USING A GENETIC ALGORITHM TO BUILD SYMMETRIC CRYPTALGORITHMS.

Zurab Kochladze
I. Javakhishvili Tbilisi State University Faculty of Exact and Natural Sciences, Department of Computer Science
0186.13 University street, Tbilisi, Georgia.

Abstract
The article discusses a symmetric encryption algorithm built using genetic algorithm operations and a pseudo-random generator. An example of how the algorithm works is given.

***Keywords**: genetic algorithm; crossover and mutasion operasions; cypher.*

1. Introduction

It is known that the speed of open key crypto algorithms is very low. Because of this, symmetric ciphers are mainly used to encrypt information [1,2]. As K. Shannon in his fundamental work [3] showed that symmetric crypto algorithms (except one) allow information to be leaked in encrypted text about plain text. So, if we do not want that the opponent to be able to crack the algorithm in polynomial time, it is necessary to perform diffusion and sconfuzion processes in the algorithm.

In modern cryptographic algorithms, in which both plain text and ciphertext represent by n-bit lengths strimgs, this practically means that if the structure of bit string, whish correspondings to the plain text is defined by plain text, the structure of the bit string, whish correspondings to the ciphertext should be as close as possible to the random bit string. This causes the use such of operations in algorithms that perform these conditions. It also causes the same text to be encrypted several times with different keys.

2. Genetic alogorithms

Genetic algorithms allow a more accurate solution to a problem by performing certain operations on candidate populations solving the problem [4]. The genetic algorithm for solving a problem consists of three stages, the first of which is to present individual potential solutions in a special form that is most convenient for performing evolutionary changes and sampling operations. More often these are usually bit (binary) strings. In the second stage, cross-breeding and mutation operations are carried out, which are characteristic of living biological forms. After using these operations, a new generation is obtained with the recommended features of their parents. In the last, third stage, based on any selection criteria, the best, ie the most accurately matching solution to the given problem will be selected. These specimens are selected for multiplication, ie to obtain a new generation of potential solutions. Eventually any generation will become the solution to the initial task.

Use of genetic algorithms in cryptology began as early as the end of the last century. Initially, genetic algorithms were actively used in cryptanalysis, in which quite good results were achieved [5,6]. Then gradually they started to be used in cryptography and practically today genetic algorithms are used in almost all directions of cryptology. Many symmetric algorithms have been developed that use genetic algorithms to encrypt [7,8]. This can be explained by the fact that genetic operations fully meet the above conditions required by symmetric cryptographic algorithms.

Today, most algorithms used for information security are based on mathematical operations and transformations. However, with the development of artificial intelligence, which has activated the principles of genetic learning based on the evolutionary model, and interest in GA, work is actively underway to use genetic algorithms to build crypto algorithms. Compared to modern crypto algorithms, which mainly operate after the representation of information in a matrix form, the main advantage of using genetic algorithms is the speed of genetic operations - their evaluation (Ō = n) is linearly related to the size of the information (message).

3. Description of the new cryptographic algorithm

The algorithm uses a pseudo-random number generator (**PRNG**) and genetic algorithm operations: crossover and mutation. Algorithm options: block size 128 bits, secret key length -128 bits, PRNG initial values: $\boldsymbol{a, x_0, b}$ (these settings are also keys).

- On the first stage, the plain text is transformed through ASCII codes to the binary string $\boldsymbol{(0,1)^n}$, which is divided into 128-bit blocks;
- Then by using the PRNG function $\boldsymbol{x_i = ax_{i-1} + b,}$ we must calculate 16 pseudo-random numbers in decimal system. It is ouer secret key. None of the key elements should be equal to 0. A piece of software code (example [1]) counts this numbers.

```
{ int a = 11, b = 5;
kay.push_back (19);
for (int i = 1; i <16; i ++)
{int y = a * (kay [i-1] + b)% (256 + i);                    (1)
while (y == 0) {
y = a * (kay [i-1] + b)% (256 + i);  }
kay.push_back (y);}
```

- In the third step, these sixteen numbers are converted to 128-bit binary string via ASCII codes. This is the secret key to a given plain text block. Using the Xor operation, the 128-bit key and the corresponding 128-bit plain text will be assembled. The key for each block of plain text is calculated again using the same function. But with each subsequent calculation the free number $\boldsymbol{b}$ increases by the magnitude $\boldsymbol{l}$. This $\boldsymbol{l}$ is also an integral part of the key.
- Software code snippet:

```
{ int e = 1, l = 0;
        while(e<=k){
        for(int i=0;i<128;i++)
        int t=text[l]||binkay[i];
        trans.push_back(t);
         l++;}
       e++;}
      for(int i=0;i<(k*128);i++)
      cout<<trans[i];
      cout<<endl;  }
```

The algorithm then starts using genetic surgeries. It takes the first number from the sixteen calculations and uses the modulus and converts it to an interval [1,7].

The number obtained is the point of a single-point crossover operation. The algorithm takes the first and second blocks of text assembled with the secret key and divides them into two parts by means of crossover operation. Then attach the first part of the first block to the second part of the second block and vice versa, attach the first part of the second block to the second part of the first block. In order for the algorithm to use the mutation operation, the information exchange subjects agree in advance on two numbers that they exchange with the key. For example $\boldsymbol{x, y. x \neq y. 0 < x, y < 8.}$ The algorithm then moves to the already modified second and irreplaceable third bytes. So he continues to work until he considers the eighth and first bytes. round.

