On improvements of r-adding walks to solve the Discrete Logarithm Problem

A thesis submitted towards partial fulfilment of
BS-MS Dual Degree Programme

by

Hardik Gajera

under the guidance of

Dr. Ayan Mahalanobis

IISER, Pune

Indian Institute of Science Education and Research, Pune

Certificate

This is to certify that this thesis entitled 'On improvements of r-adding walks to solve the Discrete Logarithm Problem' submitted towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents original research carried out by Hardik Gajera at IISER, Pune, under the supervision of Dr. Ayan Mahalanobis during the academic year 2012-2013.

Student
HARDIK GAJERA

Supervisor
DR. AYAN MAHALANOBIS

Acknowledgements

I take this opportunity to express my gratitude and deep regards to my guide Dr. Ayan Mahalanobis for his exemplary guidance, monitoring and constant encouragement throughout the course of this thesis.

I would like to acknowledge the help of John Cannon with Magma. I would like to thank Dr. Jayant Deshpande for stimulating conversations regarding QQ-plots and Jung Hee Cheon, Min Kyu Kim for their valuable reviews and suggestions through emails.

Lastly, I thank my parents, brother and friends for their constant encouragement without which this assignment would not be possible.

HARDIK GAJERA

MARCH, 2013

IISER, PUNE

Abstract

It is currently known from the work of Shoup and Nechaev that a generic algorithm to solve the discrete logarithm problem in a group of prime order must have complexity at least $k\sqrt{N}$ where N is the order of the group. In many collision search algorithms, this complexity is achieved. So with generic algorithms one can only hope to make the k smaller. This k depends on the complexity of the iterative step in the generic algorithms. The $\sqrt{N}$ comes from the fact there is about $\sqrt{N}$ iterations before a collision. So if we can find ways that can reduce the amount of work in one iteration then that is of great interest and probably the only possible modification of a generic algorithm. The modified r-adding walk does just that. It reduces the amount of work done in one iteration of the original r-adding walk. In this paper we study this modified r-adding walk, we critically analyse it and we compare it with the original r-adding walk. In the final chapter, we discuss an improvement of original r-adding walk on elliptic curve over $\mathbb{F}_p$.

Table of Contents

Chapter 0

Introduction

In a recent article in the Journal of Cryptology, Cheon et. al. [2] published a novel time-memory speed-up of the well known Teske's r-adding walk [20, 21]. Choen et. al. claim up to 10 times speed-up of the Pollard's rho algorithm to solve the discrete logarithm problem in a finite field. This has no doubt stunned the cryptography world. Ten times speed-up is a remarkable speed-up. Earlier, what would have taken ten years to solve a discrete logarithm will now take about one year. This type of claim must be verified and re-verified.

This thesis is based on our work trying to verify that claim made by the authors [2]. We **cannot support their claim** of "ten times faster". However, we found that the **modified r-adding walk**, the one proposed by Cheon et. al., is about two to three times faster (depending on what r one chooses) than the original r-adding walk proposed by Teske [20, 21]. We would reiterate that we find the idea of the modified r-adding walk interesting and novel.

As we will discuss in details later, the modified r-adding walk replaces a field multiplication in the r-adding walk by a *tag computation* and table lookup. If we only count the number of multiplications in the base field then the modified r-adding walk is way ahead. However, when we factor in the table lookup and the branching necessary to do two operations instead of one, the advantage diminished substantially.

The only way we find to compare these two algorithms – the r-adding walk [20, 21] and the modified r-adding walk [2] is to do an actually implementation of these two algorithms in an identical platform. In that implementation we must be very careful. The way we implement table lookup will influence the outcome of our experiment. We choose Magma [1] as our language of choice. The reason for that is simple, Magma probably is the best language to manipulate polynomials and has the best available large finite field implementation. In trying to implement the modified r-adding walk we implemented the algorithm many times with different design paradigms until we were reasonable certain that the implementation was optimal in speed. Then we implemented the original r-adding walk. We made sure that these two algorithms were as similar as possible. As a matter of fact we can speak with conviction that the only difference in these two implementations was, the *field multiplication in the original r-adding walk was replaced by tag computation and table lookup*. We changed the r-adding walk to use the distinguished path segment to define its distinguished points and the same index function γ.

In the original and the modified r-adding walk we need an index function γ. We define the function γ and have shown that this is one of the best choices possible (see Figure 2). This makes us confident about our findings. So the obvious question comes: why are our findings so different with that of Cheon et. al. [2]? We think the reason is that the authors used the NTL library. This library is known to be slower than Magma in the implementation of polynomial multiplication. Since the authors provide very little details about their implementation, we are unable to say more. Moreover, we have serious issues with the use of $r = 4, 8$ in the modified r-adding walk. It is known and we have re-established in Section 1 that the choice for r must be at least 16. This has substantial effect on the speed of the modified r-adding walk.

This chapter is organized in two parts. In the first part we study the r-adding walk. The rho length of a r-adding walk is defined to be the number of iterations in the walk before the first collision. In Section 1 we study the **distribution of the rho length**. Our study of the r-adding walk is different from previous studies as we are using the distinguished path segment to define the distinguished points. The purpose of this section is to (re)establish the fact that one should use large r ($r \geq 16$). We thought that it is important to establish this fact because in the modified r-adding walk uses $r = 4, 8$. They use the mean of the rho length for $r = 4, 8$ but doesn't compute the standard deviation. The variance is large, which makes their estimation ineffective.

We then explain the modified r-adding walk and many of its salient features in details in Section 2. We then do the comparison with the original r-adding walk and produce our result (Table 2.1). We developed a new table lookup method that makes the modified r-adding walk go faster.

Chapter 1

r-adding walk

Generic algorithms for solving the discrete logarithm problem are algorithms that do not use the structure or representation of the group. They use the operation of multiplication, inversion and equality in group elements. These kind of algorithms are restrictive by nature, they are often not the fastest algorithms. However, they are very powerful, they can be applied to the discrete logarithm problem in every possible scenario, be it the group of rational points of an elliptic curve or that of the group of units of a finite field. It is currently known that the complexity of solving the discrete logarithm problem in a finite cyclic group of prime order N, using any generic algorithm is at least $k\sqrt{N}$ [8,17], where k is a positive constant. So any new modification of a generic algorithm can make the k smaller. This k is tied to the amount of work done in one iteration.

The generic algorithm that we want to start our discussion with is the famous Pollard's rho algorithm. He first developed it to factor integers and then that was adapted to solve the discrete logarithm problem [9]. The idea behind the Pollard's rho algorithm is simple, create an iterated random walk in a finite cyclic group. Since the set is finite, there will always be a collision in this random walk. From that collision find the logarithm. However, there is one problem with finding the collision, store all the elements of the random walk. Not only that, every time a new node of the walk is computed one must check that with all the previous nodes. This increases both the time and space complexity of the algorithm. Pollard found a clever solution to the problem. He introduced a function, iteration by which will simulate a random walk. Let $G = \langle g \rangle$ be a group of prime order. We are given g and $h = g^x$ where x is the discrete logarithm. Pollard's function is as follows:

$$f(y) = \begin{cases} gy & \text{if} \quad y \in G_1 \\ y^2 & \text{if} \quad y \in G_2 \\ hy & \text{if} \quad y \in G_3 \end{cases}$$

where G_1, G_2 and G_3 is an almost equal sized partition of G. In this case the iterated random walk looks like the Greek letter ρ and so the name Pollard's rho algorithm. The rho structure indicates that once there is a collision, the walk will repeat itself. This changes the storage requirement dramatically as follows: we pick a few arbitrary points, and call them **distinguished points**, we will only have to look for collision in those distinguished points. Another way to think of distinguished points is laying traps. We lay a few traps and hope that some of them will be on the repeating part of the ρ. When we have two elements in our trap, we know that there is a collision and the algorithm stops. It is clear that we won't catch the first collision this way, but we don't have to do the search either, and the saving in space compensates for this increase in time. We should add here that Pollard didn't propose this distinguished point method. His idea was using the Floyd's cycle finding method. However we present Pollard's rho algorithm this way to motivate our next discussion.

Teske [20, 21] developed r-adding walk in the same spirit as the Pollard's rho algorithm (the spirit being the ρ, a repeating random walk) but in practice it works differently. Let G, g and h be the same as above. For some $r \in \mathbb{N}$, let $\{m_1, m_2, \ldots, m_r\}$ be a set of elements of G of the form $g^\alpha h^\beta$ picked uniformly randomly where α, β are integers. This is usually done by choosing α and β uniformly random. These m_is will be referred to as **multipliers** in this paper. Let $\gamma : G \to \{1, 2, \ldots, r\}$ be a function. An r-adding walk $\mathcal{F}$ is defined iteratively as follows:

$$\mathcal{F}(Y) = Y m_{\gamma(Y)}. \tag{1.1}$$

Note that computation of a node in the r-adding walk requires one group multiplication and one evaluation of the function γ.

The starting point of the r-adding walk is computed by choosing a positive integer α_0 uniformly random from the set $\{1, 2, \ldots, |G|\}$ and computing $Y_0 = g^{\alpha_0}$. To compute m_i two integers α_i and β_i are chosen uniformly randomly from $\{1, 2, \ldots, |G|\}$ and $m_i = g^{\alpha_i} h^{\beta_i}$. It is easy to notice that as the walk progresses, the nodes of the walk are of the form $g^\alpha h^\beta$ for positive integers α and β. When there is a collision we have $g^\alpha h^\beta = g^{\alpha'} h^{\beta'}$. Which forms the equation

$$\alpha + x\beta = \alpha' + x\beta' \mod |G|. \tag{1.2}$$

Since $|G|$ is prime, this equation is easy to solve for the unknown x.

Teske [21, Section 5] has shown that for large enough r and suitable γ the r-adding walk simulates a *random random walk*[1] very well. In this paper we are not repeating Teske's work. We are not looking at the randomness aspect of this r-adding walk. We want to study the distribution of the rho length of a r-adding walk.

[1]A random random walk is a random walk with a random starting point. Random being chosen uniformly random.

Iterative walks depending on functions from a finite set to itself are extensively studied by statisticians, see [4, 11, 13].

Definition 1.0.1 (Iterated walk)**.** Let Ω be a finite set of size n and $T : \Omega \to \Omega$ be a function. Let $x \in \Omega$. Then the iterative walk corresponding to T is defined as $\{x_0, T(x_0), T^2(x_0), \ldots, \}$ where T^k is defined as composition of T with itself k times.

Let us define $S_T(x_0) = \{x_0, T(x_0), T^2(x_0), \ldots, T^k(x_0), \ldots\}$ then the size of $S_T(x_0)$ is the number of steps required for the collision with T starting from x_0. If we denote the size of $S_T(x_0)$ by s, then we are interested in the distribution of s when T is chosen uniformly random from the set of all functions.

Let X be a random variable that counts the number of steps before a collision, for an iterating walk from T, starting from an arbitrary element x_0. Let us compute the probability density function for X. There is total n^n many possible functions from the set n many elements to itself. If we consider that the T is chosen randomly, then $P(T) = 1/n^n$. Let, s' denote the number of elements in the cycle contained in $S_T(x_0)$. Then for any choice of x,

$$P(s = k, s' = j) = P\{T^r \neq x, Tx, \ldots, T^{r-1}x : (0 < r \leq (k-1)); T^k x = T^{k-j}x\}$$

Using n many different elements, we can form n^k many different sequences of length k. But, we want to have sequence with $s = k, s' = j$. For fixed x, we can Tx in $(n-1)$ many ways,T^2x in $(n-2)$ and so on up to T^{k-1}. We don't have any choice for $T^k x$ because $T^k x = T^{k-j}x$ where $j > 0$ and we have already chosen $T^{k-j}x$. Hence, number of favourable sequences is $(n-1)(n-2)\ldots(n-(k-1)) = \frac{(n-1)!}{(n-k)!}$. Hence, we have

$$P(s = k, s' = j) = \frac{\text{number of favourable sequences}}{\text{total number of sequences}} = \frac{(n-1)!}{(n-k)!n^k}$$

. Note that, $P(s = k, s' = j)$ is independent of j. Hence, we have

$$P(s = k) = \sum_{j=1}^{k} \frac{(n-1)!}{(n-k)!n^k} = \frac{(n-1)!k}{(n-k)!n^k}.$$

Let us compute asymptotic probability density of s. Let, $s = \sqrt{n}x, s' = \sqrt{n}y$ and use Stirling approximation for factorials ($n! \sim \sqrt{2\pi n}(\frac{n}{e})^n$).

$$\begin{aligned}
P(s=\sqrt{n}x, s'=\sqrt{n}y) &= \frac{n!}{(n-\sqrt{n}x)!n^{\sqrt{n}x+1}} \\
&\sim \frac{\sqrt{2\pi}n^{n+1/2}e^{-n}}{\sqrt{2\pi}(n-\sqrt{n}x)^{n-\sqrt{n}x+1/2}e^{-n+\sqrt{n}x}} \\
&= \frac{n^{n-\sqrt{n}x-1/2}e^{-\sqrt{n}x}}{(n-\sqrt{n}x)^{n-\sqrt{n}x+1/2}} \\
&= \frac{n^{n-\sqrt{n}x-1/2}e^{-\sqrt{n}x}}{n^{n-\sqrt{n}x+1/2}(1-\frac{x}{\sqrt{n}})^{n-\sqrt{n}x+1/2}}
\end{aligned}$$

Write $(1-\frac{x}{\sqrt{n}})^{n-\sqrt{n}x+1/2} = e^{(n-\sqrt{n}x+1/2)\log_e(1-\frac{x}{\sqrt{n}})}$. If we expand $\log_e(1-\frac{x}{\sqrt{n}})$ and let n tends to ∞, then $\frac{e^{-\sqrt{n}x}}{e^{(n-\sqrt{n}x+1/2)\log_e(1-\frac{x}{\sqrt{n}})}} \sim e^{-\frac{1}{2}x^2}$. Hence, for large enough n, we have $P(s=\sqrt{n}x, s'=\sqrt{n}y) \sim n^{-1}e^{-\frac{1}{2}x^2}$. Hence, asymptotic density of $(s/\sqrt{n}, s'/\sqrt{n})$ is given by $f(x,y) = e^{-\frac{1}{2}x^2}$ $0 < y \leq x < \infty$. By integrating $f(x,y)$ with respect to y, we get the marginal distribution $\Gamma(x)$. Hence, $\Gamma(x) = xe^{-\frac{1}{2}x^2}$.

