Combining Montgomery Multiplication with Tag Tracing for the Pollard's Rho Algorithm in Prime Order Fields

Madhurima Mukhopadhyay[1] and Palash Sarkar[1]

[1]Applied Statistics Unit, Indian Statistical Institute, 203 B. T. Road, Kolkata, India 700108
Email:{madhurima_r,palash}@isical.ac.in

January 12, 2021

Abstract

In this paper, we show how to apply Montgomery multiplication to the tag tracing variant of the Pollard's rho algorithm applied to prime order fields. This combines the advantages of tag tracing with those of Montgomery multiplication. In particular, compared to the previous version of tag tracing, the use of Montgomery multiplication entirely eliminates costly modular reductions and replaces these with much more efficient divisions by a suitable power of two.
Keywords: Cryptography, Discrete logarithm problem, Pollard's Rho, Tag Tracing, Montgomery multiplication.
MSC (2010): 94A60.

1 Introduction

Let G be the finite cyclic group and g be a generator of G. The discrete logarithm problem (DLP) in G is the following. Given a non-zero element h of G, find i such that $g^i = h$. This i is called the discrete logarithm of h to base g which is written as $i = \log_g h$. Over suitably chosen groups, the DLP is considered to be a computationally hard problem and forms the basis for security of various cryptosystems.

The best known generic algorithm for solving DLP is the Pollard's rho algorithm [Pol78]. The resources required by the algorithm is $\sqrt{\#G}$ time and negligible space. Since its introduction, several variants of the Pollard's rho algorithm have been proposed. In particular, the tag tracing variant [CHK12] showed the possibility of obtaining practical speed-up of the Pollard's rho algorithm for certain groups. Concrete speed-ups were demonstrated for prime order subgroups of multiplicative groups of finite fields. Two kinds of fields were considered in [CHK12], namely, prime order fields and small characteristic, large extension degree fields. We focus on the application of tag tracing to prime order fields.

Let p be a prime, $\mathbb{F}_p$ be the finite field of p elements. The group G where DLP is considered is typically a prime order subgroup of $\mathbb{F}_p^\star$.

Pollard's rho algorithm performs a pseudo-random walk. For solving DLP in $\mathbb{F}_p$, each step of the walk requires performing a multiplication in $\mathbb{F}_p$. The improvement achieved by the tag tracing method is to ensure that a field multiplication is required after every ℓ steps for a suitable choice of the parameter ℓ. In the intermediate steps between two field multiplication steps, a special computation is performed by the tag tracing method. This computation is significantly faster than a field multiplication. So, tag tracing speeds up Pollard's rho algorithm by a factor of about ℓ.

A field multiplication in $\mathbb{F}_p$ consists of two phases. The first phase is an integer multiplication while the second phase is a reduction modulo p operation. For primes p not having a special structure, the reduction operation can require a substantial portion of the overall time for a field multiplication. The technique of Montgomery multiplication [Mon85, BM17] works with Montgomery representation of elements and replaces a field multiplication by a Montgomery multiplication. The advantage of Montgomery

multiplication is that all divisions are by certain powers of two and so can be implemented using right shift operations. The expensive modulo p operation is no longer required.

In this work, we show how the Montgomery multiplication can be combined with the tag tracing method. The goal is to retain the advantages achieved by tag tracing and also simultaneously replace the field multiplications required after every ℓ steps by a Montgomery multiplication. All the time consuming modulo p operations are completely eliminated. Consequently, the Montgomery multiplication version of tag tracing achieves further speed-up compared to the usual tag tracing algorithm. The combination of Montgomery multiplication and tag tracing is achieved without any trade-offs. In particular, the storage space required remains the same in both cases.

2 Background

We provide brief descriptions of Pollard's rho, tag tracing and Montgomery multiplication.

Pollard's Rho: The Pollard's rho algorithm [Pol78] is a well known method for solving DLP in prime order fields. Several variants of this algorithm have been studied. We briefly mention the variant introduced in [SJ84].

