Enhancing the Security of Cryptographic Systems by Pseudo-Random Number Generation Algorithms

Evelyn ENESCU
Faculty of Electronics, Telecommunications and Information Technology,
University Politehnica of Bucharest, Romania
evelyn.enescu@stud.etti.upb.ro

Abstract

Pseudo-random numbers play an indispensable role in the design of encryption systems, such as public and private key flow. The efficiency of crypto systems is directly proportional to the quality of the secret key generated using a random number generation algorithm. In this paper, the efficiency and applicability of a modified Linear Congruential Generator (LCG) type algorithm will be presented to increase the rate of occurrence of numbers and tend as much as possible to a truly random number. Moreover, it will be integrated into a graphical interface, which can later be integrated into the security of a larger application or even a website.

Index terms: cybersecurity, encryption systems, hazard, Linear Congruential Generator, token.

1. Concepts within cybersecurity

Cybersecurity is the discipline of preventing theft, damage, and unauthorized access to computer systems, networks, and data. Along with cybersecurity, "safety in operation" refers to making sure that computer systems and networks are run securely and safely, and that any potential risks or hazards are found and eliminated [1].

Reliability is another important concept in cybersecurity and safety in operation. It describes a system's capacity to consistently and flawlessly carry out its intended purpose over time. A trustworthy system is one that one can rely on to perform as intended, without unanticipated downtime or other problems.

Other concepts that are relevant to cybersecurity and safety in operation include resilience which refers to the ability of the system to recover fast after a security breach, but also the technique of encryption which protects data and communications from unauthorized access or interception, and other more. All these concepts are critical to ensuring the security, safety, and reliability of computer systems and networks, and to protecting against cyber threats and other risks.

To prevent risks there are some assessments that can be done, such as Fault Tree Analysis (FTA), Failure Mode Effect Analysis, Attack Tree Analysis (ATA) or Probability Risk Assessment (PRA) [2].

Depending on what the functionality is each method has its purpose. PRA is a method of assessing the probability and consequences of potential risks, assessing the likelihood of each hazard and developing strategies to manage them. Attack Tree Analysis (ATA) is a method of analyzing potential security threats by identifying the various attack paths that an attacker might use to gain unauthorized access to a system. FTA is a method of analyzing potential failures in a system identifying the potential causes of a failure, analyzing the likelihood and severity of each potential

cause, and developing strategies to prevent them. FMEA involves identifying the potential failure modes, analyzing the effects of each potential failure mode [2].

In this paper is used an LCG - Linear Congruential Generator algorithm to generate a 6-digit token with the goal of integrating it into the security of a larger software application.

2. Random number generators

A pseudo-random number generator (PRNG) is an algorithm that produces a series of numbers that mimic the properties of random numbers. These numbers are not completely random and are determined by an initial value called the seed.

Random number generators (PRNGs) are widely used in various applications such as simulations, electronic games, and cryptography. For cryptographic purposes, PRNGs must generate random outputs that cannot be derived from previous outputs.

Random number generators (PRNGs) are important to ensure that the result is reliable and accurate. Despite this, John von Neumann warned of the dangers of misinterpreting PRNGs as true random generators, saying "Anyone who relies on mathematical methods to generate random numbers commits a sin" [3]. A thorough analysis of the mathematical properties of the PRNG must be done to ensure that the output is sufficiently close to random for its intended use.

It is important to remember that even the best PRNGs cannot perfectly mimic "true" randomness. As such, for applications that require an extremely high degree of randomness, such as cryptography or scientific studies, hardware random number generators should be used.

A linear congruential generator (LCG) is an algorithm that produces a series of pseudo-random numbers based on basic arithmetic operations. The result is a sequence of pseudo-random numbers calculated with a recursive linear equation [4].

$$X_{n+1} = aX_n + c \bmod m \quad (1)$$

Hence the following conditions must always be respected:

$m > 0$ (the modulo number)

$0 < a < m$ (the multiplicator)

$0 \leq c < m$ (the increment)

$0 \leq X_0 < m$ (the seed)

LCG is defined by the following parameters: an initial base value (also known as a start value), a multiplier, an increment, and a modulus. The seed is the starting point of the sequence, the multiplier and the increment are two values that affect the output, and the modulus is used to terminate the output if it exceeds a certain maximum value. Once the initial values have been set, the LCG algorithm works by taking the current number in the sequence and multiplying it by the multiplier; then adding the increment to it; and finally taking the remainder of the result when divided by the modulus. This new number then becomes the next in the sequence. The same process is repeated until all the desired numbers in the sequence have been generated.

