Perbandingan Penggunaan Pollard-rho dan Pollard-kangaroo Method dalam penyelesaian masalah Discrete Log Problem pada Elliptic Curves

Muhammad Gilang Ramadhan and 13520137[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13520137@std.stei.itb.ac.id

***Abstrak*—Makalah ini bertujuan untuk membandingkan efektivitas algoritma *Pollard-rho* dan *Pollard-kangaroo* dalam menyelesaikan masalah *Discrete Log Problem* (DLP) pada kurva eliptik. *Pollard-rho* dikenal dengan pendekatannya yang probabilistik dan efisien dalam menemukan logaritma diskrit pada grup siklik. Sementara itu, *Pollard-kangaroo* (juga dikenal sebagai algoritma lambda) dirancang khusus untuk menyelesaikan DLP ketika nilai logaritma yang dicari berada dalam rentang yang diketahui. Pada makalah ini, dilakukan eksperimen untuk mengukur waktu komputasi, penggunaan memori, dan tingkat efektivitas. Hasil menunjukkan bahwa *Pollard-rho* lebih cepat dan lebih mudah diimplementasikan untuk sebagian besar kasus kurva eliptik. Sebaliknya, *Pollard-kangaroo* memberikan kinerja yang lebih baik ketika nilai logaritma diskrit diperkirakan berada dalam kisaran nilai tertentu.**

***Keywords*—Pollard-rho; Pollard-kangaroo; Discrete Log Problem; Kurva Eliptik; Kriptografi; Komputasi;**

I. Introduction

Masalah Discrete Log Problem (DLP) pada kurva eliptik merupakan salah satu masalah fundamental dalam kriptografi modern. DLP menjadi dasar keamanan bagi berbagai protokol kriptografi seperti Elliptic Curve Digital Signature Algorithm (ECDSA) dan Elliptic Curve Diffie-Hellman (ECDH). Penggunaan kurva eliptik dalam konsep ini didasarkan pada kemampuan masing-masing algoritma tersebut untuk menjaga tingkat keamanan yang lebih tinggi dengan ukuran kunci yang lebih kecil dibandingkan dengan sistem kriptografi yang berbasis grup pada bilangan prima.

Adapun terdapat beberapa algoritma yang biasa untuk menyelesaikan permasalahan ini, akan tetapi algoritma yang cukup terkenal digunakan untuk menyelesaikan DLP pada kurva eliptik adalah Pollard-rho dan Pollard-kangaroo. Pollard-rho sebagai algoritma probabilistik yang sangat efisien dalam menemukan logaritma diskrit pada grup siklik. Algoritma ini menggunakan pendekatan pencarian acak dengan teknik *collision detection* untuk menemukan solusi dalam waktu yang relatif lebih singkat. Sebaliknya, Pollard-kangaroo, atau yang juga dikenal sebagai algoritma lambda, dirancang khusus untuk kasus di mana nilai logaritma yang dicari berada dalam rentang yang diketahui. Algoritma ini memanfaatkan metode pencarian yang terstruktur untuk mempercepat proses penyelesaian DLP dalam kisaran nilai tertentu.

Penelitian ini bertujuan untuk membandingkan efektivitas kedua algoritma tersebut dalam berbagai kondisi. Kinerja masing-masing algoritma diukur dalam hal waktu komputasi, penggunaan memori, dan tingkat keberhasilan dalam menyelesaikan DLP pada kurva eliptik. Dengan melakukan eksperimen pada berbagai parameter kurva eliptik, diharapkan dapat diperoleh wawasan yang lebih mendalam mengenai kelebihan dan kelemahan masing-masing algoritma. Hasil penelitian ini diharapkan dapat memberikan panduan bagi para praktisi kriptografi dalam memilih algoritma yang paling sesuai untuk situasi tertentu, sehingga meningkatkan keamanan dan efisiensi sistem kriptografi yang digunakan.

II. Dasar Teori

A. Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) adalah sebuah metode kriptografi yang memanfaatkan sifat-sifat matematis dari kurva eliptik untuk melakukan operasi kriptografi seperti enkripsi, dekripsi, dan pembuatan tanda tangan digital. Metode ini didasarkan pada teori kurva eliptik dalam matematika, yang merupakan bidang yang berkembang pesat sejak ditemukan di tahun 1985 oleh Koblitz dan Miller. ECC menawarkan keamanan yang tinggi dengan panjang kunci yang relatif lebih kecil dibandingkan dengan teknik kriptografi kunci simetris seperti DES atau AES.

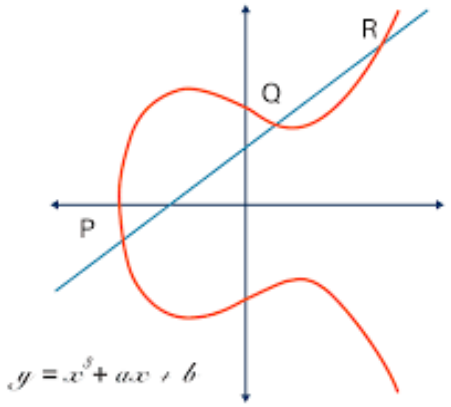
Elliptic Curve Cryptography (ECC) memanfaatkan sifat-sifat matematis dari kurva eliptik dalam sebuah lapangan untuk melakukan operasi kriptografi. Kurva eliptik dapat direpresentasikan oleh persamaan:

$$y^2 = x^3 + ax + by^2 \ldots(1)$$

Dimana a dan b adalah konstanta.

