New Technique to Generate Pseudo Random Crypto keys

Awni M. Gaftan[1] , Kholood J. Moulood[2]
[1] Mathematics Dept , College of Computer and Mathematics science , Tikrit University , Tikrit , Iraq
[2] Mathematics Dept , Education for Women , Tikrit University, Tikrit , Iraq

(Received: 5 / 5 / 2013 ---- Accepted: 28 / 10 / 2013)

Abstract

In this paper, we are building a Pseudo Random Crypto Keys Generator which depends on sequences (bank) of linear shift register (LSR), Some of them chosen randomly on each run (system operating). So we get a new cryptosystem at each run .

We will choose 4 linear shift register for each system (Si), i=1,...,4. So we will have new 16 linear shift register in all systems on each run that are chosen from 32 shift register existing in (bank) sequences of linear shift register.

Key words: Linear shift register, Cryptosystem, Pseudo random, Crypto key, Balance system, Statistical tests.

1. Introduction

With the widespread use of wireless network services and applications, security becomes a major concern. From security aspects, data integrity and confidentiality are vital issues for information systems. Confidentiality is concerned with resources being only accessed by authorized users while integrity refers to protection against unauthorized modification. Integrity and confidentiality are often related to authentication, authorization and cryptography. In fact, authentication utilizes strong cryptographic systems in order to secure itself. Thus cryptography plays a crucial part of any security system[2][4][5].

There are two basic techniques in cryptography: symmetric and asymmetric cryptography. The key of decryption can be calculated easily. In symmetric cryptography, encrypted and decrypted keys are the same. In contrast, cryptography using different encrypted keys from decrypted keys is called asymmetric cryptography. Each of them has pros and cons. Because of its characteristics, a symmetric cryptography is more secure than symmetric in key distribution and exchange. However, symmetric cryptography is significantly faster than asymmetric cryptography. Furthermore, Some papers stated that (the asymmetric cryptography key size must be ten times or more that of a symmetric cryptography key in order to have a similar level of security [2][3][6].

Modern cryptosystems depend on using cryptography algorithms or (generate crypto key algorithms) that have high complexity and very high level of security and randoming in generate outputs.

All of this depends on the contents of cryptosystem from relatively very long (LFSR) in prime number to achieve longest period.

Also they depend on the filling way that usually use the public and basic keys [3][6].

Moreover, they depend on the motion of these (LFSR)s as well as the tables and mathematical functions used in that cryptosystem.

2. Random key generator and Pseudo Random key generator

An important cryptographic function is cryptographically strong pseudorandom number generation. Pseudorandom number generators (PRNGs) are used in a variety of cryptographic and security applications.

Traditionally, the concern in the generation of a sequence of allegedly

random numbers has been that the sequence of numbers be random in some well defined statistical sense. The following two criteria are used to validate that a sequence of numbers is random [5][6]:

- **Uniform distribution:** The distribution of bits in the sequence should be uniform; that is, the frequency of occurrence of ones and zeros should be approximately equal.
- **Independence:** No one subsequence in the sequence can be inferred from the others.

Although there are well-defined tests for determining that a sequence of bits matches a particular distribution, such as the uniform distribution, there is no such test to "prove" independence. Rather, a number of tests can be applied to demonstrate if a sequence does not exhibit independence. The general strategy is to apply a number of such tests until the confidence that independence exists is sufficiently Strong [1][2][4].

3. Proposed Algorithm

In this paper, Filling system is used to fill the basic system. This filling system is stored on external part and automatically works when connected to basic system to the filling purpose, the external part is used to choose linear shifts that will be used in the basic systems.

3-1 Filling

3-1-1 External System Filling

To fill the external system, we use basic and public keys (that are saved on a special table monthly changed) with sequence of the day in the week and in the month as below:

Basic key (XOR) Public key (XOR) Day in week(1-7) (XOR) day in month (1-31)

This operation will repeat until the external system filled.