If the increment will take the value 0 then it will be called Multiplicative Congruential Generator (MCG) or Lehmer RNG [5].

The power of random generation depends on the selection of the parameters, so if the parameters have been chosen with very small values it is relatively easy to deduce the resulting sequence. For example, if a=1 and c=2 are chosen, the result will be a two-step numerator with a fairly large length, but of course it will not be random.

There are three options for choosing the parameters [6]:

1. c=0 and m a prime number

In this case, if the multiplier "a" is chosen to be an integer primitive modulo "m", then this is the original construction of the Lehmer random number generator. Its period is equal to "m-1" and the initial state must be an integer between 1 and "m-1".

2. c=0 and m a power of his

The most used values for the multiplier in this case are either 232 or 264, because the result is calculated very efficiently by simply truncation of the binary representation, thus it is no longer necessary to calculate the most significant bits. However, the approach comes with its drawbacks, as the least significant bits have a smaller period than the higher bits, so the resulting values will be able to sweep within a very small range of values. For example, if a=5 and m=8, the first bit will never change and the second bit will alternate between two states.

3. c≠0

If the values for all 3 parameters are chosen correctly, one can repeat the value of the sequence with a period m. The realization of this event will happen if and only if:

1. c and m are prime to each other
2. a-1 is divisible by all prime factors of m
3. a-1 is divisible by 4 if m is also divisible by 4

These 3 requirements are known as the Hull-Dobell theorem and work great when m is a prime number to a high power like 282, but if m were a number that is not squared or at least once, it would allow a single value of 1 and would lead to a very poor result. Although the Hull–Dobell theorem gives a maximum period, it is not sufficient to guarantee a good generator. For example, it is desirable that a − 1 is not divisible by more prime factors of m than necessary. Thus, if m is a power of 2, then a − 1 should be divisible by 4 but not divisible by 8, $a = 5 \; mod \; 8$.

Example values for LCG parameters and results obtained after 10 repetition cycles:

Table 1. LCG example

a	c	m	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
1	3	10	0	3	6	9	2	5	8	1	4	7	0
2	1	10	0	1	3	7	5	1	3	7	5	1	3
22	1	72	0	1	23	3	67	35	51	43	11	27	19
11	37	100	1	48	65	52	9	36	33	0	37	44	21

These sequences are not random, but (for the right choice of parameters) exhibit many properties of random sequences. For small parameter values, the non-randomness is particularly obvious: whenever we generate a value we've seen before, we enter a cycle where the same subsequence is continuously generated. In the first two examples above, the cycles are 0-3-6-9-2-5-8-1-4-7-0 and 1-3-7-5-1, of lengths 10 and 4 respectively. Since there are only M different possibilities, one must always enter such a cycle, but one wants to avoid short cycles.

Congruential linear generators are a popular and efficient type of pseudo-random number generator, but they have problems such as weak randomness and potential periodicity.

3. Hash map

Data requires a number of ways to be stored and accessed. One of the most important implementations includes Hash Tables. In Python, these Hash tables are implemented through the built-in data type i.e. dictionary. Hash maps are structured by indexed data. A hash map uses a hash function to calculate an index with a key in an array of slots. The key is unique. The analogy can be made with a cabinet with drawers with labels for the things stored in them.

In computer science, a Hash table or HashMap is a type of data structure that maps keys to its value pairs (implementing abstract array data types). Basically, it uses a function that calculates an

index value which in turn holds the elements to be searched for, inserted, removed, etc. This makes accessing data easy and fast. Generally, hash table stores key-value pairs and the key is generated using a hash function.

An example dictionary might be a mapping of employee names and their employee IDs, or student names and their IDs.

