Cryptogr. Commun. (2013) 5:137–162
DOI 10.1007/s12095-012-0079-1

CAR30: A new scalable stream cipher with rule 30

Sourav Das · Dipanwita RoyChowdhury

Received: 31 December 2011 / Accepted: 29 December 2012 / Published online: 7 February 2013

Abstract *CAR30* is a new stream cipher that uses classical Rule 30 of Cellular Automata (CA) along with a Maximum Length Linear Hybrid CA. This design can be implemented efficiently both in hardware and software. It has a fast initialization algorithm that makes it suitable for small messages. The generic design of the cipher enables to scale up for any length of Key and IV. This paper describes the cipher with 128-bit Key and 120-bit IV and evaluates the security and implementation aspects of it. The main advantages of the proposed cipher are the flexibility of its design, good hardware throughput in comparison with state-of-the-art hardware oriented ciphers like Grain and Trivium and better software speed than the software oriented stream cipher Rabbit.

Keywords Stream cipher · ESTREAM · Cellular automata · Rule 30 · Grain family of stream ciphers

Mathematics Subject Classification (2010) 94A60

1 Introduction

Stream ciphers provide a high encryption/decryption speed that can be used in communication channels. Cellular Automata (CA), due to their speed and randomness in their sequences, can be a very good primitive for stream ciphers. The hardware implementation of CA is simple. Since they have a regular structure, it is possible to find an efficient software implementation for eight, sixteen, thirty-two and sixty-four

S. Das (✉)
Alcatel-Lucent India Ltd, Bangalore, India
e-mail: sourav10101976@gmail.com

D. RoyChowdhury
Department of CSE, Indian Institute of Technology, Kharagpur, India

bits architectures. The CA structures can be extended to accommodate a bigger key length. CA also provide parallel transformations that help to achieve more throughput which is essential for stream ciphers. It was shown in [24] that CA can provide resistance to correlation attacks which are one of the most devastating attacks on stream ciphers.

The use of Cellular Automata in stream ciphers was proposed by Wolfram way back in 1985. Wolfram used rule 30, where the next state of the CA is defined as *(left-neighbor XOR (self OR right-neighbor))*, for building the stream cipher. He proposed to extract the temporal sequence, which is the sequence generated by the middle bit of an N bit CA with rule 30 for each cell [26, 27]. However, Meier and Staffelbach attacked this stream cipher and showed that such a construction does not provide the required security [20]. The weaknesses arise due to the linearity with the left-neighbor in the temporal sequences of such a non-linear primitive.

In this paper, we present a new scalable stream cipher which is based on Cellular Automata. Although the idea of using CA in stream ciphers is not completely new, the proposed work provides a solution which can be easily used with Keys and IVs of different sizes. This extensibility does not change the design and the structure of the cipher and this can be identified as the most original feature of the work. Further the cipher, mixing two types of CAs (linear and non-linear), and running them for a number of cycles, appears to reduce the linearity with the adjacent sequence (i.e. the sequences produced by the neighboring bits) at the moment of production of the keystream. Indeed, this was an important weakness of previous CA-based stream ciphers. Running a number of cycles in non-linear CA also acts as an "automated boolean function generator" of desired cryptographic properties. This stream cipher uses rule 30 as a primitive for non-linearity but with a structure of a modern stream cipher. In this cipher, the whole block of key stream is produced at each round that not only compensates the speed loss for running more number of cycles but also receives more throughput than Grain [16] and Trivium [5]. The proposed stream cipher, henceforth called *CAR30*, takes 128 bits Key and 120 bits of IV as input and produces 128 bits block of key stream in each iteration. The initialization process requires 160 cycles which is faster than Grain and Trivium that take 256 cycles (for 128 bits version of Grain) and 1,152 cycles, respectively. The hardware requirement of *CAR30* is more than Grain and Trivium, due to the local transformation at each bit. But the hardware cost is getting reduced day-by-day and a little extra hardware should not increase the cost of the cipher much. However, due to high throughput, the throughput to area ratio of *CAR30* is more than both Grain and Trivium. Recently, in CHES 2010, Badel et al. have introduced the Figure of Merit (FOM) [1] to give a good estimation of the area/throughput tradeoff for hardware implementations. This $FOM = throughput/area^2$ for *CAR30* is better than Grain and Trivium. Finally, due to block oriented transition functions, it provides an easy and efficient implementation in software which is difficult in existing hardware oriented stream ciphers. In fact, the software implementation shows that the speed of *CAR30* is better than software oriented eSTREAM [25] finalist Rabbit [4]. In addition, *CAR30* provides configurable security and extensibility to any Key/IV length without changing the design and analysis making it future-proof.

