CASTREAM: A New Stream Cipher Suitable for Both Hardware and Software

Sourav Das and Dipanwita Roy Chowdhury

Alcatel-Lucent India Ltd and Dept. of CSE, IIT, Kharagpur, India
sourav10101976@gmail.com, drc@cse.iitkgp.ernet.in

Abstract. A new Cellular Automata based stream cipher is proposed which is suitable for both hardware and software. It has a non-linear combiner where two non-linear blocks along with a linear block are linearly combined to produce the key-streams. Unlike Non-linear Feedback Shift Register (NFSR) based non-linear combiners, it combines 128-bit blocks using parallel evolution of Cellular Automata (CA) and small CA based S-boxes. The usage of CA prevents the correlation attack and two layers of re-usable small S-boxes prevent the algebraic attacks. The proposed stream cipher takes 128 bits Key and 128 bits of Initial Vector(IV). Theoretically, the cipher operates with an encryption speed of nearly 8 bits per cycle. The initialization process needs 96 cycles which is much faster than Grain and Trivium. This stream cipher is extensible in terms of Key size and provides configurable security and vendor specific implementation option. On implementation, the proposed cipher receives higher throughput than the existing standards.

Keywords: Stream Cipher, Security Properties, Cellular Automata, S-box.

1 Introduction

Stream ciphers have gained a lot of attention in recent years and provide fast encryption. In stream ciphers a key stream sequence is generated from a secret key. The plain-text is simply XORed with the key stream to encrypt the message at the encryption site. At the decryption site, the same key stream is generated with the same secret key. The cipher-text is XORed with the key-stream to get back the plain-text. The ESTREAM [13] project had been launched to search for a good stream cipher that can provide advantage in performance over block ciphers.

The ESTREAM project has divided the stream ciphers into two categories, namely, software oriented stream ciphers that provide fast encryption in software and hardware oriented stream ciphers that provide fast encryption with easy hardware implementation. Traditionally, the hardware oriented stream ciphers used LFSRs (Linear Feedback Shift Register) filtered with a non-linear function. But such kind of stream ciphers have been subjected to correlation attacks [10], [9], and algebraic attacks [4], [5]. Later, stream ciphers started using NFSRs,

G.C. Sirakoulis and S. Bandini (Eds.): ACRI 2012, LNCS 7495, pp. 601–610, 2012.

which are LFSR with non-linear feedback. The NFSRs are linearly combined to produce the key-streams. ESTREAM finalist Trivium [3] is an example of such stream ciphers. The stream cipher Grain [8] uses a combination of linearly filtered NFSR with a non-linearly filtered LFSR. Mickey [1] is another finalist in ESTREAM hardware profiles, and it uses variable clocking method. The software oriented stream ciphers, however, in general use block oriented state transitions that help in efficient software implementation. Examples of such stream ciphers are Rabbit [2], HC-128 [16] etc.

Cellular Automata (CA) generate statistically random sequences which are necessary for stream ciphers. CA also provide parallel transformations that help to achieve more throughput which is also essential for stream ciphers. The use of Cellular Automata in stream ciphers was proposed first by Wolfram using Rule 30 [14], [15]. However, Meier and Staffelbach found weaknesses in that stream cipher [11]. But, the weaknesses of these stream ciphers may not be because of any inherent weaknesses of CA, since CA are just primitives that generate statistically random sequences. The problem was the way CA were used in those stream ciphers. If CA are used a bit differently in the construction of stream ciphers, CA can still prove to be excellent primitive for stream ciphers.

In [7] and [6], it was shown how to generate highly non-linear S-boxes using CA. These papers demonstrate CA can be effective cryptographic primitives. These S-boxes can be used in a stream cipher to enhance the security. The S-boxes are dynamically generated, scalable and secure with efficiency in implementation. In this paper, the idea of S-box generation in [6] is extended to generate the non-linear block of a stream cipher. The uniqueness of this non-linear block is that it uses CA based small non-linear generators with CA based mixing among the generators. The advantages of such a non-linear block are higher throughput, scalability and statistically random output sequences. Using two such non-linear blocks and a linear block of maximum length linear CA, a stream cipher, *CASTREAM*, with 128 bits Key and 128 bits IV is designed in this paper.

This paper is organized as follows. Section 2 describes the cipher along with the initialization process. The security properties of the cipher is analyzed in section 3. Section 4 provides the implementation aspects of the cipher which has two subsections for hardware and software implementations.