3-1-2 Basic System Filling

When the external system is connected to basic system, some operations will be done as below:-

Algorithm 1: Basic system Filling

1- Operate the external system to get (5 bits) in order to get a shift register from 32 shift registers that will

be used in the system (2^5 = 32). i.e. if we get 0=(00000) we will use shift register (0-15), if we get 5=(00101) we will use shift register from (5-21) and so on.

2- After the shift registers are chosen, we will fill it vertically i.e. first stage from each shift registers firstly filled and second stage from each shift registers filled and so on until the shift registers will be filled. The last stage from each shift registers will be filled by 1.

3-2 Motion

Algorithm2: Motion

1- The subsystems **Si** (i=1,…,4) will be moved to get addresses to the memories **EPi** (i=1,…4) to get **BYTE-I** (i=1,…,4).

2- Execute **XOR** function between **BYTE-1**, **BYTE-2** to get **BYTE-A** and between **BYTE-4** to get **BYTE-B**.

3- Input **BYTE-A**, **BYTE-B** to **TAB256** to get **BYTE-A1** and **BYTE-B1**. Note that **TAB256** is a table with size 8*256 , we get **BYTE-A1** which depends on **BYTE-A** which is used as address to the table (TAB256) and so as **BYTE-B1** which depends on **BYTE-B** that is used as address to the table. Note that if **BYTE-A** is equal to **BYTE-B** we will add the value 1 to **BYTE-B**.

4- Execute **XOR** function between **BYTE-A1**, **BYTE-B1** to get **L-BYTE**.

5- Get byte from balance system (**BS**) and execute **XOR** function between that byte and **L-BYTE** to get **KEY- BYTE**.

6- Get **MIRR-BYTE** by rotating **KEY-BYTE** depending on 2 bits which are **2nd** position from first shift register from first system **S1** and **5th** position from fourth shift register from second system **S2** as below:-

00 → No rotating.

01 → One time left rotation.

10 → Two times left rotation.

11 → Three times left rotation.

7- After rotating, Divide **KEY-BYTE** to two parts each of them consists of 4 bits. So we get **D1,D2** and will be used as two random keys.

3-3 Statistical Tests

The sequence of crypto keys that are generated using the cryptosystem in this paper must achieve a good statistical properties and pass the random standard tests which are Frequency test, Serial test, Poker test Runs test and Autocorrelation test [1].

1- Frequency Test

The purpose of this test is to determine whether the number of 0's and 1's in s are approximately the same, as would be expected for a random sequence. Let n0, n1 denote the number of 0's and 1's in s, respectively. The statistic used is $X_1 = (n_0 - n_1)^2 / n$.

2- Serial Test

The purpose of this test is to determine whether the number of occurrences of 00, 01, 10, and 11 as subsequences of s are approximately the same, as would be expected for a random sequence. Let n0, n1 denote the number of 0's and 1's in s, respectively, and let n00, n01, n10, n11 denote the number of occurrences of 00, 01, 10, 11 in s, respectively. Note that n00 + n01 + n10 + n11 = (n − 1) since the subsequences are allowed to overlap. The static used is

$$X_2 = \frac{4}{n-1}\left(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2\right) - \frac{2}{n}\left(n_0^2 + n_1^2\right) + 1$$

3- Poker Test

Let m be a positive integer such that $\lfloor \frac{n}{m} \rfloor \geq 5 \cdot (2^m)$, and let $k = \lfloor \frac{n}{m} \rfloor$ Divide the sequence s into k non-overlapping parts each of length m, and let n_i be the number of occurrences of the i^{th} type of sequence of length m, $1 \leq i \leq 2^m$. The poker test determines whether the sequences of length m each appear approximately the same number of times in s, as would be expected for a random sequence. The statistic used is

$$X_3 = \frac{2^m}{k}\left(\sum_{i=1}^{2^m} n_i^2\right) - k$$

4- Runs Test

The purpose of the runs test is to determine whether the number of runs (of either zeros or ones) of various lengths in the sequence s is as expected for a random sequence. The expected number of gaps (or blocks) of length i in a random sequence of length n is $e_i = (n-i+3)/2^{i+2}$.

