

# Simple Algorithm to Generate Random Number Based on Image

Steve Andreas Immanuel - 13517039

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jl. Ganessa 10 Bandung 40132, Indonesia

13517039@std.stei.itb.ac.id

**Abstract**—Randomness is widely used in real life more than we realize. In science, art, some simple experiments, and more importantly computer simulation, cryptography. Computer simulation uses random number to make sure that the simulation is really applicable and the result really holds up to what happen in real life. In cryptography, one of the application of random number is to generate random password that is safe and secured. Therefore, such algorithm is required to generate random number that is not just seem random, but really random. This paper shows one of the simple algorithm that can be used to generate such random number

**Keywords**—Random number, computer simulation, cryptography, algorithm.

## I. INTRODUCTION

Randomness is something that we cannot predict, contains no patterns or regularities whatsoever. One of the main usage of randomness lies in random number. As said before, random number can be very useful in computer simulation and cryptography. In real life, to generate random number, we can simply ask several people to choose a number from a certain range. The result should be totally random. However, in computer, we can't really do such think. To achieve totally random number, computer usually has predetermined function which is used. That predetermined function makes the result of random number not totally random, in fact hackers can detect the pattern rather easily. All they need to do is get some number samples and then their cracking algorithm can show them the predetermined function.

Dealing with these kind of problems, many cyber security company try to develop such algorithm that the randomness cannot be detected. There are two kind of random number that computer can generate. One is pseudo-random number and the other is true random number.

## II. BASIC THEORY

One of the main application of number theory in discrete mathematics is to generate random number. As mentioned before, there are two kind of random number. They are pseudo-random number and true random number. Pseudo-random number can be generated using pseudo-random number generator whereas true random number can be generated using true random generator.

### A. Pseudo-random Number Generator

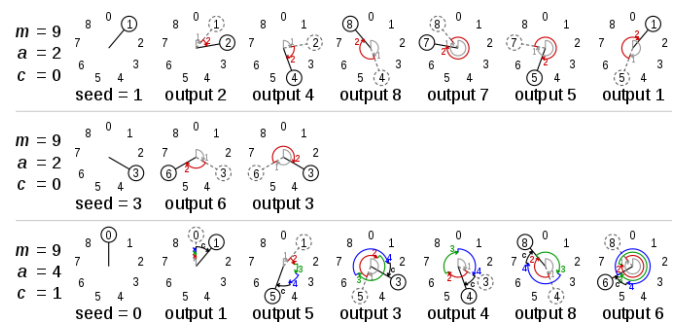
Also known as deterministic random bit generator, pseudo-random number generator is a program written for, and used in, probability and statistics applications when large quantities of random digits are needed [4]. Pseudo-random generator works based on certain predetermined function and seeds. Seeds are value that is also predetermined in order to generate the number using the predetermined function. The weakness of this kind of generator is that you will get the same random number every time you give the same input, thus the number generated is not totally random.

One of the simplest algorithm for pseudo-random number generator is called linear congruential generator (LCG). Linear congruential generators use this formula:

$$r_{n+1} = (a \times r_n + c) \text{ mod } m$$

Where:

- $r_0$  is a seed.
- $r_1, r_2, r_3, r_4, r_5, \dots$ , are the generated random numbers.
- $a, c, m$  are predetermined constants.



**Fig. 1** Linear Congruential Generator (LCG)

Source:

[http://www.wikiwand.com/en/Linear\\_congruential\\_generator](http://www.wikiwand.com/en/Linear_congruential_generator)

As shown above the linear congruential generator needs a seed and some constant predetermined in order to generate number. This makes the  $r_n$  and  $r_{n+1}$  has certain connectivity with one another. Hence, anyone who knows the formula would easily predict the resulted generated number. That is the reason why this kind of algorithm is called pseudo-random number generator, because the result is not totally random.

