavior as that of a random function; distinguished point probability will be high enough; for any specific type of walk (tame or wild), we have to choose the bounds and the *walk* function so that child of a tame (or wild) point will also be tame (or wild respectively) for a sufficient number of steps.

2.1 Collision detection

The process of finding a match between tame and wild walks will require the storage of the previously generated elements of the walk. There are several strategies [49, 42, 29, 52] to keep this storage size optimal.

However, the most efficient of all these methods is the method of distinguished points [46]. A *distinguished point* is any element of the group G that satisfies certain conditions. 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. Technically, we can say that a point is distinguished if its image by a second carefully chosen hash function is zero. We can design this hash function 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 the entire set of 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 [44] 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 hash table.

As complete information regarding each point is available, we 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. To know whether a point is distinguished or not, we can easily test 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. We need to optimally choose δ for the efficiency of the algorithm.

We should continue each pseudo-random walk until we reach a distinguished point. We store the distinguished points that occur in the tame and wild in two different tables. We have to maintain sorted tables so that finding a common element is easier.

2.1.1 Abandoning a walk

There is a possibility that we can encounter a walk that does not reach a distinguished point after many iterations. To deal with such a situation, Van Oorschot and Wiener [44] 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

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

This means that any abandoned trail is 20 times longer than the average length. Thus proportion of walks that are abandoned is about

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

The quantity on the right side is negligible and so we can abandon such long fruitless walks.

2.2 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 $(x'_1, x'_2, \dots, x'_D) \in \mathcal{T}_{GS}$ and $(y'_1, y'_2, \dots, y'_D) \in \mathcal{W}_{GS}$, 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

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

2.3 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.

2.3.1 Offline phase

We require the knowledge of the pre-computed table P (equation 8) to perform the pseudo-random walk (equations 6). In the offline phase (Algorithm 1), we aim to compute a table consisting of n_s random elements $\prod_{i=1}^D g_i^{e_i}$, $|e_i| < M_{gs} \quad \forall 1 \leq i \leq D$.

2.3.2 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 we give in Algorithm 2, 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

Algorithm 1: Precomputation in Gaudry-Schost algorithm.

Input: $D, g_1, g_2, \dots, g_D, n_s, M_{gs}$.

Output: Table P consisting of n_s products of elements g_i 's for $i = 1, 2, \dots, D$.

- 1 Set P as an empty set
 - 2 **for** $i := 1$ to n_s **do**
 - 3 Choose D integers e_1, e_2, \dots, e_D randomly from $(-M_{gs}, M_{gs})$ //Choosing random exponents.
 - 4 Compute the product as $prod \leftarrow \prod_{i=1}^D g_i^{e_i}$ //Computing product with chosen exponents.
 - 5 Append the product $prod$ to P
 - 6 **Return** P
-

for that side.

We describe the algorithm for the tame and wild processor 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.

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

Input: $g_1, g_2, \dots, g_D, h \in G, N_1, N_2, \dots, N_D \in \mathbb{N}$.

Output: Integers a_1, a_2, \dots, a_D such that $h = \prod_{i=1}^D g_i^{a_i}$.

- 1 $D_T := [], D_W := []$ //Empty tables for storing distinguished points
 - 2 **while** (*no collision between tame and wild points has been found*) **do**
 - 3 Receive a point (z, b_1, \dots, b_D) from the tame or wild processor
 - 4 **if** (*z is received from a tame processor*) **then**
 - 5 **if** $(z, y_1, \dots, y_D) \in D_W$ for some y_1, \dots, y_D **then**
 - 6 Send terminate signal to all client processors
 - 7 **return** $(b_1 - y_1, b_2 - y_2, \dots, b_D - y_D)$
 - 8 **else**
 - 9 Append (z, b_1, \dots, b_D) to D_T
 - 10 **else**
 - 11 **if** $((z, x_1, \dots, x_D) \in D_T$ for some $x_1, \dots, x_D)$ **then**
 - 12 Send terminate signal to all client processors
 - 13 **return** $(x_1 - b_1, x_2 - b_2, \dots, x_D - b_D)$
 - 14 **else**
 - 15 Append (z, b_1, \dots, b_D) to D_W
-

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

Input: $g_1, g_2, \dots, g_D, h \in G, D, N_1, N_2, \dots, N_D \in \mathbb{N}$, 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, \dots, 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 (no terminate signal received from server) do
2   Choose  $(x_1, x_2, \dots, x_D)$  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 ( $z$  is not a distinguished point and  $Length$  is less than  $L$ ) 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      $(x_1, x_2, \dots, x_D) = (x_1, x_2, \dots, x_D) + (e_{1,j}, e_{2,j}, \dots, e_{D,j})$ //Updating the current
     exponents by adding it with exponents from line 7
10     $Length \leftarrow Length + 1$ 
11   Supply  $(z, x_1, \dots, x_D)$  to the server.

```

2.4 Different choices of tame and wild sets

Gaudry and Schost [24] 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 [17] 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 [55] 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 [55] defined the tame and wild sets for dimension 1 as:

$$\mathcal{T} = [0, N], \quad (11)$$

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

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.

$$\mathcal{T} = [0, N]^D, \quad (13)$$

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

where N is same as in Definition 1.

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, [55]) *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 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 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 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 [9] 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 [45]. The Pollard Rho algorithm is a generic algorithm for solving the discrete logarithm problem. The basic idea 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.

A 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. 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.

The concept of tag tracing reduces the work per iteration, by reducing the number of multiplications in the larger group. As a result, the overall cost decreases. Here we desire to investigate whether tag-tracing is effective in the Gaudry-Schost algorithm to compute multi-dimensional discrete logarithms.

4 Application of tag tracing in the online phase of Gaudry-Schost algorithm

The tag tracing method reduces the number of field multiplications with the help of a larger pre-computed 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 field multiplication. A full product computation is required after a certain number of steps. Let us discuss

the adaption of this method for solving multi-dimensional discrete logarithms using Gaudry-Schost [24] algorithm.

4.1 Offline phase

Initially, we compute a table structurally similar to P in equation 5 of the original Gaudry-Schost algorithm. For constructing the table, we decide two positive integers r for size and M_{tt} for the bound on exponents. Our purpose in creating a table of a designated size is to ensure that there is an almost equal representation of the elements from each partition, created by the partition function s . A possible choice of r that we can take is n_s . The table that we initially compute for applying tag tracing is of the form:

$$\mathcal{M} := \{w_j = \prod_{i=1}^D g_i^{e_{i,j}} \mid -M_{tt} < e_{i,j} < M_{tt}; j = 0, 1, \dots, r-1\} \quad (15)$$

We now choose an integer l after careful analysis. In the next step, we aim to compute all those elements that are products of at most l elements from \mathcal{M} . The initial set-up is:

$$\begin{aligned} \mathcal{M}^{(0)} &= \{1_G\} \\ \mathcal{M}^{(1)} &= \mathcal{M} \end{aligned}$$

For $k > 1$, each $\mathcal{M}^{(k)}$ that we shall compute will be a product of exactly k elements from \mathcal{M} .

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

We note that we can compute $\mathcal{M}^{(k+1)}$ from $\mathcal{M}^{(k)}$ by multiplying one element of \mathcal{M} to an element of $\mathcal{M}^{(k)}$. This will parallelize computation of each $\mathcal{M}^{(k)}$.

We denote the union of these sets by \mathcal{M}_l , the multiplier set consisting of at most l products of elements of \mathcal{M} .

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

$$= \{w_{j_1} w_{j_2} \dots w_{j_k}, 1_G \mid 0 \leq j_k \leq (r-1) \forall k = 1, 2, \dots, l\} \quad (17)$$

We give the algorithm for the offline phase in Algorithm 4. Steps 2 to 5 computes the table \mathcal{M} of random products. Steps 7 to 17 calculates $\mathcal{M}^{(k)}$ for each $k = 2, \dots, l$.

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

$$l.M_{tt} \approx \frac{N}{\Gamma \log_2(N)} \quad (18)$$

where we can define Γ suitably so that the table \mathcal{M} contains all distinct elements. This choice makes $l.M_{tt} \approx M_{gs}$ and M_{tt} here is only dependent upon N . We can also define M_{tt} by taking the tame and wild sets into account.

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

Proof. To estimate the the size of \mathcal{M}_l we keep in mind that:

1. We are choosing the first r elements randomly.
2. At the next phase, we choose k elements at a time, for $2 \leq k \leq l$.
3. Each of the k elements is either the identity of the group or some element that we chose randomly at the first step.
4. We have used Lex ordering of indices in Algorithm 4 to avoid the same element appearing more than once.
5. We note that once we chose 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 $\binom{r+l}{r}$. ■

4.1.1 Storing the table \mathcal{M}_l :

Each entry of table \mathcal{M}_l will comprise of three components. The product of elements corresponding to indices j_1, j_2, \dots, j_k of \mathcal{M} (equation 15) is $\prod_{t=1}^k w_{j_t} = \prod_{i=1}^D g_i^{\sum_{t=1}^k e_{i,j_t}}$. We store the exponent information and this product in \mathcal{M}_l as:

$[(j_1, j_2, \dots, j_k), \prod_{t=1}^k w_{j_t}, (\sum_{t=1}^k e_{1,j_t}, \sum_{t=1}^k e_{2,j_t}, \dots, \sum_{t=1}^k e_{D,j_t})]$. We can sort the table \mathcal{M}_l according to multiplier combination information, or a hash table technique so that we can easily maintain this table.

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

Proof. Let us consider the following facts to evaluate the number of group operations:

1. We are constructing the first r elements by multiplying the D elements.
2. 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. ■

4.2 The pseudo-random walk and the functions required

When we apply tag-tracing, the pseudo-random walk will have the same type as in equation 8. We need to define some other functions for the purpose of tag-tracing.

We note that in the online phase, we shall access the necessary information of elements of \mathcal{M}_l . One such information that we use is the partition to which the element belongs. We already have an index

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 $i = 1, 2, \dots, D$ where each entry consists of a set of indices I , the product, and the exponent information.