A hash map typically includes the following functions [6]:

1. set_val(key,value): Inserts a key-value pair into the hash table. If the value already exists in the hash table, the value will be updated.
2. get_val(key): Returns the value to which the specified key is mapped, or "No record found" if this map contains no mapping for the key.
3. delete_val(key): Removes the mapping for the specified key if the HashMap contains the mapping for the key.

The hash() method returns the hash value of an object if it has one. Hash values are just integers that are used to quickly compare dictionary keys during a quick lookup.

The syntax of the hash() method is: hash(object), where “object” is the object whose hash value is to be returned (integer, string, float) [7].

4. Software implementation

To create this program, the Python IDE development environment was used. The program is one of the most popular now and was used for the purpose of generating a token and making a user-friendly GUI.

Within the project, a series of steps were followed which are represented in the block diagram in Figure 1. The first step is to enter from the keyboard the email address on which you want to generate a token, this option was chosen because a user can have many more email addresses. If the address is not valid, the program will return an error, otherwise it will continue. With the correct email address, a hashing will be done on the email address from which an integer value is returned. The modulus function is applied to this and will be used as the seed for the LCG algorithm. The output of the algorithm is a vector, so it has been converted to a string to be displayed as an integer suitable for a valid token.

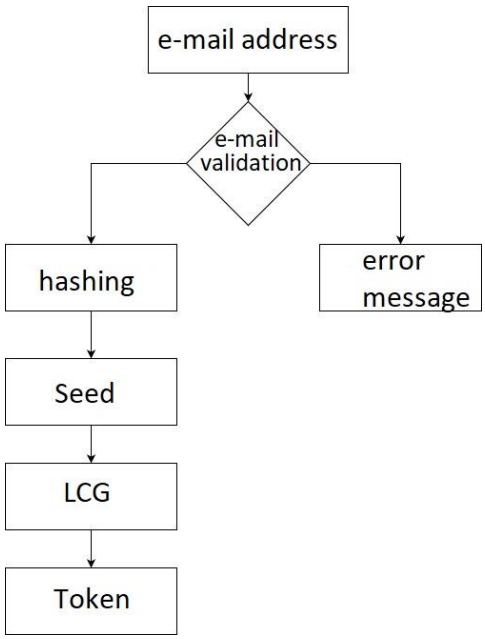

Fig. 1. Block diagram

To be able to determine if the email address is a valid one, a regexp was used. This is a regular or regular expression made up of a string of characters that specifies a pattern, a certain pattern. Which then, is to be used by another algorithm.

The following regexp was used for the email address:

Regex'([A-Za-z0-9]+[.-_])[A-Za-z0-9]+@[A-Za-z0-9-]+(\.[A-Z|a-z]{2,})+'*

Most of the time this is used inside an if structure because it returns the value 0 or 1 depending on whether the pattern was followed. Through the hashing process, a fairly large integer is generated at the output. For example, if an e-mail address is entered, it will return:

```
e-mail address: evelyn.enescu@stud.etti.upb.ro
e-mail address hashed 3650674425931386077
```

Fig. 2. The seed of an e-mail address

This resulting value will be used as the seed for the Linear Congruential Generator algorithm.

The seed value was taken and entered into the LCG algorithm where a was chosen a=1140671485 (multiplicator), c=12820116 (increment) and m=224 (module). With the help of the basic formula of the algorithm, this type of generator was implemented.

By means of a loop repeating 6 times, basically 6 numbers of variable length will be generated, so the last element of each number that resulted from the algorithm will be chosen. The resulting numbers will be concatenated to produce the token.

After testing with the email address *evelyn.enescu@gmail.com*, the following values for the LCG numbers and the token value were obtained:

```
3201383
10224022
7996681
7440048
417211
4572314
Token value 321814
```

Fig. 3. Token generation

Because the creation of an application in which you work with the command line is not very user-friendly, in this project it was also chosen to create a graphical interface (GUI) - Figure 4. This is designed to make interacting with our software as easy and intuitive as possible. It has a modern, clean design that is easy on the eyes and easy to navigate. The layout is organized in a logical manner, with all the main functionalities easily accessible from the main menu. Overall, this GUI is designed to increase productivity and make the user experience as smooth as possible. Moreover, since it is designed for login and Security use, a simple and accessible design has been implemented.