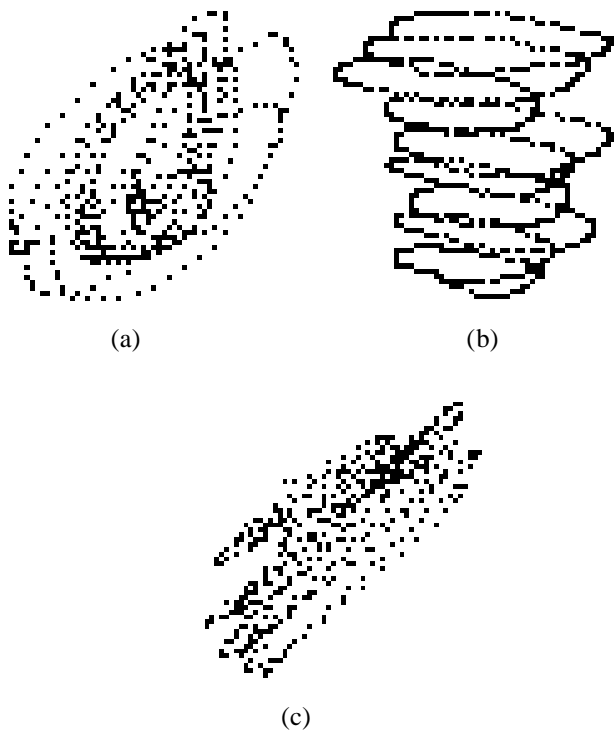
### B. True Random Number Generator

True randomness is impossible to be achieved by any kind of Turing Machines in theory [5]. That is why in order to generate true random generator, additional hardware is required by the computer to access an external source which the result then processed to generate such randomness.

Without the external source, all computer does is merely deterministic. As shown in the previous section, pseudo-random number generator needs predetermined function and predetermined seed in order to work.

The external source for the true random number generator could be anything. The more random the better.

One example of an external source is mouse movement. Based on a study [2], mouse movement could be a great source to generate random number. The background for the use of mouse movement is that it is cheap, considerably fast, and users don't need to buy additional hardware to make it work. Fig. 2 shows the tracked mouse movement done by several users.



**Fig. 2** Images Modeling Mouse Movements

(a) first image of user A (b) first image of user B (c) first image of user C [2]

The resulted tracked movement is then processed through several steps and then can be used to generate random number.

Another great example of external source is images of lava lamp. The movement part of lava lamp is truly random that we cannot predict, which is why it is so great to extract the randomness.



**Fig. 3** Lava Lamp

Source: <https://blog.cloudflare.com/randomness-101-lavarand-in-production/>

Another example of external source is physical phenomena such as thermal noise, atmospheric noise, radioactive decay, even coin-tossing or dice-tossing. [2]



**Fig. 4** Coin Tossing and Dice Tossing

Source:

