Perbandingan SecureRandom (Java), modul secrets (Python), dan crypto.randomBytes (NodeJS)

Bervianto Leo Pratama
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
bervianto.leo@gmail.com

Abstract*—Random number generator atau pembangkit bilangan acak merupakan alat yang dapat membangkitkan suatu bilangan angka secara acak. Pembangkit bilangan acak memiliki dua jenis yaitu Pseudorandom Number Generator dan True-Random Generator, perbedaan dari kedua jenis tersebut terdapat pada bagaimana menghasilkan bilangan acak misalkan dengan pola tertentu seperti Pseudorandom Number Generator. Terdapat beberapa pembangkit bilangan acak yang memiliki kekuatan yang mirip dengan true-random generator sehingga dapat digunakan pada kriptografi. Sebagai contoh yaitu SecureRandom (Java), modul secrets (Python) dan crypto.randomBytes (NodeJS). Setiap fungsi atau modul memiliki keunikan masing-masing. Pengujian dilakukan berupa pengujian frekuensi bit, pengujian frekuensi angka dan pengujian secara visual dengan menghasilkan bit-bit per pixel. Berdasarkan hasil pengujian setiap modul atau fungsi memiliki kekuatan yang tidak jauh berbeda dan cocok untuk digunakan kriptografi. Setiap fungsi atau modul memiliki kelemahan dan kelebihan masing-masing, modul secrets dan SecureRandom lebih baik dalam bagian frekuensi Bit maupun frekuensi angka dibandingkan crypto.randomBytes. Pengujian visualisasi pada setiap fungsi menghasilkan gambar acak yang baik dan tidak terlihat berpola.

Keywords—crypto.randomBytes, modul secrets, pembangkit bilangan acak, perbandingan, SecureRandom

I. INTRODUCTION

Random number generator merupakan suatu alat yang dapat membangkitkan angka acak. Pembangkit angka acak (Random Number Generator) memiliki dua jenis yaitu Pseudorandom Number Generator dan True-Random Number Generator. Pseudorandom Number Generator memiliki pola tertentu dalam menghasilkan suatu angka acak sehingga dapat ditebak dengan melihat pola yang terdapat pada Pseudorandom Number Generator. True-Random Number Generator berbeda dengan Pseudorandom number generator karena True-Random sulit untuk ditebak angka acak yang dihasilkan dan tidak memiliki pola tertentu untuk menghasilkan angkanya.

Pseudorandom Number Generator atau disebut juga sebagai *deterministic random bit generator* menghasilkan urutan bit-bit dari sesuatu nilai inisial yang rahasia dinamakan *seed* [1]. *Deterministic random bit generator* (DRBG) dapat ditambahkan beberapa property sehingga hasilnya tidak dapat diprediksi [1], hal ini juga dinamakan *cryptographic* DRBG. Hasil yang diberikan oleh DBRG bisa saja tidak diprediksi asalkan *seed* yang digunakan terjaga kerahasiaannya dan algoritma yang digunakan juga bagus [2].

Salah satu aspek yang perlu diperhatikan untuk menggunakan suatu pembangkit nilai acak yaitu bagaimana hasilnya tidak dapat diprediksi, tidak hanya hasilnya bahkan hingga urutannya tidak dapat diprediksi [3]. Selain itu juga perlu adanya sifat *random (randomness)*, sebagai contoh koin yang memiliki kemungkinan dihasilkan sisi atas maupun bawah sebesar 1/2. Probabilitas dihasilkan salah satu sisi tersebut tidak mempengaruhi perlakuan berikutnya sehingga dapat disebutkan saling *independent* [3].

NIST memberikan suatu kumpulan metode untuk melakukan pengujian pembangkit nilai acak. Beberapa pengujian yang dilakukan berupa pengujian pada level bit. Pengujian yang dilakukan terdapat 15 metode uji [3], yaitu

1. Pengujian Frekuensi (Monobit)
2. Pengujian Frekuensi dengan sebuah Blok
3. Pengujian Eksekusi (Runs Test)
4. Pengujian untuk Longest-Run-of-Ones dalam sebuah Blok
5. Pengujian Rank Matriks Binary (Binary Matrix Rank)
6. Pengujian Discrete Fourier Transform (Spectral)
7. Pengujian Pencocokan Non-overlapping Template
8. Pengujian Pencocokan Overlapping Template
9. Pengujian “Universal Statistical” Maurer
10. Pengujian Kompleksitas Linear
11. Pengujian Serial
12. Pengujian Aproksimasi Entropi
13. Pengujian Jumlah Kumulatif (Cumulative Sums)
14. Pengujian Ekskursi Acak
15. Pengujian Varian Ekskursi Acak

Bahasa pemrograman tertentu mengimplementasikan pembangkit bilangan acak untuk keperluan kriptografi. Sebelum pembangkit bilangan acak tersebut dapat digunakan perlu dilakukan pengujian terlebih dahulu terkait *randomness* maupun *unpredictable.* Beberapa contoh implementasi dari pembangkit

bilangan acak yaitu SecureRandom (Java), modul *secrets* (Python 3.6 ke atas), dan crypto.randomBytes (Node JS).

Penelitian yang dilakukan berupa perbandingan pengujian frekuensi (saat menghasilkan bit-bit), pengujian frekuensi saat menghasilkan angka, serta pengujian berupa visualisasi menghasilkan gambar mirip dengan pengujian frekuensi dalam skala bit. Pengujian yang dilakukan terhadap SecureRandom, modul *secrets*, dan crypto.randomBytes. Pengujian dilakukan untuk melihat perbedaan dari hasil setiap pembangkit nilai acak.

II. Sekilas Mengenai Implementasi Pembangkit Angka Acak dalam Bahasa Pemrograman Tertentu

Beberapa bahasa pemrograman mengimplementasi pembangkit angka acak, seperti Java, Python, maupun NodeJS. Pembangkit angka acak yang diimplementasi berupa pseudorandom maupun pembangkit untuk kebutuhan kriptografi. Java memiliki Random (https://docs.oracle.com/javase/8/docs/api/java/util/Random.html) sebagai pseudorandom, SecureRandom (https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html) sebagai *cryptographically strong random number generator* atau pembangkit angka acak untuk kebutuhan kriptografi. Python memiliki modul random (https://docs.python.org/2/library/random.html) sebagai pseudorandom, os.urandom (https://docs.python.org/2/library/os.html#os.urandom) yang menggunakan pembangkit angka acak pada sistem operasi tertentu dan dapat digunakan untuk kebutuhan kriptogafi, sedangkan modul secrets (https://docs.python.org/3/library/secrets.html) yang dikeluarkan saat versi 3.6 juga dapat digunakan untuk kebutuhan kriptografi. NodeJS memiliki pembangkit *byte* acak dengan fungsi *crypto.randomBytes* (https://nodejs.org/api/crypto.html#crypto_crypto_randombytes_size_callback) yang merupakan pseudorandom namun dapat digunakan untuk keperluan kriptografi. Fokus utama dalam penelitian ini yaitu pada *SecureRandom* milik Java, modul *secrets* milik Python, dan *crypto.randomBytes* milik NodeJS.