Let k be equal to the largest integer i for which $e_i \geq 5$ Let B_i, G_i be the number of blocks and gaps, respectively, of length i in s for each i $1 \leq i \leq k$. The statistic used is

$$X_4 = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i}$$

5- Autocorrelation Test

The purpose of this test is to check for correlations between the sequence s and (non-cyclic) shifted versions of it. Let d be a fixed integer, $1 \leq d \leq \lfloor n/2 \rfloor$. The number of bits in s not equal to their d-shifts is $A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$, where $\oplus$ denotes the XOR operator.

The statistic used is

$$X_5 = 2\left(A(d) - \frac{n-d}{2}\right) / \sqrt{n-d}$$

4- Experiment and Results

Now, if we take an enough part of the output sequence of this system and apply the statistical tests which are mentioned above we obtain the results:

Length of sequence 500 bits:-
11101100010111110101001010111100101101000110
11001111101110000001101010110000111101011100
01011000100011110100011010110101110101011100
00000000110011101111111111111111111100110010
00000011110000101100011110110000101011100000
00010011010011111011111000100000100100000100
01000110000001110010100000000111100000010110
10001001011111000010010110111010110000010000
01100000111001011101100011111001101100001000
01111010111100000110100101000111110100010000
01001110111110001000100111001111010111100110
11111010111011001111111110111

1- Frequency Test

in the above sequence n_0 = 244, n_1 = 256 and the value of the static is

X_1 = 0.288.

2- Serial Test

in the above sequence N_{00}= 132 N_{01}= 113 N_{10}= 111 N_{11}=143

X_2=2.416.

3- Poker Test

in the above sequence M=5 and k=100 so, X_3=19.04

4- Runs Test

in the above sequence e_1=62.75 e_2= 31.313 e_3= 15.6215 e_4=7.797 e_5= 3.891 E_6= 1941 e_7=0.969 e_8= 0.483 e_{16}= 0.002.

5- Conclusion

This cryptosystem is not fixed but it is changeable at each run (each operation). in spite of that, the system passed all the standard statistical tests, where the results show that they are very closed to standard states, so the system is suitable to used.

References

[1] A. Menezes, S. Vanstone , **"Handbook of Applied Cryptography"**, CRC Press, Inc., 1997.

[2] Alan G. Konheim, **"Computer Security and Cryptography"**, A John Wiley & sons, Inc. publications, 2007.

[3] Bruce schneier, **"Applied Cryptography, Second Edition"**, A John Wiley & sons, Inc. publications 1996, ISBN: 0471128457.

[4] Christof Paar, **"Applied Cryptography and Data Security"**, Ruhr-University Bochum/ Germany, 2005.

[5] Wenbo Mao Hewlett-Packard Company, **"Modern Cryptography: Theory and Practice"**, Prentice Hall PTR, 2003, ISBN: 0-13-066943-1.

[6] William Stallings, **"Cryptography and Network Security Fifth edition"**, Pearson Education, Inc., 2011.

تقنية جديدة لتوليد مفاتيح شفرية شبه عشوائية

عوني محمد كفطان[1] ، خلود جمال مولود[2]

[1] قسم الرياضيات ، كلية علوم الحاسوب والرياضيات ، جامعة تكريت ، تكريت ، العراق

[2] قسم الرياضيات ، كلية التربية للبنات ، جامعة تكريت ، تكريت ، العراق

Email: khmsc2006@yahoo.com Email: ddar_ardd@yahoo.com

(تاريخ الاستلام: 5 / 5 / 2013 ---- تاريخ القبول: 28 / 10 / 2013)

الملخص

قمنا في هذا البحث بناء منظومة توليد مفاتيح شفرية تعتمد على سلسلة (بنك) من الزواحف الخطية يتم اختيار عدد منها بشكل عشوائي عند كل تنفيذ (تشغيل للمنظومة) وبالتالي فأننا نحصل على منظومة تشفير جديدة عند كل تشغيل.

أي إننا سنقوم باختيار 4 زواحف خطية لكل منظومة (Si) أي يصبح لدينا 16 زاحف جديد في جميع المنظومات عند كل تشغيل، تم اختيارها من 32 زاحف موجودة في الخزين.