Application of Linear Congruent Generator in Affine Cipher Algorithm to Produce Dynamic Encryption

Ofelius Laia¹, Elviawaty Muisa Zamzami², Sutarman³, Fati Gratianus Nafiri Larosa⁴, Asaziduhu Gea⁵

¹Student in Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Medan, Indonesia
²Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Medan, Indonesia
³Faculty of Math and Science, Universitas Sumatera Utara, Medan, Indonesia
⁴Faculty of Computer Science, Universitas Methodist Indonesia, Medan, Indonesia
⁵Faculty of Computer Science, Universitas Methodist Indonesia, Medan, Indonesia

E-mail: ¹ofeliuslaia@gmail.com, ²elvi.zamzami@gmail.com, ³sutarman@gmail.com, ⁴fatignlarosa@gmail.com, ⁵gea.asaziduhu@gmail.com

Abstract. Modification of the original message (plaintext) with the LCG random number generator algorithm and encryption with the affine cipher algorithm to produces a dynamic and safe random message (ciphertext) compared to just the affine cipher algorithm. The plaintext is converted to ASCII code and stored in a data list. The data list that contain the plaintext changes the position of the data where each plaintext column is moved to an odd column and even columns are filled by random numbers generated by LCG. The data list is then changed again by adding value to each column with random numbers generated by the programming language and then it encrypted with the affine cipher algorithm. The total number of ciphertexts can be twice as many as the plaintext. The results obtained are not the same ciphertext character even though the plaintext has the same character. Ciphertext generated in the first run program will be different for the next run program or dynamic encryption. To restore a ciphertext to the plaintext is done based on the affine cipher decryption function and the steps to modify the message that was done before.

1. Introduction

Cryptography is art or science which includes the principles and methods of changing a message that is understood (plaintext) to a message that is not understood (ciphertext) and changes the message to return to its original form (plaintext) [1]. Affine cipher is a monoalphabetic substitute type of cipher, where each letter in the alphabet is mapped to an equivalent numeric, encrypted using a simple mathematical function and converted back into a letter [2]. Encryption results from the cipher affine method are easy to guess so that the confidentiality of a message becomes insecure. In this study, plaintext was modified before doing the encryption process with the affine cipher method.

Linear Congruential Generator (LCG) represents one of the oldest and most popular pseudo random number algorithms. The advantage of LCG is that the operation is very fast and can be applied to generate a set of random values or can be used to randomize the position of a set of values. The LCG algorithm that will be used to randomize the original message (plaintext) before the encryption process is carried out. This technique will produce a dynamic ciphertext where each encryption process will produce a different ciphertext and the number of characters in the plaintext will produce more characters in the ciphertext and for the same character in plaintext, the ciphertext is not the same.
2. Basis Theory

2.1. Definition of cryptography
In cryptography the original message is called plaintext and there are terms of encryption and decryption. Encryption is a process that changes a code from what is understandable (unreadable). Encryption can be interpreted as a code or cipher. While decryption is a process to return random information to its original form using the same algorithm when encrypting.

2.2. Affine cipher
Affine Cipher is an extension of the Caesar Cipher method, which transfers plaintext with a value and adds it with a shift $P$ produces the $C$ ciphertext expressed by the congruent function shown in equation 1 [2].

$$C \equiv mP + b \pmod{n} \quad (1)$$

Where $n$ is the size of the alphabet, $m$ is an integer that must be relatively prime with $n$ (if not relatively prime, then decryption cannot be done) and $b$ is the number of shifts (Caesar cipher is special from affine ciphers with $m=1$). To do the description, Equation 2 must be solved to obtain $P$. The emptiness solution only exists if inverse $m \pmod{n}$, expressed as $m^{-1}$. If $m^{-1}$ exists, then the decryption process is carried out as in equation 2 [2].

$$P \equiv m^{-1}(C - b) \pmod{n} \quad (2)$$

2.3. Random number generator
Linear Congruential Generators (LCG) are one of the oldest and most well-known random number generators. LCG is an algorithm that is often implemented in several programming languages to generate random numbers. LCG is defined in recurrence relations [3]:

$$X_n \equiv (aX_{n-1} + b) \pmod{m} \quad (3)$$

3. Research method
With the development of science in the field of Computer Science, especially in the field of data security, the affine cipher algorithm is rarely or almost not used because it is easy to know or guess the original message contained in the message that has been encoded.