A. Java SecureRandom

SecureRandom merupakan sebuah kelas yang mengimplementasikan sebuah pembangkit angka acak yang kuat untuk kebutuhan kriptografi. Berdasarkan dokumentasi yang dituliskan untuk SecureRandom dikatakan bahwa sudah dilakukan pengujian statistika dari FIPS 140-2, Security Requirements for Cryptographic Modules (https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf), bagian 4.9.1. Pada dokumentasi SecureRandom, SecureRandom menghasilkan keluaran non-deterministic, material *seed* yang tidak dapat ditebak dan urutan keluaran yang kuat, seperti yang disebutkan pada RFC 1750: Randomness Recommendations for Security (http://www.ietf.org/rfc/rfc1750.txt).

B. Modul secrets dalam Python

Berdasarkan dokumentasi pada modul *secrets*, *secrets* digunakan untuk melakukan pembangkitan bilangan acak yang kuat sebagai keperluan kriptografi dan cocok untuk mengelola data seperti password, otentikasi akun, token keamanan, dan berbagai hal mengenai kerahasiaan. Modul *secrets* memiliki berbagai fungsi untuk melakukan pembangkitan bilangan acak bahkan dapat menghasilkan bit-bit acak, angka, maupun secara acak memilih suatu elemen dalam kumpulan tertentu misalnya memilih suatu angka dari kumpulan angka. Pada dokumentasi model *secrets* terdapat panduan penggunaan dari setiap fungsinya sehingga memanfaatkan modul tersebut dengan baik.

C. Crypto.randomBytes pada NodeJS

Crypto.randomBytes merupakan fungsi untuk menghasilkan byte-byte random pada NodeJS. Berdasarkan dokumentasi mengenai fungsi tersebut, crypto.randomBytes merupakan *cryptographically strong pseudo-random data*. Fungsi crypto.randomBytes tergabung pada modul *crypto* yang merupakan modul yang memiliki berbagai fungsi berhubungan dengan kriptografi seperti *hash*, enkripsi dan lain-lain.

III. Hasil Perbandingan

Perbandingan terhadap tiga fungsi yang terdapat dalam bahasa pemrograman yang berbeda akan diuji berdasarkan tiga kategori, yaitu perbandingan frekuensi bit seperti pada *test suite* yang diajukan oleh NIST, lalu perbandingan frekuensi angka yang dihasilkan berdasarkan range tertentu dan statistikanya, serta perbandingan gambar yang dihasilkan dari fungsi pembangkit nilai acak tersebut. Percobaan menghasilkan gambar dapat dilihat sebagai contohnya pada random.org (https://www.random.org/bitmaps/). Setiap kategori diberikan kode contoh sesuai dengan bahasa pemrograman masing-masing untuk menghasilkan pengujian tersebut.

A. Perbandingan Frekuensi Bit

Sebelum melakukan perbandingan, terdapat beberapa persiapan yang dapat dilakukan, setiap bahasa pemrograman memiliki persiapannya masing-masing. Tahapan-tahapan dijelaskan per bahasa pemrograman yang digunakan. Terdapat penggunaan *library* untuk melakukan perhitungan tertentu seperti erfc yang merupakan fungsi untuk menghitung *complementary error* [3]. Penggunaan *library* bisa diganti dengan implementasi sendiri fungsi tersebut.

1) Persiapan pada Bahasa Python: Persiapan yang dilakukan dapat mengikuti tahapan berikut.

a) Menginstall Python versi 3.6 ke atas: Versi 3.6 digunakan karena modul *secrets* baru dirilis saat versi tersebut, versi sebelumnya belum memberikan modul *secrets*.

b) Menginstall library scipy: Tahapan ini merupakan opsional dan dapat diganti menjadi mengimplementasi sendiri fungsi erfc. Jika ingin menginstall *library* Scipy dengan perintah,

```
python -m pip install --user numpy scipy matplotlib
ipython jupyter pandas sympy nose
```

c) Menuliskan script pengujian: Script pengujian dapat mengikuti kode berikut. Kode dapat dimodifikasi sehingga mencatat setiap percobaan secara otomatis. Kode ini hanya untuk satu kali percobaan dan mendapat serta menampilkan 1 data.

```
from scipy import special
import secrets, math

# prepare random
list = []
sums = 0
for i in range(100):
    bit = secrets.randbits(1)
    res = (2 * bit) - 1
    sums += res
    list.append(bit)
print("LIST:")
print(list)
# test random
sa = math.fabs(sums)/10
print("Sn:")
print(sums)
print("Sabs:")
print(sa)
temp = sa/math.sqrt(2)
print("P-value:")
print(special.erfc(temp))
```

d) Melihat hasil P-value dari setiap percobaan dan catat hasilnya: Hasil *binary* yang dihasilkan juga perlu diperhatikan apakah perhitungannya sudah benar.

2) Persiapan pada Bahasa Java: Persiapan yang dilakukan dapat mengikuti tahapan berikut.

a) Menginstall Java SDK

b) Mendownload dan menyertakan library erfc: Library yang berisi erfc dapat diunduh di http://www-us.apache.org/dist/commons/math/binaries/commons-math3-3.6.1-bin.zip. Jika tautan sudah rusak, dapat dicari melalui mesin pencari.

c) Menuliskan kode untuk pengujian: Kode pengujian dapat mengikuti kode berikut.

```
package me.berviantoleo;

import org.apache.commons.math3.special.Erf;

import java.security.SecureRandom;

public class Main {

    public static void main(String[] args) {
        SecureRandom random = new SecureRandom();
        int sums = 0;
        for (int j = 0; j < 100; j++) {
            int bit = random.nextBoolean() ? 1 : 0;
            System.out.print(bit);
            int res = (2 * bit) - 1;
            sums += res;
        }
        System.out.println();
        System.out.println("SUMS:");
        System.out.println(sums);
        double sabs = Math.abs(sums) / 10;
        System.out.println("Sabs:");
        System.out.println(sabs);
        double temp = sabs / Math.sqrt(2);
        System.out.println("P-value:");
        System.out.println(Erf.erfc(temp));
    }
}
```

d) Melihat hasil P-value dari setiap percobaan dan catat hasilnya: Hasil *binary* yang dihasilkan juga perlu diperhatikan apakah perhitungannya sudah benar.