Let r be a small positive integer. For $i = 1, \ldots, r$, randomly choose integers $\alpha_i, \beta_i \in \{0, \ldots, p-2\}$ such that both α_i and β_i are not zeros. Define $m_i = g^{\alpha_i} h^{\beta_i}$, $i = 0, \ldots, r-1$. A pre-computed table T stores the entries $(i, m_i, (\alpha_i, \beta_i))$ for $i = 0, \ldots, r-1$. Define an indexing function $s : G \to \{0, \ldots, r-1\}$. Using s, a sequence of elements of G is defined as follows. Choose $a_0, b_0 \in \{0, \ldots, \#G - 1\}$ and set $g_0 = g^{a_0} h^{b_0}$. For $j \geq 0$, define $g_{j+1} = g_j m_{s(g_j)}$. The computation of the sequence $g_0, g_1, g_2, \ldots$ is considered to be a pseudo-random walk on G.

Writing $g_j = g^{a_j} h^{b_j}$ for $j \geq 0$, we have $a_{j+1} = a_j + \alpha_{s(g_j)}$ and $b_{j+1} = b_j + \beta_{s(g_j)}$. So, it is easy to obtain a_{j+1} and b_{j+1} from a_j and b_j. Since G is finite, there must be some j and k, with $j < k$ such that $g_j = g_k$, i.e., the pseudo-random walk must lead to a collision. Denoting $\log_g h$ by $\mathfrak{d}$, the condition $g_j = g_k$ leads to the relation $a_j + \mathfrak{d} b_j = a_k + \mathfrak{d} b_k$. Under the condition that $b_j - b_k$ is invertible modulo $\#G$ (which holds with high probability for large p and appropriate group G), we have $\mathfrak{d} = (a_j - a_k)(b_k - b_j)^{-1} \bmod \#G$.

There are several methods for detecting collisions. The distinguished point method [vOW99] is the most practical of these methods and allows parallelisation.

Tag Tracing: In the pseudo-random walk defining Pollard's rho algorithm, the computation of g_{j+1} from g_j is done by multiplying g_j and $m_{s(g_j)}$. So, each step requires a field multiplication. The tag tracing method was introduced in [CHK12]. The essential idea is to increase the size of the pre-computed table so that a field multiplication is required after every ℓ steps for a suitable choice of the parameter ℓ. The computation done in the intermediate steps between two field multiplications is significantly faster than a field multiplication.

The set of multipliers $\{m_i : m_i = g^{\alpha_i} h^{\beta_i}, i = 0, \ldots, r-1\}$ is defined as in the case of the original Pollard's rho algorithm. Choose a parameter ℓ and let $\mathcal{M}_\ell$ be the set of all possible products of at most ℓ elements from $\mathcal{M}$. The elements of $\mathcal{M}_\ell$ can be indexed by vectors of the form $(i_1, \ldots, i_k)$ where $i_1, \ldots, i_k \in \{0, \ldots, r-1\}$ and $0 \leq k \leq \ell$. Given $\mathbf{x} = (i_1, \ldots, i_k)$, the element of $\mathcal{M}$ indexed by $\mathbf{x}$ is $m_{\mathbf{x}} = m_{i_1} \cdots m_{i_k}$. Note that if $\mathbf{x}'$ is obtained by permuting the components of $\mathbf{x}$, then $m_{\mathbf{x}'} = m_{\mathbf{x}}$. So, we will assume that the vector $\mathbf{x}$ satisfies $i_1 \leq i_2 \leq \cdots \leq i_k$. A pre-computed table Tab is created. The rows of Tab are as follows.

$$(\mathbf{x}, m_{\mathbf{x}}, (a.b), (\hat{m}_0, \ldots, \hat{m}_{d-1}))$$

where

- $\mathbf{x} = (i_1, \ldots, i_k)$, with $0 \leq k \leq \ell$, $i_1, \ldots, i_k \in \{0, \ldots, r-1\}$,
- $m_{\mathbf{x}} = m_{i_1} \cdots m_{i_k} \bmod p$,
- (a, b) is such that $m_{\mathbf{x}} = g^a h^b$.

We explain the component $(\hat{m}_0, \ldots, \hat{m}_{d-1})$ later. The table Tab is stored as a hash table (or, some other suitable data structure), so that given an appropriate vector $\mathbf{x}$, it is easy to locate the corresponding row of Tab.