So with that, the author developed the affine cipher algorithm by applying a Linear Congruent Generator algorithm (LCG) to randomize the position of the plaintext character, the number of ciphertext characters produced and the dynamic encryption results.

After the encryption process is successful, then get a message or data that has been encrypted or random, which then requires a process to return the message or data to its original form (plaintext). Thus, the affine cipher decryption process is carried out on ciphertext based on the affine key used when encryption and random numbers generated from the programming language. Following are the stages of research design for the encryption and decryption process:

![Figure 1. Research Design Process Encryption](image-url)
4. Results and discussion
This research is designing the development of classical cryptography algorithm with Affine Cipher involves the random number generator algorithm is an algorithm Linear Congruent Generator (LCG) and a random number generator function is provided by the programming language (in this study the programming language used is PHP).

The results of this study will be represented in 2 stages. The first stage is described in writing, step by step, the process of encryption and decryption, and the second stage by implementing a program project built using the PHP programming language. As research material for the process of encryption and decryption, messages or data to be tested are: ICoSNIKOM 2018

4.1. Stages and encryption results
In accordance with the encryption process research design (Figure 1), first the characters in plaintext are converted in decimal form ASCII code. The following table converts plaintext characters in decimal ASCII code:

Table 1. Convert plaintext characters in decimal form ASCII code

| Plaintext Character | I | C | o | S | N | I | K | O | M | Space | 2 | 0 | 1 | 8 |
|--------------------|---|---|---|---|---|---|---|---|---|-------|---|---|---|---|
| ASCII Code         | 73| 67|111| 83| 78| 73| 75| 79| 77| 32    | 50| 48| 49| 56|

From Table 1 above it is known that the number of characters in plaintext is 14 characters. After getting the ASCII code decimal value from each plaintext character, the next step is to save each column of ASCII code for the plaintext character to a data list in the odd column only.

Table 2. Data list for plaintext characters in the form of ASCII code.

| Data  | - | 73 | - | 67 | - | 111 | - | 83 | - | 78 | - | 73 | - | 75 |
|-------|---|----|---|----|---|-----|---|----|---|----|---|----|---|----|
| Column| 0 | 1  | 2 | 3  | 4 | 5   | 6 | 7  | 8 | 9  | 10| 11 | 12| 13 |
| Data  | - | 79 | - | 77 | - | 32  | - | 50 | - | 48 | - | 49 | - | 56 |
| Column| 14| 15 | 16| 17 | 18| 19  | 20| 21 | 22| 23 | 24| 25 | 26| 27 |

Information: column 0 is considered an even column

For even columns in the empty data list are filled with numbers generated from the Linear Congruent Generator (LCG) algorithm. To generate random numbers, an LCG key is needed according to the recurrence relation (equations 1). The following LCG key to generate 14 random numbers corresponds to the number of plaintext characters in table 1 by avoiding the appearance of the same number of repetitions. LCG key : \( X_n = 3; a = 1; b = 5; m = 14 \), then the resulting random number calculation is as follows:
Here is the affine cipher encryption process described in Table 6:

| Xn | f | Xn^θ |
|----|----|------|
| 3  | 1 x 3 + 5 mod 14 | X₁ = 8 |
| 8  | 1 x 8 + 5 mod 14 | X₂ = 13 |
| 13 | 1 x 13 + 5 mod 14 | X₃ = 4 |
| 4  | 1 x 4 + 5 mod 14 | X₄ = 9 |
| 9  | 1 x 9 + 5 mod 14 | X₅ = 0 |
| 0  | 1 x 0 + 5 mod 14 | X₆ = 5 |
| 5  | 1 x 5 + 5 mod 14 | X₇ = 10 |