Harris [4, §3] has also shown that the probability density function of X converges to

$$\Gamma(x) = x\exp^{-\frac{1}{2}x^2}, \quad x > 0 \tag{1.3}$$

as n tends to infinity, where $x\sqrt{n} := s$. This is the classic Rayleigh distribution and it is known that the mean $\mu(X) = 1$ and the standard deviation $s.d.(X) \approx 0.523$ in units of $\sqrt{\frac{\pi}{2}q}$. [2]

Our question is: Is the distribution of X, the random variable to count the number of steps before a collision, for the original r-adding walk close to the distribution function Γ? We answer this question in **affirmative for a large enough** r and a suitably chosen γ by an experiment. Before we discuss our experiment, let us explain the methodology for the experiment. We have a probability density function Γ. We call the function $\Gamma(x) = x\exp^{-\frac{1}{2}x^2}, \ x > 0$ the test distribution function. We intend to generate data points (number of iterations before collision in a r-adding walk for different values of r) and then see if the data fits the test distribution function Γ. We will further plot the data and see the shape of that curve and compare that with the test distribution function. In judging, if a set of data fits a distribution – the *quantile plots* (Q-Q plots) are very useful [23, §4].

[2] The theoretical mean and standard deviation of Rayleigh distribution are generally in units of $\sqrt{(q)}$ but, our experimental mean and standard deviation are in units of $\sqrt{\frac{\pi}{2}q}$. Hence, we have provided the theoretical mean and standard deviation in units of $\sqrt{\frac{\pi}{2}q}$.

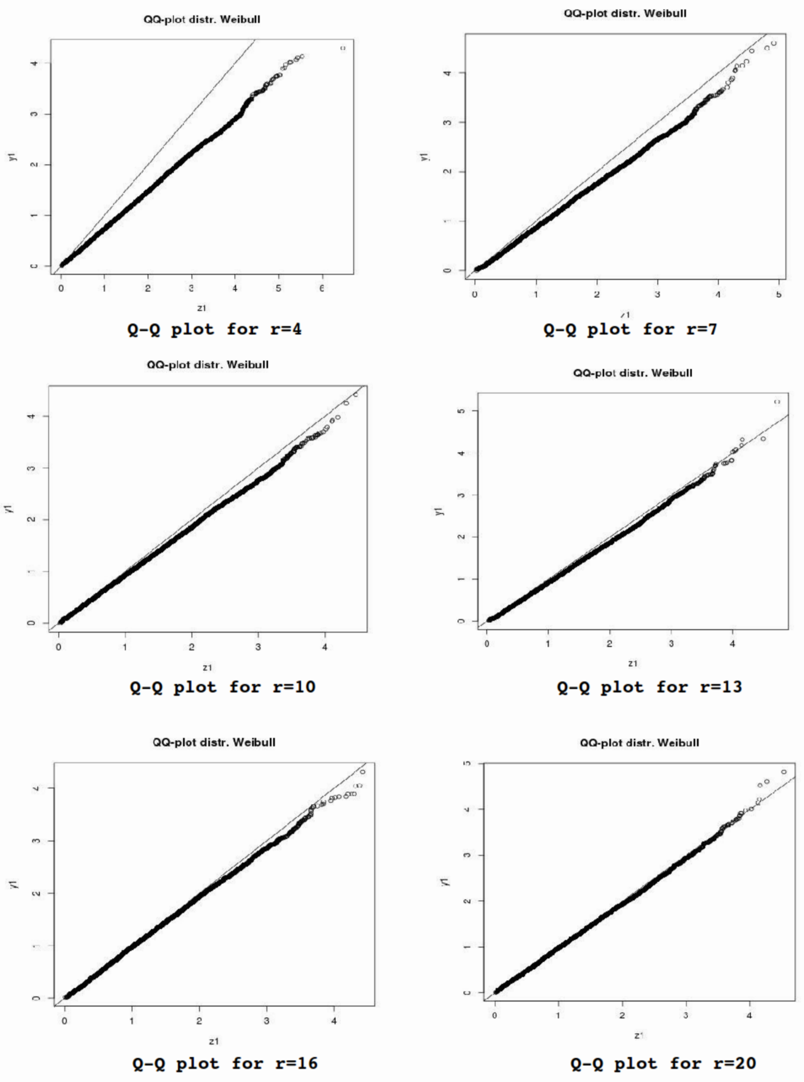

Figure 1.1: The QQ-plots for average rho-length for different values of r over finite field GF(2^{1023}) and subgroup of size 2^{35}.

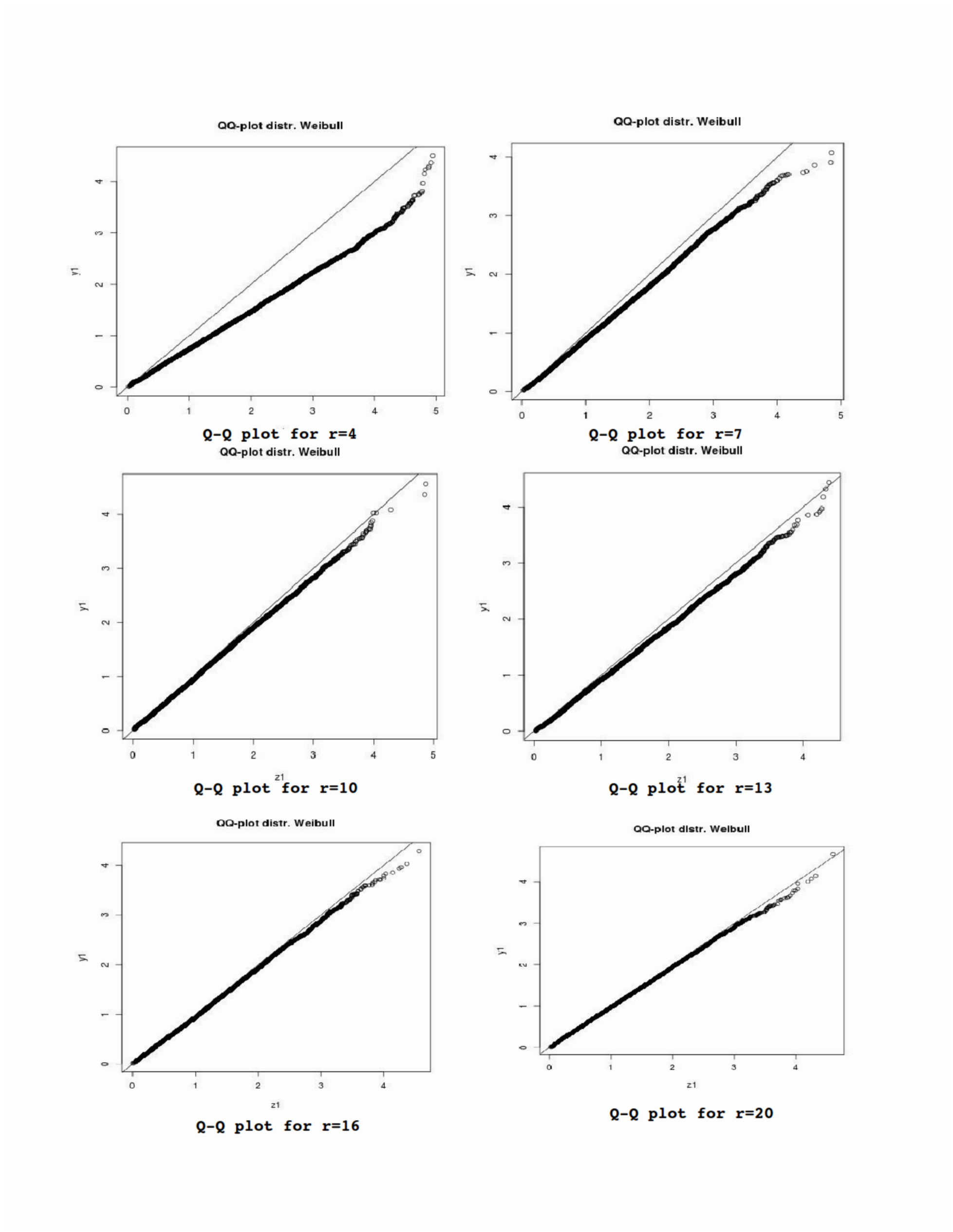

Figure 1.2: The QQ-plots for average rho-length for different values of r over finite field GF(2^{1023}) and subgroup of size 2^{37}.

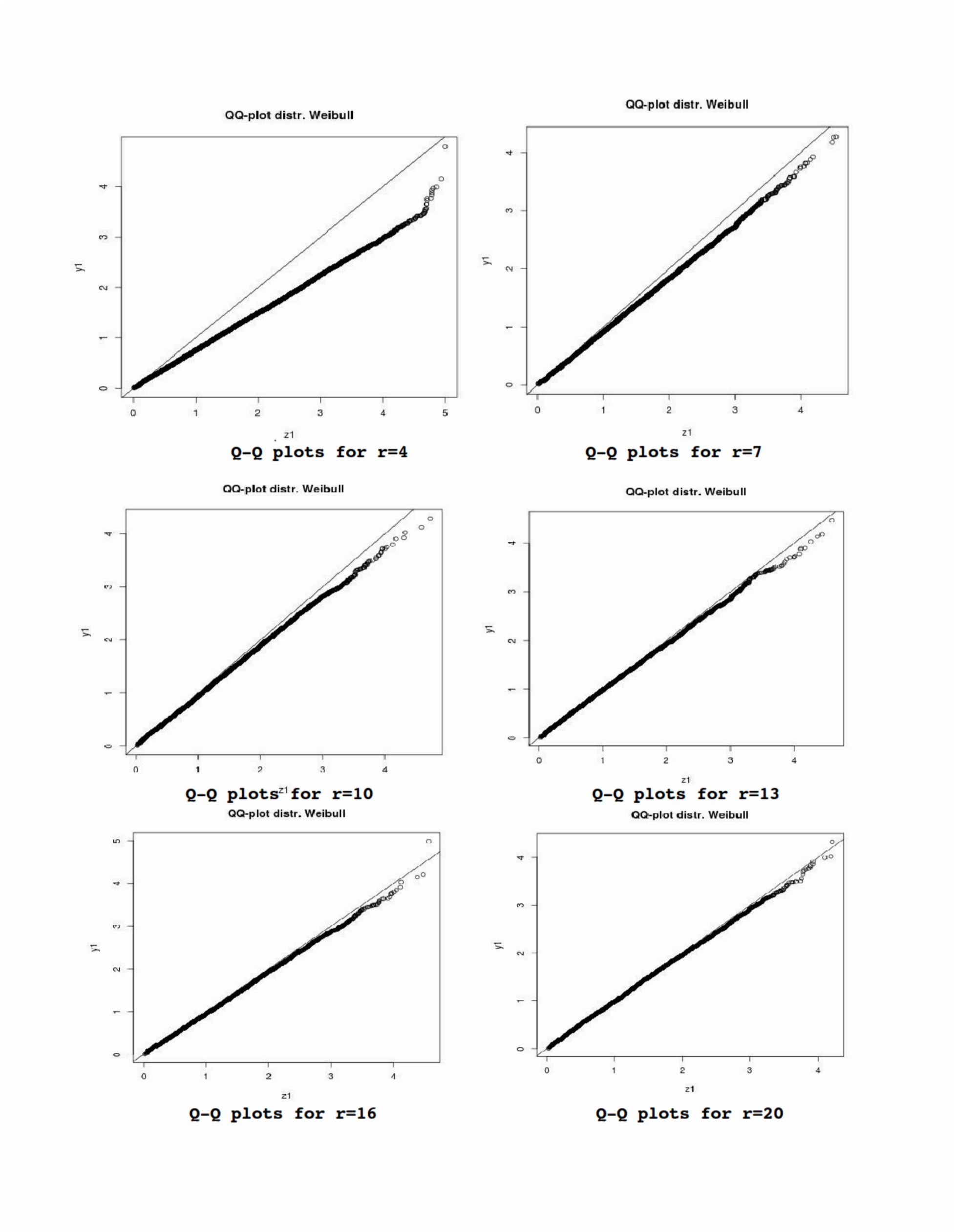

Figure 1.3: The QQ-plots for average rho-length for different values of r over finite field GF(2^{1023}) and subgroup of size 2^{39}.

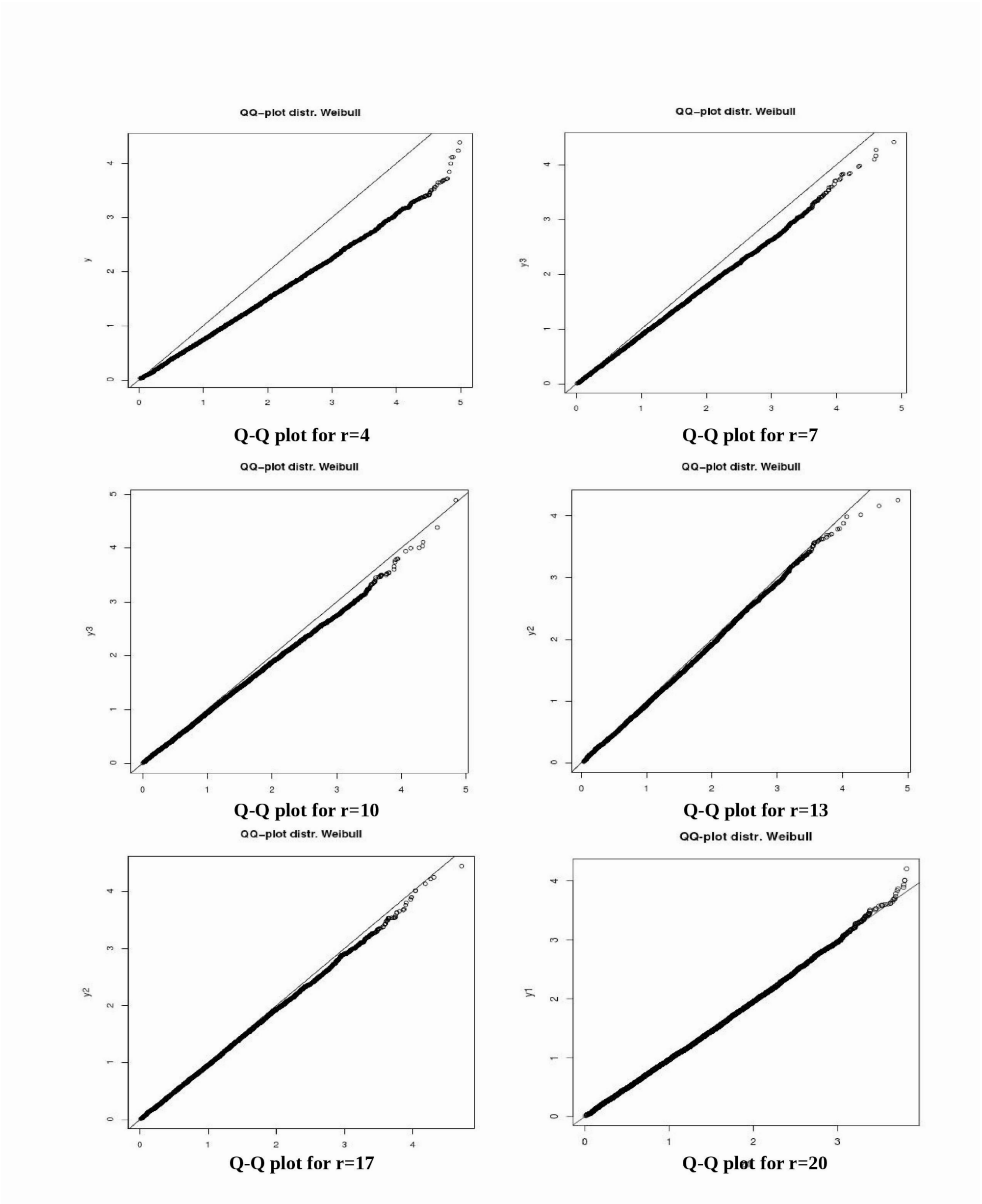

Figure 1.4: The QQ-plots for average rho-length for different values of r over finite field GF(2^{1023}) and subgroup of size 2^{40}.

