Security Evaluation of Nonlinear Filterization Stream Cipher Based on Information Theory

Sattar B. Sadkhan
University of Babylon, College of Information Technology
drensattar@ieee.org

Dhilal M. Reza
University of Babylon
dhilal.mohammad@yahoo.com

Abstract
One of the techniques for destroying the linearity inherent in LFSRs is to generate the keystream as a result of some nonlinear function of the stages of a single LFSR. Such keystream generators are called nonlinear filter generators, and the used nonlinear function is called the filtering function. We have proposed a new method to evaluate the nonlinear filtering function which is based on five parameters (length of register, seed, connection polynomial, filter structure, chosen entrances). To build the nonlinear function, the designer needs to build it in the program and change part of the chosen nonlinear function every time and see the results. We shorten this operation for the designer to write only one equation and the new method will generate all the possible forms for the input equation, and then analyzes the output sequence resulted from each state using frequency and serial tests to determine the best filter function and consequently we knew the probability distribution of the resulted key. Based on the Information Theory, security evaluation of nonlinear filtering stream cipher was computed.

Key word: Security Evaluation, Stream Cipher, nonlinear Filterization

Introduction
Stream cipher is a class of symmetric cipher that operate under the concept of one-time pad. The crucial difference is that a stream cipher only requires a small secret key, whereas a one-time pad cipher requires a key with approximately infinite length [Mark 2007]. Stream ciphers do not offer unconditional security (such as one-time pad), but the hope is that they are computationally secure [Alfred 1996]. A keystream is either randomly chosen, or is generated by an algorithm, called a keystream generator, which generates the keystream from an initial small input key called a seed [Richard 2007]. Linear feedback shift registers (LFSR) are widely used in keystream generators because they are well-suited for hardware implementation, produces sequences having large periods and good statistical properties, and are readily analyzed using algebraic techniques. Unfortunately the output sequences of LFSRs are also easily predictable using the "Berlekamp-Massey algorithm". For destroying the linearity inherent in LFSRs, three general techniques for utilizing it in the construction of stream ciphers: using a nonlinear combining function on the outputs of several LFSRs, using a nonlinear filtering function on the contents of a
single LFSR and using the output of one (or more) LFSRs to control the clock of one (or more) other LFSRs [Alfred 1996]. So the world trended toward nonlinear stream cipher and this is enough reason to think more and more toward how to evaluate nonlinear stream cipher. Information Theory can be used to evaluate the security as it was achieved in [Frédérique 2010] and [Saeed 2008]. In this paper we study the security of the system based on the second approach. The appropriate framework in which to study unconditional security is probability theory [Douglas 2007].

2- Security Evaluation

There are various approaches to evaluate the security of a cryptosystem such as computational security (this measure concerns the computational efforts required to break a cryptosystem. A cryptosystem is defined to be computationally secure if the best algorithm for breaking it requires at least N operations, where N is some specified, very large number), provable security (this approach provide evidence of security by means of a reduction) and unconditional security (a cryptosystem is defined to be unconditionally secure if it cannot be broken, even with infinite computational resources).

2.1. Shannon’s ideas (Information Theory)

- Entropy: the amount of information in a message is measured by the entropy of the message. If \( X_1, X_2, \ldots, X_n \) are n possible messages and \( P(X_1), P(X_2), \ldots, P(X_n) \) are their perspective probabilities of occurring, then the entropy of a message is defined as:

\[
H(X) = - \sum P[X_i] \log_2 P[X_i]
\]  

(1)

And the entropy of a message, \( X \), also measures its uncertainty, in that it indicates the number of bits of information that must be acquired to recover a message distorted by a noisy channel or concealed through ciphers [Jennifer 1989].

- Conditional entropy (equivocation): Shannon measured the secrecy of a cipher with respect to its key equivocation \( H(K|Y) \); for cipher text Y and key K, it may be interpreted as the degree of uncertainty in K given Y, and expressed as:

\[
H(K|Y) = - \sum_y \sum_k Pr[y] Pr[k|y] \log_2 Pr[k|y]
\]  

(2)

Where \( Pr(k|y) \) is the probability of k given y. If \( H(K|Y) \) is zero then there is no uncertainty in the cipher, making it breakable [Douglas 2007].