3) Persiapan pada NodeJS: Persiapan yang dilakukan dapat mengikuti tahapan berikut.

a) Menginstall NodeJS

b) Menginstall beberapa library: Library yang digunakan merupakan utilitas dan fungsi erfc. *Library* dapat diinstall melalui npm seperti berikut.

```
npm i biguint-format math-erfc
```

c) Menuliskan kode untuk pengujian: Kode pengujian dapat mengikuti kode berikut.

```
const crypto = require('crypto');
const format = require('biguint-format');
const erfc = require('math-erfc');


crypto.randomBytes(15, (err, buf) => {
  if (err) throw err;
  var ne = format(buf, 'bin');
  console.log(`Bit: ${ne}`);
  const len = ne.length;
  console.log(`Bit Length: ${ne.length}`);
  var sum = 0;
  for(var i=0; i<len; i++) {
      var res = (2*ne.charAt(i)) - 1;
      sum += res;
  }
  console.log(`Sum: ${sum}`)
  var sabs = Math.abs(sum) / Math.sqrt(len);
  console.log(`Sabs: ${sabs}`);
  var temp = sabs / Math.sqrt(2);
  var pVal = erfc(temp);
  console.log(`P-value: ${pVal}`);
});
```

d) Melihat hasil P-value dari setiap percoban dan catat hasilnya: Hasil *binary* yang dihasilkan juga perlu diperhatikan apakah perhitungannya sudah benar.

Berikut ini tabel perbandingan dari setiap bahasa dengan hasil P-value. Percobaan dilakukan sebanyak 10 kali, lebih banyak akan lebih baik untuk memperlihatkan bahwa fungsi pembangkit bilangan acak merupakan pembangkit bilangan acak yang cukup kuat untuk digunakan dalam kriptografi. P-value dengan nilai lebih dari 0.01 dapat dikatakan bahwa urutan yang diberikan secara acak.

Percobaan ke-	**Hasil P-value pada**		
	Java SecureRandom	***Python secrets.randbits***	***NodeJS crypto.random Bytes***
1	0.3173105078 6291404	0.3173105078 63	0.1441270348 160153

Percobaan ke-	Hasil P-value pada		
	Java SecureRandom	*Python secrets.randbits*	*NodeJS crypto.randomBytes*
2	0.3173105078 6291404	0.1095985833 99	0.2733216782 922982
3	1.0	0.8414805811 22	0.2733216782 922982
4	1.0	0.6891565167 79	0.5838824207 703652
5	1.0	0.5485062355	0.8551321405 847059
6	0.3173105078 6291404	0.3173105078 63	0.1003482464 6229079
7	1.0	0.3173105078 63	0.5838824207 703652
8	0.3173105078 6291404	0.4237107971 67	0.8551321405 847059
9	1.0	0.0718606382 259	0.2012426209 5772408
10	1.0	0.4237107971 67	0.4652088184 5214174

Percobaan yang dilakukan antara bahasa tertentu tidak mempengaruhi dengan bahasa lain. Hasil P-value pada SecureRandom hanya terdapat kedua kemunginkan nilai sedangkan modul *secrets* dan crypto.randomBytes mengasilkan hal yang berbeda. P-value yang sama tidak memberikan bahwa nilai *random* yang dihasilkan sama, nilai yang sama karena jumlah antara 1 dengan 0 sama. Perbandingan pada bagian ini ingin memberikan bahwa hasil yang diberikan cukup seimbang antara 1 dan 0. Berdasarkan 10 data tersebut terlihat Java SecureRandom cukup baik dalam menghasilkan bit-bit acak secara seimbang dan urutan acak yang baik. Fungsi lain masih cukup baik selain SecureRandom. Berdasarkan 10 data tersebut mendapatkan semua nilai lebih dari 0.01 sehingga masih dapat dikatakan bahwa cukup aman digunakan dan memberikan nilai acak yang baik.

B. *Perbandingan Frekuensi Angka dan Statistikanya*

Perbandingan dilakukan dengan berbagai teknik, yaitu sebagai berikut,

1. Melakukan pembangkitan angka dari 0 – 9 sebanyak 10 kali.
2. Melakukan pembangkitan angka dari 0 – 9 sebanyak 100 kali.
3. Melakukan pembangkitan angka dari 0 – 99 sebanyak 100 kali.
4. Melakukan pembangkitan angka dari 0 – 99 sebanyak 10.000 kali.
5. Melakukan pembangkitan angka dari 0 – 999 sebanyak 1000 kali.
6. Melakukan pembangkitan angka dari 0 – 999 sebanyak 1.000.000 kali.

Setiap fungsi akan diambil datanya lalu digambarkan grafiknya sehingga dapat dilihat distribusi yang diberikan. Fungsi dari Java dan NodeJS akan menuliskan data kepada file sedangkan Python akan langsung dilakukan visualisasi data tanpa menulis ke file.

1) Persiapan pada Python

Persiapan yang dilakukan mirip dengan percobaan frekuensi bit, yaitu

a) Melakukan Penginstalan Python

b) Melakukan penginstalan library: Library yang dibutuhkan yaitu *scipy*, *numpy* dan *matplotlib* (untuk grafik)

c) Menuliskan script: Kode dibagi dua tahap yaitu menghasilkan data dan menampilkan data. Tahap kedua dapat digunakan kembali untuk data dari *file* yang dihasilkan oleh bahasa lain. Kode yang diberikan ini hanya sebagai contoh untuk menghasilkan 1.000.000 data dari nilai 0 hingga 999.

```
from scipy import stats
import secrets
import numpy as np
import matplotlib.pyplot as plt

# Generate Data
mylist = []
for i in range(1000000):
    mylist.append(secrets.randbelow(1000))

# Showing Data
a = np.array(mylist)
z = stats.itemfreq(a)
n = z[:,0]
sn = z[:,1]

plt.bar(n,sn)
plt.show()
```