The indexing function $s : G \to \{0, \ldots, r-1\}$ defines the pseudo-random walk. Tag tracing requires an auxiliary indexing function $\overline{s} : G \times \mathcal{M}_\ell \to \{0, \ldots, r-1\} \cup \{\mathsf{fail}\}$, such that

$$\text{if } \overline{s}(y, m) \neq \mathsf{fail}, \text{ then } \overline{s}(y, m) = s(ym).$$

Suppose the element at the j-th step of the pseudo-random walk is g_j. The elements in the next ℓ steps are $g_{j+1}, \ldots, g_{j+\ell}$. For $1 \leq i < \ell$, recall that in the Pollard's rho algorithm $g_{j+i} = g_{j+i-1} m_{s(g_{j+i-1})}$. Iterating leads to the following.

$$\begin{aligned} g_{j+i} &= g_{j+i-1} m_{s(g_{j+i-1})} \\ &= g_{j+i-2} m_{s(g_{j+i-2})} m_{s(g_{j+i-1})} \\ &= \cdots \\ &= g_j m_{s(g_j)} m_{s(g_{j+1})} \cdots m_{s(g_{j+i-1})}. \end{aligned}$$

The goal of the tag tracing method is to avoid computing the intermediate elements $g_{j+1}, \ldots, g_{j+\ell-1}$ and instead jump directly from g_j to $g_{j+\ell}$. This requires obtaining the element $m_{s(g_j)} m_{s(g_{j+1})} \cdots m_{s(g_{j+\ell-1})}$ and so in particular, the index values $s(g_j), s(g_{j+1}), \ldots, s(g_{j+\ell-1})$. Since g_j is available, $s(g_j)$ can be directly obtained. For $i > 1$, the value of $s(g_{j+i})$ is obtained using the auxiliary tag function $\overline{s}$ as

$$\begin{aligned} s(g_{j+i}) &= s(g_j m_{s(g_j)} m_{s(g_{j+1})} \cdots m_{s(g_{j+i-1})}) \\ &= \overline{s}(g_j, m_{s(g_j)} m_{s(g_{j+1})} \cdots m_{s(g_{j+i-1})}). \end{aligned}$$

The elements $m_{s(g_j)} m_{s(g_{j+1})} \cdots m_{s(g_{j+i-1})}$ for $i = 0, \ldots, \ell - 1$ are elements of $\mathcal{M}_\ell$ and are part of the pre-computed table.

In the tag tracing method, a tag set $\mathcal{T}$ is identified. The index function s is defined as the composition of a tag function $\tau : G \to \mathcal{T}$ and a projection function $\sigma : \mathcal{T} \to \{0, \ldots, r-1\}$, i.e., for $y \in G$, $s(y) = \sigma(\tau(y))$. Similarly, the auxiliary index function $\overline{s}$ is defined as the composition of an auxiliary tag function $\overline{\tau} : G \times \mathcal{M}_\ell \to \mathcal{T}$ and a projection function $\overline{\sigma} : \mathcal{T} \to \{0, \ldots, r-1\} \cup \{\mathsf{fail}\}$, i.e., for $y \in G$ and $m \in \mathcal{M}_\ell$, $\overline{s}(y, m) = \overline{\sigma}(\overline{\tau}(y, m))$.

The definitions of $\tau, \sigma, \overline{\tau}$ and $\overline{\sigma}$ depend on a number of parameters. The two basic parameters are the prime p and the size of the index set r. The tag set is $\mathcal{T} = \{0, \ldots, t-1\}$ which also defines the parameter t. The parameter u is taken to be a suitable word size and d is defined to be $d = \lceil \log_u(p-1) \rceil$. An integer $\bar{t}$ is chosen such that $\bar{t} > d(u-1)$ and $t\bar{t} < p^{1/3}$. The parameter w is defined to be $w = t\bar{t}$. Finally, the parameter $\overline{r}$ is defined so that $r\overline{r} = t$. As shown in [CHK12], it is possible to choose all the parameters (other than p) to be a power of 2. Based on these parameters, the functions τ and σ are defined as follows.

$$\tau(y) = \left\lfloor \frac{y \bmod p}{\bar{t}\overline{w}} \right\rfloor; \qquad \sigma(x) = \lfloor x/\overline{r} \rfloor.$$

To define the function $\overline{\tau}$, elements of $y \in \mathbb{F}_p^\star$ are represented in base u as $y \bmod p = y_0 + y_1 u + \cdots + y_{d-1} u^{d-1}$. Given $m \in \mathcal{M}_\ell$, for $i = 0, \ldots, d-1$, define $\hat{m}_i = \lfloor (u^i m \bmod p)/\overline{w} \rfloor$. Since u is fixed, for each $m \in \mathcal{M}_\ell$, the values $\hat{m}_0, \ldots, \hat{m}_{d-1}$ are pre-computed and stored in the table Tab along with m (as mentioned earlier).