The process of calculating LCG random numbers:

\[ X_n = (aX_{n-1} + c) \mod m \]

where:
- \( a \) = 13
- \( c \) = 1
- \( m \) = 100
- \( X₀ \) = 3

Random numbers generated sequence: 8, 13, 4, 9, 0, 5, 10, 1, 6, 11, 2, 7, 12, 3

From the sequence of random numbers above which are then saved to the even column in the data list. The following changes to the data list after a random number is entered:

| Data | 8 | 73 | 13 | 57 | 4 | 111 | 9 | 83 | 0 | 78 | 5 | 73 | 10 | 75 |
|------|---|----|----|----|---|-----|---|----|---|-----|---|----|----|----|
| Column | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| Data | 1 | 79 | 6 | 77 | 11 | 32 | 2 | 50 | 7 | 48 | 12 | 49 | 3 | 56 |
| Column | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |

The next step is to generate random numbers as much as the number of data columns in the data list, namely as many as 28 columns of data generated from the programming language. 28 random numbers generated by the Hypertext Preprocessor programming language from 1 to 100 are: 93, 76, 72, 72, 35, 6, 48, 89, 60, 27, 81, 1, 53, 35, 30, 79, 48, 78, 5, 47, 30, 51, 64, 76, 32, 3, 18, 87.

| Data | 101 | 149 | 85 | 139 | 39 | 117 | 57 | 172 | 60 | 105 | 86 | 74 | 63 | 110 |
|------|-----|-----|----|-----|----|-----|---|-----|---|-----|---|----|----|----|
| Column | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| Data | 31 | 158 | 54 | 155 | 16 | 79 | 32 | 101 | 71 | 124 | 44 | 52 | 21 | 143 |
| Column | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |

List of data (Table 4) which is then used as plaintext (already in decimal data format) to be encrypted using Affine Cipher algorithm. Before encrypting, the key that will be used in the Affine Cipher formula itself is first determined. The key to encryption is: First key = 7 and Second key = 3. Here is the affine cipher encryption process described in Table 6:

| Data | 101 | (7 x 101 + 3) mod 255 | 200 |
| 149 | (7 x 149 + 3) mod 255 | 26 |
| 85  | (7 x 85 + 3) mod 255 | 88 |
| 139 | (7 x 139 + 3) mod 255 | 211 |
| 39  | (7 x 39 + 3) mod 255 | 21 |
| 117 | (7 x 117 + 3) mod 255 | 57 |
| 57  | (7 x 57 + 3) mod 255 | 147 |
| 172 | (7 x 172 + 3) mod 255 | 187 |
| 60  | (7 x 60 + 3) mod 255 | 168 |
| 105 | (7 x 105 + 3) mod 255 | 228 |

| Data | 86 | (7 x 86 + 3) mod 255 | 95 |
| 74  | (7 x 74 + 3) mod 255 | 11 |
| 63  | (7 x 63 + 3) mod 255 | 189 |
| 110 | (7 x 110 + 3) mod 255 | 8 |
| 31  | (7 x 31 + 3) mod 255 | 220 |
| 158 | (7 x 158 + 3) mod 255 | 89 |
| 54  | (7 x 54 + 3) mod 255 | 126 |
| 155 | (7 x 155 + 3) mod 255 | 68 |
| 16  | (7 x 16 + 3) mod 255 | 115 |
| 79  | (7 x 79 + 3) mod 255 | 46 |
column taken to be used as plaintext is an odd column assuming column 0 is an even column then
previous encryption the plaintext column is placed in an even column, so the data list above the
below:
from programming languages that
cipher cryptographic keys that are used during encryption and rows of random numbers generated
where the stages of the decryption process in this study have been described in the previous research
To return a message or data that has been encoded (ciphertext) a process called decryption is needed,
4.2.
Table 8. The decryption results are stored in the data list
The final step is to determine and retrieve the original plaintext column. In accordance with the
previous encryption the plaintext column is placed in an even column, so the data list above the
column taken to be used as plaintext is an odd column assuming column 0 is an even column then
converted to ASCII characters.