- The unicity distance: is the amount of ciphertext needed to uniquely determine the key, where \( R_L \) is the redundancy of the underlying language and \( n_0 \) is the unicity distance [Dorothy 1982].

\[
n_0 \approx \frac{\log_2 |K|}{R_L \log_2 |X|}
\]  

(3)

2.2. Security Evaluation parameters

To evaluate the security of a cipher based on information theory, the cipher would be treated as a black box and the parameters that are needed to achieve this task as follows:

- The probability distribution of plaintext.
- The probability distribution of ciphertext.
- The probability distribution of keytext.

In this paper we evaluate a nonlinear filterization stream cipher, but the difficulty that faced us was, what is the probability distribution for the key text? So we proposed a new method to help us to calculate it and produced the best structure for filtering function to make the nonlinear filtering stream cipher has a good security.

3- A proposed method for security evaluation of keystream generator for nonlinear filtering stream cipher.

The evaluation of the security level of this cipher depend on five factors which
are:
1- The seed (or the initial state) of LFSR.
2- The connection polynomial (primitive) of LFSR.
3- The length (number of flip flops used in structure) of LFSR.
4- The structure of the filtering function.
5- The entrances of the nonlinear filtering function.

The initial state of LFSR can be used easily to determine the previous three factors, i.e. the set of possible values for seed, connection polynomial and the length of the register, and the entrances to the nonlinear filtering function will be chosen by the designer. But the filtering function can be constructed in different ways, and It is so difficult to test all these ways especially when someone race the time, so as a result, we proposed a method to test all these ways easily in shortest possible time.

In Figure 1 the components of the filtering function consists of four major entrances come out from LFSR and three Boolean functions. So there are many different ways for connecting these Boolean functions in addition to the connection way illustrated in fig 1, and depending on permutation and combination rules we can know the number of possible rules for changing the positions of the Boolean functions.

Permutations with Repetitions: The number of permutations of n objects of which p are alike and q are alike is [Henk 2007]:
\[
\frac{n!}{p!q!}
\] ...........................................................................(4)

Combination: The number of combinations of n objects taken r at a time is defined as [Henk 2007]:
\[
\binom{n}{r} = \frac{n!}{(n-r)!r!}
\] ...........................................................................(5)

We have two binary Boolean functions (and, or), and it can be connected in three different ways ((and, and, or)-(and, or, and)-(or, and, and)), using permutation rule as follows:
\[
\frac{n!}{p!q!} = \frac{3!}{2!1!} = 3
\]

And the Boolean function (not) can be placed in seven positions as illustrated in figure 1, and we can computed that depending on combination rule.
\[
\sum_{i=0}^{m} \binom{m}{i} = C(7,0) + C(7,1) + C(7,2) + C(7,3) + C(7,4) + C(7,5) + C(7,6) + C(7,7) = 127
\]

So the total number of changing the positions of the three Boolean functions (and, or, not) will be :-
\[
\text{total}= 3*127 = 381
\]

the structure of a nonlinear filtering function must be written in formula similar to arithmetic equation and must has the following specifications :-
1- Each number must be parenthesized by circular parentheses.
2- Use specific symbols to indicate to the Boolean functions such as
("+" for OR, "." for AND, "!'" for NOT) and treat it inside program as Boolean functions.

3- Parenthesize each Boolean function either unary or binary Boolean function and theirs accompanying numbers by a circular parentheses.

4- In the successive algorithms we will name the number as operand and the Boolean function as operator.

After the equation was written, the program will analyze the equation and generate all the possible forms of the this equation using algorithm "Fil_Str_Equ(eq,arr_eq)", and then generate the keystream for each equation using algorithm "generate_outseq(new_eq,outseq,max_length)", and finally the key stream will be tested using serial and frequency tests (and also can be tested using different statistical tests) to rule on the degree of security ("good", "bad")

Up to this point we compute the security of the keystream generator, and we determined our suitable filter function, the security of the whole cipher can be computed based on information theory.