Given $y \in G$ and $m \in \mathcal{M}_\ell$, the value of $\overline{\tau}(y, m)$ is defined to be the following.

$$\overline{\tau}(y, m) = \left\lfloor \frac{\left(\sum_{i=0}^{d-1} y_i \hat{m}_i\right) \bmod w}{\bar{t}} \right\rfloor.$$

Given $x \in \mathcal{T}$, the function $\overline{\sigma}$ is defined as follows.

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

The proof of correctness of the tag tracing procedure based on the above definitions of s and $\overline{s}$ is complex. We refer to [CHK12] for details. The use of tag tracing for Pollard's rho requires a suitable definition of distinguished point. Again, we refer to [CHK12] for details.

The computation of $\overline{s}$ has a chance of failure. In case of failure, a field multiplication is required. Otherwise, a field multiplication is required after every ℓ steps. The computation of $\overline{s}$ require the computations of $\overline{\tau}$ and $\overline{\sigma}$. The quantities $\hat{m}_0, \ldots, \hat{m}_{d-1}$ are part of the pre-computed table. So, for the computation of $\overline{\tau}$, the d multiplications $y_i \hat{m}_i$, $i = 0, \ldots, d-1$ are required. Apart from these, all other computations are divisions by w, $\bar{t}$ and $\bar{r}$. Since these are chosen to be powers of 2, such computations are very fast. Overall, the computation of $\overline{s}$ is significantly faster than a field multiplication.

Our description of tag tracing has been in the context of DLP computation in a multiplicative subgroup of $\mathbb{F}_p^\star$ as given in [CHK12]. A general description of the method applicable to any finite cyclic group for which suitable tag and projection functions can be defined has been provided in [CHK12]. Further, the application of the method to small characteristic, large extension degree fields has also been described in [CHK12].

Montgomery Multiplication: Let x and y be two elements of $\mathbb{F}_p$ and the requirement is to compute the product $xy \in \mathbb{F}_p$. Typically, this is a two-stage process, where in the first stage the integer multiplication of x and y is carried out and then the result is reduced modulo p. The reduction operation can take a substantial fraction of the total time to perform the field multiplication. This is especially true if p does not have a special form. Montgomery multiplication was introduced [Mon85] to replace the costly reduction operation modulo p by much cheaper divisions by powers of two. Below we provide a brief description of Montgomery multiplication based on [BM17].

Following the notation used in the context of tag tracing, let u be a power of two representing a word size and d be such that the elements of $\mathbb{F}_p^\star$ have a d-digit representation to base u. Choose $R = u^d$ such that $u^{d-1} \leq p < u^d$. Since p is odd and u is a power of two, there exists μ satisfying $\mu = -p^{-1} \bmod u$.

The core of Montgomery multiplication is a procedure called Montgomery reduction. Given an integer x having a d-digit representation to base u, Montgomery reduction computes $xR^{-1} \bmod p$. The Montgomery multiplication is a generalisation which given two integers x and y computes $xyR^{-1} \bmod p$. Suppose x and y satisfy $0 \leq x, y < R$ and x is written as $x = \sum_{i=0}^{d-1} x_i u^i$ with $0 \leq x_i < u$ for $i = 0, \ldots, d-1$. From [BM17], the Montgomery multiplication procedure is the following.

$z \leftarrow 0$
for $i = 0$ to $d - 1$ do
  $z \leftarrow z + x_i y$
  $q \leftarrow \mu z \bmod u$
  $z \leftarrow (z + pq)/u$
end for
if $z \geq p$ then $z \leftarrow z - p$
output z.

It can be shown that the output z satisfies $z \equiv xyR^{-1} \bmod p$. For a proof of this statement, we refer to [BM17]. The point to be noted here is that the only divisions in the above procedure are by u which is a power of two. So, these divisions are simply right shift operations and are very fast.