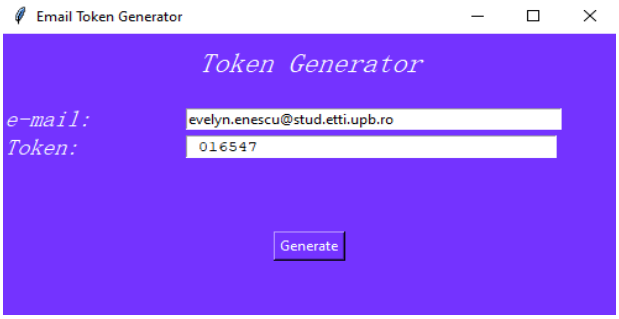

Fig. 4. Application interface

The GUI was implemented using the Tkinter package in Python, based on the TCL language. It is intuitive and easy to understand, making it an ideal choice for both novice and experienced users.

The interface is organized into several sections, with the labels, input field, button, and text field located in the center of the screen. Labels are prominently displayed to the left of the corresponding fields and are written in a clear, easy-to-read font. The input field is designed to accept user input such as text, numbers, or selections from a drop-down menu. The button is labeled with a clear and concise action such as "generate" and is positioned at the bottom of the page for easy access.

The text field sits above the button and is used to display the result of the action initiated by the button. The text field is also capable of displaying multiple lines of text, allowing the user to view detailed information or error messages. The GUI also includes a scroll bar to navigate through the text field if the text is too long to display on one screen. In addition, the GUI includes keyboard shortcuts for quick access to various functions, such as pressing the "Enter" key after entering data in the input field that will trigger the button's action.

5. Results

The results section of this article presents the findings of the study. Through a thorough analysis of the collected data, a number of significant conclusions could be drawn. Results are presented in a clear and organized manner, with graphs and figures used to enhance understanding of the data.

This study evaluated the performance of several different models on a common input data set. At the same email address, evelyn.enescu@stud.etti.upb.ro, 50 tokens were generated in order to determine if any token is repeated and the frequency of occurrence of random numbers at each position. The statistics can be found in the graphs below.

In order to make data testing more efficient, a .py file (Figure 5) exports the generated data to a .csv file (Figure 6). Quantitative and qualitative analysis will be performed on this data set.

```
com = "python main.py"
list = []
for i in range(50):
    out = os.popen(com).readlines()[0]
    out = out[:-1]
    list.append(out)
print(list)

f =  open("data.csv", "w")
for l in list:
    line = ""
    for c in l:
        line = line + "," + c
    f.write(line)
    f.write("\n")
f.close()
```

Fig. 5. Testing code

In this picture there is the code that creates the necessary files, calls for 50 times the main procedure that does the algorithms. Its purpose is to generate a .csv file to which the data will be transferred. The .csv has 50 rows representing all the output of the procedure, but the numbers are in different cells so as to see if a token was repeated and to have a graphic classification of the frequency of the numbers on each position.

Therefore, in Figure 7, the actual value of the number on the first position is entered on the X axis, and the frequency with which it appears on the first position, during 50 iterations of the same e-mail address, is marked on the Y axis.

Position 1	Position 2	Position 3	Position 4	Position 5	Position 6
9	2	9	6	3	8
6	3	8	9	0	7
3	2	3	6	1	6
0	1	0	3	4	7
6	1	0	7	0	9
7	8	5	0	1	0
9	2	1	4	5	8
7	8	5	0	5	2
1	4	1	4	7	2
9	2	5	6	3	4
4	3	2	1	2	3
7	8	5	4	3	0
6	1	4	3	2	1
5	4	9	8	1	4
1	2	3	4	9	8
7	8	1	6	9	0
7	2	7	2	7	2

Fig. 6. Exported data in .csv file

For the following figures, from Figure 8 to Figure 12, the same values corresponding to position indices 2 to 6 are illustrated. For the 1st position, the most frequent numbers are 9 and 6 followed by 5 and 7. For the 2nd position, the most frequent are 2 and 3, however for the 3rd position, the most used are 0 and 5. In the 4th position, the most used are 0 and 7, the 5th position uses the numbers 5 and 8 very often, and in the last the most frequent are 2, 7 followed by 1 and 0.