```

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

```

function $s : G \rightarrow \{0, 1, \dots, n_s - 1\}$ which partitions G . Here our strategy is to define an indexing function, which may be same as s , but which we can compute when we know the divisors of the elements. We can define an auxiliary index function \bar{s} with the help of s as:

$$\bar{s} : G \times \mathcal{M}_l \rightarrow \{0, 1, \dots, n_s - 1\}$$

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

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.

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

where $m_{s(g)}$ denotes the element of \mathcal{M}_l corresponding to the element g of the group.

We can access the product of multipliers on the right-hand side of the above equation from the table \mathcal{M}_l . To perform tag tracing we will require the design of the auxiliary index function \bar{s} in such a way that $\bar{s}(y, m)$ will be easier to compute than multiplying y and m . Once we fully compute an element by multiplying its components, the pseudo-random walk for the next l steps will be simpler.

We have to design indexing functions so that they are surjective and pre-image uniform. One possible method that we can adopt is to define additional functions to take care of these properties. Adopting the methodology in [9], we can search for:

1. Two index functions $\tau, \bar{\tau}$ with a larger image set, which we can use for constructing the functions s, \bar{s} . Our method will be to define them so that we can compute them without the complete knowledge of the element. More specifically, we must be able to compute them without knowing the full product of the divisors of a particular element.
2. Next, we can design two functions $\sigma, \bar{\sigma}$ to project $\tau, \bar{\tau}$ so that they correspond to each other and equation 19 holds by defining $s = \sigma \circ \tau$ and $\bar{s} = \bar{\sigma} \circ \bar{\tau}$ ⁵. In addition, we have to take care of the properties of pre-image uniformity and easy-computability.

More formally, we can construct the functions s, \bar{s} with the help of four other functions and a larger set \mathcal{T} (such that $\mathcal{S} = \{0, 1, \dots, n_s - 1\} \subset \mathcal{T}$) which we call the tag-set. The functions are:

$$\begin{aligned} &\text{Tag function } \tau : G \rightarrow \mathcal{T} \\ &\text{Auxiliary tag function } \bar{\tau} : G \times \mathcal{M}_l \rightarrow \mathcal{T} \\ &\text{Projection function } \sigma : \mathcal{T} \rightarrow \mathcal{S} \\ &\text{Auxiliary projection function } \bar{\sigma} : \mathcal{T} \rightarrow \mathcal{S} \cup \{\text{Fail}\} \\ & \\ &s = \sigma \circ \tau : G \rightarrow \mathcal{S} \\ &\bar{s} = \bar{\sigma} \circ \bar{\tau} : G \times \mathcal{M}_l \rightarrow \mathcal{S} \cup \{\text{fail}\} \end{aligned}$$

⁵We may not require the functions $\sigma, \bar{\sigma}$ in a fortunate situation when we can design $\tau, \bar{\tau}$ to correspond to each other.

There is a certain probability of failure, on which we have to do a full product computation. We can use the index function, which we are already computing in each step, to define distinguished points. This will ensure that deciding whether a point is distinguished or not does not incur any extra cost. The simple way by which we can define distinguished points in this set-up is:

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.*,

$$z_i \text{ is distinguished if } s(z_{i-\delta+1}) = \dots = s(z_{i-1}) = s(z_i) = 0 \quad (21)$$

We can easily check this condition from the information that we will gather from δ points. We shall only check distinguished points to find collisions, the important thing which we need to ensure is that the extra cost to find distinguished points using the above definition should be reasonable. This makes it necessary to compute the average distance between distinguished points which will provide the idea of the number of iterations that we need to do solely for the purpose of arriving at a distinguished point. As the image of the function s , is of size $n_s (= r, \text{ here})$ it will be sufficient if we know the number of steps we need to go through until the first appearance of δ run of zeroes appear when elements are non-negative. The probability of the image of a point under s to be zero is $\frac{1}{r}$. By theorem 1 of Cheon et al. [9], we obtain an idea of the additional iterations. We state the theorem here:

Theorem 4. [Theorem 1, [9]] *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 $\Delta = \frac{r}{r-1}(r^\delta - 1)$*

We define equation 19 in terms of s . In practice, we shall use \bar{s} in tag-tracing. Assuming $z_{j+1} = z_j w_j$ for any index j , and using relation given by equation 19 among \bar{s} and s , from Equation 21, a point z_i is a distinguished point if

$$\bar{s}(z_{i-\delta+1}, w_{i-\delta+1}) = \bar{s}(z_{i-\delta+2}, w_{i-\delta+2}) = \dots = \bar{s}(z_{i-1}, w_{i-1}) = 0 \quad (22)$$

Due to the connection between \bar{s} and s , this does not change the extra iterations required. Apparently, this definition seems questionable at points that occur within δ iterations. But, we note that δ is of logarithmic order in Δ . As in any case, the distinguished point method will require $O(\Delta)$ extra iterations after the point of collision, this ambiguity will not have much consequence when we consider the entire method of collision detection. In case, some applications require considering such points also, we can define distinguished points based on the output of $\bar{\tau}$, which is available for every point. There are other efficient ways that we can use to define distinguished points [Section 4.3.2 and 4.3.3 of [9]].

We note that the efficiency of the method relies on reducing the full-product computations. In order to lessen run-time when calculating s in terms of \bar{s} (equation 19), the condition, $\frac{C_{\bar{s}}}{C_{full}} < 1$ must hold, where $C_{\bar{s}}$ and C_{full} represent the time costs of obtaining \bar{s} and computing a full product, respectively..

4.3 Online phase

We describe the process for the tame and wild sides in Algorithm 5. We select random points from the tame and wild sets to initiate the tame and wild walks respectively. In this case, we can continue with the best possible choice of tame and wild sets for the Gaudry-Schost algorithm. We require D multiplications to compute random points initially. After that, we find l multipliers w (line 13) from the previously generated table \mathcal{M}_l . For finding any such multiplier, we do not require multiplication.

We check whether the product of z and the newly found multiplier w is a distinguished point (line 12). If we find any distinguished point then we send it to the server. We note that in equation 22 we have defined distinguished points in such a way that, we do not require complete multiplication of z and w to know whether $z \cdot w$ is distinguished or not. We can determine it easily from the partial knowledge of z and w .

We know the exponents of g_i 's for every point of the walk. We use this exponent tracing capability to obtain multi-dimensional discrete logarithms.

Choosing l : We note that by selecting a large l , we have the advantage of doing lesser full product computations. On the other hand, increasing l will also lead to a larger table. We should consider this trade-off along with the approximation in 18 when we fix l .

5 Cost comparison for the online phase

We see that the cost of multiplication majorly contributes to the run-time for both Algorithm 3 and Algorithm 5. When we apply the method of tag tracing to the Gaudry-Schost algorithm, we can approximate multiplication count as: Single multiplication after every l steps. When we compare this to the original Gaudry-Schost algorithm, we observe that the reduction in the number of multiplications in the full group should lead to some real gain in time.

Let us denote \bar{N} as the number of iterations for a tame or wild walk and \bar{L} as 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 should be equal to the number of iterations that we will require for distinguished points.

In case of the original Gaudry-Schost algorithm, each set of $(\bar{L} + 1)$ points of the pseudo-random walk is either an initial point of the walk or some element that we obtain by applying the *walk* function to it's previous element. We obtain the initial points by multiplying D group elements. To get the next \bar{L} points, we require a single multiplication in the full group. Let us denote C_D as the cost of the former and C_{full} as the cost of the latter. The total cost, which we denote by T_0 , for each consecutive set of $\bar{L} + 1$ points is then:

$$T_0 = (C_D + \bar{L}.C_{full}) \quad (23)$$

When we apply tag tracing, the costs are either C_D or C_{full} , but they arise in a different way depending on the function \bar{s} or completion of l iterations. Let us denote $Prob[\bar{s} \text{ fails}]$ as the probability of failure of \bar{s} . We can calculate the time for each consecutive set of $(\bar{L} + 1)$ points when we use tag tracing from:

1. We are obtaining the first element of the walk by multiplying D group elements.
2. For the rest of the \bar{L} cases, we require full product multiplication when we successfully get \bar{s} in all l iterations and we need the next element or when we fail to obtain \bar{s} in some intermediate step.

The above points indicate that we can state the total cost which we denote by T_1 as:

$$T_1 = (C_D + \bar{L}(Prob[\bar{s} \text{ fails}] + \frac{1}{l})C_{full} + \bar{L}(1 - Prob[\bar{s} \text{ fails}] - \frac{1}{l})C_{\bar{s}}) \quad (24)$$

We then have rough estimates for Algorithm 3 and Algorithm 5 as $\frac{\bar{N}}{(\bar{L}+1)}T_0$ and $\frac{\bar{N}}{(\bar{L}+1)}T_1$ respectively. The overall improvement will come from the fact that $C_{\bar{s}} < C_{full}$.

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

Input: $g_1, g_2, \dots, g_D, h \in G, d, N_1, N_2, \dots, N_D \in \mathbb{N}$, 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, \dots, 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 ( no terminate signal received from server ) do
2   Choose  $(x_1, x_2, \dots, x_D)$  from the tame set  $\mathcal{T}$  or wild set  $\mathcal{W}$  depending upon the
   walk//Set-up a random tame or wild walk
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 (  $z$  is not a distinguished point and  $Length$  is less than  $L$  ) do
6      $w \leftarrow 1_G$ .
7      $k \leftarrow 0$ .
8      $(e_{1,k}, e_{2,k}, \dots, e_{D,k}) = (0, 0, \dots, 0)$ 
9      $tmp_{\bar{s}} \leftarrow \bar{s}(z, w)$ 
10    if  $tmp_{\bar{s}}$  is not fail then
11       $j_k \leftarrow tmp_{\bar{s}}$ 
12    while (  $(k < l)$  and  $(\bar{s}$  does not fail) and  $(z.w$  is not a distinguished point) ) do
13       $E \leftarrow$  Entry of  $\mathcal{M}_l$  corresponding to  $(j_0, j_1, \dots, j_k)$ //This finds the entry with first
      component  $(j_0, j_1, \dots, j_k)$ 
14       $w \leftarrow E[2]$ //The second component is the pre-computed product  $w_{j_0}w_{j_1} \dots w_{j_k}$ 
15       $k = k + 1$ 
16       $tmp_{\bar{s}} \leftarrow \bar{s}(z, w)$ 
17      if  $tmp_{\bar{s}}$  is not fail then
18         $j_k \leftarrow tmp_{\bar{s}}$ 
19       $z \leftarrow z \cdot w$ //single multiplication in  $G$  that does a full product computation and is
      generally needed after skipping all multiplications for the previous  $l$  steps
20       $Length \leftarrow Length + l$ 
21      Update the exponent information  $(e_{1,k}, e_{2,k}, \dots, e_{D,k})$  corresponding to
       $(j_0, j_1, \dots, j_k)$ //The third component of each entry in  $\mathcal{M}_l$  contains exponent
      information.
22       $(x_1, x_2, \dots, x_D) = (x_1, x_2, \dots, x_D) + (e_{1,k}, e_{2,k}, \dots, e_{D,k})$ //Updating the exponent
      information of  $z$  using exponent information of  $w$ 
23    Send  $(z, x_1, x_2, \dots, x_D)$  to the server

```

5.1 Updating exponents

In the case of the original Gaudry-Schoat algorithm, we need to update the exponents every time. We can ignore this cost in the complexity analysis as it is less costly than iterating the walk function. When we apply tag tracing, we can update the exponents generally after l iterations and in exceptional cases when we get a distinguished point or fail to compute \bar{s} for the input. Thus, we can replace the cost of D additions at every iteration with D additions until a full-product computation is required. This is another benefit of using tag-tracing.