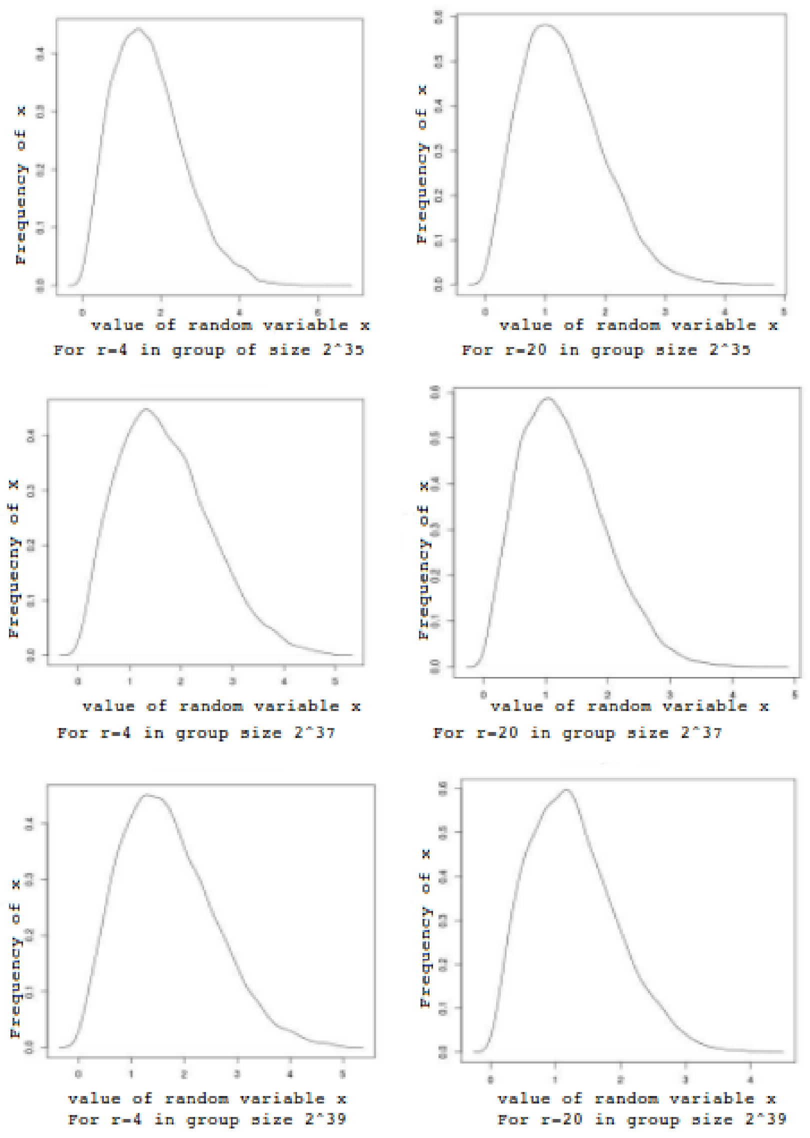

Figure 1.5: Empirical Probability Density Function for two different r-values over finite field GF(2^{1023}) and prime order subgroup.

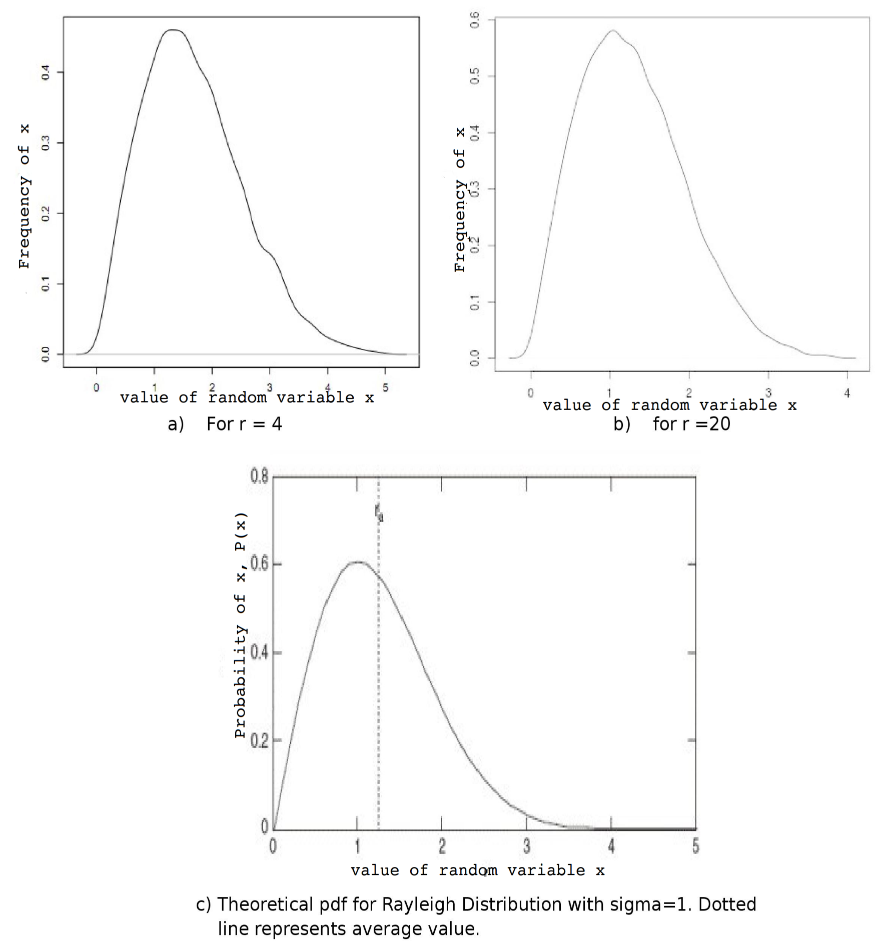

Figure 1.6: Empirical Probability Density Function for two different r-values over finite field GF(2^{1023}) and prime order subgroup of size 2^{40}.

For our experiment, We have used a subgroup of the binary field $\mathbb{F}_{2^{1024}}$ of 40-bit prime order. We have used the same r-adding walk described in [21]. We describe the index function[3] γ. For each r, choose an integer $t \approx \log_2 r$ and define the **tag function** $\tau : \mathbb{F}_{2^n} \longrightarrow \mathbb{F}_{2^t}$ where the image of τ is the vector of the coefficients of the first t many highest degree terms in the polynomial representation of an element in $\mathbb{F}_{2^n}$. Define another function $\sigma : \mathbb{F}_{2^t} \longrightarrow \{1, 2, \ldots, r\}$ where for each element $f(x) = a_0 + a_1 x + \ldots + a_{t-1} x^{t-1}$ in $\mathbb{F}_{2^t}, \sigma(f(x)) = 1 + f(2) = 1 + a_0 2^0 + a_1 2^1 + \ldots + a_{t-1} 2^{t-1}$. Note that for each i, $1 \leq i \leq r, a_i$ is in $\mathbb{F}_2$. This means that $0 \leq f(2) \leq r - 1$ ($t \approx \log_2 r$) which implies that $2^t \approx r$. Then define $\gamma = \sigma \circ \tau$. Notice that τ is an additive function. It is clear from the above that once τ is computed, it is straightforward to compute the σ and then γ. For the purpose of this paper one can use γ and τ interchangeably.

We used the R software [10] to get the empirical probability density function and the Q-Q plots for our data. We have plotted the empirical probability density function and the Q-Q plots of the variable X for $r = 4$ and $r = 20$ over finite field $\mathbb{F}_{2^{1023}}$ and *prime* subgroups of size $2^3 5$, $2^3 6$, $2^3 7$, $2^3 9$ and 2^{40}. Figures for epdf and Q-Q plots can be found in appendix.Figure 1.6 represents the empirical probability density function(epdf) of the variable X for $r = 4$ and $r = 20$ over finite field $\mathbb{F}_{2^{1023}}$ and *prime* subgroup of size 2^{40}.Notice that the maximum y-value of the function for $r = 4$ is less than 0.5 and for $r = 10$ it is greater than 0.55. Theoretically, maximum y-value of Rayleigh distribution is approximately 0.57. Also, the epdf for $r = 4$ is wider than the epdf for $r = 20$. This means that for $r = 20$, the probability of the value of X lying in the neighbourhood of the mean is higher than that for $r = 4$. Hence, epdf for $r = 20$ simulates Rayleigh distribution more closely than $r = 4$. This means that 20-adding walk is better than 4-adding walk. Figure 1.4 represents the Q-Q-plots for different r-values over finite field $\mathbb{F}_{2^{1023}}$ and subgroup of size 2^{40}. It seems clear from the Q-Q plots that as r increases the distribution of X comes closer and closer to the test distribution function Γ and for $r = 20$, it is virtually indistinguishable from Γ. Hence, one should use r close to 20 for practical purposes. Teske [21] also suggested that $r \geq 16$ for practical purposes.

The *rho length* of a r-adding walk is the number of iterations before the first collision in the walk. We ran tests to compare the average rho length of a r-adding walk for different values of r. Let us discuss in details our experiment. We have used cyclic subgroups of $G = \langle g \rangle$ of the field $\mathbb{F}_{2^{1024}}$. We have used four subgroups of prime order, they are 36-bit, 37-bit, 39-bit, 40-bit primes. For each DLP instances, an element $h \in G$ was chosen and a set of multipliers were randomly selected. Then the r-adding iteration function $\mathcal{F}$ was iterated from a random start-

[3]The index function is so chosen because we will use the same index function for the modified r-adding walk later.

Table 1.1: Average rho lengths for different r-adding walk over different subgroups of binary field $\mathbb{F}_{2^{1024}}$, in units of $\sqrt{\frac{\pi}{2}q}$ (the standard deviations of ρ_r are given in the parentheses).

r-value	36-bit	37-bit	39-bit	40-bit	average ρ_r
4	1.332(0.701)	1.353(0.705)	1.347(0.697)	1.333(0.688)	1.341
5	1.212(0.627)	1.187(0.620)	1.189(0.627)	1.191(0.620)	1.195
6	1.146(0.598)	1.137(0.583)	1.131(0.588)	1.133(0.585)	1.137
7	1.127(0.575)	1.113(0.575)	1.108(0.570)	1.121(0.585)	1.116
8	1.087(0.566)	1.092(0.565)	1.084(0.570)	1.081(0.567)	1.086
9	1.069(0.557)	1.077(0.565)	1.077(0.561)	1.072(0.560)	1.074
10	1.080(0.563)	1.062(0.556)	1.064(0.558)	1.075(0.558)	1.070
11	1.055(0.547)	1.081(0.549)	1.063(0.547)	1.053(0.544)	1.063
12	1.051(0.545)	1.049(0.554)	1.046(0.542)	1.053(0.550)	1.050
13	1.060(0.543)	1.080(0.551)	1.038(0.550)	1.050(0.533)	1.057
14	1.059(0.546)	1.057(0.548)	1.053(0.546)	1.059(0.547)	1.057
15	1.061(0.547)	1.084(0.566)	1.032(0.540)	1.033(0.537)	1.052
16	1.037(0.541)	1.037(0.542)	1.044(0.541)	1.034(0.540)	1.038
17	1.034(0.548)	1.038(0.559)	1.038(0.540)	1.040(0.545)	1.037
18	1.038(0.542)	1.035(0.538)	1.024(0.535)	1.040(0.544)	1.034
19	1.028(0.543)	1.035(0.538)	1.027(0.537)	1.027(0.537)	1.029
20	1.033(0.538)	1.031(0.538)	1.017(0.536)	1.020(0.521)	1.025

ing point until the first collision in the walk with the γ defined previously. Once the first collision was found, the rho length was recorded. Note that we were looking for the first collision in a r-adding walk, so we didn't used the distinguished point method which gives approximate position of the collision. We have repeated this processes $10,000$ times for each of the four subgroups mentioned before. In the Table 1.1, the top row represents the size of the subgroup and all other rows represents average rho length for a specific r. The rho lengths are given in units of $\sqrt{\frac{\pi}{2}q}$. Let ρ_r denotes the rho length of the r-adding walk in units of $\sqrt{\frac{\pi}{2}q}$. The table clearly shows that the ρ_r is nearly stable and almost equal to one over subgroups of different order for each $r \geq 16$. Hence, it is advisable to use $r \geq 16$ for all practical purpose. This reconfirms Teske's work [20, 21]. We assume that the ρ_r given in Table 1.1 are roughly same even on very large prime order subgroup of $\mathbb{F}_{2^{1024}}$. Cheon et. al. [2] found similar results for ρ_r.

Chapter 2

A time-memory improvement of r-adding walk over finite field

Cheon et. al. [2] found an innovative way to speed up the r-adding walk using a time-memory trade-off. Recall that one iteration of the r-adding walk requires one field multiplication and one evaluation of the function γ. The novel idea in this paper is not to do the multiplication at every step, rather do it once in a while. However multiplication is a binary operation, so it doesn't matter how often one does that, it has the same number of multiplications at the end. Authors circumvented this problem by storing various products as group elements in a table $\mathcal{M}_l$ and calling them when required.