This paper is organized as follows. First, we provide an introduction to Cellular Automata. Section 2 describes the cipher along with the initialization process and the variability of the cipher. The design rationale for the choice of various elements of

the cipher is described in Section 3. The security properties of the cipher is analyzed in Section 4. Section 5 provides the implementation aspects of the cipher for both hardware and software. Finally, Section 6 performs comparison of *CAR30* with Grain and Trivium.

1.1 Introduction to cellular automata (CA)

The CA structure can be viewed as a lattice of cells where every cell can take values either 0 or 1. Each cell evolves in each time step depending on some combinational logic on itself and its neighbors [22], which is called *three-neighborhood CA*. Throughout the paper, let $\leftarrow$, $\oplus$, $|$ and $\cdot$ denote assignment, logical XOR, logical OR and multiplication over $GF(2)$, respectively. The next state function for a three-neighborhood CA cell can be expressed as follows:

$$q_i(t+1) = f(q_i(t), q_{i+1}(t), q_{i-1}(t)),$$

where, $q_i(t)$ denotes the output state of the i-th cell at the t-th time step and f denotes the local transition function realized with a combination logic and is known as a rule of the CA.

If the rule of a CA involves XOR logic only it is called a linear rule and the corresponding CA is called a *linear CA* or an *additive CA*. Rules 90 and 150 are examples of such a CA.

$$rule90 : q_i(t+1) = q_{i+1}(t) \oplus q_{i-1}(t)$$

$$rule150 : q_i(t+1) = q_i(t) \oplus q_{i+1}(t) \oplus q_{i-1}(t)$$

The rules with AND-OR combination logic are called *non-additive CA*. Non-additive Cellular Automata are non-linear in nature. Rule 30 is one such non-linear non-additive CA, which is defined as:

$$rule30 : q_i(t+1) = q_{i-1}(t) \oplus (q_i(t)|q_{i+1}(t))$$

The characteristic matrix of a linear CA operating over GF(2) is a matrix that describes the behavior of the CA [22]. We can calculate the next state of the CA by multiplying the characteristic matrix by the present state of the CA. A characteristic matrix is constructed as:

$$T[i, j] = 1, \text{ if the next state of the } i\text{th cell depends on } j\text{th cell}$$
$$= 0, \text{ otherwise}$$

In the next section, the proposed stream cipher *CAR30* is described.

2 Description of the cipher

The construction of the stream cipher is shown in Fig. 1. It contains a non-linear block and a linear block of size 128 bits each with a total state size of 256 bits. The non-linear block consists of 128 cells (bits) CA where each cell is updated using rule 30 in every cycle. The linear block consist of 128-bit maximum length CA using rules 90/150 with rule 150 at bit positions 1 and 29 and rule 90 for the rest of the bits.

