Perbandingan Fungsi Pseudo RNG pada Python, Java, dan C++

Christian Kevin Saputra / 13516073
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
ck3putra@gmail.com

***Abstract*— Kriptografi merupakan salah satu ilmu yang sudah ada dari jaman dahulu dan berguna untuk menjaga keamanan pesan. Untuk menjaga keamanan pesan , beberapa ilmu kriptografi memanfaatkan bilangan acak yang dihasilkan oleh *random number generator (RNG)*. *Random number generator* dapat dikatakan sebagai suatu alat untuk menghasilkan bilangan – bilangan secara acak. Terdapat 2 jenis RNG yaitu *True Random Number Generator* yang benar – benar menghasilkan bilangan secara acak dan juga *Pseudorandom Number Generator* yang menghasilkan bilangan acak berdasarkan suatu state. Setiap bahasa pemrograman memiliki fungsi random bertipe *pseudo RNG* yang sudah terdapat pada library mereka. Untuk mengetahui kinerja dari masing – masing fungsi *pseudo RNG* terutama pada bahasa Python, Java, dan C++, dilakukan pengetesan dengan menggunakan beberapa *test suite* dari *A Statistical Test Suite for Random and Pseudorandom Number Generator for Cryptographic Applications oleh NIST*.**

Keywords—Kriptografi, Pseudorandom Number Generator, Python, Java, C++

I. PENDAHULUAN

Kriptografi berasal dari bahasa Yunani yaitu *cryptós* yang berarti *secret* (rahasia) dan juga *gráphein* yang berarti *writing* (tulisan) sehingga kriptografi dapat diartikan sebagai *secret writing* (tulisan rahasia). Menurut Menez (1996), kriptografi merupakan sebuah ilmu yang mempelajari teknik – teknik matematika yang berhubungan dengan aspek keamanan informasi seperti kerahasiaan, integritas data, serta otentikasi. Untuk mempersulit dipecahkannya algoritma kriptografi, beberapa algoritma kriptografi menerapakan penggunaan bilangan acak yang dihasilkan oleh *random number generator*.

Random Number Generator adalah suatu alat yang dapat menghasilkan bilangan secara acak. Sifat acak (*random*) inilah yang diinginkan dalam kriptografi untuk mempersulit pemecahan kriptografi. Terdapat 2 jenis *random number generator* yaitu *true random number generator* dan *pseudorandom number generator*. Bahasa pemrograman yang ada sekarang ini telah menyediakan *pseudorandom number generator* sebagai fungsi bawaan untuk mempermudah penggunanya. Namun, belum diketahui perbandingan kinerja dari *pseudorandom number generator* pada bahasa pemrograman. Oleh karena itu, penulis mencoba membandingkan kinerja fungsi *pseudorandom number generator* yang sudah tersedia pada bahasa pemrograman Python, Java, dan C++ dengan menggunakan *test suite* dari *A Statistical Test Suite for Random and Pseudorandom Number Generator for Cryptographic Applications* yang dipublikasikan oleh NIST. Dari 15 test yang ada, akan diambil 1 tes yaitu *Frequency (Monobit) Test* dan *The Runs Test*.

II. DASAR TEORI

A. Random Number Generator (RNG)

Random number generator adalah suatu alat yang dapat menghasilkan bilangan secara acak. Terdapat 2 jenis *random number generator* yaitu *true random number generator* dan *pseudorandom number generator*. *True random number generator* membangkitkan bilangan acak berdasarkan sumber non-deterministik. Sumber non-deterministik (*entrophy source*) mendapatkan nilainya dari kejadian yang sulit direplikasi ulang seperti suara pada sirkuit elektrik ataupun efek kuantum di dalam semikonduktor. [1]

Pseudorandom number generator (PRNG) adalah pembangkit bilangan acak yang menghasilkan bilangan secara acak berdasarkan suatu nilai atau state yang disebut dengan *seed*. Apabila *pseudorandom number generator* membangkitkan bilangan acak dengan sebuah seed yang sama dengan sebelumnya, dapat dipastikan bahwa hasil bilangan acak yang dihasilkan juga sama.