Algorithm generate Fil_Str_Equ(eq,arr_eq)

Input: eq;  //the main equation is entered by the evaluator.
Output: arr_q; //array of the possible filter structure equations that can be generated from the main equation.

Begin

Step1: Trace the main equation (eq) and count the number of binary Boolean function ("+", ".", "+*,") in it, and store it in array (seqch).

Step2 :Make a permutation for the contents of array (seqch) and construct a new equation for each permutation and store it in array of equations.

Step3 : For each equation constructed in Step2 do the following :-

Step3-1: Trace the equation and count the number (p) of occurring the left parenthesis "(".

Step3-2: Compute the number of possible positions(s) that can be preceding by unary Boolean function NOT "!'". s=2^p-1.

Step3-3: for (s) times do the following steps :-

Step3-3-1: compute the value of (p_not) in binary notation ,such that each one digit (1) indicates the position of "!'", p_not=dec2bin(i,s).

Step3-3-2: trace the equation one character at a time and do the following steps:-

Step3-3-2-1: if the character is "(" and if the corresponding position in p_not is "1" then precede the left parenthesis "(" by NOT Boolean function "!'" and store it in new equation (neweq).

else store the equation as it without change.

Step3-3-2-1: Store the new equation in array of equations(final array). x=x+1;

arr_eq(x)=eq.

End

Algorithm generate_outseq(new_eq,outseq,max_length);

Input: new_eq,max_length;  // new_eq :string represents the equation of the filter structure, max_length: represent the length of the output sequence results from nonlinear filterization stream cipher.

Output: outseq; //represent the output sequence results from non linear filterization cipher.

Begin

Step1 :Use two stacks ,the first (st1) to store operators and the second (st2) to store operands and the result will remain in it.

Step2 :Trace the equation (new_eq) one character at a time from left to right.

1865
Step3 : At each character do the following :-
Step3-1: If the character is one of these characters {+,.,*} , then store in st1.
Step3-2: If the character is operand, then store in st2.
Step3-3: If the character is ")" , then do the following:-
Step3-3-1: Pop the the contents (operator:one Boolean function) of st1 until arriving to the first "(" encountered.
Step3-3-2: Depending on the operator in Step3-3-1, if it is binary boolean function, then pop two operands from st1, else pop one operand.
Step3-3-3: Apply the operator ( Boolean function ) resulted from Step3-3-1 to the operands (numbers) resulted from Step3-3-2, and store the result in st2.
Step4 : Pop the final result in st2 and store it in string (outseq).
Step5 : Repeat the steps from 1 to 4 as many as max_length.

4- Results.
Case study (1)
In this case study as illustrated in table 1, we stay put the length of LFSR and the entrances out of LFSR.
1. For the three other factor, we stay put two factors and change the third (i.e. variable). When seed is variable then seed will take (31) values from (00001 to 11111), and when connection polynomial is variable then it will take all the primitive polynomial values related to the length of the used LFSR, and finally when filter function equation is variable then it will take all possible forms of this equation that can be generate as illustrated in table 4.

<table>
<thead>
<tr>
<th>Evaluation Factor</th>
<th>seed</th>
<th>(Connection polynomial)$_{dec}$</th>
<th>Filter function equation</th>
<th>Elapsed time</th>
<th>No. of good degree</th>
<th>No. of bad degree</th>
<th>No. of experiments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seed</td>
<td>variable</td>
<td>37</td>
<td>(((1).(3))+((4).(5)))</td>
<td>6 sec.</td>
<td>0</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>Connection polynomial</td>
<td>01101</td>
<td>variable</td>
<td>(((1).(3))+((4).(5)))</td>
<td>3 sec.</td>
<td>0</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Filter structure</td>
<td>01101</td>
<td>37</td>
<td>variable</td>
<td>50 sec.</td>
<td>6</td>
<td>375</td>
<td>381</td>
</tr>
</tbody>
</table>

Case study (2)
In this case study as illustrated in table 2, we repeat the same thing for case study 1, but change the factor (length of register).