Since tag tracing in no way interferes with the pseudo-randomness of the walk, the number of iterations \bar{N} that we need to get a collision will not change. On the other hand, as each bunch of \bar{L} iterations will be accelerated, the overall time required will be less.

We next direct the discussions towards solving the multi-dimensional discrete logarithm problem in subgroups of \mathbb{Z}_p^* .

5.2 Improvement in groups of prime order

Let G be a prime-ordered subgroup of some group containing $(p - 1)$ elements for a prime p . 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 [38] 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, [34]]. 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 [38]. This combination of tag tracing and Montgomery multiplications has been achieved without any trade-offs. We note that we can replicate the entire procedure into the Gaudry-Schoat method as well to find multi-dimensional discrete logarithms. This would eventually lead to more practical speed-ups.

6 Cryptographically relevant groups

6.1 Electronic cash and election schemes:

An interesting case of application of our proposal is in the electronic cash scheme [3] and election scheme [13]. 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 7.

6.2 Other applications

We have already mentioned some of the areas where we can apply the strategies of multi-dimensional discrete logarithm computation in Section 4. Tag-tracing in general is a generic improvement where the optimization varies according to the group. Our proposal can also be used in other applications[Section 1.1].

For example, Kim mentions in [28] 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.

7 Designing suitable functions and parameters for subgroups of \mathbb{Z}_p^*

In this section, we shall discuss how to suitably design functions s, \bar{s} for prime ordered subgroups of \mathbb{Z}_p^* . We abide by the constructions by Cheon, et. al., [9]. The tag functions $\tau, \bar{\tau}$ should be easily computable from the knowledge of divisors of the prime ordered group element of \mathbb{Z}_p^* . The straightforward way of gathering information when the binary form of an element is known, is to see it's most or least significant bit. Another important factor in this context is that these functions will be used to define distinguished points, so the chosen property must have a substantial effect on the entire process. Keeping these factors in mind, tag functions can be defined by properly projecting the most significant bits. Our initial set-up will need the information about number of partitions(say, r) and the size of tag-set(say, w). As the issue of binary representation, has already arised, the most natural choice is to fix r, w as some suitable powers of two.

We can then define the tag function as the most-significant $\log_2(w)$ bits of the input. The only issue here is that, we should be able to compute the tag function of any product zm when we know z and m separately. For example, for polynomials $z(x), m(x), p(x)$ over some field $K[x]$ we could easily write

$$(z(x)m(x)) \bmod p(x) = \sum_{i=0}^{n-1} z_i((x^i m(x)) \bmod p(x)) \quad (25)$$

where $z(x) \bmod p(x) = \sum_{i=0}^{n-1} z_i x^i$. For the analogous form for the prime-ordered subgroup of unit group of primes, we can consider the u -ary representation for u being some power of 2 and use division by suitable quantities to project it into the larger tag set.

We need⁶ to set-up the associated quantities for the tag function: $u = \sqrt{w}, d = \lceil \log_u(p-1) \rceil$, $\bar{t} = 2^{\lceil \log_2 du \rceil}$, $t = \frac{w}{\bar{t}}$, $\bar{r} = \frac{t}{r}$, $\bar{w} = \lceil \frac{p}{w} \rceil$. This implies $r\bar{r} = t, t\bar{t} = w, w\bar{w} \approx p, d \approx \log_u(p), \bar{t} = du$. We can define the functions as:

$$\tau : G \rightarrow \mathcal{T} = \{0, 1, \dots, (t-1)\} \quad \tau(z) = \lfloor \frac{z \bmod p}{\bar{t}\bar{w}} \rfloor$$

$$\sigma : \mathcal{T} \rightarrow \mathcal{S} = \{0, 1, \dots, (r-1)\} \quad \sigma(z) = \lfloor \frac{z}{\bar{r}} \rfloor$$

We can define the index function as $s = \sigma \circ \tau$. This function is roughly pre-image uniform[Proposition 1, [9]].

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

$$\bar{\tau} : G \times \mathcal{M}_l \rightarrow \mathcal{T} \quad \bar{\tau}(y, m) = \left\lfloor \frac{\left(\sum_{i=0}^{d-1} y_i \hat{m}_i \right) \bmod w}{\bar{t}} \right\rfloor \quad (26)$$

⁶It is noteworthy that we make these choices to conveniently define the functions below. There may be other ways of selection keeping the essential idea in mind.

We can verify that $\tau(zm) = \bar{\tau}(z, m)$ or $\tau(zm) = \bar{\tau}(z, m) \pmod t$ [Lemma 4, [9]]. To almost equalise the values of tag functions to get equation 19 we can define the auxiliary projection function $\bar{\sigma}$ as:

$$\bar{\sigma} = \begin{cases} \text{fail} & \text{if } x \equiv -1 \pmod{\bar{r}}, \\ \lfloor x/\bar{r} \rfloor & \text{otherwise.} \end{cases} \quad (27)$$

Clearly, whenever $\bar{s}(= \bar{\sigma} \circ \bar{\tau})$ does not fail, it is equal to $s(= \sigma \circ \tau)$ [Proposition 2, [9]]. We can take the distinguished point definition as in equation 22 for some suitable choice of δ , where we can choose δ according to our will so that the distinguished point probability is sufficient enough.

We note that the cost of a full product computation, in this case, would be $size(p)^2$ if we use Schoolbook multiplication (whereas $size(p)^{1.5}$ for Karatsuba method) where we denote by $size(p)$ the bit-size of p . In computing \bar{s} the main time we spend is in computing $\bar{\tau}$ as we can take r to be a small positive integer (generally 4, 8 for up to 3072-bit primes). $\bar{\tau}$ is essentially multiplication between w bit and u bit integers, where $size(w) \ll size(p)$ and $size(u) \ll size(p)$. 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 5.2 along with tag tracing.

7.1 Cost analysis

Let us now focus on the cost of applying tag-tracing to the Gaudry-Shost algorithm for subgroups modulo p . 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.

We compute the auxiliary index function $\bar{s}(= \bar{\sigma} \circ \bar{\tau})$, from the definitions of $\bar{\sigma}$ (equation 27) and $\bar{\tau}$ (equation 26). The cost of this computation is: d multiplications modulo w , $(d-1)$ additions modulo w , and integer divisions. Let us ignore the cheaper costs and focus on the more dominant ones. We can then approximate the cost of computing \bar{s} by d multiplications modulo w which becomes $d \text{ Mul}(\|w\|)$ according to the notations we have adopted. We see that $\bar{\tau}$ produces an output for each input. Thus the function \bar{s} fails to produce an output when $\bar{\sigma}$ fails. By definition of $\bar{\sigma}$ (equation 27), this probability of failure is $\frac{1}{\bar{r}}$. On such a failure, we have to do the costly multiplications modulo p . We will also be required to do these costly multiplications when we complete l iterations or when we reach a distinguished point. There is an essential difference between these two situations. After we reach l iterations, we have to do only one modulo p multiplication (Line 19 of Algorithm 5). But on reaching a distinguished point, we have to submit this point for storage (Line 23 of Algorithm 5) and re-initiate the pseudo-random walk (Lines 2 and 3 of Algorithm 5). In such cases, we need to multiply D elements modulo p . We can sum up the entire thing as: On reaching a point where \bar{s} fails or when l iterations are completed we do a single multiplication, whereas we do $(D-1)$ multiplications when we reach a distinguished point. The former contributes a cost of $\left(\frac{1}{l} + \frac{1}{\bar{r}}\right) \text{ Mul}(\|p\|)$ and the later costs $\frac{D-1}{\Delta} \text{ Mul}(\|p\|)$. We can thus approximate the time complexity when tag-tracing is used for such groups as:

$$\left\{ d \text{ Mul}(\|w\|) + \left(\frac{1}{l} + \frac{1}{\bar{r}}\right) \text{ Mul}(\|p\|) + \frac{D-1}{\Delta} \text{ Mul}(\|p\|) \right\} (1.0171^D \sqrt{\pi N} + \Delta) \quad (28)$$

where Δ is the number of extra iterations to reach distinguished points as in 4 and we consider 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 $u \ll w$. 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.