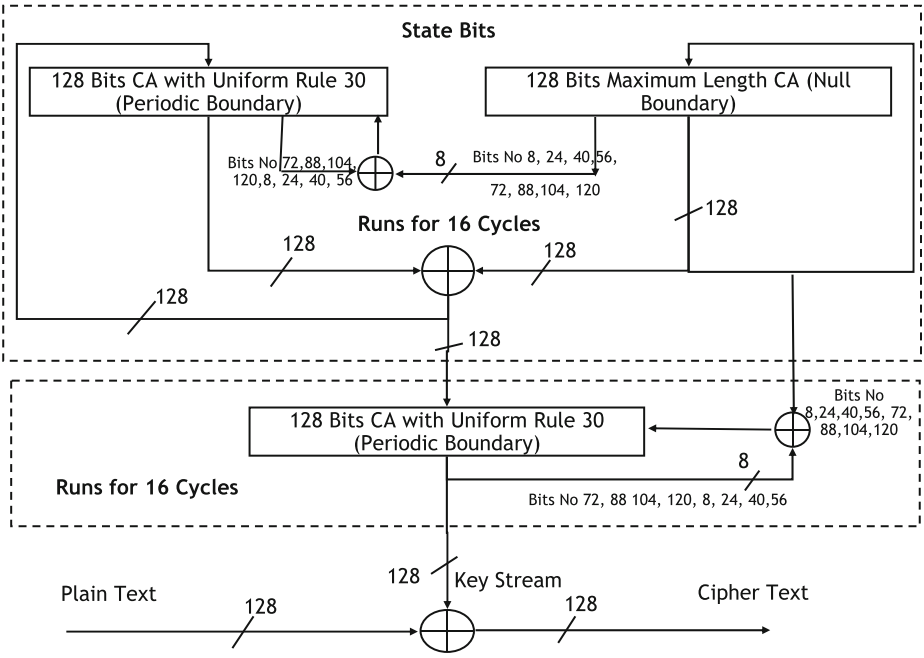

Fig. 1 The cipher description

During the first 16 cycles, the outputs of the non-linear and linear block are XORed and fed to the non-linear block. After 16 cycles, this XOR output is stored to update the non-linear block at the beginning of the next iteration (i.e. after the key stream is produced). In the next 16 cycles, the non-linear block runs without any XOR feedback. *The key-streams are produced directly from the second non-linear block*. The key-stream is finally XORed with the plain-text to produce the cipher-text.

The linear blocks have a null boundary condition, that is the left neighbor of the extreme left and the right neighbor of the extreme right bits are connected to logic zero. The non-linear block has a periodic boundary condition which means the right side of the extreme right cell is connected to the extreme left cell of the non-linear CA (NLCA) and vice versa. In addition, before the non-linear operation is performed, 8th, 24th, 40th, 56th, 72th, 88th, 104th and 120th bits of the linear CA (LCA) are XORed with the 72th, 88th, 104th, 120th, 8th, 24th, 40th and 56th bits of NLCA, respectively, to update the corresponding non-linear CA bits.

Let $\mathbf{a} = a_0, \cdots a_{127}$ denote the non-linear state bits and $\mathbf{b} = b_0, \cdots b_{127}$, denote the linear state bits at the beginning of any round and **ks** denotes 128-bit key-stream. **c** denotes an intermediate state. The key-stream generation algorithm can be described alternatively as follows:

1: **for** i=1 to 16 **do**
2: $\quad \mathbf{b} \leftarrow T \cdot \mathbf{b}$
3: $\quad \mathbf{a} \leftarrow \{a_{127}, a_0, \cdots, a_6 \oplus b_{71}, \cdots, a_{22} \oplus b_{87}, \cdots, a_{38} \oplus b_{103}, \cdots, a_{54} \oplus b_{119}, \cdots, a_{70} \oplus b_7, \cdots a_{86} \oplus b_{23}, \cdots, a_{102} \oplus b_{39}, \cdots, a_{118} \oplus b_7 \cdots a_{126}\} \oplus (\{a_0, \cdots, a_{127}\} \mid \{a_1, \cdots, a_{127}, a_0\})$

4: $\mathbf{a} \leftarrow \mathbf{a} \oplus \mathbf{b}$
5: **end for**
6: $\mathbf{c} \leftarrow \mathbf{a}$
7: **for** i=1 to 16 **do**
8: $\mathbf{b} \leftarrow T \cdot \mathbf{b}$
9: $\mathbf{a} \leftarrow \{a_{127}, a_0, \cdots, a_6 \oplus b_{71}, \cdots, a_{22} \oplus b_{87}, \cdots, a_{38} \oplus b_{103}, \cdots, a_{54} \oplus b_{119}, \cdots, a_{70} \oplus b_7, \cdots a_{86} \oplus b_{23}, \cdots, a_{102} \oplus b_{39}, \cdots, a_{118} \oplus b_7 \cdots a_{126}\} \oplus (\{a_0, \cdots, a_{127}\} \mid \{a_1, \cdots, a_{127}, a_0\})$
10: **end for**
11: **ks** $\leftarrow$ **a**
12: **a** $\leftarrow$ **c**