Adapun persamaan Kurva (1) tersebut terdiri dari operasi utama penjumlahan titik dan penggandaan titik, dalam ECC yang digambarkan oleh titik P, Q, dan R pada

gambar kurva eliptik berikut.



Gambar 1. Kurva Eliptik $y = x^3 + ax + b$
Sumber: elliptic-curve-cryptography-diagram.png

Sebagaimana aturan yang berlaku pada kurva eliptik, operasi yang ada pada persamaan kurva tersebut mengikuti aturan sebagai berikut.

1. Jika P dan Q adalah dua titik yang berbeda, garis yang melewati kedua titik tersebut akan memotong kurva eliptik pada titik ketiga R, yang merupakan hasil penjumlahan P + Q.
2. Jika P dan Q adalah titik yang sama, aturan khusus digunakan untuk menemukan hasil penjumlahannya. Titik ketiga R dalam kasus ini adalah titik yang terletak pada garis tegak lurus di titik P, dan hasil penjumlahannya adalah titik refleksi −P.
3. Garis singular yang tegak lurus pada titik P digambar, dan titik ketiga R di mana garis tersebut memotong kurva adalah hasil penggandaan 2 P dari titik P..
4. perasi penjumlahan pada kurva eliptik tidak bersifat komutatif; P+Q mungkin tidak sama dengan Q + P.

Keamanan ECC didasarkan pada kesulitan dalam menyelesaikan *Discrete Logarithm Problem* (DLP) pada kurva eliptik. Dalam konsep ECC, DLP adalah permasalahan untuk menghitung k dari P = kG, di mana G adalah titik generator pada kurva eliptik dan P adalah titik hasil perkalian skalar k dari G. Pentingnya ECC dalam keamanan terletak pada kenyataan bahwa tidak ada algoritma efisien secara umum untuk menyelesaikan DLP. Oleh karena itu, ECC menjadi fondasi untuk berbagai protokol kriptografi modern, menyediakan keamanan yang tinggi dengan panjang kunci yang relatif lebih kecil daripada teknik kriptografi kunci simetris.

B. *Discrete Logarithm Problem* (DLP)

Discrete Logarithm Problem adalah salah satu masalah yang cukup sulit diselesaikan dalam kriptografi yang mencakup berbagai persoalan persamaan matematika pada kriptografi. Pada DLP dilakukan analisis terhadap kemampuan cara untuk menemukan eksponen r pada basis tertentu, dalam modulo bilangan prima p, yang memenuhi persamaan:

$$g^r \equiv h \mod p.$$

Dalam notasi matematika, persamaan tersebut juga dapat ditulis sebagai:

$$r = log_g(h) \mod p$$

Dimana g adalah generator dari grup, h adalah elemen dalam grup, x adalah eksponen yang ingin ditemukan, dan n adalah orde grup.

Pada penyelesaian DLP, meskipun menemukan nilai x tersebut tidaklah mudah. Namun, ada algoritma khusus yang digunakan untuk menyelesaikan DLP dalam beberapa kasus, terutama ketika n adalah bilangan prima besar. Salah satu algoritma yang umum digunakan adalah algoritma Index Calculus atau algoritma Number Field Sieve (NFS).

Oleh karena itu, pada beberapa permasalahan skema kunci publik seperti *Diffie-Hellman Key Exchange*, *El-Gamal encryption*, dan *Digital Signature Algorithm* (DSA). Kemampuan untuk memecahkan DLP dalam permasalah tersebut dapat memberikan pintu gerbang yang lebih terbuka terhadap keamanan sistem kriptografi yang menggunakan skema-skema tersebut.

C. *Elliptic Curve Discrete Logarithm Problem* (ECLDP)

ECLDP mirip dengan *Discrete Logarithm Problem* (DLP), tetapi berlaku untuk operasi dalam grup titik-titik pada kurva eliptik. Misalnya, diberikan sebuah titik Q pada kurva eliptik dan sebuah titik generator P dan terdapat bilangan bulat k sedemikian rupa sehingga Q = kP, di mana k adalah eksponen yang ingin dicari.

Dalam mencari eksponen k yang ketika titik P dijumlahkan dengan dirinya sendiri sebanyak k kali, akan menghasilkan titik Q. Proses ini sering disebut "menghitung skalar" atau "mengalikan titik dengan skalar" pada kurva eliptik.

Adapun untuk menemukan k dari sebuah Q yang diberikan dan sebuah P generator pada kurva eliptik yang dipilih dengan benar, adalah tugas yang sangat sulit atau bahkan tidak mungkin dilakukan dalam waktu yang wajar. Karena kompleksitas yang dihasilkan adalah eksponensial yang bergantung kepada seberapa banyak titik-titik skalar yang dikalikan pada kurva eliptik tersebut.