7.1.1 Asymptotic complexity

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 5. *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 $O(d) \text{Mul}(\|w\|)(1.0171^D \sqrt{\pi N}) + DO(1) \text{Mul}(\|p\|)$.

Proof. Let us refer to Equation 28 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 28 can be bounded as

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

$$(1.0171^D \sqrt{\pi N}) + (D-1)O(1) \text{Mul}(\|p\|) \text{ [By using } l \approx \bar{r}\text{].} \quad (29)$$

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 its input length. Also here we

perform multiplication between w and u bit integers where $\|u\| < \|w\|$. Then,

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

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

Then, $\frac{O(1)}{\bar{r}} \text{Mul}(\|p\|) = O(d) \text{Mul}(\|w\|)$. Using this in the simplification 29, the requisite cost is obtained. \blacksquare

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

Theorem 6. *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*

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

The improved cost of using the FFT method is

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

Proof. Let us consider the statement of Lemma 5. It is enough to prove that $O(d) \text{Mul}(\|w\|) = O(\|p\| \log \|p\|)$.

By our choice of d, w , as in Section 7 and by using the fact that multiplication is at most quadratic,

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

We note that the elements of the group(modulo p) are written in u -ary notation(Section 7). Then by the above approximations: $O(d) \text{Mul}(\|w\|) = O(\|p\| \log \|p\|)$. \blacksquare

The term $(D-1)O(1) \text{Mul}(\|p\|)$ arises from the consideration of distinguished points. In practice, the number of distinguished points will be much smaller than the total number of iterations. Hence, we can present the dominating complexity by the first term $O(\|p\| \log \log \|p\|)(1.0171^D \sqrt{\pi N})$.

Space complexity: We see in Section 4.1.1, that each component of the table consists of three entries. Clearly, the space required by the first and third components in each entry being indices and exponents respectively can be ignored when compared to the second one. The second entry is the element in the group modulo p and so we can represent it by $\|p\|$ bits of space. As we have mentioned in Section 7, we also have to store the element $\hat{m}_i = \lfloor (u^i m \bmod p) / \bar{w} \rfloor$ for each $m \in \mathcal{M}_l$ corresponding to the u -ary notation for $i = 0, 1, \dots, (d-1)$. Also, by our choice of parameters $w\bar{w} = p$. Then the storage for each of these entries is $d\|w\|$ bits. We can now approximate the total storage as:

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

Now by using the fact $d \approx \frac{\|p\|}{\|u\|}$ and $\|w\| = O(\|u\|)$ we get $(\|p\| + d\|w\|) = O(\|p\|)$.

$$\text{The storage complexity is } \binom{r+l}{r} \times O(\|p\|) = l^r O(\|p\|) = O(\|p\|^{r+1}) \text{ [} \because l \approx \|p\| \text{]} \quad (30)$$

In the next section, we shall present the results of experiments where we applied tag-tracing to the Gaudry-Schoat algorithm. These experiments were necessary as evaluating the efficiency of tag-tracing [9] was previously assessed in the case of the Pollard-Rho algorithm. The basic Pollard-Rho algorithm does not re-initiate the walk on the appearance of a distinguished point. This is an essential difference between the Pollard-Rho and Gaudry-Schoat algorithms. The other fact is that, unlike the Gaudry-Schoat algorithm having two types of walks (tame and wild) in the Pollard-Rho algorithm walks are of the same type. This suggests, that to explore the impact of the tag-tracing algorithm we would require a comparison of the Gaudry-Schoat algorithm, with and without tag-tracing on groups of workable size.

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

In this section we present the results of implementation of Algorithm 3 and Algorithm 5 for computing multi-dimensional discrete logarithms in prime order subgroups G_q (of order q) of \mathbb{Z}_p^* . The codes for each of the algorithms can be accessed from the [GitHub link](#). The primes q that we chose for our experiments ranged from 256 to 2076 bits while p was about 1024 bits for a complete discrete logarithm computation. We aimed to compare the original Gaudry-Schoat method with the latest choice of tame and wild sets [55] and the tag tracing method applied to the Gaudry-Schoat algorithm. Our experiments were for dimension $D = 2$. We see that the theoretical estimate of speed-up provided by Theorem 6 matches with our experimental observations. We obtain acceleration in the factor by which time requirement reduces as the size of the group increases.

Methodology adopted and computational resource: We performed two types of experiments. At first, we resorted to the method of walking over a large number of tame and wild points to collect some distinguished points from each. We noted the time to gather the number of distinguished points we desired for each of the algorithms. In the second kind, we performed a full discrete logarithm computation. We ran the computations on a single core of server Intel Xeon E7-8890 @ 2.50 GHz using Magma version V2.22-3.

Crux of the impact of our experimentation: We present below the observations from our experiments. We shall delve into the exact details in the next subsections.

1. When we apply tag tracing to the Gaudry-Schoat algorithm, we observe a significant speed-up, i.e., the time we needed was much less for the Gaudry-Schoat algorithm modified with tag-tracing.
2. The factor by which the time-requirement reduced, increased as we enlarged the size of the prime of group operations. This is quite intuitive in the sense that the cost reduction happens in the tag-tracing algorithm as the number of multiplications in the group is lessened and the cost of multiplication depends on the group size.

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. 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-Schoat algorithm. In Appendix A, we jot down the exact choice of

primes, the definition of distinguished points, and the average value of the time it took to walk past each point.

The time to pre-compute the table of random points for both algorithms, may not be negligible when we consider a large-sized group. But we can safely ignore this time when we compare it with the corresponding total time for finding the complete solution of the multi-dimensional discrete logarithm problem. Thus, we focussed on measuring the time requirement of the online phase of both algorithms. We have also tested the correctness of both algorithms. We find that both the algorithms terminate in expected time for the corresponding size of the group, and provide valid multi-dimensional discrete logarithms. For e.g., for the 2-dimensional discrete logarithm problem on input g_1, g_2, h , where we take the notations from Definition 1, we tested that the output (a_1, a_2) of the algorithms were valid:

1. Both a_1 and a_2 were within the interval $[0, N_1)$ and $[0, N_2)$ respectively.
2. h satisfied the condition that $h = g_1^{a_1} g_2^{a_2}$.