Initialization The initialization algorithm is shown in Fig. 2. It is similar to the key stream generation algorithm. The difference is that the key-streams are fed to the linear block. The internal structure remains the same with both the non-linear CA and the linear CA run for 16 cycles at each stage. At the beginning the 128-bit Key is uploaded to the non-linear block. The 120-bit IV is extended to 128 bits as follows. A logic bit 1 is added after every fifteenth position of the 120-bit IV. That is bit numbers $16 * i$, $(i = 1 \ldots 8)$ are made to logic one in the 128-bit block. Since each block of the cipher runs for sixteen cycles, the above arrangement will ensure that each linear state bit is affected by these logic bits 1. The extended IV is uploaded in the linear block. The initialization algorithm is iterated for five times before producing any key stream.

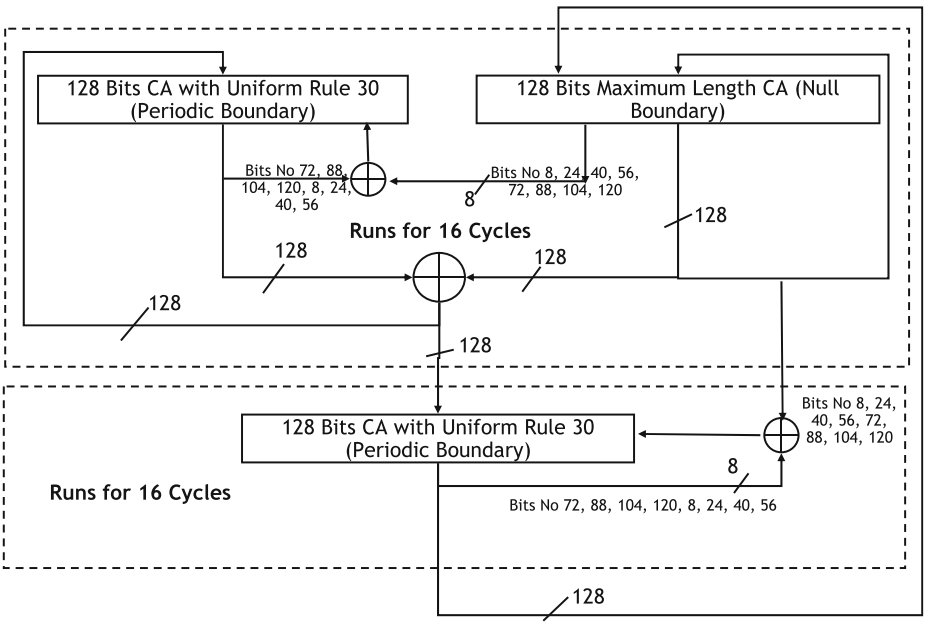

Fig. 2 The cipher initialization

The initialization process is mathematically shown as follows ($\mathbf{IV} = \{IV_0, \cdots, IV_{120}\}$, denotes 120 bits initial value):

1: $\mathbf{a} \leftarrow \mathbf{KEY}$
2: $\mathbf{b} \leftarrow \{IV_0, IV_1, \cdots, IV_{14}, 1, IV_{15}, \cdots, IV_{120}, 1\}$
3: **for** j=1 to 5 **do**
4: **for** i=1 to 16 **do**
5: $\mathbf{b} \leftarrow T \cdot \mathbf{b}$
6: $\mathbf{a} \leftarrow \{a_{127}, a_0, \cdots, a_6 \oplus b_{71}, \cdots, a_{22} \oplus b_{87}, \cdots, a_{38} \oplus b_{103}, \cdots, a_{54} \oplus b_{119}, \cdots, a_{70} \oplus b_7, \cdots a_{86} \oplus b_{23}, \cdots, a_{102} \oplus b_{39}, \cdots, a_{118} \oplus b_7 \cdots a_{126}\} \oplus (\{a_0, \cdots, a_{127}\} \mid \{a_1, \cdots, a_{127}, a_0\})$
7: $\mathbf{a} \leftarrow \mathbf{a} \oplus \mathbf{b}$
8: **end for**
9: $\mathbf{c} \leftarrow \mathbf{a}$
10: **for** i=1 to 16 **do**
11: $\mathbf{b} \leftarrow T \cdot \mathbf{b}$
12: $\mathbf{a} \leftarrow \{a_{127}, a_0, \cdots, a_6 \oplus b_{71}, \cdots, a_{22} \oplus b_{87}, \cdots, a_{38} \oplus b_{103}, \cdots, a_{54} \oplus b_{119}, \cdots, a_{70} \oplus b_7, \cdots a_{86} \oplus b_{23}, \cdots, a_{102} \oplus b_{39}, \cdots, a_{118} \oplus b_7 \cdots a_{126}\} \oplus (\{a_0, \cdots, a_{127}\} \mid \{a_1, \cdots, a_{127}, a_0\})$
13: **end for**
14: $\mathbf{b} \leftarrow \mathbf{a}$
15: $\mathbf{a} \leftarrow \mathbf{c}$
16: **end for**