D. Algoritma *Pollard Rho*

Algoritma *Pollard Rho* adalah algoritma yang menggunakan probabilitas untuk menyelesaikan masalah logaritma diskrit dalam beberapa kasus tertentu hingga mendapatkan solusi secara efisien. Adapun ide dasar di balik algoritma ini adalah menggunakan pola siklik dalam urutan bilangan untuk menemukan nilai yang diulang,

yang mencerminkan penyelesaian masalah logaritma diskrit.

Algoritma *Pollard Rho* memanfaatkan konsep bahwa dalam urutan bilangan, ada kecenderungan untuk menciptakan pola siklik. Dalam konteks logaritma diskrit, ini berarti jika dua bilangan yang berbeda tetapi hasil perhitungan mereka sama, maka mereka telah berada dalam jarak yang lebih dekat dalam langkah-langkah sebelumnya.

Secara umum, berikut adalah langkah yang digunakan pada algoritma Pollard Rho untuk menyelesaikan DLP.

1. Inisialisasi: pilih sebuah titik awal acak x_0 pada kurva eliptik. Kemudian, tentukan dua fungsi iteratif $f(x)$ dan $g(x)$, yang akan menghasilkan urutan bilangan.

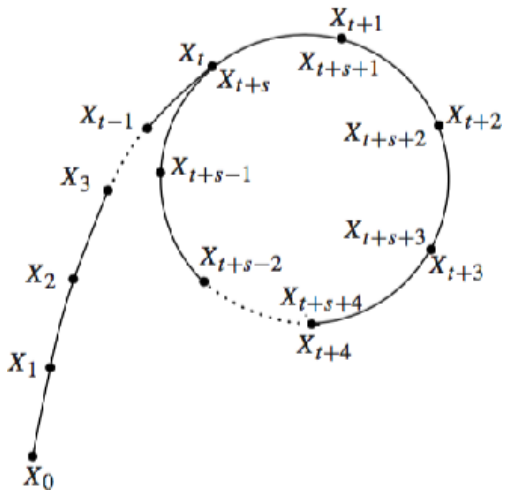
2. Perhitungan Nilai Iteratif: hitung $f(x)$ dan $g(x)$ dari x_0 menggunakan fungsi iteratif yang telah ditentukan. Misalnya, $f(x) = x^2 + c$dan $g(x) = x^2 + c$, dengan c adalah konstanta yang dipilih secara acak.

3. Pencarian Siklik: lakukan iterasi pada nilai xx, menghitung $f(x)$ dan $g(x)$ secara berulang-ulang. Kemudian, cari titik-titik x_i dan x_{2i} di mana $f(x_i) \equiv g(x_{2i}) \ mod\, n$, di mana n adalah modulus. Setelah itu, jika kesamaan ini terpenuhi, kita telah menemukan pola siklik dalam urutan bilangan.

4. Deteksi Siklik: gunakan algoritma pengujian siklik, seperti metode Brent, untuk mendeteksi dan memverifikasi keberadaan siklik.

5. Penyelesaian: jika siklik terdeteksi, hitung selisih dari $x_{2i} - x_i x_{2i} - x_i$. Dengan menggunakan *Backward Euclidean Algorithm*, temukan pembagi bersama antara perbedaan ini dan modulus n. Jika pembagi bersama bukan 1, telah ditemukan faktor non-trivial dari modulus n, yang dapat digunakan untuk menyelesaikan masalah logaritma diskrit. Jika siklik tidak ditemukan pada iterasi pertama, lanjutkan dengan mengatur nilai awal baru x_0. dan ulangi langkah-langkah di atas. Anda dapat menyesuaikan parameter seperti fungsi iteratif dan konstanta c untuk meningkatkan kemungkinan menemukan siklik dalam urutan bilangan.

Adapun ilustrasi terkait dengan langkah-langkah tersebut dapat dilihat pada gambar 2 berikut.



Gambar 2. Ilustrasi Pola Siklik pada Algoritma Pollard Rho
Sumber: images

Oleh karena itu, kompleksitas waktu dari algoritma *Pollard Rho* tergantung pada panjang siklik yang terbentuk dalam urutan bilangan. Dalam kasus terburuknya, kompleksitas waktu adalah sub-eksponensial.

E. Algoritma *Pollard Kangaroo*

Algoritma *Pollard Kangaroo* adalah varian dari algoritma *Pollard Rho* yang dirancang untuk menyelesaikan masalah logaritma diskrit dengan lebih efisien dalam beberapa kasus. Adapun nama "Kangaroo" dan "Wallaby" digunakan untuk mewakili dua jenis pemindai yang digunakan dalam algoritma ini.

Kangaroo melompat sepanjang kurva eliptik menggunakan suatu fungsi iteratif. Setiap langkah, nilai dari fungsi tersebut dihitung berdasarkan posisi sebelumnya, dan kangaroo berpindah ke titik baru pada kurva. Sementara itu, Wallaby melakukan gerakan yang mirip, tetapi beroperasi dalam interval yang lebih kecil. Tugasnya adalah untuk memonitor pola yang lebih spesifik dalam hasil fungsi iteratif.

Secara umum, berikut adalah langkah yang digunakan pada algoritma Pollard Kangaroo untuk menyelesaikan DLP.

1. Inisialisasi: pilih dua titik awal acak pada kurva eliptik: $x_{kangaroo}$ dan $x_{wallaby}$. Kemudian, ditentukan dua fungsi iteratif f(x) dan g(x), yang akan menghasilkan urutan bilangan.