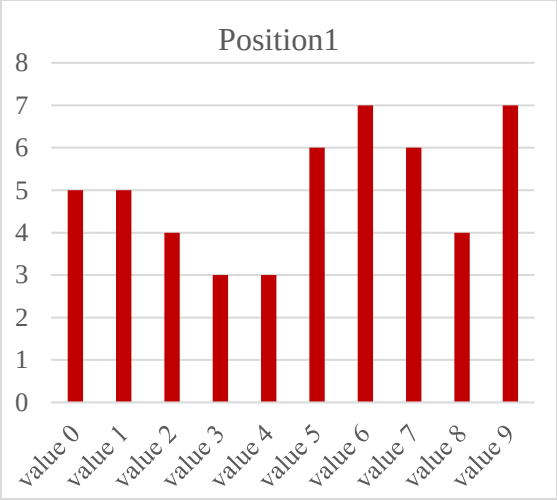

Fig. 7. The frequency on 1st position

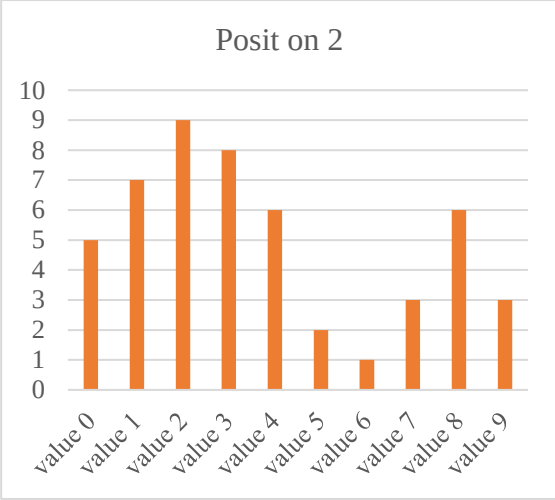

Fig. 8. The frequency on 2nd position

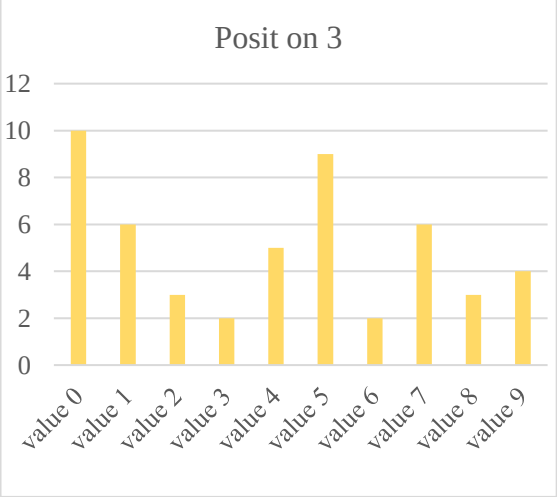

Fig. 9. The frequency on 3rd position

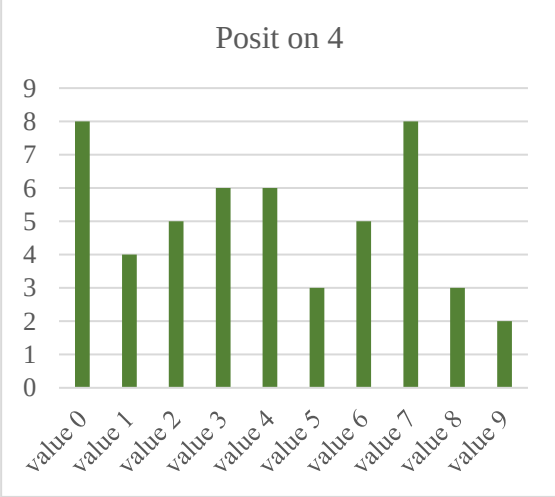

Fig. 10. The frequency on 4th position

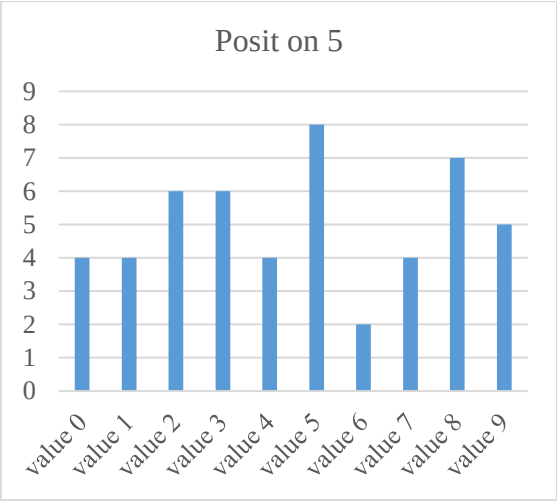

Fig. 11. The frequency on 5th position

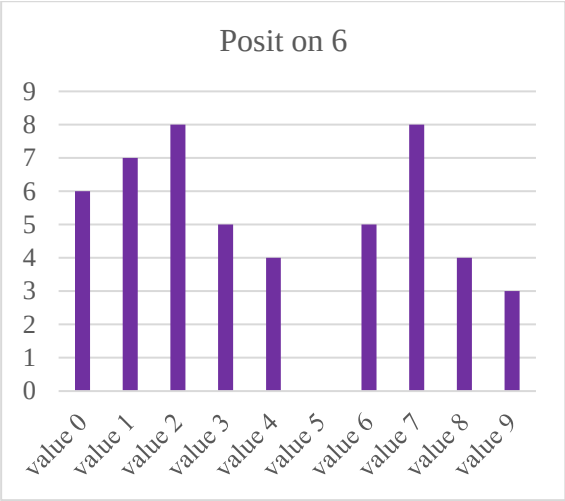

Fig. 12. The frequency on 6th position

6. Conclusions

The results of this evaluation showed that the overall algorithm performed best on the given dataset and could be used to generate repeating tokens and random numbers for the purpose of email authentication. Regardless of the type of pseudorandom number generator used, it is important to understand the strengths and weaknesses of each generator and the selection of the most appropriate one for the application. The advantage of using the LCG algorithm is that it generates pseudo-random numbers in a very accessible and fast way, but it comes with the disadvantage that its mass use can overload the program and generate the same corresponding token for two different email addresses.

In this paper, a modification was made to the classic LCG algorithm in order to generate a token. This algorithm takes an email address as input, checks that it is valid, and then generates 6 strings of numbers according to the algorithm shown. It takes the last digit of each number and creates a new number that will represent the authentication value.