2 Description of the Stream Cipher

The high level block diagram of the construction is shown in Figure 1. This stream cipher has three 128 bits blocks of state which are linearly combined to produce 128 bit key-stream in each round. Two of the three blocks perform non-linear transitions and the third block performs linear transition which guarantees the period. The linear block is a 128-bit maximum length CA using rules 90/150. This CA run for sixteen cycles in each iteration. The non-linear parts are designed using a series of dynamically generated CA based S-boxes. The S-boxes are generated by using the principle of input governing the number of

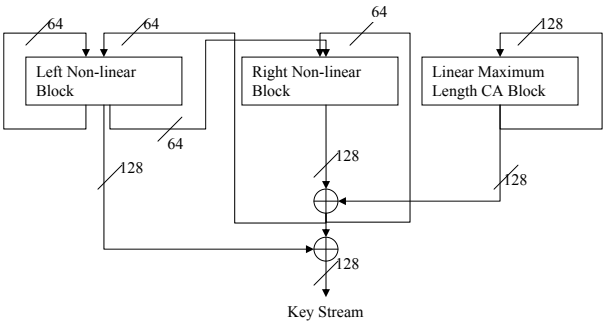

Fig. 1. The Stream Cipher Block Diagram

cycles of a maximum length CA with a constant initial seed as shown in [7]. Each non-linear block also has a 128-bit maximum length linear CA to achieve mixing among the S-boxes with good statistical properties. Both the non-linear blocks contain two non-linear layers with a maximum length linear CA sandwiched between them to achieve mixing. The linear CA in the non-linear block also run for sixteen cycles. The non-linear layers contain a series of 4-bit, 5-bit and 6-bit S-boxes. These S-boxes are mixed in such a way that they give rise to four 32-bit blocks of S-boxes. Two different combinations of the small S-boxes are used to form the thirty-two bits blocks. The first combination contains 5-bit, 4-bit, 5-bit, 4-bit, 5-bit, 4-bit and 5-bit S-boxes in order. The second combination contains 4-bit, 5-bit, 4-bit, 6-bit, 4-bit, 5-bit and 4-bit S-boxes in order. Each combination is repeated four times to constitute the non-linear layer. In the left hand side non-linear block, the first combination makes the first non-linear layer and the second combination constitutes the second non-linear layer as shown in figure 2. The right hand side non-linear block consists of the second combination in the first non-linear layer and the first combination in the second non-linear layer as shown in Figure 3.

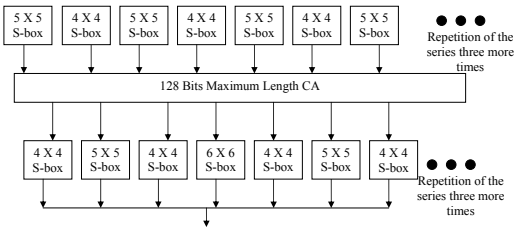

Fig. 2. The Processing of Left Hand Side Non-linear Block

In each iteration, the linear block feedbacks to itself. The two non-linear blocks send feedback to the other block as well as to itself as follows. Let us divide the each 128 non-linear state bits into eight 16-bits blocks. Then these blocks are

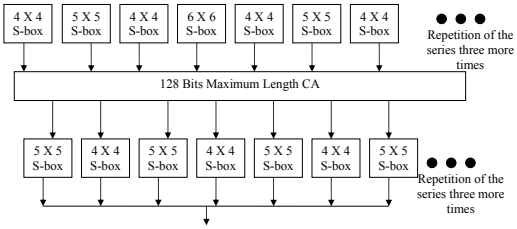

Fig. 3. The Processing of Right Hand Side Non-linear Block

permuted so that each alternate block is taken from the left and right hand side non-linear blocks. If we denote the left state bits $(L_1 \cdots L_{128})$, the right state bits as $(R_1 \cdots R_{128})$ and the linear state bits as $(C_1 \cdots C_{128})$, then after permutation the left state bits become:

$(L_1 \cdots L_{16}, R_1 \oplus C_1 \cdots R_{16} \oplus C_{16}, L_{17} \cdots L_{32}, \cdots, L_{49} \cdots L_{64}, R_{49} \oplus C_{49} \cdots R_{64} \oplus C_{64})$.