Recall that at any intermediate step, the iteration computes the product $Ym_{i_1}m_{i_2} \ldots m_{i_k}$ where $i_j \in \{1, 2, \ldots, r\}$ and k a positive integer. If there is a table that can stores the values of product $m_{i_1}m_{i_2}m_{i_3} \ldots m_{i_k}$ for various k, then these k multiplications can be reduced to one multiplication. This is the central idea that makes tag tracing go faster than the normal r-adding walk. Select a positive integer l, and compute the product after every l steps using a table $\mathcal{M}_l$ that will be described soon.

There is one more thing that needs mention, the function $\gamma : G \to \{1, 2, \ldots, r\}$ can only be computed after the product is computed. Recall that the iteration in r-adding walk is $\mathcal{F}(Y) = Ym_{\gamma(Y)}$. So until Y is available one cannot compute $\gamma(Y)$ and the iteration cannot work. This gets in the way of the idea, "multiply after every l steps". Authors [2] solved this problem by introducing a **tag** which is associated with every group element in the table ($\mathcal{M}_l$). Then γ is a function from this tag to $\{1, 2, \ldots, r\}$. This involves a table lookup in the modified r-adding walk that will slow things down. We describe the algorithm of the modified r-adding walk in details later.

2.1 Tag

Recall that we are working in a group $G \subset \mathbb{F}_{2^\eta}^\times$. What we discuss will work for any field extension of prime characteristic, however we do not know how to make this work outside of finite fields.

In this case we represent the vector space $\mathbb{F}_{2^\eta}$ as a η-dimensional vector space over the field of two elements $\mathbb{F}_2$. We take the polynomial basis $\{1, x, x^2, \ldots, x^{\eta-1}\}$ as the basis of $\mathbb{F}_{2^\eta}$ and any element $x \in \mathbb{F}_{2^\eta}$ can be written uniquely as a polynomial of degree less than η with coefficients over $\mathbb{F}_2$. Fix a small positive integer t, the tag corresponding to t for an element $f(x) = a_0 + a_1 x + \ldots, +a_{\eta-1}x^{\eta-1}$ in $\mathbb{F}_{2^\eta}$, is the coefficients of $\{x^{\eta-t}, x^{\eta-t+1}, \ldots, x^{\eta-1}\}$. In short, the tag is a binary vector of length t consisting of coefficients of the highest t powers of the polynomial $f(x)$. Notice that our polynomials are always of degree less than η. One can also define the tag as an **additive function** $\tau : \mathbb{F}_{2^\eta} \to \mathbb{F}_2^t$ where the image of τ is the vector of coefficients of t highest degree terms in the polynomial basis representation of an element in $\mathbb{F}_{2^\eta}$. It is easy to verify that this tag is an additive function and respects scalar multiplication.

Now assume that $\tau(x^0 m_i), \tau(xm_i), \tau(x^2 m_i), \ldots, \tau(x^{\eta-1}m_i)$ is known, then we can compute $\tau(fm_i)$ using the following formula:

$$fm_i = a_0 m_i + a_1(xm_i) + a_2(x^2 m_i) + \ldots + a_{\eta-1}(x^{\eta-1}m_i)$$

and from the know properties of τ we have

$$\tau(fm_i) = a_0\tau(m_i) + a_1\tau(xm_i) + a_2\tau(x^2 m_i) + \ldots + a_{\eta-1}\tau(x^{\eta-1}m_i) \quad (2.1)$$

The above statement follows from the distribution of multiplication over addition in the polynomial algebra $\mathbb{F}_2[x]$ and is even true if we replace m_i by a product $m_{i_1}m_{i_2}\ldots m_{i_k}$ for any positive integer k.

2.2 The table $\mathcal{M}_l$

The table $\mathcal{M}_l$ has l rows. The first row contains r cells. Each cell is numbered by $\{1, 2, \ldots, r\}$ corresponding to $\{m_1, m_2, \ldots, m_r\}$ stating from the left. The second row has $\binom{2+r-1}{2} = \binom{r+1}{2}$ cells. Total number of all possible $m_i m_j$ where $1 \leq i, j \leq r$ is r^2. Notice that we are in a abelian group and $m_i m_j = m_j m_i$. There are exactly r-many elements of the form $m_i m_i$ where $1 \leq i \leq r$. Hence, after removing duplicates, we are left with $\frac{r^2-r}{2} + r = \frac{r^2+r}{2} = \binom{r+1}{2}$ many elements in second row. Continuing in this way we have the last row as the l^{th} row. This row has $\binom{l+r-1}{l}$ cells. Each cell in the table corresponds to a vector

$(i_1, i_2, \dots, i_k)$ for some positive integer k and this vector correspond to the group element $m_{i_1}m_{i_2}\dots m_{i_k}$ where each $i_j \in \{1, 2, \dots, r\}$.

Each cell in the above table has four sets of information attached to it:

Multiplier Information This is a vector $(i_1, i_2, \dots, i_k)$ of integers, where $k \leq l$ and $i_j \in \{1, 2, \dots, r\}$ for all j. We can assume that the vector is ordered. It contains the information on the multipliers involved in this cell.

Group element The group element formed from multiplication of the multipliers involved in a cell, i.e., $m = m_{i_1}m_{i_2}\dots m_{i_k}$ is computed and stored.

Exponent Recall that each multiplier $m_i = g^{\alpha_i}h^{\beta_i}$ for some integers α_i and β_i. When these multipliers are multiplied, the exponents are added up in the product. This information (α, β) is the exponent, where $\alpha = \sum_j \alpha_{i_j}$ and $\beta = \sum_j \beta_{i_j}$. One needs the exponent information when the walk reaches a distinguished point.

Tag The vector $(\tau(m), \tau(xm), \dots, \tau(x^{\eta-1}m))$ is stored.

2.3 An overview of the algorithm

The modified r-adding walk proposed by Cheon et. al. [2] follows the original r-adding walk closely. The only difference is, in the modified one the multiplication is done after l iterations and the iteration uses a table lookup. In the original r-adding walk multiplication is performed every iteration.

Let g, h and m_i be as defined earlier. We compute the table $\mathcal{M}_l$ as described above. Once that computation is done, we start the iterated walk. An intermediate step in the iteration looks like

$$Y' = Ym_{i_1}m_{i_2}\dots m_{i_k}.$$

Now we need to find $\gamma(Y') = i_{k+1}$ where γ is a index function from $\mathbb{F}_2^t$ to $\{1, 2, \dots, r\}$. The function γ was defined earlier. Assume that $Y = y_0 + y_1x + \dots + y_{\eta-1}x^{\eta-1}$, and we know $(i_1, i_2, \dots, i_k)$. The novel idea in the modified r-adding walk algorithm is, **we do not have to compute the product Y', to find** i_{k+1}. Notice that $(i_1, i_2, \dots, i_k)$ is the multiplier information in the table $\mathcal{M}_l$. Let us denote $m_{i_1}m_{i_2}\dots m_{i_k}$ by m. Let us warn the reader that this is just a notation to increase readability not the product of $m_{i_1}m_{i_2}\dots m_{i_k}$. Now we do a **table lookup** and find the cell containing $(i_1, i_2, \dots, i_k)$ as multiplier information. To that cell is attached the tag

$$\left(\tau(m), \tau\left(xm\right), \tau\left(x^2m\right), \dots, \tau\left(x^{\eta-1}m\right)\right).$$

Now notice that,

$$Y' = y_0 m + y_1 (xm) + y_2 \left(x^2 m\right) + \ldots + y_{\eta-1} \left(x^{\eta-1} m\right)$$

and from the additivity property of the tag function we have that

$$\begin{aligned} i_{k+1} = \gamma(Y') = \;& y_0 \tau(m) + y_1 \tau(xm) + y_2 \tau(x^2 m) + \\ & \ldots + y_{\eta-1} \tau(x^{\eta-1} m) \end{aligned} \tag{2.2}$$

It is clear that the tag of Y' can be computed without computing Y'. So now Y'' can be determined the same way Y' was determined. We can continue this process l times and then compute the product from the pre-computed group element in the table $\mathcal{M}_l$, that requires a table lookup. The full product is also computed when one reaches a distinguished point. However that is a rare event and we will totally ignore that.

Why is the modified r-adding walk faster?

In computing an iteration in the original r-adding walk, we need to do about η^2 multiplication in $\mathbb{F}_2$. We refer to Equation 2.2 to find the number of multiplications in $\mathbb{F}_2$ for a single iteration in the modified r-adding walk. Recall that $\tau(x^i m)$ is already computed in $\mathcal{M}_l$ for each i and is a vector of size t. So $\gamma(Y')$ can be computed in $t\eta$ multiplications in $\mathbb{F}_2$. Since t is significantly smaller than η, we have that $\eta^2 \gg t\eta$ and an iteration in the modified walk is faster than the original r-adding walk. Recall that at every l step there is a full product computation and assume that it takes η^2 multiplications. So to complete l steps, in the modified walk we need $lt\eta + \eta^2$ filed multiplications compared to $l\eta^2$ multiplication in the original walk. However, one has to do a table lookup as well and that can be time consuming.

2.4 Comparison of modified r-adding walk with the original r-adding walk

We now describe the original r-adding walk by Teske and its modification by Cheon et. al. in more details. Cheon et. al. described a novel idea of *distinguished path segment* to find the distinguished point. We start with that.

Definition 2.4.1 (Distinguished Path Segment). Let $(g_i)_{i\geq 0}$ be a random walk over a finite group G. Let $\gamma : G \longrightarrow \{1, 2, \ldots, r\}$ be an onto function. Fix a positive integer δ. For $i \geq \delta - 1$, the sequence $\{g_{i-\delta+1}, g_{i-\delta+2}, \ldots, g_i\}$ is called distinguished path segment if

$$\gamma(g_{i-\delta+1}) = \gamma(g_{i-\delta+2}) = \ldots = \gamma(g_i) = 1 \qquad \text{and} \qquad \gamma(g_{i-\delta}) \neq 1.$$

Cheon et. al. [2] show that the expected number of function iterations before the appearance of the first distinguished path segment is $\frac{r}{r-1}(r^\delta - 1)$. So, the probability of a sequence $\{g_{i-\delta+1}, g_{i-\delta+2}, \ldots, g_i\}$ to be a distinguished path segment is $\frac{r}{(r-1)(r^\delta-1)}$.

2.4.1 r-adding walk

In this section, we briefly discuss the r-adding walk to solve the discrete logarithm problem. Throughout this section $G = \langle g \rangle$ denotes a finite cyclic group of prime order and $h = g^x$ where x is the discrete logarithm.

Original r-adding walk requires an index function $\gamma : G \longrightarrow \{1, 2, \ldots, r\}$. For our experiment, we had considered $\gamma = \sigma \circ \tau$ where τ is as discussed in Section 3. We want σ to be a surjective function which is roughly pre-image uniform, i.e., the pre-image of each element is roughly the same size. Choose $t \approx log_2 r$ and assign an unique non-negative integer less than 2^t as an image of an element of $\mathbb{F}_{2^t}$ under the map σ. This makes γ pre-image uniform. As described in Algorithm 2.4.1, for each $i = 1, 2, \ldots, r$ choose non-zero integers $1 \leq \alpha_i, \beta_i < |G|$ and set multiplier m_i equal to $g^{\alpha_i} h^{\beta_i}$. The r-adding iterating function $\mathcal{F}$ is as discussed in Section 2. Start iteration from $Y = g^{\alpha_0}$ where α_0 is the random integer between 1 and $|G|$. Compute $\mathcal{F}(Y)$ and set $Y = \mathcal{F}(Y)$. Since $m_{\gamma(Y)}$ is of the form $g^{\alpha_{\gamma(Y)}} h^{\beta_{\gamma(Y)}}$, it is easy to keep track of the exponents of g and h. Fix some positive integer $\delta > 0$ and define current element Y as a distinguished point if it is the last element of a distinguished path segment of r-adding walk. The r-adding walk is travelled until the current element Y is found to be a distinguished point. Whenever a distinguished point is reached, the current element is searched for in the table of distinguished points and is added to the table if it is not found. When there is a collision among distinguished points, we can use (1.2) to find the unknown x.

As discussed in Section 2, for a randomly chosen iteration function, the expected rho length is $1.253\sqrt{q}$ where q is the order of the group. Since we are using distinguished points approach, one would expect to compute $1.253\sqrt{|G|} + \frac{r}{r-1}(r^\delta - 1)$ iterations until a collision detection. Let MAX be the maximum number of distinguished points stored. Since the number of iterations until the appearance of the first distinguished point is $\frac{r}{r-1}(r^\delta - 1)$, one should choose δ in such a way that MAX$\frac{r}{r-1}(r^\delta - 1) \approx 1.253\sqrt{|G|}$. At the same time one should keep in mind that $\frac{r}{r-1}(r^\delta - 1)$ has to be much less than $1.253\sqrt{|G|}$.

Algorithm 2.4.1 (Original r-adding Walk).
Input:

Field $\mathbb{F}_{2^n}$.

Subgroup G=$\langle g \rangle$.

An element h of the subgroup G.

Two positive integers r and t.

$\mathcal{M} = \{(m_i = g^{\alpha_i} h^{\beta_i}, \alpha_i, \beta_i) : 1 \leq i \leq r\}$

The main algorithm

```
1. Start with an empty table of distinguished points.
2. $(Y, \alpha, \beta) = (g^{\alpha_0}, \alpha_0, 0)$
3. While there is no duplicates among distinguished points
   do { $i = \gamma(\lfloor \frac{Y}{x^{n-t}} \rfloor)$.
        $(Y, \alpha, \beta) = (Y * m_i, \alpha + \alpha_{m_i}, \beta + \beta_{m_i})$.
        if Y is distinguish point
           then add $(Y, \alpha, \beta)$ to the table of distinguished points.
   Solve DLP using exponents $\alpha$ and $\beta$ of duplicate elements and Equation 2.
```

2.4.2 Modified r-adding walk

On each iteration, the original r-adding walk computes a field multiplication whereas modified r-adding walk does not. Instead it requires a table lookup where the size of the table is quite large and a tag computation. As described in Algorithm 4.2, one needs to compute the table $\mathcal{M}_l$ before starting the algorithm. For a large enough group, time required to compute this table $\mathcal{M}_l$ is negligible compared to the time required to solve the discrete logarithm problem.