Later, this application can be inserted into any other type of Web or mobile application to secure user authentication. Some examples of applications they could be embedded in are banking, medical results, and other applications that require increased attention to protecting customer or user data.

References

[1]. swarnavo09, “Elements of Cybersecurity,” 2022. Accessed: Mar. 15, 2023. [Online]. Available: https://www.geeksforgeeks.org/elements-of-cybersecurity/

[2]. H. Kavak, J.J. Padilla, D. Vernon-Bido, S.Y. Diallo, R. Gore, S. Shetty, “Simulation for cybersecurity: state of the art and future directions,” *Journal of Cybersecurity*, Vol. 7, Issue 1, 2021. Accessed Apr. 4, 2023. [Online]. Available: https://academic.oup.com/cybersecurity/article/7/1/tyab005/6170701.

[3]. J. V. Neumann, *Various techniques used in connection with random digits*, 1951.

[4]. B. Fathi-Vajargah, R. Asghari, "A Novel Pseudo-Random Number Generator for Cryptographic Applications," *Indian Journal of Science and Technology*, 9(6), 2016.

[5]. H. Tang, “Reverse multiple recursive random number,” *European Journal of Operational Research*, 164, 2005.

[6]. akshisaxena, Hash Map in Python, 2023. Accessed Mar. 22, 2023. [Online]. Available: https://www.geeksforgeeks.org/hash-map-in-python/

[7]. Python hash(). Accessed Mar. 20, 2023. [Online]. Available: https://www.programiz.com/python-programming/methods/built-in/hash