Di dalam bahasa pemrograman Python, PRNG dapat diakses melalui modul Random dengan melakukan “import random”. Java mengakses PRNG dengan mengakses kelas Random yang didapat dari “java.util.Random”. C++ mengakses PRNG dengan menggunakan fungsi rand() yang terdefinisi pada <stdlib.h>

B. A Statistical Test Suite for Random and Pseudorandom Number Generator for Cryptographic Applications

Paper yang memiliki judul *A Statistical Test Suite for Random and Pseudorandom Number Generator for Cryptographic Applications* ditulis oleh Andrew Rukhin , Juan Soto , James Nechvatal , Miles Smid , Elaine Barker , Stefan Leigh , Mark Levenson , Mark Vangel , David Banks , Alan Heckert , James Dray , dan San Vo dan dipublikasikan oleh NIST (National Institute of Standards and Technology). Paper ini ditulis dengan tujuan untuk mempublikasikan tes – tes yang dapat digunakan untuk menentukan apakah sebuah RNG cocok untuk digunakan dalam bidang kriptografi. Terdapat 15 tes di dalamnya yaitu :

1. *The Frequency (Monobit) Test,*
2. *Frequency Test within a Block,*
3. *The Runs Test,*
4. *Tests for the Longest-Run-of-Ones in a Block,*
5. *The Binary Matrix Rank Test,*
6. *The Discrete Fourier Transform (Spectral) Test,*
7. *The Non-overlapping Template Matching Test,*
8. *The Overlapping Template Matching Test,*
9. *Maurer's "Universal Statistical" Test,*
10. *The Linear Complexity Test,*
11. *The Serial Test,*
12. *The Approximate Entropy Test,*
13. *The Cumulative Sums (Cusums) Test,*
14. *The Random Excursions Test,* dan
15. *The Random Excursions Variant Test.*

Tes yang dilakukan untuk pengujian PRNG pada paper ini hanya menggunakan *The Frequency (Monobit) Test*

C. *The Frequency (Monobit) Test*

Tes dilakukan dengan melakukan pemanggilan fungsi PRNG untuk menghasilkan angka 0 atau 1 sebanyak n kali. Fokus dari tes ini adalah untuk menentukan apakah jumlah dari angka 0 dan 1 di dalam sebuah urutan sama seperti yang akan dihasilkan oleh *truly random number generator*. Tes akan menyatakan urutan yang dihasilkan adalah tidak random apabila menghasilkan P-value < 0,01 , selain itu maka urutan adalah random.

D. *The Runs Test*

Tes dilakukan dengan melakukan pemanggilan fungsi PRNG untuk menghasilkan angka 0 atau 1 sebanyak n kali. Fokus dari tes ini adalah untuk menentukan jumlah total dari *runs. Run* adalah urutan angka yang identikal. Dengan mengetahui jumlah dari *runs of zeroes* dan *runs of ones*, dapat diketahui apakah jumlah *runs* sama seperti yang akan dihasilkan oleh urutan bilangan random.

III. Hasil Percobaan

A. *The Frequency (Monobit) Test*

Tes dilakukan pada 3 bahasa pemrograman yaitu Python, Java, dan C++.

Pada Python, program menggunakan library “scripy” untuk mendapatkan nilai P-Value serta random untuk menggunakan PRNG milik bahasa Python dan math untuk menghitung absolut. Berikut implementasi dari program :

```
from scipy import special
import random, math
urutan = []
Sn = 0
for i in range(100):
 bit = random.randrange(2)
 bit_value = (2 * bit) - 1
 Sn = Sn + bit_value
 urutan.append(bit)
print("Urutan bilangan:")
print(urutan)
s_abs = math.fabs(Sn)/10
print("Sn = " , Sn)
print("Sabs =" , s_abs)
print("P-value:" , special.erfc(s_abs/math.sqrt(2)))
```

Pada Java, kelas Random untuk menggunakan PRNG dipanggil dari “java.util.Random” ditambahkan pula jar dari website yang dapat dilihat pada referensi [2] untuk fungsi erfc. Program diimplementasikan sebagai berikut :

```
import
org.apache.commons.math3.special.Erf;
import java.util.Random;
import java.lang.Math;
public class Kripto {


        public static void main(String[]
args) {
                Random random = new Random();
                int Sn = 0;
                for (int i = 0; i < 100; i++)
{
                        int bit =
random.nextInt(2);
                        System.out.print(bit);
                        int bit_value = (2 *
bit) - 1;
                        Sn += bit_value;
                }
                System.out.println();
                System.out.print("Sn = ");
                System.out.println(Sn);
                double S_abs = (double) Sn /
10;
                System.out.print("S_abs = ");
                System.out.println(S_abs);
                double P_value = S_abs /
Math.sqrt(2);
                System.out.println("P-
value:");
```

```
System.out.println(Erf.erfc(P_value));
                }


}
```

Pada CPP, program untuk mendapatkan P-Value diimplementasikan sebagai berikut :

```
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <math.h> //for erfc
#include <time.h>
#include <iostream>
using namespace std;


int main () {
        vector<int> urutan;
        int Sn = 0;
        srand( time(NULL));
        for (int i = 0; i < 100; i ++){
                int bit = rand() % 2;
                int bit_value = (2 * bit) - 1;
                Sn = Sn + bit_value;
                urutan.push_back(bit);
        }


        for (auto i = urutan.begin(); i != urutan.end(); ++i) {
       cout << *i;
        }
        cout << endl;
        float s_abs = float(abs(Sn)) /float(10);
        cout << "Sn =" << Sn << endl;
        cout << "Sabs =" << s_abs << endl;
        float P_value = erfc(s_abs/sqrt(2));
        cout << "P-value =" << P_value << endl;


}
```