Tag computation

We have described the concept of tag in details in Section 3.3. It is clear that faster the tag computation, faster the modified r-adding walk. In implementing our algorithms we are using Magma [1]. One of the reasons we choose magma is that polynomial arithmetic and finite field implementation is the fastest in this package. We tried three different methods for this tag computation. We discuss those methods briefly. Recall that $Y = y_0 + y_1 x + \ldots + y_{\eta-1} x^{\eta-1}$ and m is $m_{i_1} m_{I_2} \ldots m_{i_k}$. We are using m as a shorthand not the product. To compute $\tau(Ym)$, one needs to compute $y_0 \tau(m) + y_1 \tau(xm) + y_2 \tau(x^2 m) + \ldots + y_{\eta-1} \tau(x^{\eta-1} m)$. The table $\mathcal{M}_l$ contains $(\tau(m), \tau(xm), \ldots, \tau(x^{\eta-1} m))$. So we have to do this scalar multiplication and the addition.

Method 1 An obvious way to compute the tag is to loop over $y_i\tau(x^i m)$ from $i = 0 \ldots \eta - 1$. Advantage of this is that this will use polynomial arithmetic, which is fast, but the length of this loop will be equal η which is 1023 in our case. It turns out to be much slower than the later methods explained.

Method 2 Another method is to use the in-built inner product function in Magma. Let v be the vector of coefficients of the polynomial representation of $Y = y_0 + y_1 x + \ldots + y_{\eta-1}x^{\eta-1}$, i.e., $v = (y_0, y_1, \ldots, y_{\eta-1})$ and from $\mathcal{M}_l$ we obtain $w = (\gamma(x^0 m), \gamma(x^1 m), \ldots, \gamma(x^{\eta-1}m))$. Then the inner product of v and w is $\gamma(Ym)$. This method won't require us to define a loop and at the same time, the inner product computation will use polynomial arithmetic. However, there is a serious disadvantage to this method. To use inner product, both the vectors v and w have to be in the same vector space. Note that the coefficients of w are in $\mathbb{F}_{2^t}$, whereas the coefficients of v are in $\mathbb{F}_2$. Hence, we need to *coerce* (use an inbuilt embedding function in Magma) the vector v into the vector space of the dimension η over $\mathbb{F}_{2^t}$ and that makes it slower.

Method 3 The fastest tag computation that we could achieve was using t-many inner products instead of one. Again we were using the inbuilt Magma function for inner products. Recall that we need to compute $y_0\gamma(m) + y_1\gamma(xm) + y_2\gamma(x^2m) + \ldots + y_{\eta-1}\gamma(x^{\eta-1}m)$ and $\gamma(x^i m)$ is a binary vector of size t. The idea is to compute these t vectors in the sum independently. Each inner product has one input $(y_0, y_1, \ldots, y_{\eta-1})$ and the other a vector of size η of bits, where the i^{th} entry comes from $\gamma(x^i m)$. Which entry from $\gamma(x^i m)$ gets chosen is decided by a loop. The first iteration of the loop uses the first entry of each $\gamma(x^i m)$, the second the second entry from each $\gamma(x^i m)$, and so on, the last entry in the t^{th} iteration of the loop. So, each loop gives the corresponding entry in the sum which is a binary vector of size t. For this we had to write an external loop in our program which runs for t iterations. However, since t is small this method was the fastest among all that we tried and was implemented.

We keep computing the tag value in each iteration until the walk reaches a distinguished point or l consecutive iterations are performed. Whenever the walk reaches a distinguished point or l consecutive iterations are performed, one computes the full product Ym and set $Y = Ym$. Everything else is the same as in the original r-adding walk.

Here, the value of l determines the speed up factor. If we increase the value of l, the number of consecutive iterations without product computation increases. One extreme value of l is $\sqrt{|G|}$ where $|G|$ is the order of the group $G = \langle g \rangle$. However the size of the table $\mathcal{M}_l$ is $\binom{r+l}{r}$, increasing the value of l increases the size of the table. Pre-computation time and the time required for a table looklup on each iteration increases with that. One needs to choose a value of l in such a way that it balances the table lookup time and the storage availability.

Algorithm 2.4.2 (Modified r-adding Walk)**.**
Input:

Field $\mathbb{F}_{2^n}$

Subgroup G=$\langle g \rangle$

An element h of the subgroup G

Three positive elements r, l and t.

$\mathcal{M}_l$

The main algorithm

1. Start with an empty table of distinguished points.

2. $(Y, \alpha, \beta, v) = (g^{\alpha_0}, \alpha_0, 0, (a_0, a_1, \dots, a_{n-1}))$ where, $g^{\alpha_0} = a_0 + a_1 x + \dots + a_{n-1}x^{n-1}$.

3. **while** there are no duplicates among distinguished points

```
do {
    $i = 1$
    while $i \leq l$ and $Y$ is not distinguished point
    do {
        set $m = m m_{s_i} m_{s_{i-1}} \dots m_{s_0}$
        look up for the vectot $u = (\tau(m), \tau(xm), \tau(x^2 m), \dots,$
          $\tau(x^{n-1}m))$ in precomputed table $\mathcal{M}_l$.
        compute $ss := InnerProduct(v, u)$
        $s_i = \sigma(ss)$.
        $i = i + 1$.
    }
    $(Y, \alpha, \beta, v) = (Ym, \alpha + \alpha_m, \beta + \beta_m, (c_0, c_1, \dots, c_{n-1}))$
    where, $Ym = c_0 + c_1 x + c_2 x^2 + \dots + c_{n-1}x^{n-1}$.
    if Y is a distinguished point
    then add $(Y, \alpha, \beta)$ to the table of distinguished points.
}
```

Solve DLP using exponents α and β of duplicate elements and Equation (2).

2.4.3 Table lookup

In the modified r-adding walk, the multiplication in field $\mathbb{F}_{2^\eta}$ is replaced by few multiplications in $\mathbb{F}_2$ (tag computation) and a table lookup. The size of the table $\mathcal{M}_l$ is $\binom{l+r}{r}$. For $l = 10$ and $r = 4, 8, 16, 20$, the size of the table is $\binom{10+4}{4} \approx 2^{10}$, $\binom{10+8}{8} \approx 2^{15.4}$, $\binom{10+16}{16} \approx 2^{22.3}$ and $\binom{10+20}{20} \approx 2^{24.8}$, respectively. We can not ignore the time required for table lookup in the modified r-adding walk because the size of the table large. So during implementing the modified r-adding walk algorithm, we need to use the most efficient way for table lookup.

In general, the best method for table lookup is the binary search, but we came up with something even better. So, let us count the number of basic operations required for the table lookup in the binary search method.

Binary search method

Let, the size of the table be approximately equal to 2^w. Then on each iteration, binary search method will require w steps of vector comparisons. Since the length of the vector $(i_1, i_2, ..., i_k)$ varies from 1 to l, each vector comparison requires on an average $l/2$ integer comparisons. Hence, the binary search method requires total $lw/2$ many integer comparisons on each iteration. For large value of l and r, $lw/2$ is significantly large. For example, for $l = 10, r = 16$, $lw/2 \approx 110$. It takes a significant number of steps even though we are using the best known method. We tried many other approaches as well including the inbuilt index search algorithm in Magma. Finally, we developed our own algorithm using some pre-computation.

Pre-computation method

Note that we can always write an element m in the table $\mathcal{M}_l$ as $m_1^{a_1} m_2^{a_2} \dots m_r^{a_r}$. We can pre-compute the vector V of length $(l+1)^r$ whose element $V[a_1 * (l+1)^{r-1} + a_2 * (l+1)^{r-2} + \dots + a_r]$ store the position i of the element $m = m_1^{a_1} m_2^{a_2} \dots m_r^{a_r}$. Note that to find the position of $m = m_1^{a_1} m_2^{a_2} \dots m_r^{a_r}$, we need to look at $V[a_1 * (l+1)^{r-1} + a_2 * (l+1)^{r-2} + \dots + a_r]$. Let, $s = a_1 * (l+1)^{r-1} + a_2 * (l+1)^{r-2} + \dots + a_r$. For any given $1 \le i \le r$, to find the position of $m' = mm_i$, we need to look at $V[s + (l+1)^{r-i}]$. We can also store this s of previous multiplier m and $(l+1)^i$ for each $0 \le i \le (r-1)$. Hence, on each iteration, the table look up requires just one integer addition. We know that integer addition requires around N steps where N is the maximum of the number of digits of integers to be multiplied or added. For $l = 10, r = 16$, $\binom{l+r-1}{r} = \binom{25}{16} \approx 2^{20.9} \approx 10^6$. Hence, for $r = 16, l = 10$, this method requires around 6 basic steps on each iteration but, look at the size of the vector V. If we assume that each element of V consumes 1-bit of space, then for $r = 16, l = 10$, the vector v consumes 11^16 bits of space and 11^16 bits $> 5 \times 10^6$ GB which is a huge space. Hence, this method is feasible only for small value of r and we have used this method for $r = 4, l = 10$. We need to use different method for $r = 16, l = 10$. Let us describe the another method for large value of r.

Let, Y be the last fully computed element and $m = m_{i_1} m_{i_2} \dots m_{i_k}$ be the last multiple. The index of m in the table $\mathcal{M}_l$ is $\{i_1, i_2, \dots, i_k\}$ and we denote it by ind_m. Let $\gamma(Ym)$ be i_{k+1}. This means the next element on the walk is $Ymm_{i_{k+1}}$. Set $m' = mm_{i_{k+1}}$. We need to find the index of the m'. Note that given an element $m = m_{i_1} m_{i_2} \dots m_{i_k}$ and a multiplier $m_{i_{k+1}}$, there is an unique element $m' = mm_{i_{k+1}}$ in the table $\mathcal{M}_l$. We can store the index of m' for each possible pair (m, m_{k+1}). Note that if the multiple m is of the form $m_{i_1} m_{i_2} \dots m_{i_l}$, i.e., m is the combination of the l-many multipliers, then we compute the γ-value of the next element by computing full product. So, we are not considering pairs of the form (m, m_{k+1}) where $m = m_{i_1} m_{i_2} \dots m_{i_l}$. We are left with $\binom{l+r-1}{r}$ many multiples. For each multiple m, we have r many choice for m_{k+1}. Hence, total number of pairs of the form (m, m_{k+1}) is $r\binom{l+r-1}{r}$. We pre-compute a vector V of length $r\binom{l+r-1}{r}$. We divide V in r equal parts of length $\binom{l+r-1}{r}$ where each part corresponds to a particular multiplier m_{k+1}. For example, first $\binom{l+r-1}{r}$ many entries corresponds to the pairs of the form (m, m_1). If index of m is ind_m, then we store the index of $m' = mm_{k+1}$ in the table $\mathcal{M}_l$ as the $\left(k\binom{l+r-1}{r} + \text{ind}_\text{m}\right)^{\text{th}}$ entry of V. On each iteration, we can find the index of $m' = mm_{k+1}$ using the index of m and i_{k+1}. So the index of m' is $V[(s-1)\binom{l+r-1}{r} + \text{ind}_\text{m}]$ where ind_m is the index of m in the table $\mathcal{M}_l$. Note that we need to compute $\binom{l+r-1}{r}$ once because l and r are fixed. Hence, on each iteration this method requires one integer multiplication and one integer addition. We know that integer multiplication requires around N^2 steps and integer addition requires around N steps where N is the maximum of the number of digits of integers to be multiplied or added. For $l = 10, r = 16$, $\binom{l+r-1}{r} = \binom{25}{16} \approx 2^{20.9} \approx 10^6$. Hence, for $r = 16, l = 10$, this method requires around $6^2 + 6 = 42$ basic steps on each iteration. In Section 5.3.1, we have shown that the binary search method requires 110 basic steps for the same parameter. So, this method is faster than the binary search and we have used this method in our experiment for $r = 16, l = 10$.

2.4.4 Results of our experiments

Theoretically, it seems that the modified r-adding walk is faster than the original r-adding walk. To answer, how fast is the modified r-adding walk compared to original r-adding walk, we have tested both these algorithms on the prime order

Table 2.1: Average time required for 10^7 iterations of various methods on 47-bit prime order subgroup of $\mathbb{F}_{2^{1023}}$ and the average rho length as in Table 1.1.

r	l	t	time	ρ_r	$\frac{time \times \rho_r}{10^7}$
20, original	-	6	121.37	1.025	$\frac{124.404}{10^7}$
4, modified	10	2	20.68	1.341	$\frac{27.718}{10^7}$
16, modified	10	4	51.6	1.038	$\frac{53.415}{10^7}$

subgroup of the binary field $\mathbb{F}_{2^\eta}$.

Speed Comparison

We want to compare the time required for the original r-adding walk and the modified r-adding walk to solve the discrete logarithm problem in the same group. If we aim for practical parameters, i.e., DLP in a subgroup of size a 80-bit prime, then it won't be possible to solve the DLP. So, we have measured the speed of the first few iteration of the walk for large parameters. One can find the time required to solve a DLP for each walk in large group using the average speed of each iteration and the expected rho length. Let v be the speed of each iteration and L be the expected rho length, then the a DLP can be solved in time $v \times L$. As the results should not be biased toward modified r-adding walk, we have implemented both the method on same platform using Magma [1]. We use the binary field arithmetic functions from the Magma.

Let us explain the details of the experiment. We have selected a cyclic group $\langle g \rangle \subseteq \mathbb{F}_{2^{1023}}$ of order a 47-bit prime. All other parameters were chosen as described in Section 4. Both the algorithms are same as far as possible. We ran the 20-adding walk algorithm and the modified r-adding walk for different values of r and l. Timing was started after the full computation of the respective multiplication tables $\mathcal{M}$ and $\mathcal{M}_l$. For each parameter set, we ran 100 different DLP instances with 100 tests for each instance. We measured the time of the first 10^7 iterations in each case. Table 2.1 provides the time require for 10^7 iterations by both the algorithms. Here, we have excluded the pre-computation time for modified r-adding walk because time required for pre-computation is negligible compare to the time required to solve a discrete logarithm problem in the large group.

From Table 2.1, the average ratio to solve a DLP by original and modified r-adding walk with $r = 4$ is $\frac{124.404}{27.718} \approx 4.49$ and for $r = 16$ is $\frac{124.404}{53.415} \approx 2.33$.