Set-up: We present the two types of experiments and their observations in Sections 8.1 and Section 8.2. In all the experiments, we use $l = 20, r = 4, w = 2^{32}$ along with $u = \sqrt{w}, d = \lceil \log_u p - 1 \rceil, \bar{t} = 2^{\lceil \log_u p - 1 \rceil}, t = \frac{w}{\bar{t}}, \bar{r} = \frac{t}{r}, \bar{w} = \lceil \frac{w}{\bar{w}} \rceil$ as in Section 7. For the Gaudry-Schoat algorithm, we used Equation 9 to define distinguished points, *i.e.*, we choose δ_1 optimally and check whether 2^{δ_1} divides the present walk point or not. For tag tracing, we chose a positive integer δ_2 and checked whether Equation 22 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 $\bar{r} + \frac{r}{r-1}(r^{\delta_2} - 1)$ (where \bar{r} arises due to cases when \bar{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-Schoat algorithm than when tag-tracing was applied. As we wanted to compare the run-times to collect a fixed number of distinguished points⁷, 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-Schoat algorithm so that the modulo operation becomes easier. This is another advantage that the implementation of Gaudry-Schoat algorithm enjoyed due to our choice. We calculated the child point for Algorithm 5 with the help of \bar{s} that we defined previously.

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

In this section, we report the observations, when we walk over a large number of points ranging from 2 lakhs to more than 4 million. We performed our experiments for groups $G_q(\subseteq \mathbb{Z}_p^*)$ with $p = 2q + 1$. We aimed to collect a designated number of distinguished points. In Table 4, we have noted the exact values of the 256, 512, 1024, 2076⁸ bit primes q . We used (δ_1, δ_2) as (12, 6) (for q with bit-size 256, 2076)

⁷This 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.

⁸The 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.

and $(10, 5)$ (for q with bit-size 512, 1024). We see from Table 5 in the Appendix that with such selections, the theoretical estimates of the number of iterations to reach a distinguished point required for tag tracing was more and so to make the results neutral, we compute the average time to walk past each point. For each such group G_q , we set the dimension as two and randomly fix five targets h along with bases g_1, g_2 to collect the same number of distinguished tame and wild points for both the algorithms.

In Table 1, we have noted the time along with the total number of points of each group through which we walk when we perform pseudo-random tame and wild walks of both Algorithms 3 and 5. We see that the number of points we had to traverse (column 7 of Table 1) when we adopted tag-tracing was more, simply due to the choice of δ_1, δ_2 which resulted in lesser iterations to get a distinguished point in case of original Gaudry-Schost algorithm (Table 5). It has nothing to do with the tag-tracing algorithm and we can change it as we desire.

In columns 5 and 8, we noted the average time taken to travel each point of the walk, whether tame or wild in the case of both the algorithms. 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.

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).

We detect that this ratio of average time 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. Also, we observe that the more we increase the size of the prime related to the group operation, the more benefit we obtain by using tag-tracing.

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 of time to traverse single point
		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	
(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