<table>
<thead>
<tr>
<th>Length of LFSR</th>
<th>seed</th>
<th>No. of good degree</th>
<th>No. of bad degree</th>
<th>Elapsed time</th>
<th>Connection polynomial</th>
<th>No. of good degree</th>
<th>No. of bad degree</th>
<th>Elapsed time</th>
<th>Filter structure</th>
<th>No. of good degree</th>
<th>No. of bad degree</th>
<th>Elapsed time</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>31</td>
<td>6 sec</td>
<td>0</td>
<td>6</td>
<td>3 sec</td>
<td>6</td>
<td>375</td>
<td>50 sec</td>
<td>0</td>
<td>381</td>
<td>92 sec</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>63</td>
<td>15 sec</td>
<td>0</td>
<td>6</td>
<td>5 sec</td>
<td>0</td>
<td>381</td>
<td>92 sec</td>
<td>0</td>
<td>381</td>
<td>180 sec</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>127</td>
<td>44 sec</td>
<td>0</td>
<td>18</td>
<td>7 sec</td>
<td>0</td>
<td>381</td>
<td>180 sec</td>
<td>0</td>
<td>381</td>
<td>360 sec</td>
</tr>
</tbody>
</table>

Case study (3)
In this case as illustrated in table 3, we repeat the same thing for case study 1, but change the factor.
(the entrances out of LFSR to the nonlinear filtering function, length of LFSR).

**Table 3**

<table>
<thead>
<tr>
<th>Lenght of LFSR</th>
<th>Chosen intrances</th>
<th>Filter function equation</th>
<th>seed</th>
<th>Connection polynomial</th>
<th>Filter structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1,3,4,5</td>
<td>(((1),(3))+((4),(5)))</td>
<td>0</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>375</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1,2,3,5</td>
<td>(((1),(2))+((3),(5)))</td>
<td>10</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>381</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1,2,4,5</td>
<td>(((1),(2))+((4),(5)))</td>
<td>0</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>369</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1,2,2,5</td>
<td>(((1),(2))+((2),(5)))</td>
<td>0</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>333</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1,3,2,4,5,7</td>
<td>((((1)+(3)),((2)+(4)))+((5),(7)))</td>
<td>0</td>
<td>127</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4,6,7,1,5</td>
<td>((((6),(6)),(7)))+((1),(5))</td>
<td>127</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1023</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4**

<table>
<thead>
<tr>
<th>([(1),(3)]+([4],[5]))</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Journal of Babylon University/Pure and Applied Sciences/ No.(7)/ Vol.(22): 2014**
Case study (4)

In this case study as illustrated in table 5, a comparison between the security of two filterization stream cipher have the following equations as filtering functions.

Table 5

| Filter structure equation | Security degree | Probability (K=0) | Probability (K=1) | H(p) | H(C) | H(K|C) | Unicity distance |
|---------------------------|----------------|-------------------|-------------------|------|------|--------|-----------------|
| (((4.6), (7)) + ((1), 5)) | good           | 0.472441          | 0.527559          | 0.973578 | 0.702805 | 1.29287 | 9.31825         |
| (((4.6), (7)) + ((1), 5)) | bad            | 0.1575            | 0.84252           | 0.973578 | 0.832767 | 1.8586 | 9.31825         |

5- Conclusions

For case studies 1, 2, 3, we have the following conclusions:

1. From the results we see that the chosen entrances play an important role, and the suitable arrangement for the Boolean function in nonlinear function determine the best structure.
2. The length of the LFSR do not affect on the security if the results in point 1 not verify.
3. Using all the values of primitive polynomial related to the length of LFSR for connection polynomial have the same effect.

According to information theory criteria, the decreasing in the value of entropy and conditional entropy give us a pointer towards strong security.

The security of stream cipher depends on the security of key generators, and consequently it will produce a good probability distribution for the key text, which the information theory approach for security evaluation depend on.

The unicity distance has no change because we compare between two ciphers with the same key space.

References