2.5 Conclusion

Cheon et. al. [2] used binary field arithmetic functions from NTL library and measured the average time required by both the algorithm for 10^8 iterations on 206-bit prime order subgroup of the binary field $\mathbb{F}_{2^{1024}}$. For $r = 4, l = 10$, they found that modified r-adding walk is around 8.6 times faster than the original r-adding walk. Our results are different from their results. In our case, the original 20-adding walk took on an average 121.37 seconds for 10^7 iterations while in [2], the average time for original 20-adding walk is around 406 seconds per 10^8 iterations. In our implementation, there are three main steps on each iteration, namely, field multiplication, conversion of field element in to vector and tag computation. Out of 121.37 seconds, field multiplication and conversion of field element in to vector each consumes around 30 seconds and around 60 seconds consume by tag computation and distinguished point searching. If we ignore the γ function, then our time is around 40.4 seconds for 10^7 iterations means around 404 seconds for 10^8 iterations which is nearly equal to the respective claim in [2]. This proves that our implementation is as optimal as theirs. In our implementation, we have to do field element to vector conversion to access the coefficients of x^i in polynomial representation of the element but, in NTL library, it might be possible to access this coefficients directly from the field element. Even though our implementation in Magma are almost optimised, our findings are different than that of Cheon et. al. [2] because of the use of different platform for implementation.

There are some advantages and disadvantages for using $r = 4$ instead of $r = 16$ in the modified walk. An advantage of $r = 4$ is that the modified walk gives a speed-up of the factor of 4.49 and one can use large values of l compared to that of $r = 16$. The size of the table $\mathcal{M}_l$ is given by $\binom{l+r}{r}$. One disadvantage of $r = 4$ is that the 4-adding walk has large expected rho length and variance. This means that finding the collision 4-adding method is quite uncertain. An advantage of $r = 16$ is that the 16-adding walk has small expected rho length and variance. This means that finding the collision using 16-adding method is more certain compared to the 4-adding walk. One disadvantage of $r = 16$ is that the modified method gives a speed-up to the factor of 2.33 and one can not use large values of l. For example, with 4 GB memory, we were restricted to use $l \leq 10$ for $r = 16$ but we were able to use $l = 50$ for $r = 4$. Since both values of r have some advantage and disadvantage, on have to choose cautiously for practical purposes.

Chapter 3

Morified r-adding method over Elliptic Curve ($E(\mathbb{F}_p)$)

We got some speed up with modified r-adding method for attacks on DLP over finite field but, one can use Index calculus method, which is faster than r-adding method, to solve DLP over finite field.The index calculus method can not be use to attack DLP over elliptic curve over finite field [19] but, one can always use the r-adding walk because it is a generic. It is important to check whether this modified r-adding method can be use to attack DLP over elliptic curve over finite field. Hence, if we can use modified r-adding method over elliptic curve and find some speed up, then it would be significant. In this chapter, we have describe the elliptic curve version of modified r-adding method.

3.1 Group structure on rational points of an Elliptic curve

General equation of elliptic curve over some field K is given by

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6. \tag{3.1}$$

where, $a_1, a_2, a_3, a_4, a_6 \in K$. Equation 3.1 is called the general Weierstrass equation for elliptic curves. The elliptic curve E of the form 3.1 is called non-singular or smooth if for each (x, y) with coordinates in the algebraic closure of K, the partial derivatives $2y + a_1x + a_3$ and $3x^2 + 2a_2x + a_4 - a_1y$ do not vanish simultaneously. A point on the curve is called singular if both the partial derivative vanish at that point.

Let us now describe the addition law to turn the set of points of E over some field K into a group. Let P and Q be any two points of E. As shown in figure

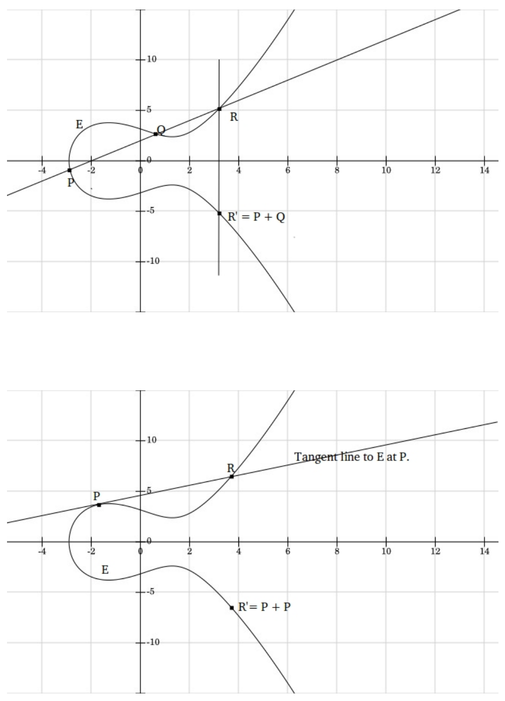

Figure 3.1: Addition law on elliptic curve

3.1, to add P and Q, draw a line connecting P and Q. This line intersects curve at third point R, say $R = (x_3, y_3)$. Using quadratic formula, it is easy to check that if (x_3, y_3) satisfies Equation 3.1, then $(x_3, -(a_1x_3 + a_3) - y_3)$ also satisfies Equation 3.1. Denote the point $(x_3, -(a_1x_3 + a_3) - y_3)$ by R' and set this R' as sum of P and Q so, $R' = P+Q$. In similar way, we can add P and P by replacing connecting line by a tangent to the curve E at P. For more details see [3, Chapter 13].

If the x-coordinates of P and Q are same then line connecting P and Q do not intersect E at any third point because given a *x*-coordinate, there are at most two points which satisfies Equation 3.1. We need to define sum of two points with same *x*-coordinate. Note that any line passing through two points with same *x*-coordinate is parallel to *y*-axis. Visualize a point lying far out on the *y*-axis such that any line parallel to *y*-axis passes through it. This point is denoted by $\mathcal{O}$, the point at infinity. One can check that if (x_1, y_1) lies on the curve E, then the point $(x_2, -(a_1x_1 + a_3) - y_1)$ also lies on the curve E. Let (x_1, y_1) and (x_1, y_2) are two points on E then define $(x_1, y_1) + (x_1, y_2) = \mathcal{O}$. One can check that $\mathcal{O}$ is the neutral element which means for any point P in E, $P + \mathcal{O} = P$. Hence, additive inverse of $P = (x_1, y_1)$ is given by $(x_1, -(a_1x_1 + a_3) - y_1)$ and is denoted by $-P$. We have defined addition law using geometry. The geometrical method of addition can be translated in to algebraic method using the theorem 3.1.1.

Theorem 3.1.1. Let, $E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ be an elliptic curve over K. Let, P and Q be two points on E. Then following statements are true.

1. If $P = \mathcal{O}$, then $P + Q = P$.
2. If $Q = \mathcal{O}$, then $P + Q = Q$.
3. If $P \neq \mathcal{O}$ and $Q \neq \mathcal{O}$, then write $P = (x_1, y_1)$ and $Q = (x_2, y_2)$.
   (a) If $x_1 = x_2$ and $y_1 \neq y_2$, then $P + Q = \mathcal{O}$.
   (b) else, define λ and c by

$$\lambda = \begin{cases} \frac{(y_2-y_1)}{(x_2-x_1)} & \text{if} \quad P \neq Q \\ \frac{3x_1^2+2a_2x_1+a_4-a_1y_1}{2y_1+a_1x_1+a_3} & \text{if} \quad P = Q \end{cases}$$

$$c = \begin{cases} \frac{y_1x_2-y_2x_1}{x_2-x_1} & \text{if} \quad P \neq Q \\ \frac{-x_1^3+a_4x_1+2a_6-a_3y}{2y_1+a_1x_1+a_3} & \text{if} \quad P = Q \end{cases}$$

and let $x_3 = \lambda^2 + a_1\lambda x - a_2 - x_1 - x_2$ and $y_3 = -(a_1 + \lambda)x_3 - c - a_3$
Then, $P + Q = (x_3, y_3)$.

We postpone the proof of the above theorem for a while and describe some important concepts. By definition of projective planes, $\mathbb{P}^2(K)$ consist all triples $(X : Y : Z)$ where not all X, Y, Z are zero and $(X : Y : Z)$ is equivalent to $(\lambda X : \lambda Y : \lambda Z)$ for all $\lambda \in K^*$. We choose the line $Z = 0$ as the line at infinity in $\mathbb{P}^2(K)$. Then we have two types of points in $\mathbb{P}^2(K)$, the points with $Z \neq 0$: $(X : Y : Z) = (x, y, 1)$ where $x = X/Z$ and $y = Y/Z$ and the points at infinity with $Z = 0$: $(X : Y : 0)$.

By substituting $x = X/Z$ and $y = Y/Z$ in Equation 3.1 and multiplying it by Z^3, we get the following weierstrass equation in projective coordinates:

$$E1 : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \qquad (3.2)$$

The point at infinity has coordinates $(X : Y : 0)$ and at the point of infinity above equation reduces to $X^3 = 0$ means $X = 0$. Hence, above equation has unique point at infinity which is $(0 : Y : 0)$. Note that in projective coordinates, the point $(0 : Y : 0)$ is equivalent to $(0 : 1 : 0)$ for all Y. Hence, we choose $(0 : 1 : 0)$ to represent the point of infinity. One can represents points on elliptic curve over K as points in projective coordinates as follows:

$$(x, y) = (X : Y : 1).$$

Lemma 3.1.2. The lines in the projective plane that intersect E at the point $\mathcal{O} = (0 : 1 : 0)$ at infinity are the lines of the form $x = c$ where $c \in K$.

Proof. The general equation of line in the plane is given by

$$L : \alpha y = \beta x + \gamma$$

where $\alpha, \beta, \gamma \in K$ with either α or β is non-zero.The equation of L in projective coordinates is given by

$$\alpha Y = \beta X + \gamma Z.$$

Now, $\mathcal{O} = (0 : 1 : 0)$ lies on the line L means $\alpha = 0$. Since $\alpha = 0$, β cannot be zero. Hence, equation for L in affine plane that meets E at $\mathcal{O}$ is of the form $x = c$. ●

PROOF OF THEOREM 3.1.1. (1) To find $\mathcal{O} + Q$, we need to draw a line passing through $\mathcal{O}$ and Q. Let L be the line passing through $\mathcal{O}$ and Q. Since $\mathcal{O}$ lies on the line L, from Lemma 3.1.2, the equation of the line is given by $L : x = c$ for some $c \in K$. Let, $Q = (x_2, y_2)$. Since Q also lies on the line L, the line L is given by $L : x = x_2$.

Now, there are at most two points on E with *x*-coordinate x_2. If one point is (x_2, y_2), then another point is (x_2, y_3) with $y_3 = -(a_1x_2 + a_3) - y_2$. Hence, third point of intersection of L with E is given by $R = (x_2, y_3)$. As describe in geometrical method, take the point $R' = (x_2, -(a_1x_2 + a_3) - y_3)$ and set $R' = \mathcal{O} + Q$. Since $y_3 = -(a_1x_2 + a_3) - y_2$, $R' = (x_2, y_2) = Q$. Hence, $\mathcal{O} + Q = Q$. This implies if $P = \mathcal{O}$, then $P + Q = Q$.

(2) Since line passing through P and Q is same as line passing through Q and P, $P + Q = Q + P$. Hence, if $Q = \mathcal{O}$, then we have $P + Q = P + \mathcal{O} = \mathcal{O} + P$. Using part (1) of this theorem, $\mathcal{O} + P = P$. Hence, if $Q = \mathcal{O}$, then $P + Q = P$.

(3) Let, $P \neq \mathcal{O}, Q \neq \mathcal{O}$ and $P = (x_1, y_1), Q = (x_2, y_2)$.
(a) If $x_1 = x_2$ and $y_1 \neq y_2$, then line passing through P and Q is $L : x = x_1$. We know that any line parallel to y-axis intersects E at $\mathcal{O}$ at infinity. Hence, third point of intersection of L with E is $\mathcal{O}$. Hence, $P + Q = \mathcal{O}$ and $-P = Q$.

(b) Assume $x_1 = x_2$ and $y_1 \neq y_2$ is not true. If $P \neq Q$, then the slope of the line passing through $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ is given by $\lambda = \frac{(y_2 - y_1)}{(x_2 - x_1)}$. If $P = Q$, then the connecting line is tangent to the curve E at P. Hence, slope of the line is the derivative of E with respect to x at P and is given by $\lambda = \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3}$. The line passing through P, Q is of the form $L : y = \lambda x + c$ for some $c \in K$. Now, we need to find third point of intersection of L with E.

CASE I: $P \neq Q$. By substituting $x = x_1$ and $y = y_1$ in the equation of line

L, we get $c = \frac{y_1x_2 - y_2x_1}{x_2 - x_1}$. Hence, the equation of the line L becomes

$$L : y = \frac{y_2 - y_1}{x_2 - x_1}x + \frac{y_1x_2 - y_2x_1}{x_2 - x_1}.$$

CASE II: $P = Q$. By substituting $x = x_1$ and $y = y_1$ in the equation of line L, we get $c = \frac{-x_1^3 + a_4x_1 + 2a_6 - a_3y}{2y_1 + a_1x_1 + a_3}$. Hence, the equation of the line L becomes

$$L : y = \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3}x + \frac{-x_1^3 + a_4x_1 + 2a_6 - a_3y}{2y_1 + a_1x_1 + a_3}.$$

Now substituting $y = \lambda x + c$ with respective λ and c in the equation of E, we get the cubic equation in the variable x,

$$x^3 + (a_2 - \lambda^2 - a_1\lambda)x^2 + (a_4 - 2\lambda c - a_1c - a_3\lambda)x + (a_6 - c^2 - a_3c) = 0 \quad (3.3)$$

Since line L intersects E at three points, above cubic equation has exactly three roots. we know the two roots, x_1 and x_2. Let third root by x_3. We know that sum of the roots of cubic equation is given by the negative of the coefficient of x^2 in the cubic equation. Hence, $x_1 + x_2 + x_3 = a_2 - \lambda^2 - a_1\lambda$. This gives

$$x_3 = \lambda^2 + a_1\lambda x - a_2 - x_1 - x_2.$$

By substituting $x = x_3$ in the equation of L, we get

$$y = \lambda x_3 + c.$$

Hence, third point of intersection of L and E is (x_3, y). Now, as described in geometrical method, $P + Q = (x_3, y_3)$ where $y_3 = -(a_1x_3 + a_3) - y$. Since $y = \lambda x_3 + c$, we get $y_3 = -(a_1 + \lambda)x_3 - c - a_3$. ●

Notation: For any positive integer k and a point P on E, we denote the k many addition of P by kP. This is also known as scalar multiplication.