<https://www2.palomar.edu/users/warmstrong/coinflip.htm>

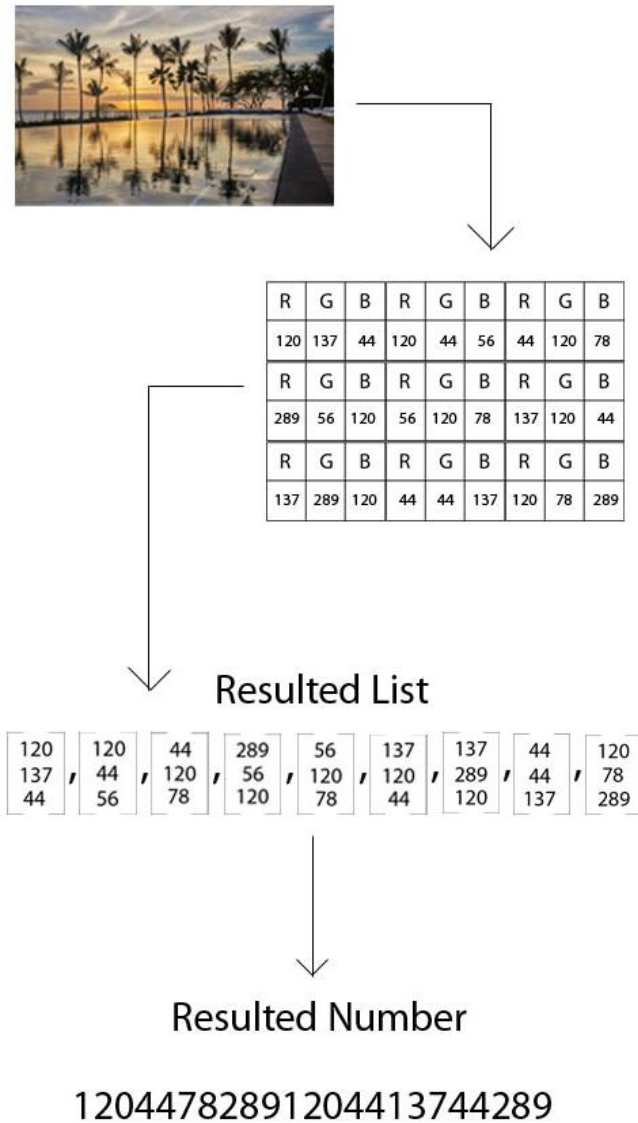
In this paper, the algorithm to generate random number is very simple yet pretty effective. The main reason for the use of image is that because all images are unique which is why it makes sense to use it.

### III. IMPLEMENTATION OF THE ALGORITHM

#### A. The Way it Works

The algorithm that the author develops is simply extract the RGB from each pixel of an image, then develop such random sequence of number which then processed.

The fact that the value of a pixel is based on the color of the image makes it pretty random and it will differ from one image to another. Fig. 5 shows the visualization of how the algorithm works.



**Fig. 5** Visualization of The Algorithm  
Source: author's document

All of the pixel values are then appended into a long number sequence. To generate random number from this sequence, nine consecutive numbers will be selected and processed. The reason why the author take only nine numbers is considering the size of an integer is only up to  $2^{31}-1$  (10 digits).

#### B. The Algorithm

The algorithm is developed using python 3.7.0. Using the Python Imaging Library (PIL), the program then extracts all the image value of an image. Below is the pseudo-code.

```

from PIL import Image

def removerpeated(X):
    loop=True
    while(loop):
        changed=X
        for i in range(0,10):
            if (i==0):

                changed=changed.replace('000000000','')
            else:

                changed=changed.replace(str(i*111111111),
                ,'' )

            if(changed==X):
                loop=False
            else:
                X=changed
        return changed

filename =
'F53402_FW18_fvqz_abisko_view_2_21.jpg'
image = Image.open(filename)

allpix=list(image.getdata())

sequenceseries=''

for i in range(0,len(allpix)):
    modthree=i%3
    sequenceseries+=str(allpix[i][modthree])

sequenceseries=removerpeated(sequenceseries)
print(sequenceseries)

```

First, the program will open the desired image. Then, it extracts all the pixel value from the image and put them to a list called `allpix`. Because each pixel contains three value (Red, Green, and Blue), each element of `allpix` will then contains three values as shown in Fig. 5. The program will iterate for every value in the list and take the pixel value based on modulo three (to increase the randomness). The value taken is then appended to a string variable called `sequenceseries`. When the iteration has finished, `sequenceseries` would become a very long string of random number sequence that depends on the image size. The final step is to remove any unwanted repeated number. Repeated number could exist when certain area of the image contains only one or two colors.

Fig. 6 shows the image that the author use to generate random number.