8.2 Complete multi-dimensional discrete logarithm computation

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:=898846567431157953864652595394512366808988489471153286367150405788663379027
5048156635423866120376801056005693993569667882939488440720831124642371531973706
2188883946712432742638151109800623047059726541476042502884419075341171231440736
956555270413618581675255342293149119973622969239858152417678164812132075497;
q:=1094833
```

We did these experiments for 10 random targets and generators. Similarly to what we did before, we fixed suitable values of parameters for defining distinguished points so that the number of iterations needed by tag-tracing counterpart would be more than the original Gaudry-Schost algorithm. This also reflected experimentally, where the variation was a bit more as it should be, due to the pseudo-randomness of the walk instead of being a perfectly random one. We kept the values of the other parameters same as we have noted in the paragraph of this section containing the details of the set-up phase. 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: $\frac{\text{Time per iteration using Gaudry-Schost algorithm (Algorithm 3)}}{\text{Time per iteration with the version modified with tag-tracing (Algorithm 5)}}$. For the 1024 bit prime chosen by us, we observe that the time required by Gaudry-Schost is about 8 to 9 times more in all the cases than the time for tag-tracing modified version.

When the value in the rightmost column of Table 2 is compared with the value in the rightmost column for 1024 bit row in Table 1, we see that these values are almost the same. We note that the former has arisen due to full discrete logarithm computation while the latter is for walking over a large number of points to collect a fixed number of distinguished points. But in both the savings in time occurs as the number of costly multiplications are less than the original Gaudry-Schost algorithm. Intuitively, these values should be close enough when the modulo sizes for which multiplications are done are proximate to each other. This indicates the experimental accuracy of our method. Experiments for full multi-dimensional discrete logarithm computation for higher sizes are time-wise costly with limited computational resources. We note that similar to our observations in Section 8.1, this ratio will increase with the increase in values of p, q .

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

8.3 Comparison of experimental and theoretical estimates

We observe from Theorem 6, that tag-tracing requires an approximate cost of $O(\|p\| \log \|p\|)(1.0171^D \sqrt{\pi N})$ under certain consideration of choosing various parameters [Lemma 5]. The corresponding cost of the original Gaudry-Schost algorithm is $O(\|p\|^2)(1.0171^D \sqrt{\pi N})$. The trade-off lies in more storage requirements in the tag-tracing scenario. However, these two costs are for the complete multi-dimensional discrete logarithm computation in the ideal situation when all approximations holds true. If we want to compare the time for a pseudo-random walk for the cases we are doing, the theoretical estimate of the ratio of time required by the original Gaudry-Schost algorithm and the modified version, *i.e.*, $\frac{\|p\|}{\log(\|p\|)}$ has to be adjusted with the following:

1. The probabilities of reaching distinguished points: For realistic cases, the probabilities in both cases are not equal. We approximate by the factor $\frac{Pr_{gs}}{Pr_{tt}}$ for this purpose where Pr_{gs}, Pr_{tt} denotes the probabilities to reach a distinguished point for the Gaudry-Schost algorithm and the modified version respectively. We have calculated this approximation by considering the inverse of the actual number of iterations.
2. The value of l : Lemma 5 assumes $l \approx \bar{r}$. If this does not hold, we can modify the approximation in Equation 29 by dividing $Mul(\|p\|)$ with l . The modified complexity using the classical method of multiplication from Theorem 6 is $O(\|p\| \log \|p\|)(1.0171^D \sqrt{\pi N}) + \frac{D-1}{l} O(1) Mul(\|p\|)$. The term $\frac{D-1}{l} O(1) Mul(\|p\|)$ arises due to the occurrence of distinguished points. As the number of

distinguished points is negligible when we compare it with the total number of iterations, in practice, we can ignore the second term with the caveat of considering l .

Considering the above modifications, the theoretical ratio of time per iteration required by the original and tag-tracing modified version of the Gaudry-Schost algorithm would be

$$\frac{\|p\|}{\log(\|p\|)} \frac{Pr_{gs}}{Pr_{tt}} \frac{1}{l} \quad (31)$$

We have computed the above ratio for all our experimental cases in the last column of Table 3. Here, $Pr_{gs} = \frac{1}{i_{gs}}$, $Pr_{tt} = \frac{1}{i_{tt}}$ where $i_{gs} = 2^{\delta_1}$, $i_{tt} = \bar{r} + \frac{r}{r-1}(r^{\delta_2} - 1)$. We obtained the actual values of i_{gs}, i_{tt} from Table 5 in the first four cases and from Table 2 in the last case. We see that these iterations differ for the two algorithms which makes it necessary to multiply the theoretical estimate of Theorem 6 by the corresponding probabilities that we can obtain from the inverse of iterations. Also, the value of l that we used in our experiments was 20, which is quite less than the value of \bar{r} that we fixed for each of the cases [Table 4]. This makes it necessary to divide the modified estimate with l , by what we discussed just above. We see that this ratio $\frac{\|p\|}{\log(\|p\|)} \frac{Pr_{gs}}{Pr_{tt}} \frac{1}{l}$ [noted in last column of Table 3] which is a theoretical estimate of the speed-up is quite close to the experimental one, as in the last column of Table 1 (for the first four cases) and Table 2 (for the last case of full discrete logarithm computation).

The 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. We have also mentioned the reason for such behavior in Section 5.1. Our tag-tracing modification, experimentally confirms this.

8.4 Summary of the observations:

As our aim was simply to compare the relative run-time, we have not focussed on fine-tuning the parameters for the algorithms. For both algorithms, we consider the ratio of average time per iteration. We observe that:

1. Tag-tracing accelerates the computation of logarithms. The speed-up obtained experimentally is quite close to the theoretical estimations that we derived in Theorem 6. It is about 12 times faster for the largest group that we have considered.
2. It is also noteworthy that as the size of the underlying prime (modulo which the group operations are done), increases, tag tracing becomes more and more efficient.
3. Full multi-dimensional discrete logarithm computations in larger groups with larger dimensions will also involve a good amount of communication costs, which is more in the Gaudry-Schost algorithm as exponents and elements need to be updated every time.

This suggests that the gain in time in practical scenarios would be quite significant when tag-tracing is applied. We recall once again that this decrease in time requirement happens despite the benefits that Algorithm 3 enjoyed due to our choices. For example, we defined distinguished points so that reaching them and testing them, was relatively easier in the original algorithm than our modified version. We also chose the pre-calculated table size as a power of two due to which calculation of the next element of the walk was easier in the original Gaudry-Schost algorithm. This means in strict terms, if we choose the values of associated quantities so that any of the algorithms do not gain privilege compared to the