d) Melihat hasil distribusi dan bandingkan

2) Persiapan pada Java

Persiapan yang dilakukan yaitu sebagai berikut,

a) Menginstall Java

b) Menuliskan kode: Contoh yang diberikan dalam penelitian ini yaitu menuliskan hasil angka ke dalam suatu *file* yang digunakan oleh bahasa Python untuk menampilkan datanya. Namun tidak terbatas oleh bahasa Python, boleh dengan bahasa apapun untuk menampilkan data tersebut. Kode yang dituliskan dapat melihat contoh berikut.

```
package me.berviantoleo;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.security.SecureRandom;

public class DataNum {
    public static void main(String[] args) {
        try {
            BufferedWriter writer = new
BufferedWriter(new FileWriter("java6.txt"));
            SecureRandom random = new
SecureRandom();
            for (int j = 0; j < 1000000; j++) {
                int number = random.nextInt(1000);
```

```
writer.write(String.valueOf(number));
                writer.newLine();
            }
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

c) Melihat hasil distribusi dan bandingkan

3) Persiapan pada NodeJS

a) Menginstall NodeJS

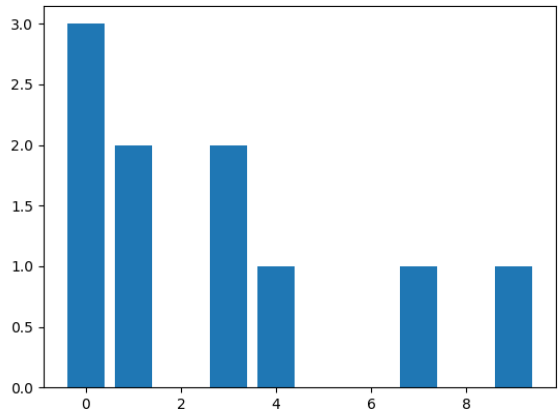
b) Menulis kode: Sebagai contoh dapat dilihat kode seperti berikut untuk menuliskan data acak kepada suatu file.

```
const crypto = require('crypto');
const fs = require('fs');

function randomRange(maximum) {
    var minimum = 0;
    var maxBytes = 6;
    var maxDec = 281474976710656;
    var randnum =
parseInt(crypto.randomBytes(maxBytes).toString('he
x'),16);
    var res = Math.floor(randnum/maxDec*(maximum-
minimum+1)+minimum);
    if (res>maximum) {
        res = maximum;
    }
    return res;
}

var numbers = []
for (var i=0;i<1000000;i++) {
    numbers.push(randomRange(999));
}
console.log(numbers)
let writeStream =
fs.createWriteStream("node6.txt");
numbers.forEach(function(element) {
    writeStream.write(element.toString());
    writeStream.write("\n");
});
writeStream.end();
```

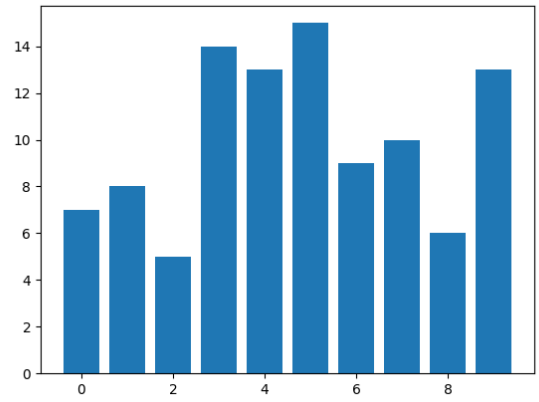
c) Melihat hasil distribusi dan bandingkan

Setelah persiapan sudah selesai, sudah saatnya untuk melihat hasilnya. Berikut ini hasil dari setiap fungsi,

1) Hasil dari modul secrets