Similarly, the right hand side state bits become:

$(L_{65} \cdots L_{80}, R_{65} \oplus C_{65} \cdots R_{80} \oplus C_{80}, \cdots, L_{113} \cdots L_{128}, R_{113} \oplus C_{113} \cdots R_{128} \oplus C_{128})$.

The key stream is generated by XORing all the three blocks. The encryption is performed after XORing the key stream with the plain-text. The decryption is performed after XORing the key stream with the cipher-text.

2.1 Initialization

The initialization of the cipher uses the same method as the key stream generation. On the left hand side non-linear block, the Key bits are loaded into the state registers. On the right hand side non-linear block, the IV bits are loaded. For the linear side, XOR of Key and IV bits are loaded. After loading the Key and the IV bits, the cipher is iterated six times without generating any key stream. While initializing, the only difference from the key stream generation is that the linear block is fed from the XOR output of the internal states. Also, after the Key and the IV are loaded, the last bit in the linear CA is always

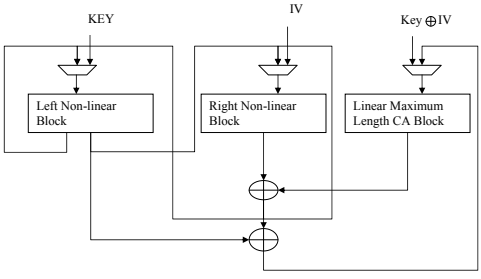

Fig. 4. The Cipher Initialization

kept at logic one during initialization. The key-streams are taken out from the seventh iteration. The initialization process is shown in Figure 4.

2.2 Parameterizations

The parameterizations of the cipher help different vendors implement the cipher in a different way. A different parameter value will result in a different key-stream sequence even with the same Key-IV pairs in the following way. Each of the S-boxes takes the seed as a parameter. There are four 128-bits layers of S-boxes in the non-linear blocks. Each layer takes a parameter of 128 bits. Hence, the total size of the parameter is $128 \times 4 = 512$ bits. However, care should be taken that none of the S-boxes is parameterized with all zeros. To make the parameterizations simple, every fourth bit of the 128-bits parameter is made to logic one. This makes the user parameter size as $96 \times 4 = 384$.

3 Security Analysis

In this section, we provide security analysis of the cipher.

3.1 An Analysis of the Cipher

Let, f_{NL}, f_L denote the non-linear function of the S-boxes and linear function of maximum length CA, respectively. Also, let K_i, and IV_i denote the i-th, $(i \in (1 \cdots 128))$, bit of the key and the initial value. $L_{i,j}$, $R_{i,j}$ and $C_{i,j}$ denote the i-th, $(i \in (1 \cdots 128))$, bit of the the state bits of left hand side non-linear block, right hand side non-linear block and linear CA block, respectively, during some iteration j. For clarity we analyze the evolution of first bit (i=1) for all the blocks below. The evolution of other bits can easily be extended from this analysis.

The Evolution of the First Bit of All the Blocks. The equation for the left hand side non-linear block after the first S-box layer transformation (denoting it as $LS1_{i,j}$) is, since the first S-box transforms five bits:

$$LS1_{1,j} = f_{NL}(L_{1,j}, L_{2,j}, L_{3,j}, L_{4,j}, L_{5,j}) \tag{1}$$

After the linear transformation (denoting the bits as $LC1_{i,j}$), the equation for the first bit becomes, since after sixteen cycles each bit in the CA will depend on itself, 16 more bits from left and 16 more bits from right:

$$LC1_{1,j} = f_L(LS1_{1,j}, \cdots, LS1_{17,j}) \tag{2}$$

Expanding, as $LS1_{17,j}$ is the part of 4×4 S-box,

$$LC1_{1,j} = f_L(f_{NL}(L_{1,j}, \cdots, L_{5,j}), \cdots, f_{NL}(L_{15,j}, \cdots, L_{18,j})) \tag{3}$$

Then occurs the second layer of S-box substitution. If $LS2_{i,j}$ denotes the output of this layer, then the expression for the first bit is:

$$LS2_{1,j} = f_{NL}(LC1_{1,j}, \cdots, LC1_{4,j}) \tag{4}$$

Expanding, since $LC1_{4,j}$ will depend on sixteen bits from right side and will also be affected by all the bits in the left hand side,

$$LS2_{1,j} = f_{NL}(f_L(LS1_{1,j}, \cdots, LS1_{17,j}), \cdots, f_L(LS1_{1,j}, \cdots, LS1_{20,j})) \tag{5}$$