Table 3: Calculation of the theoretical estimate of acceleration in time from Theorem 6 when tag-tracing is used.

p	i_{gs}	i_{tt}	$\frac{\ p\ }{\log(\ p\)} \frac{Pr_{gs}}{Pr_{tt}} \frac{1}{l}$
115792089237316195423570985008687907853269984665640564039457 584007913216334807	4096	6484	2.54
134078079299425970995740249982058461274793658205923933777235 614437217640300735469768018742981669034276900318581864860508 53753882811946569946433649012611839	1024	1876	5.22
179769313486231590772930519078902473361797697894230657273430 081157732675805500963132708477322407536021120113879871393357 658789768814416622492847430639474124377767893424865485276302 219601246094119453082952085005768838150682342462881473913110 540827237163350510684586298239947245938479716304835356329624 227998859	1024	1620	8.11
136974931083420025453067958003661046808348497807617722297183 685792073160569883865627704828870239029973056684124419376052 555690053382262359800245473693936631750667998048046179244363 831860529838904020010582141317074590601516390406759409789687 682319740835825865619765024282282049679136875302174861756178 796074779247084094016366584623580671114253583630341772457950 699030671519211810779554409953910496233478224095478256606763 209315665484048500218889153859663345221697060555905044047181 868164905261188760494018821779376768630966139207409246819563 484771785995221498911492659377904171808521981720806692862053 98643821974676362576394739	4096	5524	12.71
898846567431157953864652595394512366808988489471153286367150 405788663379027504815663542386612037680105600569399356966788 293948844072083112464237153197370621888839467124327426381511 098006230470597265414760425028844190753411712314407369565552 704136185816752553422931491199736229692398581524176781648121 32075497	4096	5716	7.15

other, then it should lead to a further decrease in practical run-time when tag-tracing is used. Also, the l value chosen by us is quite small. Optimal size of l will lead to walking through more points without performing complete multiplications.

9 Future works

We see that tag tracing leads to gain in time when applied in the original Gaudry-Schoat algorithm. We list a few other things that could be done for further betterment.

1. We can think of designing tame and wild sets so that they are more suited towards tag tracing. We have seen cases of non-constant [55] and constant overlap [17]. These improvements have used the variables N, α [according to the convention in Section 2.4]. We can also construct tame and wild sets based on the value of l , the variables $u, w, d, \bar{r}, t, \bar{t}, w, \bar{w}$ which are the quantities we used for defining suitable functions for tag-tracing (Section 7). If we can couple efficient structures of tame and wild sets with the benefits of tag tracing that results in lesser work per iteration, it should reduce overall practical cost.
2. We can also think of efficient function choices when we define the functions $\tau, \sigma, \bar{\tau}, \bar{\sigma}$ (as in Section 7) so that it is more suited for multi-dimensional scenario.
3. As we need to check the distinguished point condition at each iteration, another possible venue that we can utilize to refine the tag-tracing part is to define distinguished points more adapted for multi-dimensional setting and easier to check.
4. We can combine the above along with replacing modulo p reductions by divisions by powers of two in case of prime fields (Section 5.2).

We implemented both algorithms in subgroups of \mathbb{Z}_p^* . We can also apply the tag tracing procedure when we perform pseudo-random walks of multi-dimensional situations. The other areas that seem interesting are: multi-dimensional problems over elliptic curve groups⁹. One vital area where we can deploy our algorithm would be to access the security of the post-quantum cryptographic scheme CSIDH¹⁰.

10 Conclusion

We have proposed modifications to the Gaudry-Schoat algorithm, which is the state-of-the-art algorithm to compute multi-dimensional discrete logarithms. Our refinement is based on the principle of tag-tracing, which has not been previously used anywhere for dimensions exceeding one and having different types of pseudo-random walks.

⁹The papers [1, 31] have some latest research about Diffie-Hellman key exchange and discrete logarithms over elliptic curves respectively.

¹⁰Besides application in CSIDH, class-groups play an important role in post-quantum cryptography. Pseudo-random walks have also been used recently [39] to compute class groups.

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

Bits in q	q	d	\bar{r}
256	578960446186580977117854925043439539266349 92332820282019728792003956608167403	16	1024
512	6703903964971298549787012499102923063739682910296196688861780721860882 015036773488400937149083451713845015929093243025426876941405973284973 216824506305919	32	512
1024	8988465674311579538646525953945123668089884894711532863671504057886633 790275048156635423866120376801056005693993569667882939488440720831124 642371531973706218888394671243274263815110980062304705972654147604250 288441907534117123144073695655527041361858167525534229314911997362296 9239858152417678164812113999429	64	256
2076	$(1846389521368) + (11^{600})$	130	64

We have theoretically derived the improvement in time requirement that our proposal will lead to. We have calculated the estimates more precisely in subgroups modulo prime.

Our experiments in subgroups of \mathbb{Z}_p^* , confirm the improvement in time complexity that we predicted. An essential characteristic that we noted theoretically and observed practically is that, the factor by which we gain in time, increases as the size of the underlying prime increases. This property should be useful when the algorithm is applied on large sized groups that are used for practical purposes.

Our improvements can be directly applied to the electronic voting and cash schemes. There are other areas of application in cryptographic schemes with groups created from algebraic curves as well as in recent post-quantum cryptography.

We have also noted some future research directions for overall improvement.

A Appendix

In this appendix, we list the numerical values of various associated quantities we have used in our experiments. In Table 4, we list the group size which is a prime and d, \bar{r} that we needed for Algorithm 5. In Table 5, we mention the theoretical estimates of the number of extra iterations that we needed 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.

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

We derived in Section 8.3, 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.

Table 5: Comparison of iteration determining factors 2^{δ_1} (for Gaudry-Schost algorithm) and $\bar{r} + \frac{r}{r-1}(r^{\delta_2} - 1)$ (for our modified algorithm).

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

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 first focus on the cost analysis in the case of the Gaudry-Schost algorithm. As the asymptotic cost for a prime p is $O(\|p\|^2)(1.0171^D \sqrt{\pi N})$, it is sufficient for us 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 twice that of p_1 , this estimate is about 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 $\frac{\|p_2\| \text{Log}(\|p_2\|)}{\|p_1\| \text{Log}(\|p_1\|)}$ 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 $\frac{\|p_2\| \text{Log}(\|p_2\|)}{\|p_1\| \text{Log}(\|p_1\|)}$. 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 the two algorithms, it is sufficient to look at the corresponding costs of multiplication.