2. Gerakan Kangaroo dan Wallaby: Kangaroo akan melompat sepanjang kurva eliptik menggunakan fungsi iteratif f(x). Kemudian, Wallaby akan melakukan gerakan yang mirip, tetapi beroperasi dalam interval yang lebih kecil, menggunakan fungsi iteratif g(x).

3. Deteksi Siklik: Lakukan iterasi pada nilai xx, menghitung f(x) dan g(x) secara berulang-ulang.

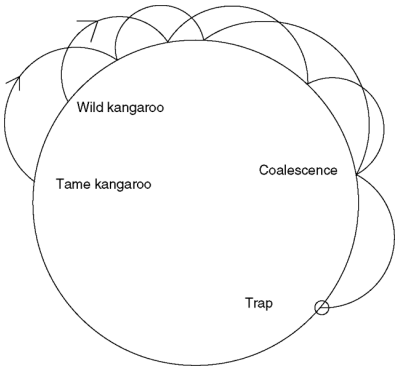
Kemudian, periksa apakah ada titik-titik x_i dan x_{2i} di mana $f(x_i) \equiv g(x2i) \ mod\, n$, di mana n adalah modulus. Setelah itu, jika pola siklik terdeteksi, lanjutkan ke langkah berikutnya. Jika tidak, lanjutkan gerakan Kangaroo dan Wallaby. Jika siklik terdeteksi, hitung perbedaan $x_{2i} - x_i x_{2i} - x_i$. Dengan menggunakan *Backward Euclidean Algorithm*, ditemukan pembagi bersama antara perbedaan ini dan modulus n. Jika pembagi bersama bukan 1, Anda telah menemukan faktor non-trivial dari modulus nn, yang dapat digunakan untuk menyelesaikan masalah logaritma diskrit.

4. Penyelesaian: jika siklik tidak ditemukan pada iterasi pertama, lanjutkan dengan mengatur nilai awal baru $x_{kangaroo}$ dan $x_{wallaby}$, serta ulangi langkah-langkah di atas. Dapat disesuaikan parameter seperti fungsi iteratif dan konstanta untuk meningkatkan kemungkinan menemukan siklik dalam urutan bilangan.

Pada Pollard Kangaroo, terdapat 2 jenis gerakan, yaitu sebagai berikut.

1. Kangaroo Liar (*Wild Kangaroo*)
   Kangaroo liar adalah kangaroo yang bergerak secara acak di sepanjang kurva eliptik dengan menggunakan fungsi iteratif tertentu. Pergerakan kangaroo liar tidak terikat oleh aturan tertentu, dan ia melompat sepanjang kurva tanpa mempertimbangkan apakah jalur yang ditempuh telah dilewati sebelumnya atau tidak. Tugas utama kangaroo liar adalah untuk mencoba menemukan titik yang bersamaan dengan jalur yang telah dilewati oleh kangaroo jinak.
2. Kangaroo Jinak (*Tame Kangaroo*)
   Kangaroo jinak adalah kangaroo yang bergerak secara terstruktur dan deterministik di sepanjang kurva eliptik dengan menggunakan fungsi iteratif yang sama. Kangaroo jinak mengikuti jalur yang telah dilewati sebelumnya oleh kangaroo lain, seperti kangaroo liar atau kangaroo jinak lainnya. Kangaroo jinak memiliki tujuan untuk mencapai titik tertentu dalam interval yang ditentukan dan kemudian melaporkan hasilnya kepada algoritma.

Adapun ilustrasi terkait dengan langkah-langkah tersebut dapat dilihat pada gambar 2 berikut.



Gambar 3. Ilustrasi Pola Siklik pada Algoritma Pollard Kangaroo
Sumber: 978-0-387-23483-0_4_Part_Fig5_HTML.gif

Oleh karena itu, kompleksitas waktu algoritma *Pollard Kangaroo* bergantung pada berbagai faktor, termasuk panjang siklik yang terbentuk dalam urutan bilangan pada kurva eliptik. Namun, secara umum, algoritma ini sering kali lebih cepat daripada pendekatannya yang lebih sederhana, seperti *Pollard Rho*, dalam banyak kasus.

III. Perbandingan Algoritma Pollard Rho dan Pollard Kangaroo pada ECDLP

A. Analisis Solusi Algoritma pada ECDLP

1. Algoritma Pollard Rho

Pada Algoritma *Pollard Rho*, setiap langkah pergerakannya dengan menghitung:

triplet (Ri, ai, bi) dan (R2i, a2i, b2i)

serta, sambil memeriksa kebenaran dari Ri = R2i.

Perhatikan bahwa di setiap "langkah", dapat dihitung tiga operasi grup (lihat baris 23, 24, 25 di Algoritma 1 pada gambar 4). Ketika sebuah tabrakan ditemukan, dapat diperoleh:

$$[ai]P + [bi]Q = [a2i]P + [b2i]Q$$

dan dapat dihitung dengan logaritma diskrit (selama (b2i − bi) ≠ 0 mod r) sebagai berikut:

$$n = (ai - a2i)(b2i - bi)^{-1} mod\, r$$

Karena r besar, dapat diasumsikan bahwa probabilitas $b_{2i} \equiv b_i$ mod r cukup kecil, dan bisa diabaikan.