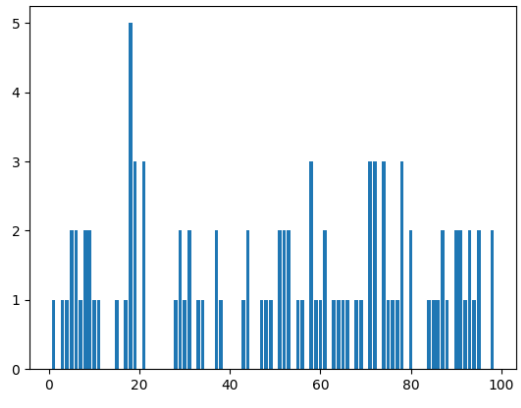


Gambar 1. Hasil fungsi secrets.randbelow pada data rentang 0 hingga 9 sebanyak 10 kali

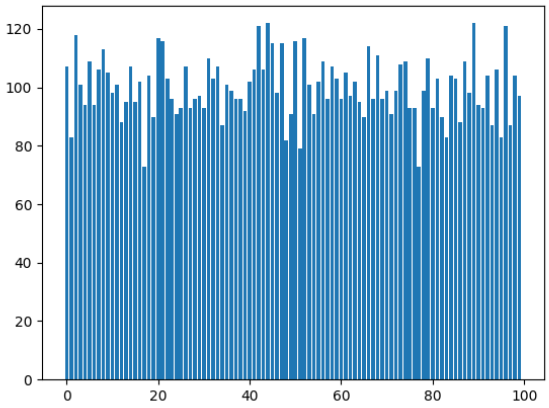


Gambar 2. Hasil fungsi secrets.randbelow pada data rentang 0 hingga 9 sebanyak 100 kali

Gambar 1 dan 2 memperlihatkan bahwa untuk memberikan nilai dengan rentang yang kecil tidak terlalu acak. Jika cukup acak distribusi yang diberikan mendekati sama dan perbedaan antara bilangan terbanyak dengan bilangan tersedikit tidak besar. Bilangan terbanyak sebaiknya tidak melebihi 50% dari jumlah total data. Namun dapat dilihat bahwa menghasilkan angka 0 pada Gambar 1 menjadi terbanyak namun saat Gambar 2 menjadi lebih sedikit sehingga ini dapat memenuhi bahwa sulit ditebak.

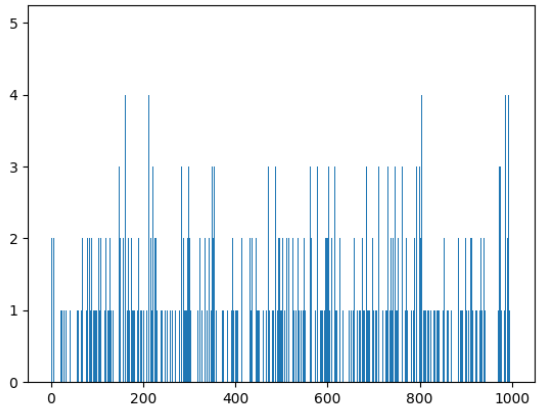


Gambar 3. Hasil fungsi secrets.randbelow pada data rentang 0 hingga 99 sebanyak 100 kali

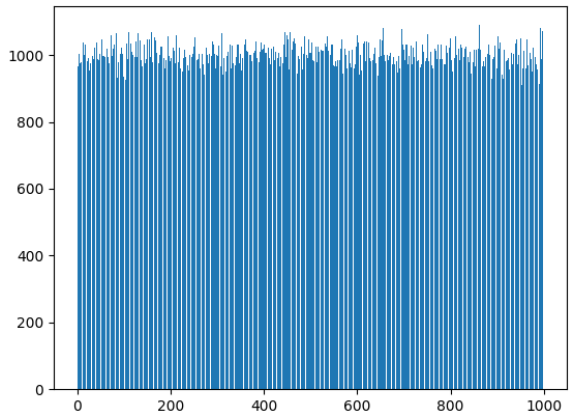


Gambar 4. Hasil fungsi secrets.randbelow pada data rentang 0 hingga 99 sebanyak 10.000 kali

Gambar 3 memperlihatkan hasil dari fungsi *secrets.randbelow* untuk rentang 0 hingga 99. Data yang dihasilkan masih tidak rata. Gambar 4 maupun Gambar 3 tidak memberikan atau menunjukkan ada suatu angka yang selalu lebih banyak sehingga dapat memenuhi kriteria sulit ditebak. Gambar 4 mulai menunjukkan kesetaraan jumlah bilangan yang dihasilkan.



Gambar 5. Hasil fungsi secrets.randbelow pada data rentang 0 hingga 999 sebanyak 1.000 kali



Gambar 6. Hasil fungsi secrets.randbelow pada data rentang 0 hingga 999 sebanyak 1.000.000 kali

Gambar 5 menunjukkan bahwa tidak terdapat angka yang dominan saat rentang diperbesar sehingga cukup baik untuk digunakkan sebagai angka acak. Saat dilakukan percobaan lebih banyak yaitu sebanyak 1.000.000 kali memberikan hasil yang relatif lebih setara jumlah yang dihasilkan sehingga akan sulit ditebak angka yang muncul antara 0 hingga 999 walaupun dilakukan sebanyak 1.000.000.

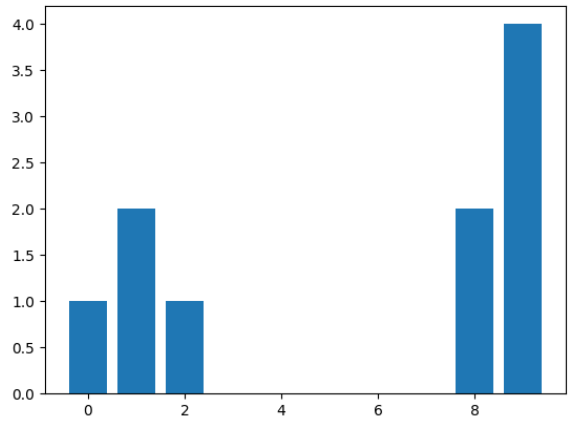
2) Hasil dari Java SecureRandom

Kode yang digunakan untuk menghasilkan data dari file yaitu seperti berikut,

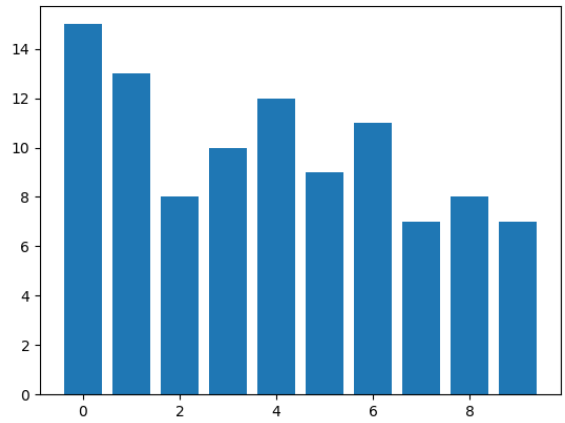
```
from scipy import stats
import secrets
import numpy as np
import matplotlib.pyplot as plt