If the characteristic of the field K is not equal to 2, then one can get simpler form of E by some basic transformation. By replacing y by $(y - \frac{(a_1x+a_3)}{2})$ in Equation 3.1, we get

$$E : y^2 = x^3 + b_1x^2 + b_2x + b_3 \quad (3.4)$$

where

$$b_1 = \frac{(a_1^2 + 4a_2)}{4}, b_2 = \frac{a_1a_3 + 2a_4}{2}, b_3 = \frac{a_3^2 + 4a_6}{4}.$$

If the characteristic of the field K is not equal to 2 and 3, then one can get even simpler form of E. By replacing x by $(x + \frac{b_2}{12})$ in Equation 3.4, we get

$$E : y^2 = x^3 + Ax + B \tag{3.5}$$

where $A = -\frac{b_2^2 - 24b_4}{48}$ and $B = -\frac{-b_2^3 + 36b_2b_4 + -216b_6}{864}$. It is easy to work with the elliptic curve of the above form. Hence, now onwards we assume characteristic of K is not equal to 2 and 3 and $y = x^3 + Ax + B$ as equation for elliptic curve over K.

Definition 3.1.1 (m-torsion set)**.** Let, $m \geq 1$ be an integer. The set of points elliptic curve of order m on elliptic curve is called the m-torsion set. We denote the m-torsion set by $E[m] = \{P \in E : mP = \mathcal{O}\}$.

Note that if $mP = \mathcal{O}$ and $mQ = \mathcal{O}$, then $m(P + Q) = \mathcal{O}$ and $m(-P) = \mathcal{O}$. Hence, m-torsion set $E[m]$ is a subgroup of E. If elliptic curve is defined over some field K, then we write the m-torsion set as $E(K)[m]$.

Definition 3.1.2 (ECDLP)**.** Let, p be a prime and P be a point in $E(\mathbb{F}_p)$. Let $G = \langle P \rangle$. Then G is the cyclic subgroup of $E(\mathbb{F}_p)$ generated by P. Given a point Q of the subgroup G, the ***Elliptic Curve Discrete Logarithm Problem***(ECDLP) is the problem of finding the smallest positive integer n such that $Q = nP$.

We want to solve ECDLP using modified r-adding method. Note that modified r-adding is based on the linearity of the tag function defined in Chapter 3. We know that there are bilinear maps defined over $E(\mathbb{F}_p)$ but, image of these bilinear maps lies in extension field. The idea is to use this bilinear map with some pre-computation table and then to use tag function defined in Chapter 3 for converting an image of bilinear map into an element of the prime field. Before discussing about attacks on ECDLP, let us describes bilinear maps.

3.2 Bilinear maps

3.2.1 Rational functions and divisor

A rational function is a ratio of polynomials

$$f(x) = \frac{a_0 + a_1x + \ldots + a_mx^m}{b_0 + b_1x + \ldots + b_nx^n}.$$

As is customary in algebraic geometry, we assume $m = n$. We know that any polynomial can be factored completely over the complex numbers. So, one can write $f(x)$ as

$$f(x) = \frac{a_0(x - \alpha_1)^{e_1}(x - \alpha_2)^{e_2} \ldots (x - \alpha_r)^{e_r}}{b_0(x - \beta_1)^{d_1}(x - \beta_2)^{d_2} \ldots (x - \beta_r)^{d_s}}$$

for some $\alpha_1, \alpha_2, \ldots, \alpha_r, \beta_1, \beta_2, \ldots, \beta_s$ distinct complex numbers. The zeros of the rational function are all those points at which the rational function vanishes and the poles of the rational function are all those points at which value of the function is infinite. Hence, the numbers $\alpha_1, \alpha_2, \ldots, \alpha_r$ are the zeros of $f(x)$ and $\beta_1, \beta_2, \ldots, \beta_s$ are the poles of $f(x)$.

In general, a divisor is a finite linear combination of points on the surface with integer coefficients. Now, define the divisor of $f(x)$, $div(f(x))$, to be the formal sum

$$div(f(x)) = e_1[\alpha_1] + \ldots + e_r[\alpha_r] - d_1[\beta_1] - \ldots - d_s[\beta_s]$$

One can check that if f, g, h are rational functions such that $h = fg$, then $div(h) = div(f) + div(g)$. Also, $div(1/f) = -div(f)$. Hence, divisor of rational functions forms a group under this addition operation.

If $E : y^2 = x^3 + Ax + B$ is an elliptic curve and $f(x, y)$ is a rational function of two variables, then divisor of f can be written as

$$div(f) = \sum_{P \in E} n_P[P]$$

where, coefficients n_P are integers and only finitely many of the n_P are non-zero. If n_P is positive, then f has zero of order n_P at $P \in E$ and if n_P is negative, then f has pole of order n_P at $P \in E$. The degree of the divisor is the sum of its coefficients i.e. $deg(div(f)) = \sum_{P \in E} n_P$ and the sum of the divisor is given by $Sum(div(f)) = \sum_{P \in E} n_P P$. Note that if D_1 and D_2 are two divisors, then $deg(D_1 + D_2) = deg(D_1) + deg(D_2)$ and $Sum(D_1 + D_2) = Sum(D_1) + Sum(D_2)$.

A divisor is called principal divisor if its degree is zero. Since $deg(D_1 + D_2) = deg(D_1) + deg(D_2)$ and $deg(-D_1) = -deg(D_1)$, if D_1 and D_2 are principal divisors then $D_1 + D_2$ is a principal divisor and $-D_1$ is also a principal divisor. Hence, all principal divisors of rational functions forms a subgroup of the divisor group.

Theorem 3.2.1. Let D= $\sum_{P \in E} n_P[P]$ be a divisor on E. Then D is the divisor of a rational function on E if and only if deg(D)= 0 and sum(D)= $\mathcal{O}$.

For a proof of this theorem see [18, Corollary 3.5]. According to the above theorem, **all rational functions on E has principal divisors** and hence, the degree of polynomial in numerator of the rational function is same as the degree of polynomial in denominator of the rational function. Let us now describe the algorithm to generate a rational function with divisor $(\eta[P] - [\eta P] - (\eta - 1)[\mathcal{O}])$ for any given $P \in E$ and an integer η.

3.2.2 Miller's algorithm

Victor Miller [7] gave an algorithm to quickly create rational functions of specified divisors. Let E be an elliptic curve. Let $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ be any non-zero points on E. Define a rational function $g_{P,Q}$ as follows:

$$g_{P,Q} = \begin{cases} \frac{y - y_P - \lambda(x - x_P)}{x + x_P + x_Q - \lambda^2} & \text{if} \quad \lambda \neq \infty \\ x - x_P & \text{if} \quad \lambda = \infty \end{cases}$$

where λ is the slope of the line connecting P and Q or the slope of the tangent to the elliptic curve E if $P = Q$. Now, let us define Miller's algorithm which takes a point P of elliptic curve E and an integer m and gives a rational function f with div(f) $= \eta[P] - [\eta P] - (\eta - 1)[\mathcal{O}]$.

Algorithm 3.2.2 (Miller's algorithm)**.**
Input:

An integer $\eta = \eta_0 + \eta_1 2 + \eta_2 2^2 + \ldots + \eta_{\zeta-1} 2^{\zeta-1}$ where $\eta_i \in 0, 1$ for all $0 \leq i < \zeta$.

An elliptic curve E

A point P on E

The main algorithm

```
1. Set $T = P$, $f = 1$ and $i = \zeta - 2$.
2. while $i \geq 0$
   do
      Set $f = f^2.g_{T,T}$
      Set $T = 2T$
      if $m_i = 1$
      then
         Set $f = f.g_{T,P}$
         Set $T = T + P$
      $i = i - 1$
3. Return $f$.
```

Theorem 3.2.3. Let E be an elliptic curve. Let $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ be any non-zero points on E. Then

1. div($g_{P,Q}$)$= [P] + [Q] - [P + Q] - [\mathcal{O}]$.

2. Let $m \geq 1$ and let η as the binary expression is

$$\eta = \eta_0 + \eta_1 2 + \eta_2 2^2 + \ldots + \eta_{\zeta-1} 2^{\zeta-1}$$

with $\eta_i \in 0, 1$ and $\eta_{\zeta-1} \neq 0$. Then miller's algorithm returns a rational function f_P with

$$div(f_P) = \eta[P] - [\eta P] - (\eta - 1)[\mathcal{O}].$$

For proof see [6, Chapter 5]. If we have a point P of order η ($\eta P = \mathcal{O}$), then Miller's algorithm returns a rational function f_P with div(f_P)= $\eta[P] - [\eta P] - (\eta - 1)[\mathcal{O}] = \eta[P] - \eta[\mathcal{O}]$.

3.2.3 Bilinear Pairing on elliptic curve

Definition 3.2.1 (Bilinear map)**.** Let, V_1, V_2, V_3 be three vector spaces. A bilinear map is a function B from product of two vector spaces, $V_1 \times V_2$ to the third vector space, V_3 with the following property:

$$\begin{aligned} B(\alpha_1 v_1 + \alpha_2 v_2, \beta_1 w_1 + \beta_2 w_2) &= \alpha_1 B(v_1, \beta_1 w_1 + \beta_2 w_2) + \alpha_2 B(v_2, \beta_1 w_1 + \beta_2 w_2) \\ &= \alpha_1\beta_1 B(v_1, w_1) + \alpha_1\beta_2 B(v_1, w_2) + \alpha_2\beta_1 B(v_2, w_1) + \alpha_2\beta_2 B(v_2, w_2) \end{aligned}$$

for all $v_1, v_2 \in V_1$, $w_1, w_2 \in V_2$ and $\alpha_1, \alpha_2, \beta_1, \beta_2$ in base field.

The Weil pairing that we discuss in this section takes two points of elliptic curve as inputs and gives a field element as output. However, the bilinarity condition in Weil pairing is slightly different where additivity in output is replaced by the multiplicity.

Definition 3.2.2 (The Weil pairing)**.** Let, $P, Q \in E[\eta]$ and f_P, f_Q be rational functions on E such that $div(f_P) = \eta[P] - \eta[\mathcal{O}]$ and $div(f_Q) = \eta[Q] - \eta[\mathcal{O}]$. The Weil pairing of P, Q is defined as

$$e_\eta(P, Q) = \frac{\frac{f_P(Q+S)}{f_P(S)}}{\frac{f_Q(P+(-S))}{f_Q(-S)}}$$

where $S \in E$ but, $S \notin \{\mathcal{O}, P, -Q, P - Q\}$. This Weil pairing has many useful properties.

Theorem 3.2.4. 1. For all $P, Q \in E[\eta]$, $e_\eta(P, Q)^\eta = 1$. This means $e_\eta(P, Q)$ is an η^{th} root of unity.

2. (Bilinear property)
   Let, $P, Q, R \in E[\eta]$. Then $e_\eta(P+Q, R) = e_\eta(P, R)e_\eta(Q, R)$ and $e_\eta(P, Q+R) = e_\eta(P, Q)e_\eta(P, R)$.

3. For all $P \in E[\eta]$, $e_\eta(P, P) = 1$. This implies that $e_\eta(P, Q) = e_\eta(Q, P)^{-1}$ for all $P, Q \in E[\eta]$.

4. (Non-degeneracy) If $e_\eta(P, Q) = 1$ for all $Q \in E[\eta]$, then $P = \mathcal{O}$.

Note that if $Q = nP$ for some positive integer n, then we have $e_\eta(Q, P) = e_\eta(nP, P) = e_\eta(P, P)^n = 1^n = 1$. Hence, Weil pairing is useful only if both input points comes from different cyclic subgroup of same order.

Proposition 3.2.5. Let $\eta \geq 1$ be an integer.Then

a. If $K = \mathbb{Q}$ or $\mathbb{R}$ or $\mathbb{C}$, then $E(K)[\eta] \cong \mathbb{Z}/\eta\mathbb{Z} \times \mathbb{Z}/\eta\mathbb{Z}$ is a product of two cyclic groups of order η.

b. If $K = \mathbb{F}_p$ and p doesn't divide η, then there exist a value k such that $E(\mathbb{F}_{p^{jk}})[\eta] \cong \mathbb{Z}/\eta\mathbb{Z} \times \mathbb{Z}/\eta\mathbb{Z}$ for all $j \geq 1$. The smallest such k is called the embedding degree of $E(\mathbb{F}_p)$ with respect to η.

Hence, whenever we need to use Weil pairing over $E(\mathbb{F}_p)[\eta]$, we first define it over $E(\mathbb{F}_{p^k})[\eta]$ where k is the embedding degree of $E(\mathbb{F}_p)$ with respect to η. The question is how to find this embedding degree k for a given η and the elliptic curve over $\mathbb{F}_p$. The following proposition describe the method to find this k for different possible cases.

Proposition 3.2.6. Let, E be an elliptic curve over $\mathbb{F}_p$ and η be a prime not equal to p such that $E(\mathbb{F}_p)$ contains a point of order η. Then one of the following statement is true regarding the embedding degree of E with respect to η.

1. The embedding degree is 1. This are called anomalous curves and DLP in these curves is trivial [12, 15, 16].

2. $p \equiv 1 \pmod{\eta}$ and the embedding degree is η.

3. $p \not\equiv 1 \pmod{\eta}$ and the embedding degree is the smallest value of $k \geq 2$ such that $p^k \equiv 1 \pmod{\eta}$.

For proof see [22]. The main application of the Weil pairing is to reduce ECDLP in $E(\mathbb{F}_p)$ to the DLP in $\mathbb{F}_{p^k}$ where k is the embedding degree. The algorithm of Menezes, Okamoto and Vanstone(MOV algorithm) solves ECDLP over $E(\mathbb{F}_p)$ by reducing ECDLP to DLP in $\mathbb{F}_{p^k}$ using Weil pairing. Let us describe the MOV attack.

MOV algorithm

Given an elliptic curve E over $\mathbb{F}_p$, a point $P \in E(\mathbb{F}_p)$ of order η and $Q \in E(\mathbb{F}_p)$ such that $Q = nP$ for some integer n, our aim is to find this integer n. First compute the embedding degree k of E with respect to η and then compute $N = \#(E(\mathbb{F}_{p^k}))$. Since there is a point $P \in E(\mathbb{F}_p)$ of order η and $E(\mathbb{F}_p) \subset E(\mathbb{F}_{p^k})$, $\eta | N$.