Perlu diperhatikan juga bahwa dalam metode Pollard-rho, probabilitas keberhasilan terdefinisi dengan baik (yaitu, tabrakan akan terjadi jika urutan tersebut cukup besar), tetapi waktu eksekusi diharapkan, bukan mutlak dibatasi. Dari algoritma tersebut, kita dapat menyajikan metode Pollard-rho yang dijelaskan di atas dengan asumsi bahwa f berperilaku seperti fungsi random untuk implementasi komputasinya. Namun, dalam analisis, f dapat didefinisikan sebagai fungsi acak dari ⟨P⟩ × Z/rZ ×

Z/rZ ke dirinya sendiri. Untuk selengkapnya dapat melihat *pseudocode* pada gambar 4 berikut.

Algorithm 1 Pollard-rho discrete log

Input: group $E(\mathbb{F}_q)$; points $P, Q \in E(\mathbb{F}_q)$; integers a_0, b_0; point $R_0 = [a_0]P + [b_0]Q$
Output: integer n, such that $Q = [n]P$

function $\phi(R)$
  $\phi = x_R \bmod 3$
  Return ϕ
end function

function $f(R, a, b)$
if $\phi(R) = 0$ **then**
    $f \leftarrow (R + P, a + 1, b)$
else
    if $\phi(R) = 1$ **then**
      $f \leftarrow ([2]R, 2a, 2b)$
    else
      $f \leftarrow (R + Q, a, b + 1)$
    end if
end if
Return f
end function

main
$(R, a, b) \leftarrow f(P, 1, 0)$
$(R', a', b') \leftarrow f(R, a, b)$
while $R \neq R'$ **do**
    $(R, a, b) \leftarrow f(R, a, b)$
    $(R', a', b') \leftarrow f(R', a', b')$
    $(R', a', b') \leftarrow f(R', a', b')$
end while
if $(b' - b) \equiv 0 \pmod{\mathbf{r}}$ **then**
    Output "failure"
else
    Output $n = (a - a')(b' - b)^{-1} \pmod{\mathbf{r}}$
end if
end main

Gambar 4. Solusi Algoritma Pollard Rho pada ECDLP

2. Algoritma Pollard Kangaroo

Dari bagian sebelumnya pada teori dasar, melalui Konfigurasi gerakan *Wild Kangaroo* dan *Tame Kangaroo*, dapat diperoleh pendekatan sebagai berikut.

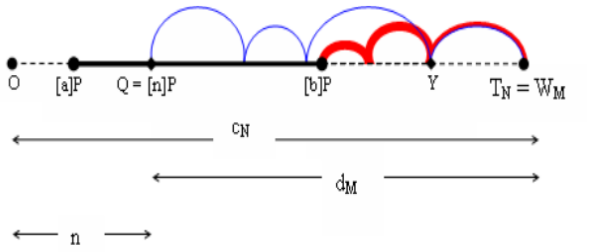
1. Pertemuan Antara *Wild Kangaroo* dan *Tame Kangaroo*: Jika pada suatu titik *Wild Kangaroo (W)* bertemu dengan jalur yang telah dilalui oleh *Tame Kangaroo* (T), maka W akan terus bergerak sepanjang jalur yang semula dilalui oleh T hingga mencapai titik *Wm = TN* untuk beberapa m yang kurang dari atau sama dengan M. Dalam konteks kurva eliptik, hal ini berarti bahwa ada titik TN yang dicapai oleh T dan titik Wm yang dicapai oleh W yang merepresentasikan [n+dm]P, di mana nn adalah solusi dari masalah DLP yang ingin ditemukan.

2. Solusi untuk Masalah DLP: Dengan demikian, kita memiliki solusi untuk masalah DLP, karena kita dapat menyatakan bahwa *n = cN−dm*, di mana c dan d adalah langkah-langkah yang diambil oleh masing-masing kangaroo dan N adalah panjang interval yang ditentukan.

3. Pemberhentian *Wild Kangaroo*: Jika jarak yang ditempuh oleh Wild Kangaroo (*dm*) melebihi batas tertentu (misalnya, *Mα*), yang lebih besar dari perbedaan antara cN dan aa, maka kita dapat menghentikan pergerakan W, karena ini menunjukkan bahwa ia telah melewati "perangkap" yang ditentukan sebelumnya.

4. Jika Tidak Ada Tabrakan: Jika tidak ditemukan tabrakan, kita dapat memulai *Wild Kangaroo* lainnya dari titik baru, misalnya *Q'=Q+[z]P*, di mana z adalah bilangan bulat yang diketahui.

Adapun untuk konfigurasi tersebut dapat diilustrasikan melalui gambar 4 berikut.



Gambar 4. Ilustrasi Pergerakan Kangaroo Jinak dan Kangaroo Liar

Sehingga pada konfigurasi pertemuan atau kebebasan kedua kangaroo tersebut, dapat diperoleh algoritma berikut untuk menyelesaikan permasalahan ECDLP

Algorithm 2 Pollard-kangaroo discrete log

Input: group $E(\mathbb{F}_q)$, points $P, Q \in E(\mathbb{F}_q)$, integers a, b
Output: integer n such that $a \leqslant n \leqslant b$ and $Q = [n]P$.