Hasil dari tes ini ditunjukkan dengan tabel berisi nilai P-Value berikut:

Percobaan	Python	C++	Java
1	1.0	0.109599	0.16151331846754213
2	0.8414805811217939	0.317311	0.6891565167793516
3	0.5485062355001471	0.230139	0.5485062355001471
4	0.4237107971667935	0.423711	0.6891565167793516
5	0.0718606382258516 2	0.548506	1.0

Nilai P-Value yang dihasilkan dari semua nilai pemrograman dapat dilihat pada tabel di atas. Dari 5 percobaan, PRNG pada bahasa Python dan Java yang menghasilkan bilangan acak dengan perbandingan angka 1 dan 0 yang sama (ditunjukkan dengan nilai P-Value 1.0). Hal ini dapat dikatakan bahwa Python menghasilkan bilangan acak yang baik. Walaupun tidak mendapatkan nilai 1.0, namun nilai P-Value yang lebih besar dari 0.01 (P-Value>0.01) menyatakan bahwa bilangan acak yang dihasilkan sudah cukup random. Semakin besar P-Value menyatakan bahwa semakin banyak persebaran antara angka 0 dan 1. Sedangkan P-Value memiliki nilai 1.0 menyatakan bahwa bilangan yang dihasilkan memiliki angka 0 dan 1 yang sama banyak jumlahnya.

B. The Runs Test

Tes dilakukan dengan membangkitkan 100 bilangan antara 0 dan 1 dengan menggunakan PRNG milik Python, Java, dan C++. Dari bilangan yang terbentuk akan dihitung jumlah bilangan 1 yang keluar kemudian dibandingkan dengan banyaknya bilangan. Contoh : 1001011100 memiliki 5 bilangan 1 maka $\pi = 5 / 10 = 0.5$. Untuk menghitung P-value digunakan fungsi erfc pada persamaan berikut :

$$\left(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)} \right).$$

Dengan n adalah jumlah bilangan dalam urutan dan $V_n(obs)$ adalah test statistic yang didapat dari menghitung bilangan pada urutan yang dibangkitkan yang berbeda dari bilangan selanjutnya. Contoh 1001011100 maka $V_{10}(obs) = (1 + 0 + 1 + 1 + 1 + 0 + 0 + 1 + 0) + 1 = 6$.

Implementasi menggunakan bahasa Python :

```
from scipy import special
import random, math


urutan = []
```

```
    count = 0
    one_counter = 0
    for i in range(100):
        bit = random.randrange(2)
        if bit == 1 :
            one_counter+=1
        urutan.append(bit)



    print("Urutan bilangan:")
    print(urutan)
    pi = one_counter/100
    print("pi = ", pi)
    for i in range(99):
        if urutan[i] != urutan[i+1]:
            count+=1

    count+=1
    print("count = ", count)
    pembilang = count - (2 * 100 * pi * (1 - pi))
    penyebut = 2 * math.sqrt(200) * pi * (1-pi)
    print(pembilang)
    print(penyebut)
print("P-value:" , special.erfc(pembilang/penyebut))
```

Implementasi pada bahasa Java :

```
    import org.apache.commons.math3.special.Erf;
    import java.util.Random;
    import java.lang.Math;
    public class Kripto {



        public static void main(String[] args) {
            Random random = new Random();
            int[] urutan = new int[100];
            int counter = 0;
            int one_counter = 0;
            for (int i = 0; i < 100; i++) {
                int bit = random.nextInt(2);
                urutan[i]= bit;
                if (bit == 1) {
                    one_counter++;
                }


            }
            for (int i=0; i< (urutan.length - 1);i++) {

                System.out.print(urutan[i]);
                if(urutan[i] != urutan[i+1]){
                    counter += 1;
                }
                    }
            counter +=1;
            System.out.println();

            double pi = (double) one_counter / 100;
            System.out.print("pi=");
            System.out.println(pi);


            double pembilang = counter - (2 * 100 * pi
* (1 - pi));
            double penyebut = 2 * Math.sqrt(200) * pi
* (1-pi);


            System.out.println("P-value:");

System.out.println(Erf.erfc(pembilang/penyebut));
            }
}
```