Now, choose a random point $T \in E(\mathbb{F}_{p^k})$ but not in $E(\mathbb{F}_p)$ and computes $T' = (N/\eta)T$. Check whether T' is equal to $\mathcal{O}$. If $T' = \mathcal{O}$, then choose a random point T again. If $T' \neq \mathcal{O}$, then the order of T' is η and there isn't any integer x such that $T' = xP$.

Now, we have two points from different cyclic subgroups of order m. Compute $\alpha = e_\eta(P, T') \in E(\mathbb{F}_{p^k})$ and $\beta = e_\eta(Q, T') \in E(\mathbb{F}_{p^k})$. Using bilinear property of Weil pairing, we have $\beta = e_\eta(Q, T') = e_\eta(nP, T') = e_\eta(P, T')^n = \alpha^n$. Hence, solving $Q = nP$ for n is reduced to solving $\beta = \alpha^n$ in $\mathbb{F}_{p^k}$. If p^k is not large then $\beta = \alpha^n$ can be solved using index calculus method which is a sub-exponential algorithm and if p^k is large then we need to use r-adding method which is an exponential algorithm. As in [6], the MOV algorithm has exponential time complexity if $k > (ln(p))^2$. Since most of the time a randomly chosen elliptic curve over $\mathbb{F}_p$ has embedding degree that is much larger than $(ln(p))^2$, the MOV algorithm is not much useful. In next section, we have describe the modified r-adding method to solve ECDLP over $E(\mathbb{F}_p)$ which might be faster than the original r-adding method. We have to define only tag function related to elliptic curve, everything else remains same as in modified r-adding method described in previous chapter.

3.3 The Tag function for the modified r-adding walk over elliptic curve

Let, p, η be two primes and P be a point in $E(\mathbb{F}_p)$ such that $\eta P = \mathcal{O}$. Let $G = \langle P \rangle$. Then G is the cyclic subgroup of $E(\mathbb{F}_p)$ generated by P. Let, $Q \in G$ be a point such that $Q = xP$ for some positive integer x. Our aim is to find this x using modified r-adding method. The main idea of the modified r-adding method is to compute full product after every l iterations only. Hence, we need to find a function τ_1 over cyclic subgroup $E[\eta]$ such that given $P, Q \in E[\eta]$, one can compute $\tau_1(P + Q)$ without computing $P + Q$.

3.3.1 The tag function

The tag of a point on the curve E is a vector consisting of first few significant coefficients of the image under the Weil map. Let, E and P be as defined above. Let, k be the embedding degree of $E(\mathbb{F}_p)$ with respect to η. Compute the number of points $N = \#(E(\mathbb{F}_{p^k}))$. If k is very small, then it is easy to compute the number of points using SEA algorithm [14]. Since there is a point of order η in $E(\mathbb{F}_p)$ and $E(\mathbb{F}_p) \subset E(\mathbb{F}_{p^k})$, $\eta | N$. Now, choose a random point $T \in E(\mathbb{F}_{p^k})$ with $T \notin E(\mathbb{F}_p)$. Compute $T' = (N/\eta)T$. If $T' = \mathcal{O}$, then choose another T until you get $T' \neq \mathcal{O}$. Note that this T' is the point of order η. Now, define a tag function

$$\tau_1 : E(\mathbb{F}_p)[\eta] \longrightarrow \mathbb{F}_{p^k}$$

as $\tau_1(P) = e_\eta(T, P)$ for all $P \in E(\mathbb{F}_p)[n]$. Since w_η is a bilinear map, for any $P_1, P_2 \in E(\mathbb{F}_p)[n]$, $\tau_1(P_1 + P_2) = e_\eta(T, P_1 + P_2) = e_\eta(T, P_1)e_\eta(T, P_2) = \tau_1(P_1)\tau_1(P_2)$. Hence, we can compute tag value of sum of two points in $E(\mathbb{F}_p)[\eta]$ without computing their sum. Now, suppose at some step in modified r-method we need to compute the tag of $P + m_{i_1} + m_{i_2} + \ldots + m_{i_k}$. Assume that $e_\eta(T, P)$ and $e_\eta(T, m_{i_1} + m_{i_2} + \ldots + m_{i_k})$ is known. Then we can compute $\tau_1(P + m_{i_1} + m_{i_2} + \ldots + m_{i_k})$ using the formula:

$$\tau_1(P + m_{i_1} + m_{i_2} + \ldots + m_{i_k}) = e_\eta(T, P)e_\eta(T, P + m_{i_1} + m_{i_2} + \ldots + m_{i_k}).$$

Hence, on each iteration we need to compute one multiplication in the finite field $\mathbb{F}_{p^k}$. There is a way to replace this field multiplication by few multiplication in $\mathbb{F}_p$. We will discuss this in later part of this section.

For given values of r and l, the structure of the table $\mathcal{M}'_l$ is same as described in Section 2.2. Only the information regarding tag is different. Note that $e_\eta(T, m)$ is an element in $\mathbb{F}_{p^k}$ and we have already defined a function $\tau : \mathbb{F}_{p^k} \longrightarrow \mathbb{F}_{p^t}$ where $t \approx \log_p r$. Hence, the vector $(\tau(x^0 e_\eta(T, m)), \tau(x e_\eta(T, m)), \ldots, \tau(x^{k-1} e_\eta(T, m)))$ is stored as tag information of m.

3.3.2 The modified r-adding function

We need to first define the index function $\gamma_1 : E(\mathbb{F}_p)[\eta] \longrightarrow \{1, 2, \ldots, r\}$. We have already defined a function $\tau_1 : E(\mathbb{F}_p)[\eta] \longrightarrow \mathbb{F}_{p^k}$. Now consider the function $\gamma : \mathbb{F}_{p^k} \longrightarrow \{1, 2, \ldots, r\}$ as described in Chapter 2. Then, we can define $\gamma_1 = \gamma \circ \tau_1$.

Let P, Q be as defined earlier. Let m_i, $1 \leq i \leq r$, be randomly chosen points of $E(\mathbb{F}_p)[\eta]$ of the form $\alpha_i P + \beta_i Q$ for some $1 \leq \alpha_i, \beta_i \leq \eta$. We compute the table $\mathcal{M}'_l$ as described above. Once that computation is done, we start the iterated walk. A modified r-adding walk $\mathcal{F}'$ is defined iteratively as follows:

$$\mathcal{F}'(Y) = Y + m_{\gamma_1(Y)}. \qquad (3.6)$$

An intermediate step in the iteration looks like

$$Y' = Y + m_{i_1} + m_{i_2} + \ldots + m_{i_k}.$$

Now we need to find $\gamma_1(Y') = i_{k+1}$ where γ_1 is a index function from $E(\mathbb{F}_p)[\eta]$ to $\{1, 2, \ldots, r\}$. Assume that we know $(i_1, i_2, \ldots, i_k)$. The novel idea in the modified r-adding walk algorithm is, **we do not have to compute the product Y', to find i_{k+1}**. Notice that $(i_1, i_2, \ldots, i_k)$ is the multiplier information in the table $\mathcal{M}_l$. Let us denote $m_{i_1} + m_{i_2} + \ldots + m_{i_k}$ by m. Now we do a **table lookup** and find the cell containing $(i_1, i_2, \ldots, i_k)$ as multiplier information. To that cell is attached the tag information $(\tau(x^0 e_\eta(T, m)), \tau(x e_\eta(T, m)), \ldots, \tau(x^{k-1} e_\eta(T, m)))$. Let, $e_\eta(T, Y) = y_0 + y_1 x + \ldots + y_{k-1} x^{k-1}$ From the linearity of the Weil map we have that

$$\tau_1 (Y') = \tau_1 (Y) \tau_1 (m) = e_\eta(T, Y) e_\eta(T, m) \tag{3.7}$$

Hence, from the additivity property of the tag function we have that

$$\begin{aligned}(\tau \circ \tau_1)(Y') = \; & y_0 \tau \left(e_\eta(T, m)\right) + y_1 \tau \left(x e_\eta(T, m)\right) + y_2 \tau \left(x^2 e_\eta(T, m)\right) + \\ & \ldots + y_{k-1} \tau \left(x^{k-1} e_\eta(T, m)\right)\end{aligned} \tag{3.8}$$

We can get $i_{k+1} = \sigma((\tau \circ \tau_1)(Y'))$ without computing Y'. Now, it is clear that the index of Y' can be computed without computing Y'. So now Y'' can be determined the same way Y' was determined. We can continue this process l times and then compute the product from the pre-computed group element in the table $\mathcal{M}_l$, that requires a table lookup. The full product is also computed when one reaches a distinguished point. However that is a rare event and we will ignore that in our analysis.

Algorithm 3.3.1 (Modified r-adding Walk over Elliptic curve)**.**
Input:

Elliptic curve over $E(\mathbb{F}_p)$.

Subgroup G=$\langle P \rangle$ and $\eta = \#(G)$.

Embedding degree k of $E(\mathbb{F}_p)$ with respect to η.

An element Q of the subgroup G.

An element $T \in E(\mathbb{F}_{p^k})$ but not in $E(\mathbb{F}_p)$ of order η.

Three positive elements r, l and t.

$\mathcal{M}'_l$

The main algorithm

1. Start with an empty table of distinguished points.

2. $(Y, \alpha, \beta, v) = (\alpha_0 P, \alpha_0, 0, (a_0, a_1, \dots, a_{n-1}))$ where, $e_\eta(T, \alpha_0 P) = a_0 + a_1 x + \dots + a_{k-1}x^{k-1}$.

3. **while** there are no duplicates among distinguished points

```
do {
    $i = 1$
    while $i \leq l$ and $Y$ is not distinguished point
    do {
        set $m = m + m_{s_i} + m_{s_{i-1}} + \dots + m_{s_0}$
        look up for the vectot $u = (\tau(e_\eta(T, m)), \tau(xe_\eta(T, m)),$
         $\tau(x^2 e_\eta(T, m)), \dots, \tau(x^{k-1}e_\eta(T, m)))$ in precomputed
        table $\mathcal{M}'_l$.
        compute $ss := InnerProduct(v, u)$.
        $s_i = \sigma(ss)$.
        $i = i + 1$.
    }
    $(Y, \alpha, \beta, v) = (Ym, \alpha + \alpha_m, \beta + \beta_m, (c_0, c_1, \dots, c_{k-1}))$
    where, $e_\eta(T, Ym) = c_0 + c_1 x + c_2 x^2 + \dots + c_{k-1}x^{k-1}$.
    if Y is a distinguished point
    then add $(Y, \alpha, \beta)$ to the table of distinguished points.
}
```

Solve DLP using exponents α and β of duplicate elements and Equation (2).

In original r-adding walk over elliptic curve, we need to compute one addition in $E(\mathbb{F}_p)$. If we use the modified r-adding walk as described above, then this addition is replaced by one table look up and either $k * t$ many multiplication in $\mathbb{F}_p$. The modified r-adding walk will be faster than original r-adding walk over elliptic curve if the time require by computation of one addition in $E(\mathbb{F}_p)$ is more than the time require by either kt many multiplication in in $\mathbb{F}_p$. One need to check this by implementing both the method.

Bibliography

[1] Wieb Bosma, John Cannon, and Catherine Playoust, *The Magma algebra system. I. The user language*, J. Symbolic Comput. **24** (1997), no. 3-4, 235–265.

[2] Jung Hee Cheon, Jin Hong, and Minkyu Kin, *Accelarating pollard's rho on finite fields*, Journal of Cryptology **25** (2012), no. 2, 195–242.

[3] H. Cohen and G. et.al. Frey, *Handbook of elliptic and hyperelliptic curve cryptography*, Chapman & Hall/CRC, Boca Raton, 2006.

[4] Bernard Harris, *Probability distribution related to random mapping*, Annals of Mathematical Statistics **31** (1960), 1045–1062.

[5] William B. Hart, *Fast library for number theory: An introduction*, Mahtematical Software - ICMS 2010, Third International Congress on Mathematical Software, Kobe, Japan, September 13-17, 2010. Proceedings, LNCS, vol. 6327, 2010, pp. 88–91.

[6] J. Hoffstein, J. Pipher, and J. H. Silverman, *An introduction to mathematical cryptography*, Springer, New York, 2008.

[7] Victor S. Miller, *The weil pairing, and its efficient calculation*, Journal of Cryptology **17** (2004), no. 4, 235–261 (English).

[8] V.I. Nechaev, *Complexity of a determinate algorithm for the discrete logarithm*, Mathematical Notes **55** (1994), no. 2, 165–172.

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

[10] R Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, 2012, ISBN 3-900051-07-0.

[11] Sheldon Ross, *A random graph*, Journal of applied probability **18** (1981), 309–315.

[12] T. Satoh and K. Araki, *Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves*, comment. Math. Univ. St. Paul **47** (1998), 81–92 (English).

[13] Eric Schmutz, *Period lengths for iterated functions*, Combinatorics, Probability and Computing **20** (2011), 289–298.

[14] Ren Schoof, *Elliptic curves over finite fields and the computation of square roots* $\bmod p$, Mathematics of Computation **44** (1985), no. 170, pp. 483–494 (English).

[15] I. A. Semaev, *Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p*, Mathematics of Computation **67** (1998), no. 221, pp. 353–356 (English).

[16] I. A. Semaev, *Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p*, Mathematics of Computation **67** (1998), 353–356.

[17] Victor Shoup, *Lower bounds for discrete logarithms and related problems*, EUROCRYPT '97, LNCS, vol. 1233, 1997, pp. 256–266.

[18] J. H. Silverman, *The arithmetic of elliptic curves*, vol. 106, Springer-Verlag, New York, 1986.

[19] J. H. Silverman and Joe Suzuki, *Elliptic curve discrete logarithms and the index calculus*, Advances in Cryptology ASIACRYPT98 (Kazuo Ohta and Dingyi Pei, eds.), Lecture Notes in Computer Science, vol. 1514, Springer Berlin Heidelberg, 1998, pp. 110–125 (English).

[20] Edlyn Teske, *Speeding up Pollard's rho method for computing discrete logarithm*, Algorithmic Number Theory Symposium, LNCS, vol. 1423, 1998, pp. 541–553.

[21] ———, *On random walks for Pollard's rho method*, Mathematics of Computation **70** (2000), no. 234, 809–825.

[22] L. C. Washington, *Elliptic curves: Number theory and cryptography*, Chapman & Hall/CRC, 2003.

[23] M.B. Wilk and R. Gnanadesikan, *Plotting methods for analysis of data*, Biometrika **55** (1968), no. 1, 1–17.