Choose number of partitions s
Choose starting point of tame kangaroo $T_0 = [c_0]P$
Choose steps s_i /* Usually, $s_i = 2^i$ */
Choose walk lengths $M, N \in \mathbb{N}$ $\mathcal{S}_j$, $0 \leq j < s$.
function $\phi(R)$
  $\phi = x_R \bmod s$
  Return ϕ
end function

Precompute $S_i = [s_i]P$ for $0 \leq i < s$
[*Tame Kangaroo*]
$T \leftarrow T_0$
$c \leftarrow c_0$
for $i = 1$ to N **do**
  Compute $j = \phi(T)$
  $T \leftarrow T + S_j$
  $c = c + s_j$
end for
[*Wild Kangaroo*]
$W \leftarrow Q$
$d \leftarrow 0$
$i \leftarrow 0$
repeat
  Compute $j' = \phi(W)$
  $W \leftarrow W + S_{j'}$
  $d \leftarrow d + s_{j'}$
  $i \leftarrow i + 1$
until $(T = W)$ or $(i > M)$
if $T = W$ **then**
  Output $n = (c - d)$
else
  Output "failure"
end if

Gambar 5. Solusi Algoritma Pollard Kangaroo pada ECDLP

B. Implementasi Pada Bahasa Pemrograman C++

1. Pustaka Operator & Tipe Data
   Terdapat operasi GCD yang diimplementasikan secara manual sebagai operasi yang digunakan untuk menangani modularitas kelipatan dari setiap gerakan untuk menghitung pola siklik yang ada yaitu sebagai berikut.

```
ll gcd(ll a, ll b) {
    if (b == 0)
        return a;
    return gcd(b, a % b);
}
```

Terdapat tipe data point untuk mengimplementasikan posisi titik koordinat dari kurva eliptik yang digunakan sebagai tipe data untuk setiap pergeseran titik pada ECDLP.

```
struct Point {
    ll x, y;
    bool operator==(const Point& p) const {
        return x == p.x && y == p.y;
    }
};
```

2. Pustaka Fungsi/Prosedur Bantuan
   Terdapat prosedur *measurePerformance* yang digunakan untuk mengukur performansi rata-rata dari masing-masing algoritma dengan masukan test case digunakan secara random, yaitu sebagai berikut.

```
vector<double> measurePerformance(int
numTests) {
    vector<double> pollardTimes;
    vector<double> rhoTimes;

    for (int i = 0; i < numTests; ++i) {
        EllipticCurve p = rand() % 1000 + 1000;
// random prime number
        point g = rand() % (p - 1) + 1; // random
generator
        point q = rand() % (p - 1) + 1; // random
target
ll n = rand() % (p - 1) + 1; // random target

        clock_t start, end;

        start = clock();
        ll result1 =
pollard_kangaroo_algorithm(g, h, p);
        end = clock();
        pollardTimes.push_back((double)(end -
start) / CLOCKS_PER_SEC * 1000);

        start = clock();
        ll result2 = pollardRho(g, p, q, h);
        end = clock();
        rhoTimes.push_back((double)(end - start)
/ CLOCKS_PER_SEC * 1000);
    }

    return {accumulate(pollardTimes.begin(),
pollardTimes.end(), 0.0) / numTests,
            accumulate(rhoTimes.begin(),
rhoTimes.end(), 0.0) / numTests};
}
```

Dari fungsi tersebut dapat dihasilkan seluruh rata dari keseluruhan percobaan sebanyak numTest kali. Setelah itu, dikembalikan rata-rata time execution dari setiap masukkan random pada masing-masing algoritma untuk penggunaan test case yang sama.

3. Pustaka Elliptic Curve
   Diimplementasikan kelas kurva eliptik untuk memudahkan dalam implementasi kurva eliptic dan melakukan operasi yang ada pada kurva eliptik secara sistematis dan clean code.

```
class EllipticCurve {
public:
    ll a, b, p; // Parameters of the curve: y^2 =
x^3 + ax + b (mod p)

    EllipticCurve(ll a, ll b, ll p) : a(a), b(b), p(p)
{}

    Point add(const Point& P, const Point& Q)
const {
        if (P.x == -1 && P.y == -1) return Q; // P
is the point at infinity
        if (Q.x == -1 && Q.y == -1) return P; //
Q is the point at infinity

        ll m;
        if (P == Q) {
            m = (3 * P.x * P.x + a) * modInverse(2
* P.y, p) % p;
        } else {
            m = (Q.y - P.y) * modInverse(Q.x -
P.x, p) % p;
        }

        ll xr = (m * m - P.x - Q.x) % p;
        ll yr = (m * (P.x - xr) - P.y) % p;
        return {xr < 0 ? xr + p : xr, yr < 0 ? yr + p
: yr};
    }

    Point multiply(const Point& P, ll k) const {
        Point R = {-1, -1}; // Point at infinity
        Point Q = P;
        while (k > 0) {
            if (k % 2 == 1) {
                R = add(R, Q);
            }
            Q = add(Q, Q);
            k /= 2;
        }
        return R;
```

```
    }

private:
    ll modInverse(ll a, ll p) const {
        ll m0 = p, t, q;
        ll x0 = 0, x1 = 1;
        if (p == 1) return 0;
        while (a > 1) {
            q = a / p;
            t = p;
            p = a % p, a = t;
            t = x0;
            x0 = x1 - q * x0;
            x1 = t;
        }
        if (x1 < 0) x1 += m0;
        return x1;
    }
};
```