# Load Data
f = open("java1.txt","r")
```

```
mylist = []
for anything in f.readlines():
    mylist.append(int(anything))

# Showing Data
a = np.array(mylist)
z = stats.itemfreq(a)
n = z[:,0]
sn = z[:,1]

plt.bar(n,sn)
plt.show()
```

Gambar 7 terlihat bahwa nilai 9 muncul lebih banyak namun ini tidak dapat membuktikan bahwa nilai 9 pasti muncul lagi, hal ini dapat dibantah pada Gambar 8 yang menunjukkan angka 0 menjadi lebih sering muncul saat dilakukan pembangkitan bilangan lebih banyak. Pembangkitan nilai acak pada SecureRandom untuk jumlah sedikit dan nilai sedikit tidak cukup baik dibandingkan pada modul *secure* milik Python.

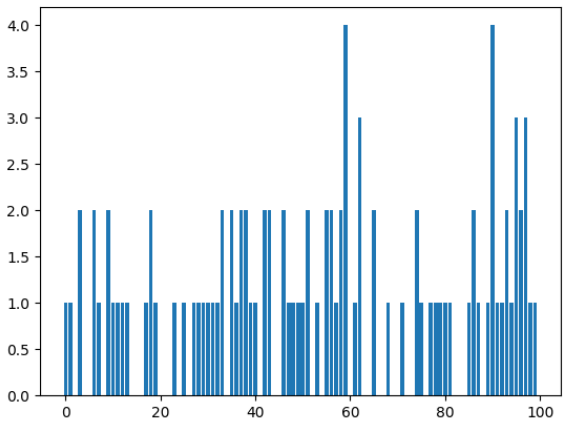


Gambar 7. Hasil fungsi randint pada SecureRandom pada data rentang 0 hingga 9 sebanyak 10 kali

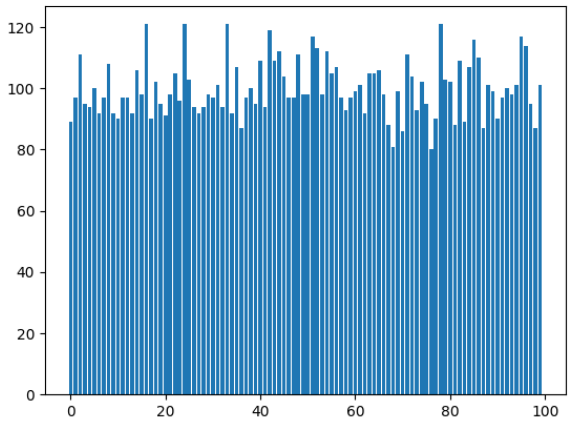


Gambar 8. Hasil fungsi randint pada SecureRandom pada data rentang 0 hingga 9 sebanyak 100 kali

Rentang 0 hingga 99 masih cukup baik dan sulit untuk ditebak angka yang sering muncul. Gambar 9 menunjukkan hasil dari rentang 0 hingga 99 untuk satu kali putaran. Walaupun tidak dapat menghasilkan angka unik, namun hasil yang diberikan tidak ada yang dominan sehingga masih dapat dikatakan bersifat *random*.

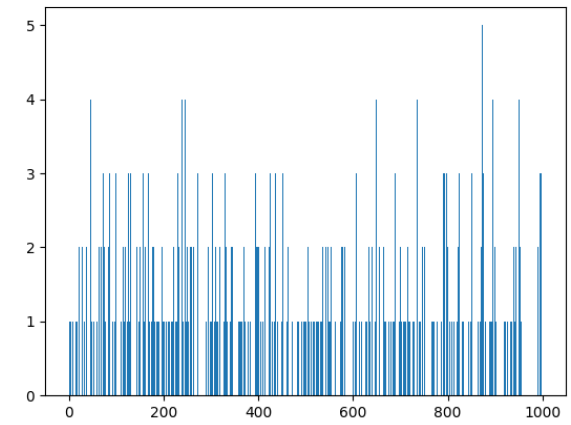


Gambar 9. Hasil fungsi randint pada SecureRandom pada data rentang 0 hingga 99 sebanyak 100 kali

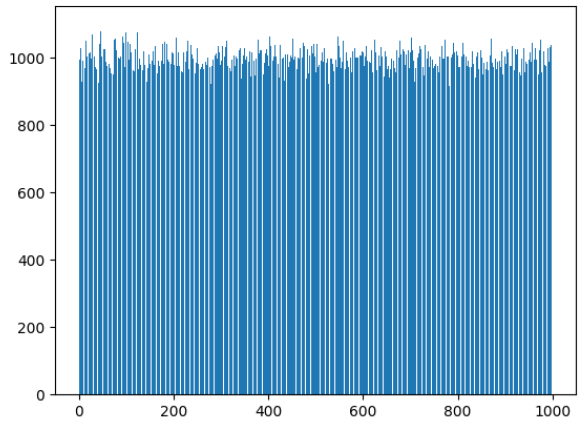


Gambar 10. Hasil fungsi randint pada SecureRandom pada data rentang 0 hingga 99 sebanyak 10.000 kali

Rentang yang lebih besar masih terdapat beberapa angka yang lebih banyak dihasilkan. Gambar 10 menampilkan untuk rentang 0 hingga 999 untuk eksekusi sebanyak 1.000.000 kali. Hasil yang diberikan cukup merata dan tidak terlihat ada yang dominan. Sehingga dapat dikatakan bahwa SecureRandom ini memiliki sifat sulit ditebak.



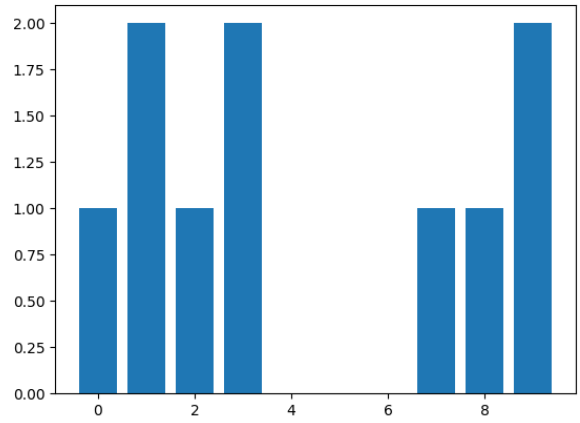
Gambar 11. Hasil fungsi randint pada SecureRandom pada data rentang 0 hingga 999 sebanyak 1.000 kali



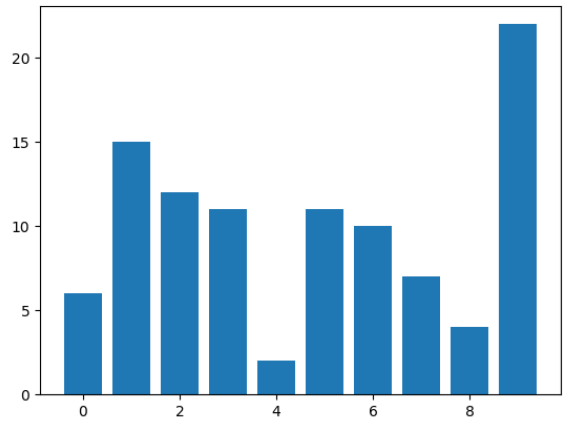
Gambar 12. Hasil fungsi randint pada SecureRandom pada data rentang 0 hingga 999 sebanyak 1.000.000 kali

3) Hasil dari NodeJS crypto.randomBytes