Given two field elements x and y, one way to multiply them is to first convert them to Montgomery representation by computing $\tilde{x} = xR \bmod p$ and $\tilde{y} = yR \bmod p$, then performing a Montgomery multiplication of $\tilde{x}$ and $\tilde{y}$ to obtain $\tilde{z} = \tilde{x}\tilde{y}R^{-1} = xyR \bmod p$ and then performing a Montgomery reduction (or, performing Montgomery multiplication of $\tilde{z}$ and 1) on $\tilde{z}$ to obtain $\tilde{z}R^{-1} \bmod p = xy \bmod p$. This procedure has the overhead of converting x and y to Montgomery representation and at the end applying a Montgomery reduction to $\tilde{z}$. So, for performing a single multiplication, this procedure is not very useful. Instead, Montgomery multiplication turns out to be effective when a sequence of multiplications can be done in the Montgomery representation.

3 Combining Montgomery multiplication with Tag Tracing

Pollard's rho algorithm in G consists of a sequence of multiplications modulo p. So, it is an ideal application case for Montgomery multiplication. Let us first consider how this can be done.

As described earlier, the pseudo-random walk of the Pollard's rho algorithm starts with g_0 and continues by computing $g_1, g_2, \ldots$, where for $j \geq 0$, $g_{j+1} = g_j m_{s(g_j)}$. Recall that for each $i \in \{0, \ldots, r-1\}$, the values α_i and β_i are known such that $m_i = g^{\alpha_i} h^{\beta_i}$. As before, a pre-computed table T stores $(i, m_i, (\alpha_i, \beta_i))$ for $i = 0, \ldots, r-1$.

To perform Pollard's rho algorithm using Montgomery multiplication, the multipliers are converted to Montgomery representation. This requires a change in the pre-computed table T. Denote the modified table by modT. Then the rows of modT are $(i, \tilde{m}_i, (\alpha_i, \beta_i))$ for $i = 0, \ldots, r-1$, where $\tilde{m}_i = m_i R \bmod p$.

As in the Pollard's rho algorithm described above, randomly choose a_0 and b_0 and define $z_0 = g^{a_0} h^{b_0}$. Let $\tilde{z}_0 = z_0 R \bmod p$ be the Montgomery representation of z_0. For $j \geq 0$, we define $z_{j+1} = z_j m_{s(\tilde{z}_j)} \bmod p$. Note that in this case, the indexing function s is applied to $\tilde{z}_j$ instead of being applied to z_j. This is because the element computed at the $(j+1)$-th step of the walk is $\tilde{z}_{j+1}$. The quantity $\tilde{z}_{j+1}$ is computed by applying Montgomery multiplication to $\tilde{z}_j$ and $\tilde{m}_{s(\tilde{z}_j)}$, i.e., $\tilde{z}_{j+1} = \tilde{z}_j \tilde{m}_{s(\tilde{z}_j)} R^{-1} \bmod p = z_j m_{s(\tilde{z}_j)} R \bmod p = z_{j+1} R \bmod p$.

With the above modification, all the multiplications required in the pseudo-random walk are Montgomery multiplications. So, at no stage the reduction operation modulo p is required.

The exponent information can be obtained from the walk. For $j \geq 0$, let $z_j = g^{a_j} h^{b_j}$. Note that a_0 and b_0 are known. Let $i = s(\tilde{z}_j)$. Then from the pre-computed table, it is possible to obtain (m_i, α_i, β_i). By definition, we have $z_{j+1} = z_j m_i$ and so, $a_{j+1} = a_j + \alpha_i$ and $b_{j+1} = b_j + \beta_i$.

Now, suppose there is a collision in the pseudo-random walk, i.e., there are j and k with $j < k$ such that $\tilde{z}_j = \tilde{z}_k$. Using the definition of $\tilde{z}_j$ and $\tilde{z}_k$, we have $z_j R = z_k R \bmod p$ implying $z_j = z_k \bmod p$ since R is co-prime to p. Using $z_j = z_k \bmod p$, we obtain $a_j + \mathfrak{d} b_j = a_k + \mathfrak{d} b_k$, where $\mathfrak{d} = \log_g h$. From this relation, it is possible to obtain $\mathfrak{d}$ as mentioned earlier.

The distinguished point method for detecting collisions can be applied to this modified pseudo-random walk by defining distinguished points based on $\tilde{z}_j$ for $j \geq 0$.

The above description shows that using Montgomery multiplication to define the pseudo-random walk for the Pollard's rho algorithm results in replacing all the relatively expensive modulo p operations with divisions by powers of two. We next consider, how the tag tracing method can be applied to this version of the Pollard's rho algorithm.