**Fig. 6** Sample Image That is Used

Source: <https://www.fjallraven.com/shop/fjallraven-abisko-view-2-F53402/>

From the image in Fig. 6, the program will generate sequence of random number that is shown in Fig. 7.

```

Command Prompt - python imageprocess.py
6790358785521017525127184575943057834110010449794314110137361
1910928636537617287146116731811701191581456111089391101002811
4160741611381563595290459921146271772861562187615286370496216
5593508767153622172873964856980444373414776644775735613213170
1401405275110345680367964165386717266974610103134801421233170
7096988418888106559353602360702339380788132055591051307713110
9591048155111501579945170684310058277977639588491231448210169
2810372176374501481042779674286592498521176985491784871473897
7930908422798607051275101208289561011088312710248101765475022
9411460115630655905252583740312305830664413014214010415811054
6220118014811616917251115835264685443352552757622663594389852
6989243961033428191340402034362463715410187325653215790328183
2061671513416357899733103111567986671331574372560323962154143
667477305280711371030699418112113218911572112651855038865230
6040175565347856191121658791640183038433212386114141411201045
9125864911882471311176710075439259066119114171109146179231541
5280104713011210640961326211810245858780110653789520243215728
1541351104511810972186168681001105811396651038282945931515439

```

**Fig. 7** The Sequence of Number Generated from Fig. 6

Source: author's document

The resulted number is truly random as it doesn't depend on any predetermined function or seed.

But this very long sequence of number is not what we wanted to have. So in order to generate number from certain range, the author use algorithm below.

```

generated=randomnumber[i]
generated%=(high-low+1)
generated+=low
print('Generated number :
',generated)
i+=1
else:
print('All sequence has been
used, please use new image.')
elif(userinput==2):
low=int(input('Insert lowest number :
'))
high=int(input('Insert highest number
: '))
result=[]
for j in range(1,1001):
generated=randomnumber[i]
generated%=(high-low+1)
generated+=low
i+=1
result.append(generated)
for j in range(low,high+1):
print(j, ' :
',result.count(j),'\n')

```

Using the algorithm above, the author can generate random number in a certain range and test the result.

#### IV. EXPERIMENT AND ANALYSIS

##### A. Experiment

The author tested the developed algorithm to check the randomness of the resulted generated number. At first glance, the resulted number seems to be truly random, but upon further examination there are some regularities in the resulted number. To test the randomness that is generated, the author generates one thousand random number in certain range and then count how many times each number appears in the result.

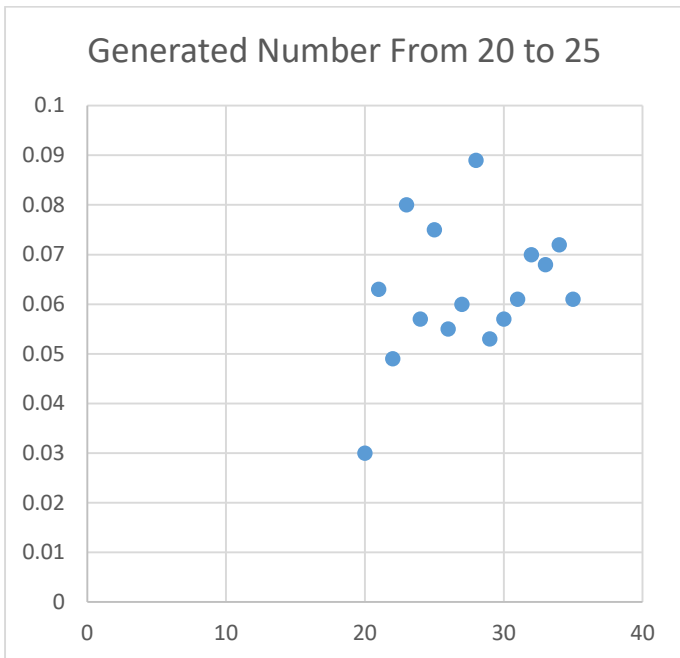
The algorithm to do so is also provided in the previous algorithm. From the result, a graph is made to show whether the distribution of number generated even or not. The X-axis represents the number that is generated, whereas the Y-axis represents the probability of the number generated to appear in the result. If the distribution is even, the graph is expected to form a straight horizontal line.