4. Pustaka Algoritma Pollard Rho
   Sesuai dengan algoritma yang telah disajikan sebelumnya, berikut diimplementasikan Algoritma Pollard Rho yang diimplementasikan dengan parameter kurva eliptik yang pada kedua algoritma tersebut dapat diambil kurva eliptik yang sama, titik p, titik q, dan nilai n.

```
ll pollardRho(const EllipticCurve& curve,
const Point& P, const Point& Q, ll n) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<ll> dis(0, n - 1);

    ll a = dis(gen), b = dis(gen);
    ll A = dis(gen), B = dis(gen);

    Point X = curve.add(curve.multiply(P, a),
curve.multiply(Q, b));
    Point Y = curve.add(curve.multiply(P, A),
curve.multiply(Q, B));

    while (true) {
        if (X == Y) {
            ll r = (a - A) % n;
            ll s = (B - b) % n;
            if (s == 0) continue;
            return (r * curve.modInverse(s, n)) %
n;
        }
        X = curve.add(X, X);
        Y = curve.add(Y, Y);
        Y = curve.add(Y, Y);
    }
}
```

5. Pustaka Algoritma Pollard Kangaroo
   Berdasarkan Algoritma 2, yaitu pada gambar 5, dapat diimplementasikan skema Wild Kangaroo dan Tame Kangaroo yang diimplementasikan dengan menggunakan parameter kurva eliptik sebagai input fungsi, titik P dan Q sebagai titik pergeseran, serta nilai n, a, dan b, masing-masing sebagai nilai derajat dan koefisien grup polinomial yang digunakan. Adapun pada fungsi ini dikembalikan nilai x sebagai solusi ECDLP.

```
ll pollardKangaroo(const EllipticCurve&
curve, const Point& P, const Point& Q, ll n, ll
a, ll b) {
    vector<Point> T(32);
    vector<ll> D(32);
    for (int i = 0; i < 32; ++i) {
        T[i] = curve.multiply(P, 1LL << i);
        D[i] = 1LL << i;
    }

    ll xTame = a;
    Point Tame = curve.multiply(P, a);
    while (xTame <= b) {
        int j = Tame.x % 32;
        Tame = curve.add(Tame, T[j]);
        xTame += D[j];
    }

    Point Wild = Q;
    ll xWild = 0;
    while (true) {
        if (Tame == Wild) return (xTame -
xWild) % n;
        int j = Wild.x % 32;
        Wild = curve.add(Wild, T[j]);
        xWild += D[j];
    }
}
```

C. Hasil Pengujian

Test Case 1

```
Curve Parameters: y^2=x^3+2x+3y^2 mod 97

Base Point (P): (3,6)(3,6)

Resulting Point (Q): kP=(80,10) for k=5

Order (n): 5

Time Execution Pollard Rho: 842.146 ms

Time Execution Pollard Kangaroo: 780.273 ms
```

Test Case 2

Curve Parameters: y^2=x^3+4x+20 mod 101

Base Point (P): (2,22)

Resulting Point (Q): kP=(85,9), k=7

Order (n): 7

Time Execution Pollard Rho: 1476.529 ms

Time Execution Pollard Kangaroo: 1291.376 ms

Test Case 3

Curve Parameters: y2=x3+5x+7y2=x3+5x+7 mod 103103

Base Point (P): (3,6)

Resulting Point (Q): kP=(45,78) for k=10

Order (n): 10

Time Execution Pollard Rho: 1783.245 ms

Time Execution Pollard Kangaroo: 1936.185 ms

Test Case 4

Curve Parameters: y^2=x^3+x+1y^2 mod 89

Base Point (P): (2,4)(

Resulting Point (Q): kP=(58,23) for k=11

Order (n): 11

Time Execution Pollard Rho: 2518.872 ms

Time Execution Pollard Kangaroo: 2036.678 ms

Test Case 5

Curve Parameters: y ^2 = x^3 + 3x + 10 mod 113

Base Point (P): (6 , 15)

Resulting Point (Q): k P = (75 , 25), k=13

Order (n): 13

Time Execution Pollard Rho: 3678.561 ms

Time Execution Pollard Kangaroo: 2748.135 ms

Dari kelima test case tersebut dapat dilihat bahwa untuk kasus derajat grup yang semakin tinggi dan nilai k yang semakin tinggi akan mengakibatkan algoritma Pollard Rho dan Algoritma Pollard Kangaroo Semakin lambat. Akan tetapi, untuk pengurangan performa waktu eksekusi lebih signifikan pada algoritma Pollard Rho.

Hal tersebut dikarenakan untuk nilai k yang semakin besar, Algoritma Pollard Rho mencari pemetaan titik-titik pada kurva untuk menemukan siklus yang akhirnya mengarah pada solusi.Ketika nilai k meningkat, jumlah titik yang harus dipetakan dan diperiksa juga meningkat, yang berarti lebih banyak langkah yang diperlukan sebelum menemukan siklus.