Let us first consider the difficulties in applying Montgomery multiplication to the setting of tag tracing. Suppose the pseudo-random walk is at an element $\tilde{z}_j$ for some $j \geq 0$. The goal of tag tracing is to perform a single field multiplication to move to the element $\tilde{z}_{j+\ell}$. For the intermediate points of the walk, the index values $s(\tilde{z}_j), s(\tilde{z}_{j+1}), \ldots, s(\tilde{z}_{j+\ell-1})$ are required.

The goal is to replace the usual field multiplication with a Montgomery multiplication. On the other hand, recall that the function s is obtained from the auxiliary function $\overline{s}$, such that for $y \in G$ and $x \in \mathcal{M}_\ell$, if $\overline{s}(y, m) \neq$ fail, then $s(ym) = \overline{s}(y, m)$. The product ym in the argument of s is the usual field multiplication. So, there are two apparently conflicting requirements. For the movement from $\tilde{z}_j$ to $\tilde{z}_{j+\ell}$, a Montgomery multiplication is to be applied, while the indexing function s is defined with respect to the usual field multiplication.

We show a simple resolution of this problem. The first thing to note is that the product in the

argument of s is not actually performed. Instead, $s(ym)$ is computed as $\overline{s}(y, m)$. For $1 \leq i \leq \ell$, we have

$$\begin{aligned}
s\left(\tilde{z}_{j+i}\right) &= s\left(\tilde{z}_{j+i-1}\tilde{m}_{s(\tilde{z}_{j+i-1})}R^{-1} \bmod p\right) \\
&= s\left(\tilde{z}_{j+i-1}m_{s(\tilde{z}_{j+i-1})}RR^{-1} \bmod p\right) \\
&= s\left(\tilde{z}_{j+i-1}m_{s(\tilde{z}_{j+i-1})} \bmod p\right) \\
&= s\left(\tilde{z}_{j+i-2}\tilde{m}_{s(\tilde{z}_{j+i-2})}R^{-1}m_{s(\tilde{z}_{j+i-1})} \bmod p\right) \\
&= s\left(\tilde{z}_{j+i-2}m_{s(\tilde{z}_{j+i-2})}m_{s(\tilde{z}_{j+i-1})} \bmod p\right) \\
&= \cdots \\
&= s\left(\tilde{z}_j m_{s(\tilde{z}_j)} \cdots m_{s(\tilde{z}_{j+i-2})}m_{s(\tilde{z}_{j+i-1})} \bmod p\right) \\
&= \overline{s}\left(\tilde{z}_j, m_{s(\tilde{z}_j)} \cdots m_{s(\tilde{z}_{j+i-2})}m_{s(\tilde{z}_{j+i-1})} \bmod p\right).
\end{aligned}$$

Let $m = m_{s(\tilde{z}_j)} \cdots m_{s(\tilde{z}_{j+i-2})}m_{s(\tilde{z}_{j+i-1})} \bmod p$. The element m is in the set $\mathcal{M}_\ell$. For computing $\overline{\tau}$, the quantities $\hat{m}_0, \ldots, \hat{m}_{d-1}$ derived from m are required, but, the actual value of m is not required. The fourth component of the pre-computed table Tab corresponding to the entry for m has the values $\hat{m}_0, \ldots, \hat{m}_{d-1}$. So, using the entries in Tab, it is possible to compute $\overline{\tau}(\tilde{z}_j, m)$ and hence $\overline{s}(\tilde{z}_j, m)$ which provides the value for $s\left(\tilde{z}_{j+i}\right)$.

Now let us consider the computation of $\tilde{z}_{j+\ell}$ from $\tilde{z}_j$.

$$\begin{aligned}
\tilde{z}_{j+\ell} &= \tilde{z}_{j+\ell-1}\tilde{m}_{s(\tilde{z}_{j+\ell-1})}R^{-1} \bmod p \\
&= \cdots \\
&= \tilde{z}_j m_{s(\tilde{z}_j)} \cdots m_{s(\tilde{z}_{j+\ell-2})}m_{s(\tilde{z}_{j+\ell-1})} \bmod p \\
&= \tilde{z}_j m_{s(\tilde{z}_j)} \cdots m_{s(\tilde{z}_{j+\ell-2})}m_{s(\tilde{z}_{j+\ell-1})}RR^{-1} \bmod p \\
&= \tilde{z}_j \mathfrak{m} RR^{-1} \bmod p \\
&= \tilde{z}_j \tilde{\mathfrak{m}} R^{-1} \bmod p
\end{aligned}$$