Fig. 8 shows the randomness test result in range 20 to 35. From Fig. 8, even though it is not a straight horizontal line, it shows a pretty decent distribution of the generated number.

```

i=0
userinput=1
randomnumber=[]
while (i+9<len(sequenceseries)):
randomnumber.append(int(sequenceseries[i:i+9]
))
i+=9
i=0
while (userinput!=3):
userinput=int(input('1. Generate Number\n2.
Test Result\n3. Exit\n'))
if (userinput==1):
low=int(input('Insert lowest number :
'))
high=int(input('Insert highest number
: '))
if (i<len(randomnumber)):

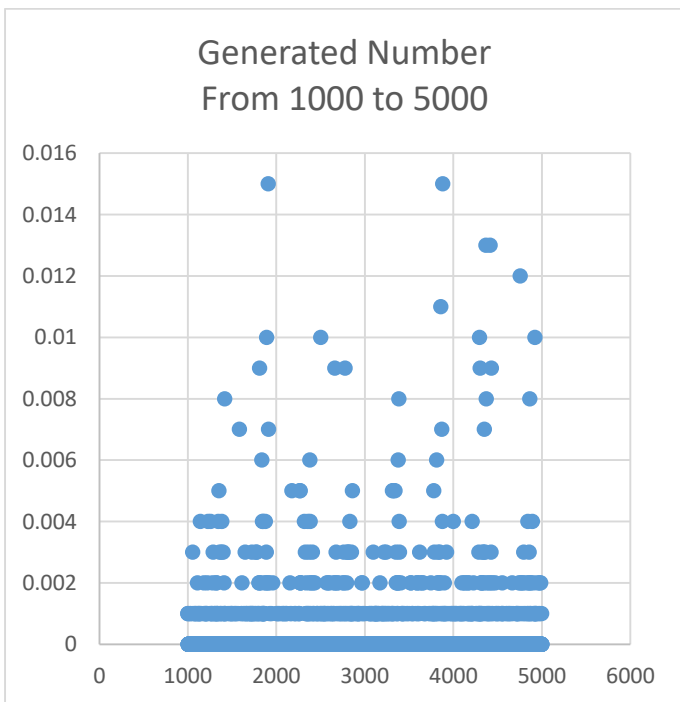
```



**Fig. 8** Graph Probability of Appearance of Numbers from 20 to 35 in the Generated Result  
Source: author's document

However, problem arises when the program is used to generate number in a big range. For instance, when the program is used to generate number from range 1000 to 5000, there are certain regularities that occurs.

As shown in Fig. 9, certain numbers are prone to appear more than another number. The graph fluctuation shows that the resulted generated number are not very random.



**Fig. 9** Graph Probability of Appearance of Numbers from 1000 to 5000 in the Generated Result  
Source: author's document

### B. Analysis

The algorithm gives some good results when generating number in small range. This could be very useful as the algorithm is very simple and easy to understand.

However, when the algorithm is used to generate number in big range, the results are not very good. There are a lot of factors that could make this happen. These are the main factor that the author thinks could affect the result :

#### 1. The Image Itself

The variety of color in image has direct effect to the result of the program. This is because the program merely takes the pixel value and processed them.

For instance, an image that consists of many different colors will theoretically have a better result than an image that consists of some colors.

#### 2. The Color Distribution of the Image

This is almost the same as the previous reason. Even though an image consists of many different color, if the same color converges in certain area of the image, it will make the result less random. This is why in the real complex algorithm, the randomness of an image is not extracted from the pixel value. Instead, it is extracted from the noise of the image, exposure, and many other things. The reason is because a slight change in those aspects have big impact in the result, thus can increase the randomness.