Consider the process of working this algorithm on a simple example. Suppose we want to encrypt open source text: **domein parameters**. The parameters of the pseudo-random number generator $X_{i+1}(a \cdot X_i + b) \bmod m$ should be as follows:

$a = 37, x_0 = 4, b = 11, m = 101$.

$X_1 = (4 \cdot 37 + 11) \bmod 101 = 58;$ $X_2 = (4 \cdot 58 + 11) \bmod 101 = 41;$

$X_3 = (4 \cdot 41 + 11) \bmod 101 = 74;$ $X_4 = (4 \cdot 74 + 11) \bmod 101 = 4;$

$X_5 = (4 \cdot 4 + 11) \bmod 101 = 27;$ $X_6 = (4 \cdot 27 + 11) \bmod 101 = 18;$

$X_7 = (4.18 + 11) \bmod 101 = 83;$ $X_8 = 4 \cdot 83 + 11) \bmod 101 = 40;$

$X_9 = (4 \cdot 40 + 11) \bmod 101 = 70;$ $X_{10} = (4 \cdot 70 + 11) \bmod 101 = 89;$

$X_{11} = (4 \cdot 89 + 11) \bmod 101 = 64;$ $X_{12} = (4 \cdot 64 + 11) \bmod 101 = 65;$

$X_{13} = (4 \cdot 65 + 11) \bmod 101 = 69;$ $X_{14} = (4 \cdot 69 + 11) \bmod 101 = 85;$

$X_{15} = (4 \cdot 85 + 11) \bmod 101 = 48;$ $X_{16} = (4 \cdot 48 + 11) \bmod 101 = 1.$

Add the numbers generated by the generator and the decimal numbers corresponding to the plane text letters with modulus 256:

M[1] = (100 + 58) **mod** 256 = 158; M[2] = (111 + 41) **mod** 256 = 152;
M[3] = (109 + 74) **mod** 256 = 183; M[4] = (97 + 4) **mod** 256 = 101;
M[5] = (105 + 27) **mod** 256 = 132; M[6] = (110 + 18) **mod** 256 = 128;
M[7] = (32 + 83) **mod** 256 = 115; M[8] = (112 + 40) **mod** 256 = 152;
M[9] = (97 + 70) **mod** 256 = 167; M[10] = (114 + 89) **mod** 256 = 203;
M[11] = (97 + 64) mod 256 = 161; M[12] = (109 + 65) **mod** 256 = 174;
M[13] = (101 + 69) **mod** 256 = 170; M[14] = (116 + 85) **mod** 256 = 201;
M[15] = (101 + 48) **mod** 256 = 149; M[16] = (114 + 1) **mod** 256 = 115.

Convert the obtained numbers to a binary system and divide bytes. We get:

10011110, 10011000, 10110111, 01100101, 10000100, 10000000, 01110011, 10011000, 10100111, 11001011, 10100001, 10101110, 10101010, 11001001, 10010101, 01110011

Convert plain text to a binary string, we get:

01100100 01101110 01101101 01100101 01101001 01101110 01110000 01100001 01110010 01100001 01101101 01100101 01110100 01100101 01110000 01110011.

Assemble the first two blocks of the secret key and open text with the xor operation, we get:

11111010 ***11110101***

Perform crossover operation on encrypted text. Let's take the first number x_1, generated by the PRNG and calculate $x_1 (\bmod 8) = 158 \bmod 2 = 6.$

The number obtained indicates the crossover position. Take the first and second blocks of encrypted text and divide this point into two parts and exchange these parts

First block 111110 10 ***second block 111101 01***

We get it: ***11111001*** ***11110110***

Suppose there are mutation points (3,6) then we get new blocks:

11011101 ***1100010***

4. Conclusion and future work

As we can see, the algorithm consists of very simple operations, which means that the speed of the algorithm will be high, which is one of the important features. The operations used satisfy K. well. Shannon requirements. Experiments now need to be performed to determine how many rounds we need to perform to make the bit string $(0,1)^n$ containing the received encrypted text as close as possible to the random $(0,1)^n$ string.

REFERENCES:

1. A. G. Konheim Computer security and cryptography .A.J.Wiley &sons, 2007.

2. W. Stallings, L. Brown Computer Security Third Edition. Pearson, 2015.

3. C.Shannon Comunication theory of secrecy systems. Bell System Technology, J. 28, №4 (1949), pp. 656-715.

4. Goldberg D.E. *Genetic Algorithms in Search, Optimisation and Machine Learning,* Boston, Addison-Wesly, 1989.

5. Spillman R.,Janssen M., Nelson B., Kepner N.,, Use of Genetic Algorithm in Cryptanalysis of Simple Substituion Cipher, Cryptologia, Vol.17, No.4, pp. 367-377, 1993.

6. Kochladze Z., Beselia L., Cracking of the Merkle–Hellman Cryptosystem Using Genetic Algorithm, Transections on Sciense and Tecnology , Volume 3, No. 1-2: Science and Natural Resources [pp. 291-296] .2016.

7. Sindhuja K , Pramela Devi S. “A Symmetric Key Encryption Technique Using Genetic Algorithm”. (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (1), 414-416, 2014.

8. Tragha, A.; Omary, F.; Mouloudi, A. 2006. ICIGA: Improved Cryptography Inspired by Genetic Algorithms, in *Proc. International Conference on Hybrid Information Technology*. Cheju Island, 335–341.

Article received: 2021-03-09