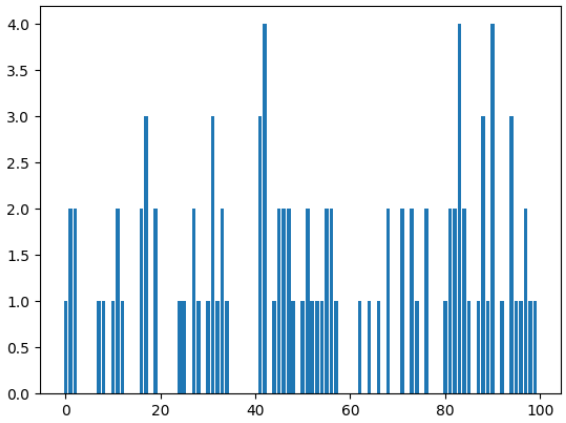
Pada bagian ini juga sama dengan bagian SecureRandom untuk menghasilkan grafis dari data yang dihasilkan oleh randomBytes. randomBytes pada NodeJS cenderung menghasilkan angka yang sama namun tidak pada salah satu angka. Angka 9 menjadi sering muncul pada Gambar 14 walaupun saat Gambar 13 terlihat sama dengan angka lain. Sementara dapat dikatakan fungsi ini menghasilkan angka yang mudah ditebak, namun perlu melihat untuk rentang yang lebih luas. Rentang 0 hingga 99 masih terlihat ada angka yang dominan seperti Gambar 16. Rentang 0 hingga 999 cukup membaik dalam pembangkitan angka acaknya walaupun tidak sebaik pembangkit angka acak yang sebelumnya.



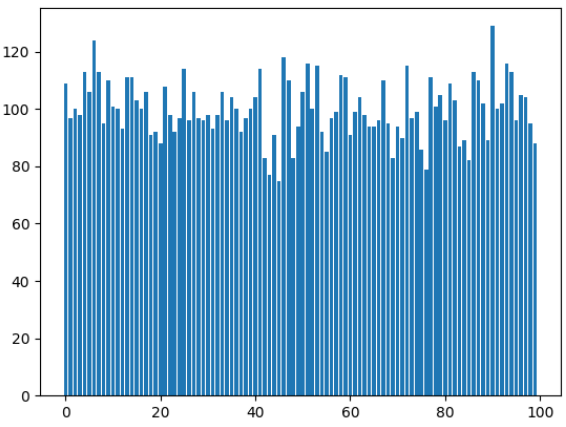
Gambar 13. Hasil crypto.randBytes pada data rentang 0 hingga 9 sebanyak 10 kali



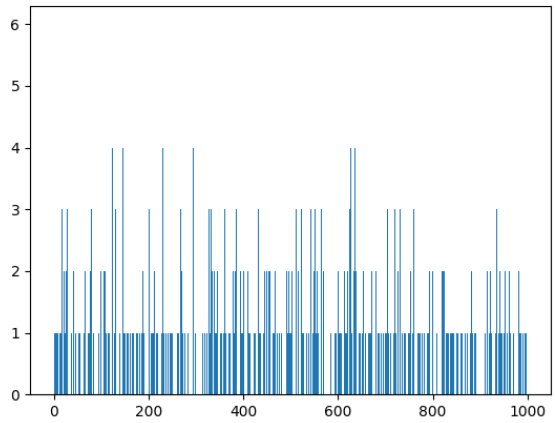
Gambar 14. Hasil crypto.randBytes pada data rentang 0 hingga 9 sebanyak 100 kali



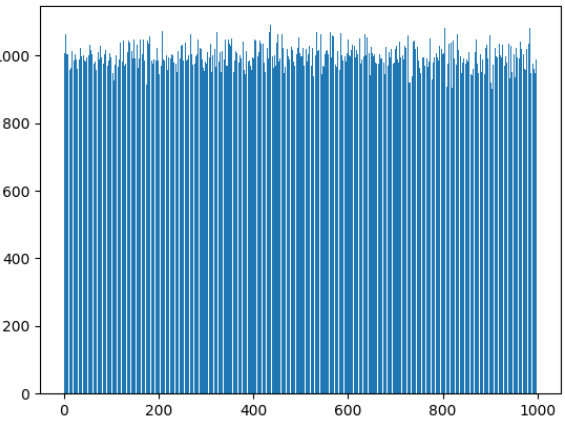
Gambar 15. Hasil crypto.randBytes pada data rentang 0 hingga 99 sebanyak 100 kali



Gambar 16. Hasil crypto.randBytes pada data rentang 0 hingga 99 sebanyak 10.000 kali



Gambar 17. Hasil crypto.randBytes pada data rentang 0 hingga 999 sebanyak 1000 kali



Gambar 18. Hasil crypto.randBytes pada data rentang 0 hingga 999 sebanyak 1.000.000 kali

Banyak faktor yang mempengaruhi pembangkit bilangan acak tidak mendapatkan hasil yang lebih, misalnya penerapan bilangan acak yang lebih baik. Seperti crypto.randBytes yang tidak secara langsung menghasilkan bilangan dan melalui fungsi perantara yang dibuat sendiri.

C. Perbandingan Gambar yang Dihasilkan

Secara umum perlu mempersiapkan terlebih dahulu OpenCV untuk menampilkan gambar sehingga gambar dapat divisualisasikan. Setiap bahasa memiliki implementasi masing-masing. Kode untuk melakukan percobaan dapat dilihat pada bagian masing-masing. Perbandingan pada bagian ini terinspirasi dari situs random.org yang dapat memberikan gambar acak berwarna hitam putih.

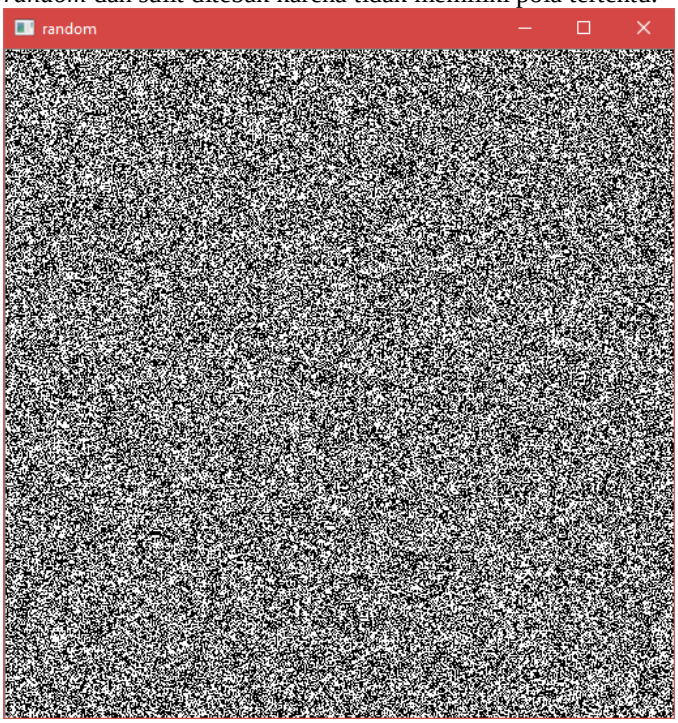
1) Modul secrets

Kode yang dapat digunakan untuk menghasilkan gambar acak pada Python seperti berikut,