Expanding, since $LS1_{20,j}$ depends on $L_{19,j}, \cdots, L_{23,j}$

$$LS2_{1,j} = f_{NL}(f_L(f_{NL}(L_{1,j}, \cdots), \cdots, f_{NL}(\cdots, L_{23,j})), \cdots) \tag{6}$$

Note that, $LS2_{i,j}$ is nothing but $L_{1,j+1}$. Also, $L_{17,j} \cdots L_{23,j}$ are $R_{1,j} \oplus C_{1,j} \cdots R_{7,j} \oplus C_{7,j}$ due to the swap of bits at the end of every round. Hence, we can write, denoting F_{NLL} as the transformation function of the left hand side non-linear block (which is the right hand side of the equation 6):

$$L_{1,j+1} = F_{NLL}(L_{1,j}, \cdots, L_{16,j}, R_{1,j}, \cdots, R_{7,j}, C_{1,j} \cdots C_{7,j}) \tag{7}$$

In a similar manner, we can show that in the right hand side also, the expression for the first bit is:

$$R_{1,j+1} = F_{NLR}(L_{65,j}, \cdots, L_{80,j}, R_{65,j}, \cdots, R_{71,j}, C_{65,j}, \cdots C_{71,j}) \tag{8}$$

where, F_{NLR} is the transformation function of the right hand side, which is similar to the expression in equation 6, but the functions are applied in the reverse order.

Then, it is easy to derive the equation for the first bit of the linear CA block, which is, since this block runs for 16 cycles:

$$C_{1,j+1} = f_L(C_{1,j}, \cdots, C_{17,j}) \tag{9}$$

Finally, the equation for the first bit of the 128 block of key stream is:

$$KS_{i,j+1} = L_{1,j+1} \oplus R_{1,j+1} \oplus C_{1,j+1} \tag{10}$$

where, $L_{1,j+1}, R_{1,j+1}, C_{1,j+1}$ are as in equations 7, 8 and 9.

Expression for other Bits. Since the non-linear blocks use asymmetric S-box transformation, the expression of each bit will be different. Hence, it is difficult to write generalized expression for all the 128 bits. If we simplify the S-boxes to have only uniform 4-bits S-boxes, then we can make an attempt. In that case, the first bit is affected by four bit tuple it belongs to and another four 4-bits tuples from immediate right. It can be also be shown that another four 4-bits tuple affect each state bit from the left for the bit numbers $i \in (17 \cdots 128)$ for both left hand side and the right hand side non-linear blocks. So, we can write a

generic expression for any non-linear next state bit in the simplified non-linear block, $X_{i,j+1}$ in terms of the previous state bits $X_{i,j}$, where i denotes the i-th state bit ($i \in (33 \cdots 96)$) and j denotes the iteration number.

$$X_{i,j+1} = F(X_{i-(16+(imod4)),j}, \cdots, X_{i+20-(imod4),j}) \tag{11}$$

where, F denotes the transformation function of the non-linear block. Hence, each state bit will depend on at least 36 bits from the previous state.

For the linear state bits the corresponding equation is:

$$C_{i,j+1} = f_L(C_{i-16,j}, \cdots, C_{i+16,j}) \tag{12}$$

Key and IV Mixing During Initialization. We show now that each state bit after initialization depends on all the Key and the IV bits. We divide the left hand side non-linear block and the right hand side non-linear block into eight sixteen bits blocks. Let us denote them $L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8$ for the left hand side non-linear block and $R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8$ for the right hand side non-linear block. We use the following facts.

1. After sixteen cycles in the linear CA of the non-linear block, every bit in the each sixteen bit block depends on all the bits in that block.

2. After both the S-box transformations and linear CA transformation running for sixteen cycles, all the bits from the immediate neighboring blocks affect at least one bit in the current block. In addition, a few bits from neighbor's neighbor block affect at least one bit in the current block.

We also simplify the analysis by ignoring the XOR from the linear block. However, due to this XOR, if we prove that after the initialization the right block affects all the bits in the left block, then, it will automatically imply that the linear block also affects all the state bits in the non-linear blocks. Also, due to the XOR feedback to the linear block, the proof will also imply that the linear block is also affected by all the state bits.