Sebaliknya, Algoritma Pollard Kangaroo dirancang untuk skenario dimana logaritma diskrit diketahui berada dalam interval tertentu [a,b]. Dalam kasus ini, waktu eksekusi tetap relatif stabil dan tidak terlalu bergantung pada nilai k secara keseluruhan, selama intervalnya diketahui dan dapat diperkirakan.

D. Analisis Kompleksitas

Dari analisis pada bagian analisis solusi serta dari studi literatur melalui beberapa paper, dapat diperoleh kompleksitas waktu eksekusi dan penggunaan memori dari masing-masing algoritma tersebut adalah sebagai berikut.

1. Algoritma Pollard Rho

   Waktu Eksekusi: Algoritma Pollard Rho memiliki waktu eksekusi yang diharapkan $O(\sqrt{n})$ untuk menemukan solusi. Dalam praktiknya, menggunakan metode Floyd untuk penemuan siklus, waktu eksekusi diharapkan sekitar $3.76\sqrt{n}$. Jika menggunakan metode Brent, waktu eksekusi dapat diperbaiki menjadi sekitar $2.86\sqrt{n}$.

   Analisis Memori: Algoritma Pollard Rho menggunakan jumlah memori yang konstan karena hanya membutuhkan beberapa variabel tambahan untuk menyimpan titik pada kurva dan koefisien yang terkait dengan perhitungan iteratif. Ini berarti penggunaan memori adalah $O(1)$.

2. Algoritma Pollard Kangaroo

Waktu Eksekusi: Algoritma Pollard Kangaroo juga memiliki waktu eksekusi yang diharapkan $O(\sqrt{n})$, namun waktu ini lebih tepat diterapkan saat logaritma diskrit terletak dalam interval tertentu [a,b]. Dalam implementasi umum, waktu eksekusi diharapkan sekitar $3.28\sqrt{n}$.

Analisis Memori: Algoritma *Pollard Kangaroo* cenderung menggunakan lebih banyak memori dibandingkan Pollard Rho karena memerlukan penyimpanan tabel pre-komputasi untuk lompatan kangaroo (wild dan tame). Penggunaan memori adalah O(m), di mana m adalah jumlah lompatan yang telah disiapkan.

V. Kesimpulan dan Saran

Dari hasil eksperimen dan analisis yang dilakukan pada makalah ini, dapat disimpulkan bahwa Algoritma Pollard Rho dan Algoritma Pollard Kangaroo memberikan pendekatan yang efektif untuk menyelesaikan masalah ECDLP. Akan tetapi, jika dibandingkan, Algoritma Pollard-Kangaroo cenderung memberikan kinerja yang lebih baik, terutama ketika pada kasus pola siklik yang cenderung lebih pendek yang terdapat urutan bilangan pada kurva eliptik. Meskipun demikian, Komputasi Pollard-Rho juga cenderung lebih baik ketika pada kasus pola siklik yang dihasilkan cenderung lebih sedikit atau dalam situasi dimana kompleksitas ruang lebih diutamakan daripada kompleksitas waktu. Algoritma Pollard-rho akan selalu memberikan solusi, kecuali jika kondisi tertentu terpenuhi dengan probabilitas kecil. Kondisi tersebut adalah $(b' - b) \equiv 0 \ mod \ r$, yang jarang terjadi jika r cukup besar. Efektivitas keduanya tergantung pada karakteristik kasus *test case* yang dihadapi, dengan Pollard-Kangaroo sering kali lebih bagus pada penanganan kasus-kasus di mana pola siklik dapat ditemukan dengan relatif cepat.

Lampiran

Akses source code program yang diimplementasikan dapat diakses melalui pranala pada link berikut.

Ucapan Terima Kasih

Penulis berterima kasih kepada Tuhan yang Maha Esa karena rahmat dan karunia-Nya, penulis diberikan kesempatan untuk menyelesaikan makalah IF4020 Kriptografi. Serta Penulis mengucapkan terima kasih kepada Pak Rinaldi Munir yang membimbing penulis pada mata kuliah IF4020. Penulis mengucapkan terima kasih terhadap semua pihak yang terlibat dalam membantu Penulis dalam menyelesaikan tugas ini.

References

R. Munir, "ECC-Bagian 1." 5 April 2024. Diakses: 12 Juni 2024. [Daring]. Tersedia pada: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2023-2024/22-ECC-Bagian1-2024.pdf

Edlyn Teske. Computing discrete logarithms with the parallelized kan- garoo method. Discrete Applied Mathematics, 130(1):61–82, 2003.

D. Stinson. Some Baby-Step Giant-Step Algorithms for the Low Ham- ming Weight Discrete Logarithm Problem. Mathematics of Computa- tion, 71(237):379–391, 2002.

J. Pollard. Kangaroos, Monopoly and Discrete Logarithms. Journal of Cryptology, 13(4):437–447, December 2000.

W. D. B. Junior. Applications of Frobenius Expansions in Elliptic Curve Cryptography. Department of Mathematics Royal Holloway, University of London, 2008.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

Muhammad Gilang Ramadhan