Code 004.9

CREATING AN ENCRYPTION BLOCK ALGORITHM USING PRNG AND GENETIC OPERATIONS

Kochladze Zurab[1], Benidze Nana[2], Beselia Lali[2]

[1]Ivane Javakhishvili Tbilisi State University Faculty of Exact and Natural Sciences, Department of Computer Science
[2]Sokhumi State University, Faculty of Mathematics and Computer Sciences

Abstract

This paper describes a new block encryption algorithm that uses the Hill's modified algorithm for faster efficiency process. This allows us to increase the encryption and decryption speeds so as not to reduce the algorithm's resistance to cryptanalytic attacks.

Анотація

В статье обсуждается построение симметричного алгоритма шифрования с помощью генетических операций и псевдослучайной последовательности. Работа такого алгоритма показана на примере шифрования одного 128-битного блока. Основным преимуществом этого алгоритма является его скорость.

Introduction

Algorithm options: block size 128 bits, secret key length -128 bits, PRNG initial values: $a,\ x_0, b$ (these settings are the key).

Description of the algorithm :

- In the first stage, the encrypted text is transferred through ASCII codes to the binary system, which is divided into 128-bit blocks;
- In the second step, using $y = ax_{i-1} + b(modm)$ function, we calculate the secret key, which represents 16 pseudo random numbers in decimal system. None of the key elements should be equal to 0. A piece of software code (example [1]) counts the secret key.

```
{ 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);
 while (y == 0) {
 y = a * (kay [i-1] + b)% (256 + i);  }
 kay.push_back (y);}
```

(1)

- In the third step, the secret key transforms into the binary system by the ASCII codes. As already mentioned the key length is 128 bits. Using xor operation, each block of key and encrypted text is collected, given in (example [2]) 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;  }
```

Where k is the number of blocks. The vector elements obtained by the bitwise assembly of the keys and blocks are divided into bytes and placed in the *array [8] [16 * k]* matrix;

- In the fourth step $x_n = 2^{13}(x_{n-1} + x_{n-2} + x_{n-3}) \bmod (2^{32} - 5)$ using PRNG, we calculate the $point[16]$ sixteen pseudo random numbers and subtract their values using $mod8$. The 16 random pseudo-numbers obtained are the points of genetic operation between the bytes - the crossover[1,2]. Like the secret key, we exclude zero values here. At these points the crossover is held on the following principle[]: it intersects the first and second bytes of the first block at $point[0]$, then intersects at the second and third $point[1]$, and so on until the sixteenth byte intersects with the first byte of the block. A crossover is held for all k blocks.

For example, let's say we have two bytes:

1	0	0	1	1	1	1	1

1	0	1	0	1	1	1	0

If the intersection point is $point\ [0]\ =\ 3$, the crossover in this text As a result we get:

1	0	0	0	1	1	1	0
1	0	1	1	1	1	1	1

- In the fifth stage, the mutation operation on one bit is completed at the $point[i]$ points found by PRNG in each byte[3,4]. For example, if we perform a mutation operation on the third bit of our first byte, we get:

1	0	1	0	1	1	1	0

- In the sixth stage, if each byte of each block of the new matrix is different from the corresponding byte of the corresponding block of open text, which means that this is enough to hide the information about the open text well in the resulting cipher text, then the bytes are converted to symbols. Otherwise we repeat several rounds and during all rounds the key and crossing points are changed[5].

Let's give an example of encrypting a 128-bit block:

Encrypted text: „domain reporters“

Encrypted text in binary system:

01100100 01101111 01101101 01100001 01101001 01101110 00100000 01110010
01100101 01110000 01101111 01110010 01110100 01100101 01110010 01110011

Encrypted text in binary system:
01001111 01111101 11100101 11111111 00111110 01110001 01111111 11110111
11110001 00101111 11010010 11110101 11000101 11101110 00110011 00101111

In the case of the given example, none of the bytes of the encrypted and encrypted text match. In this case, we only need to do one round.

Decryption is performed using the reverse sequence of encryption operations.

References

1.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.

2.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.

3.Delman B., Genetic Algorithms in Cryptography, Master of Science Thesis, Rochester Institute of Technology, 2004.

4.Garg P., Genetic algorithm Attack on Simplified Data Encryption Standard algorithm, International journal Research in Computing Science, ISSN1870-4069, 2006.

5.Gorodilov A., Morozenko B., genetic algorithm for finding the key’s length and cryptanalysis of the permutation cipher, International Journal "Information Theories & Applications" Vol.15 / 2008.