#### 3. The Algorithm to Generate Number

The way the author generates random number after acquiring the sequence random number is simply doing a modulo operation to the number. The base of the modulo is the highest range minus the lowest range plus one. This will create certain regularities because the base of the modulo of certain range can be the same as another range.

For example, when generating number from range 1 to 5, the base of the modulo will be 5 ( $5-1+1$ ). When generating number from range 101 to 105, the base of the modulo will also be 5 ( $105-101+1$ ).

The program also has some other weaknesses. One of the main one is the time execution. This is because when processing image in big resolution (1920x1080, or higher), the iteration to get all pixel value from the image will be repeated so many times. The sample image that is used is also very important in this algorithm. To get the best result, it is recommended to use image that has a lot of color variety.

## V. CONCLUSION

In conclusion, the algorithm to generate random number that is introduced in this paper is quiet powerful when dealing with small data range. However, the algorithm is not very effective when being used to generate data in big range. For further development, it is best to use not only the pixel value of image but also the noise, exposure, and so on in order to generate true random number.

## VII. ACKNOWLEDGMENT

Praise to God Almighty, for the presence of His mercy and grace, so that the author has the chance to create and complete this paper. The author would also like to thank Dr. Judhi Santoso as the author's lecturer in IF2120 Discrete Mathematics class for his guidance and knowledge that has been given. Last but not least, the author would like to thank all family members and friends for their support in making this paper.

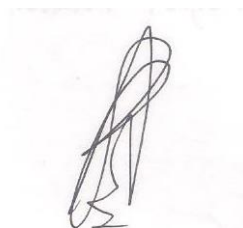
## REFERENCES

- [1] Rongzhong Li. *A True Random Number Generator Algorithm Using Digital Camera Noises Under Varying Lighting Conditions*. WakeSpace [Online]. Available from : [https://wakespace.lib.wfu.edu/bitstream/handle/10339/62642/Li\\_wfu\\_0248M\\_10943.pdf](https://wakespace.lib.wfu.edu/bitstream/handle/10339/62642/Li_wfu_0248M_10943.pdf) [Accessed 8<sup>th</sup> December 2018]
- [2] Qing Zhou, Xiaofeng Liao, Kwok-wo Wong, Yue Hu. *A True Random Number Generator Based on Mouse Movement and Chaotic Cryptography*. ResaearchGate [Online]. Available from : [https://www.researchgate.net/publication/222856978\\_A\\_true\\_random\\_number\\_generator\\_based\\_on\\_mouse\\_movement\\_and\\_chaotic\\_cryptography](https://www.researchgate.net/publication/222856978_A_true_random_number_generator_based_on_mouse_movement_and_chaotic_cryptography) [Accessed 7<sup>th</sup> December 2018]
- [3] [https://rosettacode.org/wiki/Linear\\_congruential\\_generator](https://rosettacode.org/wiki/Linear_congruential_generator) [Accessed 8<sup>th</sup> December 2018]
- [4] <https://whatis.techtarget.com/definition/pseudo-random-number-generator-PRNG> [Accessed 8<sup>th</sup> December 2018]
- [5] <https://cs.stackexchange.com/questions/7729/how-can-it-be-detected-that-a-number-generator-is-not-really-random> [Accessed 8<sup>th</sup> December 2018]
- [6] <https://lemire.me/blog/2017/08/22/cracking-random-number-generators-xoroshiro128/> [Accessed 8<sup>th</sup> December 2018]
- [7] <https://crypto.stackexchange.com/questions/43115/how-can-i-extract-randomness-from-a-jpeg-file> [Accessed 8<sup>th</sup> December 2018]
- [8] P. Murali, R. Palraj. *True Random Number Generator Method Based on Image for Key Exchange Algorithm*. Available from : <https://pdfs.semanticscholar.org/f623/539dd7905e54935e3215686f9856ec544f5c.pdf> [Accessed 8<sup>th</sup> December 2018]

## 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, 9 Desember 2018



Steve Andreas Immanuel - 13517039