| Data  | Result |
|-------|--------|
| 32    | 227    |
| 101   | 200    |
| 71    | 245    |
| 124   | 106    |
| 44    | 56     |
| 52    | 112    |
| 21    | 150    |
| 143   | 239    |

From the encryption process in Table 6, ciphertext is obtained in decimal form which is then
converted to ASCII characters. The following encrypted messages (ciphertext) have been converted in
ASCII characters:

| Data  | f          | Result |
|-------|------------|--------|
| 200   | (73 x (200-3)) mod 255 | 101 |
| 26    | (73 x (26-3)) mod 255 | 149 |
| 88    | (73 x (88-3)) mod 255 | 85 |
| 211   | (73 x (211-3)) mod 255 | 139 |
| 21    | (73 x (21-3)) mod 255 | 39 |
| 57    | (73 x (57-3)) mod 255 | 117 |
| 147   | (73 x (147-3)) mod 255 | 57 |
| 187   | (73 x (187-3)) mod 255 | 172 |
| 168   | (73 x (168-3)) mod 255 | 60 |
| 228   | (73 x (228-3)) mod 255 | 105 |
| 95    | (73 x (95-3)) mod 255 | 86 |
| 11    | (73 x (11-3)) mod 255 | 74 |
| 189   | (73 x (189-3)) mod 255 | 63 |
| 8     | (73 x (8-3)) mod 255 | 110 |

The decryption results above are reduced by the value of each data column with a random number of
data column generated from the programming language then stored in a data list as in the table
below:

| Data  | Column |
|-------|--------|
| 8     | 73     |
| 73    | 13     |
| 67    | 4      |
| 111   | 9      |
| 83    | 0      |
| 78    | 5      |
| 73    | 10     |
| 75    | 11     |
| 12    | 13     |
| 1     | 79     |
| 6     | 77     |
| 11    | 32     |
| 200   | 2      |
| 50    | 7      |
| 48    | 12     |
| 49    | 3      |
| 56    | 25     |
| 26    | 27     |

| Data  | Column |
|-------|--------|
| 14    | 15     |
| 16    | 17     |
| 18    | 19     |
| 20    | 21     |
| 22    | 23     |
| 24    | 25     |
| 26    | 27     |

4.2. Stages and decryption results
To return a message or data that has been encoded (ciphertext) a process called decryption is needed,
where the stages of the decryption process in this study have been described in the previous research
design as shown in Figure 2.
The data needed for the decryption process are messages that will be decrypted (ciphertext), affine
cipher cryptographic keys that are used during encryption and rows of random numbers generated
from programming languages that are also used during the encryption process. Here is the affine
cipher decryption process described in Table 7:

| Data  | f          | Result |
|-------|------------|--------|
| 220   | (73 x (220-3)) mod 255 | 31 |
| 89    | (73 x (89-3)) mod 255 | 158 |
| 126   | (73 x (126-3)) mod 255 | 54 |
| 68    | (73 x (68-3)) mod 255 | 155 |
| 115   | (73 x (115-3)) mod 255 | 16 |
| 46    | (73 x (46-3)) mod 255 | 79 |
| 227   | (73 x (227-3)) mod 255 | 32 |
| 200   | (73 x (200-3)) mod 255 | 101 |
| 245   | (73 x (245-3)) mod 255 | 71 |
| 106   | (73 x (106-3)) mod 255 | 124 |
| 56    | (73 x (56-3)) mod 255 | 44 |
| 112   | (73 x (112-3)) mod 255 | 52 |
| 150   | (73 x (150-3)) mod 255 | 21 |
| 239   | (73 x (239-3)) mod 255 | 143 |