Now let us see how the permutation at the end of the round takes place for different rounds during initialization.

Round 1: $L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8,\ R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8$
Round 2: $L_1, R_1, L_2, R_2, L_3, R_3, L_4, R_4,\ L_5, R_5, L_6, R_6, L_7, R_7, L_8, R_8$
Round 3: $L_1, L_5, R_1, R_5, L_2, L_6, R_2, R_6,\ L_3, L_7, R_3, R_7, L_4, L_8, R_4, R_8$
Round 4: $L_1, L_3, L_5, L_7, R_1, R_3, R_5, R_7,\ L_2, L_4, L_6, L_8, R_2, R_4, R_6, R_8$
Round 5: $L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8,\ R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8$

Now, if we start backtracking from round 5 for each of the blocks, we can easily see that all sixteen blocks have affected each of the block. We show this for the first block below.

After Round 1, the block L_1 is affected by all the bits in the block L_2 (plus some more bits from L_3, which we ignore for this example). At Round 2, L_1 is affected by R_1. But R_1 is already affected by all the bits in R_2 in Round 1. Hence, after Round 2, L_1 depends on L_2, R_1, R_2.

At Round 3, L_1, depends on all the bits from L_5. But, in Round 2 $L5$ depends on all the bits from R_5. In Round 1, L_5 depends on all the bits of L_4 and L_6.

Again, in Round 1 R_5 depends on all the bits of R_4 and R_6. Hence, at the end of Round 3, L_1 depends on $L_2, L_4, L_5, L_6, R_1, R_2, R_4, R_5, R_6$.

At Round 4, L_1, depends on all the bits from L_3. But, in Round 3 $L3$ depends on all the bits from L_7 which depends on L_6 and L_8 at the first round itself. In Round 2, L_7 depends on R_6 and R_7. In Round 1, R_7 depends on R_6 and R_8. Again, in round 2 L_3, depends on R_2 and R_3. Hence, at Round 4, L_1 depends on all the sixteen 16-bits blocks, namely, $L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8$, $R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8$.

In a similar manner, it can be shown that all other blocks are affected by rest of the blocks after four round of initialization. Before the Round 1 all the blocks are initialized with the Key bits and IV bits. Hence, after four rounds of initialization, all state bits are affected by all the key bits and the IV bits.

3.2 Statistical Tests

The output of the stream cipher was evaluated for statistical randomness using NIST [12] statistical test tool. This test tool contains sixteen statistical tests to evaluate a random number generator. We generated 128 million bits of key streams and used the tool for the evaluation with 10 bitstreams of size 12.8 million each. The following Key and IV pairs are used:
$Key : 0xFEDCBA98765432100123456789ABCDEF$
$IV : 0x0123456789ABCDEFFEDCBA9876543210$.
The default parameters of the tools were used except for the block length of Approximate Entropy where the block length used was 4. With these inputs, the tool performed a number of statistical tests out of which 15 tests failed and 1829 tests passed. Table 1 shows the result of the statistical tests with the first bit stream. $p - value > 0.01$ implies that the tests are passed.

Table 1. Statistical Tests Summary

Test	p-val	Test	p-val
Frequency	0.032	Longest Run	0.782
Block Freq	0.726	Non-overlapping Template	0.369
Runs	0.789	Overlapping Template	0.356
Rank	0.564	Random Excur	0.065
FFT	0.664	Random Excur Var	0.548
Universal	0.747	Approx Entropy	0.247
Serial	0.560	Lin Complexity	0.513
CuSums	0.062	CuSums(Reverse)	0.050

4 Implementation and Performance

In this section we provide the architecture for hardware implementation along with the software implementation.

4.1 Hardware Implementation

The hardware architecture of the key stream generation algorithm of the proposed cipher is shown in Figure 5. The hardware layout is made same as the structure of the cipher. The S-boxes are implemented by using ROMs. The CA are implemented using combinational logic. There is a 128 bit flip-flop layer in each part of the state that perform the sequential part of the CA of running sixteen cycles. The flip-flops in the non-linear layer takes the input from the first S-box layer on Reset and from the output of the CA combinational circuit at each cycle. The output of the CA combinational part is sent to the second S-box layer whose output is connected to a three input XOR layer. The XOR output is fed to the Key Stream flip-flops. After running sixteen cycles, a counter enables the Key Stream flip-flops where the key-stream can be extracted. The state flip-flops are Reset again with the wire permutation of the their contents at the end of sixteen cycles. The linear block also consists of a series of flip-flops with CA combinational logic. The CA in the linear block always take their inputs from the outputs of the CA combinational logic.