Implementasi pada bahasa C++ :

```
    #include <stdio.h>
    #include <stdlib.h>
    #include <vector>
    #include <math.h> //for erfc
    #include <time.h>
    #include <iostream>
    using namespace std;
```

```
int main () {
	vector<int> urutan;
	int count = 0;
	int one_counter = 0;
	srand( time(NULL));
	for (int i = 0; i < 100; i ++){
		int bit = rand() % 2;
		if (bit == 1) {
			one_counter = one_counter + 1;

		}
		urutan.push_back(bit);
	}

	for (int i=0; i< urutan.size()-1;i++) {
	cout << urutan[i];
	if(urutan[i] != urutan[i+1]){
	count += 1;
	}
	}
	count += 1;
	cout << endl;
	float pi = float(one_counter) /float(100);
	cout << "pi =" << pi << endl;



	cout << "count =" << count << endl;


	float pembilang = float(count) - float((2 * 100 * pi *
(1 - pi)));
	float penyebut = float(2 * sqrt(200) * pi * (1-pi));


	cout << pembilang << endl;
	cout << penyebut << endl;
	cout << "P-value =" << erfc(pembilang/penyebut)
<< endl;


}
```

Hasil dari percobaan yang dilakukan ditampilkan pada tabel hasil P-Value berikut :

Hasil dari tes ini ditunjukkan dengan tabel berisi nilai P-Value berikut:

Percobaan	Python	C++	Java
1	0.9711786469808221	0.500798	0.7270698098362818
2	0.9486419854895918	0.841481	0.6616942006581116
3	0.8349687063 58955	0.661694	0.029605794690274422
4	0.1351095376 0937222	0.737739	0.7377389114860018
5	0.9872139357396792	0.363302	0.2984592823927015

Sama seperti percobaan yang pertama, agar suatu urutan bilangan dikatakan bersifat acak adalah jika P-Value memiliki nilai lebih besar dari 0.01. Dari tabel hasi percobaan diatas, dapat dikatakan bahwa semua PRNG pada ketiga bahasa melalui tesnya dan dapat dikatakan merupakan PRNG yang baik untuk kriptografi, walaupun pada percobaan ke 3 terdapat nilai P-Value yang sangat kecil yaitu 0.02.

IV. KESIMPULAN

Dengan menggunakan *The Frequency (Monobit) Test* dan *The Runs Test* dari *A Statistical Test Suite for Random and Pseudorandom Number Generator for Cryptographic Applications* yang dipublikasikan oleh NIST, 3 *pseudorandom number generator* dari masing – masing bahasa C++, Python, dan Java digunakan untuk menghasilkan bilangan acak yang terdiri dari angka 1 dan 0 sebanyak 100 secara berurutan. Tujuan dari *The Frequency (Monobit) Test* adalah untuk menentukan apakah jumlah dari angka 0 dan 1 di dalam sebuah urutan sama seperti yang akan dihasilkan oleh *truly random number generator*. Sedangkan tujuan dari *The Runs* Test adalah untuk melihat jumlah *runs* yang terdapat pada urutan bilangan yang dibangkitkan. Hasil P-Value yang dihasilkan dari tes dan memiliki nilai lebih besar dari 0.01 menyatakan bahwa urutan bilangan acak tersebut adalah random.

Dari hasil percobaan yang dilakukan, dapat dikatakan semua lolos *the frequency test* karena semua PRNG menghasilkan P-Value yang lebih besar dari 0.01. Namun, hanya bahasa Python dan Java yang memiliki P-Value bernilai 1.0 sedangkan C++ tidak. Walaupun begitu, P-Value yang lebih besar dari 0.01 telah menyatakan bahwa PRNG pada bahasa Python, Java, dan C++ dapat digunakan untuk kriptografi dan bersifat acak (random). Begitu pula dengan *the runs test* dimana ketiga PRNG pada 3 bahasa pemrograman lolos tes dan membuktikan ketiga PRNG dapat digunakan untuk kriptografi dan bersifat random.

UCAPAN TERIMA KASIH

Penulis ingin mengucapkan terima kasih pertama – tama kepada Tuhan Yang Maha Esa atas berkat dan penyertaan-Nya,

penulis dapat menyelesaikan makalah ini. Penulis juga berterima kasih kepada orang tua yang telah memberikan bantuan moral serta doa – doa yang diberikan sehingga penulis dapat terus berusaha menyelesaikan makalah ini. Tak lupa penulis mengucapkan terima kasih kepada Dr. Ir. Rinaldi Munir, MT selaku dosen mata kuliah Kriptografi yang telah membimbing dan mengajarkan materi yang diperlukan untuk menyelesaikan makalah ini.

REFERENCES

[1] R. Andrew, dkk., "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications", NIST, April 2010

[2] http://commons.apache.org/proper/commons-math/javadocs/api-3.3/org/apache/commons/math3/special/Erf.html diakses pada 10 Mei 2019

[3] http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2018-2019/Pengantar-Kriptografi-(2019).pdf, Dibuat oleh Rinaldi Munir, Diakses pada 10 Mei 2019 Pukul 14.20

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, 10 Mei 2019

Christian Kevin Saputra - 13516073