References

- [1] Huda Kadhim M. Aljader and Ruma Kareem K. Ajeena. The optimized diffie-hellman key exchange using the graphical method. *Journal of Discrete Mathematical Sciences and Cryptography*, 26(6):1691–1697, 2023.

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

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\ \text{Log}(\ p_2\)}{\ p_1\ \text{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

- [2] 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.
- [3] Stefan A Brands. An efficient off-line electronic cash system based on the representation problem. 1993.
- [4] Wouter Castryck and Thomas Decru. Csidh on the surface. In *International Conference on Post-Quantum Cryptography*, pages 111–129. Springer, 2020.
- [5] 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.
- [6] Wouter Castryck, Thomas Decru, Marc Houben, and Frederik Vercauteren. Horizontal racewalking using radical isogenies. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 67–96. Springer, 2022.
- [7] 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.
- [8] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith. Stronger and faster side-channel protections for csidh.

- In *Progress in Cryptology–LATINCRYPT 2019: 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2–4, 2019, Proceedings 6*, pages 173–193. Springer, 2019.
- [9] Jung Hee Cheon, Jin Hong, and Minkyu Kim. Accelerating Pollard’s Rho algorithm on finite fields. *Journal of cryptology*, 25(2):195–242, 2012.
- [10] Jesús-Javier Chi-Domínguez and Krijn Reijnders. Fully projective radical isogenies in constant-time. In *Cryptographers’ Track at the RSA Conference*, pages 73–95. Springer, 2022.
- [11] 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.
- [12] Jean-Marc Couveignes. Hard homogeneous spaces. *Cryptology ePrint Archive*, 2006.
- [13] 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.
- [14] 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.
- [15] Xuejun Fan, Song Tian, Bao Li, and Xiu Xu. Csidh on other form of elliptic curves. *Cryptology ePrint Archive*, 2019.
- [16] Steven Galbraith, John Pollard, and Raminder Ruprai. Computing discrete logarithms in an interval. *Mathematics of Computation*, 82(282):1181–1195, 2013.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] 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.

- [22] Robert P Gallant, Robert J Lambert, and Scott A Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In *Annual International Cryptology Conference*, pages 190–200. Springer, 2001.
- [23] Pierrick Gaudry and Robert Harley. Counting points on hyperelliptic curves over finite fields. In *International Algorithmic Number Theory Symposium*, pages 313–332. Springer, 2000.
- [24] 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.
- [25] Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.
- [26] 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.
- [27] Benits Junior and Waldyr Dias. *Applications of Frobenius expansions in elliptic curve cryptography*. PhD thesis, Royal Holloway, University of London, 2008.
- [28] 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.
- [29] Donald E Knuth. *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.
- [30] Neal Koblitz. CM-curves with good cryptographic properties. In *Annual international cryptology conference*, pages 279–287. Springer, 1991.
- [31] Qasim Mohsin Luhaib and Ruma Kareem K. Ajeena. Elliptic curve matrices over group ring to improve elliptic curve–discrete logarithm cryptosystems. *Journal of Discrete Mathematical Sciences and Cryptography*, 26(6):1699–1704, 2023.
- [32] Luciano Maino and Chloe Martindale. An attack on sidh with arbitrary starting curve. *Cryptology ePrint Archive*, 2022.
- [33] 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.
- [34] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2018.
- [35] Michael Meyer, Fabio Campos, and Steffen Reith. On lions and elligators: An efficient constant-time implementation of csidh. In *Post-Quantum Cryptography: 10th International Conference, PQCrypto 2019, Chongqing, China, May 8–10, 2019 Revised Selected Papers 10*, pages 307–325. Springer, 2019.
- [36] Madhurima Mukhopadhyay. *Aspects of Index Calculus Algorithms for Discrete Logarithm and Class Group Computations*. PhD thesis, Indian Statistical Institute, Kolkata, 2021.

- [37] Madhurima Mukhopadhyay and Palash Sarkar. Faster initial splitting for small characteristic composite extension degree fields. *Finite Fields and Their Applications*, 62:101629, 2020.
- [38] 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.
- [39] Madhurima Mukhopadhyay and Palash Sarkar. Pseudo-random walk on ideals: practical speed-up in relation collection for class group computation. In *International Symposium on Cyber Security, Cryptology, and Machine Learning*, pages 18–31. Springer, 2023.
- [40] 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.
- [41] Kazuo Nishimura and Masaaki Sibuya. Probability to meet in the middle. *Journal of Cryptology*, 2(1):13–22, 1990.
- [42] Gabriel Nivasch. Cycle detection using a stack. *Information Processing Letters*, 90(3):135–140, 2004.
- [43] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. (short paper) a faster constant-time algorithm of csidh keeping two points. In *Advances in Information and Computer Security: 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28–30, 2019, Proceedings 14*, pages 23–33. Springer, 2019.
- [44] C Paul and J Wiener Michael. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):01–28, 1999.
- [45] John M Pollard. Monte Carlo methods for index computation (\pmod{p}). *Mathematics of computation*, 32(143):918–924, 1978.
- [46] Jean-Jacques Quisquater. How easy is collision search. In *Advances in Cryptology-CRYPTO’89: Proceedings*, volume 435, page 408. Springer, 1995.
- [47] Damien Robert. Breaking sidh in polynomial time. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 472–503. Springer, 2023.
- [48] Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies. *Cryptology ePrint Archive*, 2006.
- [49] Robert Sedgewick, Thomas G Szymanski, and Andrew C Yao. The complexity of finding cycles in periodic functions. *SIAM Journal on Computing*, 11(2):376–390, 1982.
- [50] Boris I SELIVANOV. On waiting time in the scheme of random allocation of coloured particies. 1995.
- [51] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. of Symp. Math. Soc.*, 1971, volume 20, pages 41–440, 1971.

- [52] Edlyn Teske. Speeding up Pollard’s rho method for computing discrete logarithms. In *International Algorithmic Number Theory Symposium*, pages 541–554. Springer, 1998.
- [53] Paul C Van Oorschot and Michael J Wiener. Parallel collision search with application to hash functions and discrete logarithms. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 210–218, 1994.
- [54] Annegret Weng. A low-memory algorithm for point counting on Picard curves. *Designs, Codes and Cryptography*, 38(3):383–393, 2006.
- [55] Haoxuan Wu and Jincheng Zhuang. Improving the Gaudry–Schost algorithm for multidimensional discrete logarithms. *Designs, Codes and Cryptography*, pages 1–13, 2021.
- [56] 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.