where $\mathfrak{m} = m_{s(\tilde{z}_j)} \cdots m_{s(\tilde{z}_{j+\ell-2})}m_{s(\tilde{z}_{j+\ell-1})}$. So, $\tilde{z}_{j+\ell}$ is obtained by applying Montgomery multiplication to $\tilde{z}_j$ and $\tilde{\mathfrak{m}}$. The element $\mathfrak{m}$ is in the set $\mathcal{M}_\ell$ and so is in the pre-computed table Tab. Note however, the value of $\tilde{\mathfrak{m}}$ is required which is not present in Tab. One may, of course, obtain $\tilde{\mathfrak{m}}$ from $\mathfrak{m}$ by performing the product $\mathfrak{m}R \bmod p$. This would be costly and would defeat the whole purpose of utilising Montgomery multiplication. So, a better option would be to include the element $\tilde{\mathfrak{m}}$ in the table Tab as part of the entry corresponding to the row for $\mathfrak{m}$. This would increase the size of the table Tab. Instead, we propose that in the table Tab, the entry $\tilde{\mathfrak{m}}$ is to be stored in place of $\mathfrak{m}$.

Let us denote the modified table by modTab. Based on the above discussion, the rows of the table modTab are as follows.

$$(\mathbf{x}, \tilde{m}_{\mathbf{x}}, (a.b), (\hat{m}_0, \ldots, \hat{m}_{d-1}))$$

where

- $\mathbf{x} = (i_1, \ldots, i_k)$, with $0 \leq k \leq \ell$, $i_1, \ldots, i_k \in \{0, \ldots, r-1\}$,
- $m_{\mathbf{x}} = m_{i_1} \cdots m_{i_k} \bmod p$ and $\tilde{m}_{\mathbf{x}} = m_{\mathbf{x}}R \bmod p$,
- (a, b) is such that $m = g^a h^b$,
- $\hat{m}_i = \lfloor (u^i m \bmod p)/\overline{w} \rfloor$ for $i = 0, \ldots, d-1$.

So, modTab stores $\tilde{m}$ instead of m while the quantities $\hat{m}_0, \ldots, \hat{m}_{d-1}$ in modTab are derived from m and not from $\tilde{m}$. In particular, the only difference between Tab and modTab is that Tab stores m whereas modTab stores $\tilde{m}$. All other entries of Tab and modTab are identical. So, the storage requirements of both Tab and modTab are also the same.

Using modTab, tag tracing can proceed as follows. For the jump from $\tilde{z}_j$ to $\tilde{z}_{j+\ell}$, the entry $\tilde{m}$ is to be used, whereas for the computations of the outputs of the function s, the entries $\hat{m}_0, \ldots, \hat{m}_{d-1}$ are to be used. Consequently, the advantage of tag tracing is retained, i.e., all computations required for computing the output of s are divisions by powers of two. Additionally, there is an efficiency gain where the field multiplication required in tag tracing for the jump from the j-th step of the walk to the $(j+\ell)$-th step of the walk is replaced by a Montgomery multiplication. As explained earlier, this replaces the costly reduction operations modulo p by inexpensive divisions by powers of two.

4 Conclusion

In this work, we have shown how to combine Montgomery multiplication to the tag tracing variant of Pollard's rho algorithm for solving DLP in $\mathbb{F}_p$. This results in replacing costly modulo p operations with divisions by a power of two which will lead to practical speed-ups in actual implementations.

References

[BM17] J. W. Bos and P. L. Montgomery. *Montgomery arithmetic from a software perspectiv*, pages 10–39. Cambridge University Press, 2017.

[Bos17] Joppe W Bos. Montgomery arithmetic from a software perspective. *IACR Cryptology ePrint Archive*, 2017:1057, 2017.

[CHK12] J. H. Cheon, J. Hong, and M. Kim. Accelerating Pollard's rho algorithm on finite fields. *Journal of cryptology*, 25(2):195–242, 2012.

[Mon85] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[Pol78] J. M. Pollard. A Monte Carlo method for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[SJ84] C. Schnorr and H. Lenstra Jr. A Monte Carlo factoring algorithm with linear storage. *Mathematics of Computation*, 43(167):289–311, 1984.

[vOW99] P. van Oorschot and M. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12:1–28, 1999.