```
import numpy as np
import cv2
import secrets
```

```
# Generate Pixel
img = np.zeros((512,512,1), np.uint8)

for i in range(512):
    for j in range(512):
        if secrets.randbits(1) == 1:
            img[i,j] = 255

# Show Image
cv2.imshow('random', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Gambar 19 menunjukkan hasil dari kode tersebut. Gambar 19 menunjukkan bahwa modul *secrets* tersebut sudah cukup *random* dan sulit ditebak karena tidak memiliki pola tertentu.

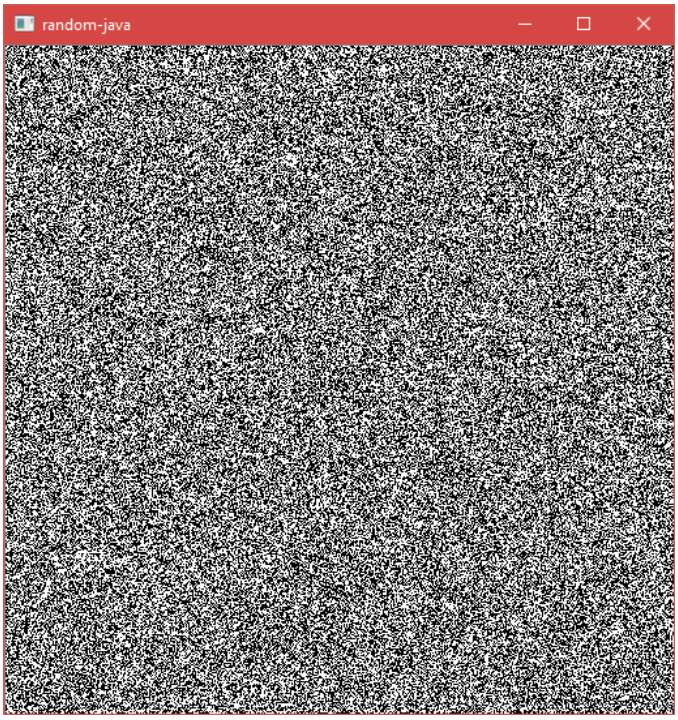


Gambar 19 Hasil Gambar Acak pada Python

2) Java SecureRandom

Data yang digunakan untuk memberikan nilai acak setiap piksel dilakukan dengan menuliskan kepada file dengan kode berikut,

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.security.SecureRandom;

class RandomBitTest {
    public static void main(String[] args) {
        // Prepare Bit Random
        try {
            BufferedWriter writer = new
BufferedWriter(new FileWriter("image-java.txt"));
            SecureRandom random = new SecureRandom();
            for (int j = 0; j < 262144; j++) {
                int bit = random.nextBoolean() ? 1 : 0;
                writer.write(String.valueOf(bit));
                writer.newLine();
                }
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



Gambar 20 Hasil Gambar Acak pada Java

Gambar 20 terlihat bahwa gambar masih terlihat acak tidak dapat dilihat adanya pola dalam gambar tersebut.

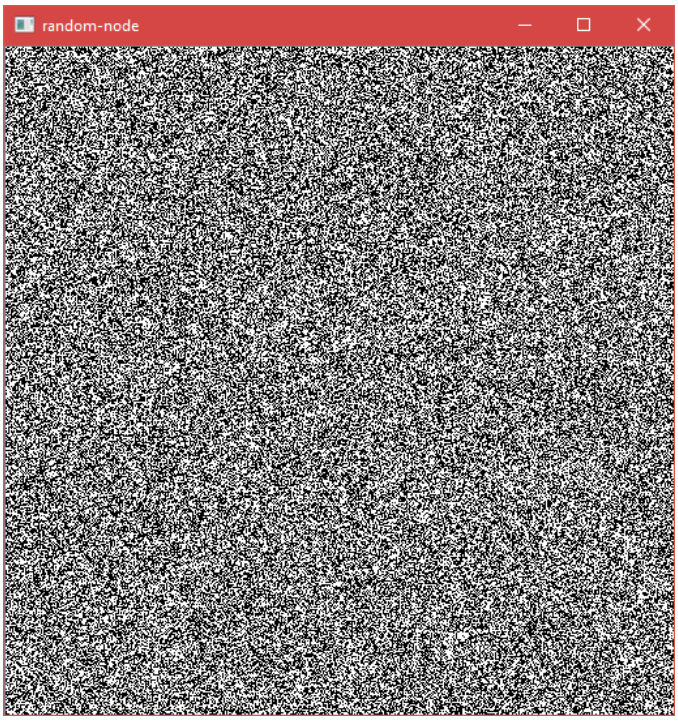
3) randomBytes Node JS

Data yang digunakan untuk memberikan nilai acak setiap piksel dilakukan dengan kode berikut,

```
const crypto = require('crypto');
const format = require('biguint-format');
const fs = require('fs');
```

```
crypto.randomBytes(32768, (err, buf) => {
  if (err) throw err;
  var ne = format(buf, 'bin');
  const len = ne.length;
  console.log(len);
  let writeStream = fs.createWriteStream("image.txt");
  for(var i=0; i<len; i++) {
      writeStream.write(ne.charAt(i));
      writeStream.write("\n");
  }
  writeStream.end();
});
```

Dari data tersebut lalu dibaca oleh Python untuk menampilkan data tersebut melalui OpenCV. Gambar 21 memperlihat hasil dari data yang diberikan melalui kode penghasil bilangan acak.



Gamber 21 Hasil Gambar Acak pada NodeJS

IV. Simpulan dan Rekomendasi

Hasil yang diberikan oleh setiap fungsi yang diuji sudah memberikan nilai acak yang sesuai dengan kriteria pembangkit bilangan acak. Dalam menghasilkan bit-bit, SecureRandom memiliki keunggulan yang cukup baik dibandingkan yang lain karena mendapatkan *score* yang besar. Dalam menghasilkan angka acak, crypto.randomBytes kurang memberikan bilangan acak yang baik atau kurang merata. Menghasilkan visualisasi dari bit-bit tidak memberikan perbedaan pada setiap gambar. Setiap fungsi memberikan hasil gambar yang tidak berpola. Secara umum dapat dikatakan setiap fungsi tidak dapat ditebak, tidak memberikan pola, serta memberikan hasil yang acak.

Penelitian ini dapat dilanjutkan dengan melakukan pengujian bagian lain pada 15 kumpulan kasus uji yang dapat digunakan. Penelitian ini juga dapat dilanjutkan dengan melakukan pengujian lebih banyak tidak terbatas 10 data seperti yang dilakukan pada pengujian frekuensi bit.

Acknowledgment

Puji Syukur kepada Tuhan Yang Maha Esa sehingga *paper* ini dapat diselesaikan dengan baik. Terima kasih kepada Rinaldi Munir yang sudah memberikan materi pada mata kuliah IF4020 Kriptografi terutama materi mengenai pembangkit angka acak. Terima kasih kepada pengurus *random.org* melalui situs tersebut yang sudah memberikan inspirasi beberapa hal untuk melakukan pengujian terhadap pembangkit bilangan acak.

References

[1] B. Elaine, “Recommendation for Random Number Generation Using Deterministic Random Bit Generators”, NIST, Juni 2015

[2] B. Elaine, “Recommendation for Key Management, Part 1: General”, NIST, Januari 2016

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