| Data  | f          | Result |
|-------|------------|--------|
| 220   | (73 x (220-3)) mod 255 | 31 |
| 89    | (73 x (89-3)) mod 255 | 158 |
| 126   | (73 x (126-3)) mod 255 | 54 |
| 68    | (73 x (68-3)) mod 255 | 155 |
| 115   | (73 x (115-3)) mod 255 | 16 |
| 46    | (73 x (46-3)) mod 255 | 79 |
| 227   | (73 x (227-3)) mod 255 | 32 |
| 200   | (73 x (200-3)) mod 255 | 101 |
| 245   | (73 x (245-3)) mod 255 | 71 |
| 106   | (73 x (106-3)) mod 255 | 124 |
| 56    | (73 x (56-3)) mod 255 | 44 |
| 112   | (73 x (112-3)) mod 255 | 52 |
| 150   | (73 x (150-3)) mod 255 | 21 |
| 239   | (73 x (239-3)) mod 255 | 143 |
Table 9. Plaintext results from decryption in decimal form and ASCII characters.

| ASCII Code | 73 | 67 | 111 | 83 | 78 | 73 | 75 | 79 | 77 | 32 | 50 | 48 | 49 | 56 |
|------------|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|
| Plaintext Character | I  | C  | o   | S  | N  | I  | K  | O  | M  | Space | 2  | 0  | 1  | 8  |

Decryption Results (Plaintext): ICoSNIKOM 2018.

4.3. Test result

From the stages and results of the encryption and decryption above, then implemented in an application built with the web-based Hypertext Preprocessor programming language. The following is the program output display for message encryption and decryption.

![Encryption and decryption program output](image)

Figure 3. Encryption and decryption program output

5. Conclusions

In accordance with the results and testing of the plaintext modification with the Linear Congruent Generator algorithm for the Affine Cipher cryptographic algorithm, Affine Cipher's encryption and decryption process was successfully executed both for encoding the message (ciphertext) and for returning messages that had been encrypted to their original form.

Plaintext modification with Linear Congruent Generator algorithm greatly guarantees the security level of a message because it changes the number of messages to more and the addition of random number values from the programming language to the plaintext value makes the process of encryption with dynamic or variable ciphertext according to the number of program execution for the encryption process.

6. References

[1] Acharya, Bibhudendra, S.K. Panigrahy, S.K. Patra and G. Panda. 2009. Image Encryption Using Advanced Hill Cipher Algorithm, International Journal of Recent Trends in Engineering, Vol. 1, No. 1, May 2009.
[2] Math 135, Affine Ciphers, Decimation Ciphers, and Modular Arithmetic, Summer 2006.
[3] Niels Ferguson and Bruce Schneier. Practical Cryptography. Wiley Publishing, Inc, Indianapolis, Indiana, 2003.
[4] Sidorenko, Andrey and Schoenmakers, Berry. 2005. “Concrete Security of The Blum-Blum-Shub Pseudorandom Generator”. Cryptography and Coding: 10th IMA International Conference. Springer-Verlag: Computer Science 3796. Eindhoven University of Technology. P.O. box 513, 5600MB Eindhoven. The Netherlands.
[5] C. Kenny, Random Number Generators: An Evaluation and Comparison of Random.org and Some Commonly Used Generators Project Report, April 2005.
[6] D.E. Knuth, The Art of Computer Programming Volume 2: Seminumerical Algorithms, MA:Addison-Wesley, 1997.
[7] Thanigaivelan Balavelan, Postula Adam and Hamilton Tara Julia. A modified MOSFET small-signal model based on affine arithmetic concepts, 2009 Asia Pacific Conference on Postgraduate Research in Microelectronics & Electronics (PrimeAsia), Shanghai, China.
[8] Sitjongsataporn Suchada, Nurarak Piyporn. A modified adaptive step-size affine projection algorithm based on QR-decomposition. 2017 International Electrical Engineering Congress (iECON), Pattaya, Thailand.