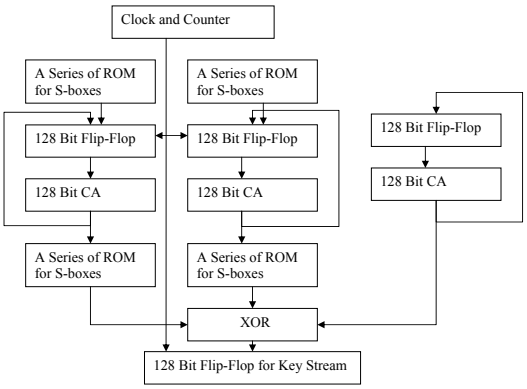

Fig. 5. The Hardware Architecture

The above circuit was implemented using Xilinx 7.i synthesis tool. The speed was 183 MHz and the number of slices required was 1196. Since, at each iteration it provides a key stream block of 128 bits, the throughput is $183 \times 128 = 23.4$ GBPS, which is very high and better than all the ESTREAM candidates. The throughput to area ratio is 19.4 which is again higher than all the existing stream ciphers. On post place and route (PPR) results, the area required was more and the speed was less than the synthesis results.

5 Conclusion

In this paper, a new stream cipher is proposed based on Cellular Automata (CA). The cipher is secure and suitable for both hardware and software. It operates at

a good speed. The cipher can be used both in small and large devices. It is also extensible to a higher key size.

References

1. Babbage, S., Dodd, M.: The stream cipher MICKEY 2.0., http://www.ecrypt.eu.org/stream/mickeyp3.html
2. Boesgaard, M., Vesterager, M., Pedersen, T., Christiansen, J., Scavenius, O.: Rabbit: A New High-Performance Stream Cipher. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 307–329. Springer, Heidelberg (2003)
3. De Canniere, C., Preneel, B.: Trivium Specification, http://www.ecrypt.eu.org/stream/triviump3.html
4. Courtois, N.T.: Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 176–194. Springer, Heidelberg (2003)
5. Courtois, N., Meier, W.: Algebraic Attacks on Stream Ciphers with Linear Feedback. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 345–359. Springer, Heidelberg (2003)
6. Das, S., Roy Chowdhury, D.: An Efficient, Parameterized and Scalable S-box for Stream Ciphers. In: Lai, X., Yung, M., Lin, D. (eds.) Inscrypt 2010. LNCS, vol. 6584, pp. 77–94. Springer, Heidelberg (2011)
7. Das, S., Roy Chowdhury, D.: Generating Cryptographically Suitable Non-linear Maximum Length Cellular Automata. In: Bandini, S., Manzoni, S., Umeo, H., Vizzari, G. (eds.) ACRI 2010. LNCS, vol. 6350, pp. 241–250. Springer, Heidelberg (2010)
8. Hell, M., Johansson, T., Meier, W.: Grain - A Stream Cipher for Constrained Environments, http://www.ecrypt.eu.org/stream/Grainp3.html
9. Johansson, T., Jönsson, F.: Improved Fast Correlation Attacks on Stream Ciphers via Convolutional Codes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 347–362. Springer, Heidelberg (1999)
10. Meier, W., Staffelbach, O.: Fast Correlation Attacks on Stream Ciphers. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 301–314. Springer, Heidelberg (1988)
11. Meier, W., Staffelbach, O.: Analysis of Pseudo Random Sequences Generated by Cellular Automata. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 186–199. Springer, Heidelberg (1991)
12. NIST Statistical Test Suit, http://csrc.nist.gov/rng/
13. The Estream Project, http://www.ecrypt.eu.org/stream/
14. Wolfram, S.: Random Sequence Generation by Cellular Automata. Advances in Applied Mathematics 7, 123 (1986)
15. Wolfram, S.: Cryptography with Cellular Automata. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 429–432. Springer, Heidelberg (1986)
16. Wu, H.: The Stream Cipher HC-128, http://www.ecrypt.eu.org/stream/p3ciphers/hc/hc128_p3.pdf