Variability The cipher is variable with the number of cycles that are run at each stage of the cipher. This provides the configurable security and the flexibility to tradeoff between speed and the security without changing the hardware or the software implementation code. We have shown 16 cycles as that is required for 128-bit security with sufficient security margin.

3 Design rationale

In this section, we describe the design rationale of the proposed stream cipher. We used CA for both non-linear blocks and the linear blocks because CA provide very good statistically random sequences and are easy to implement in hardware. Note that the ease of hardware implementation is an important criterion for stream ciphers. As stated by Meier and Staffelbach [20], an important difference between CA and feedback shift registers is that even if CA are invertible, it is not possible to find the predecessor of a state by simply reversing the rule to find out the next state [9]. The problem of deducing the initial configuration from partial output is NP-Complete [26]. However, we show later that these statements are incorrect. The non-linear CA employing Rule-30 are subjected to inversion attack [13]. Hence special considerations are taken in this cipher by injecting the linear bits to non-linear CA to make this attack much more complex than brute force. The stream cipher uses two stages of non-linear transformations. The first stage is required for the update of the non-linear state bits with a linear masking so that a better period is guaranteed. The second stage, on the other hand, acts as a filter to the state bits before producing the key-streams. The injection of the linear bits in the non-linear

block guarantees a better period, a better randomness to the sequences produced and prevents correlation attacks by balancing the key streams. This also helps to prevent an attacker to walk backwards from the key-streams to the state bits by increasing the number of unknown bits. The non-linear CA is made periodic boundary so that there is no weakness at the end bits. The bit numbers 8, 24, 40, 56, 72, 88, 104 and 120 from the linear CA are sent to the non-linear block so that maximum number of linear bits affect the non-linear CA. These linear bits help to prevent the inversion attack on the cipher.

The choice of number of cycles Cellular Automata perform parallel transformations in every cycle unlike LFSRs. Hence, all the state bits are transformed in every cycle. So CA provide a large throughput with less number of cycles. On the other hand, each CA transformation involves only three bits which will not provide sufficient security for the CA based primitive. Hence, to get enough security, it is important to run the CA for a few cycles before the output is produced. So, the number of cycles is the trade-off parameter between the speed and the security. In case of *CAR30*, in each stage of the cipher sixteen cycles are used because each output bit in each stage will depend on 33 bits from its input. Out of these 33 bits, it will depend non-linearly on 32 bits for the non-linear CA block. We state some Theorems below for some properties of CA with straightforward proofs which will show the reason for choosing a total of 32 cycles between successive rounds .

Theorem 1 *For non-linear n-cell periodic-boundary CA transformation with rule 30 running for p cycles ($p < n/2$), each state bit depends on $2p + 1$ neighboring bits.*

Theorem 2 *For non-linear n-cell periodic-boundary CA transformation with rule 30 running for p cycles ($p < n/2$), each non-linear state bit depends on $2p$ bits non-linearly.*

Now let us prove Theorems 1 and 2, by induction.

Proof Let us assume the following at the pth ($p < n/2$) cycle on a periodic boundary CA with uniform Rule 30:

1. $q_{i,t+p}$ depends on $2p + 1$ bits which are all bits from $q_{i-p,t}$ to $q_{i+p,t}$. This is Theorem 1.

Due to symmetric nature, this assumption implies:

2. $q_{i-1,t+p}$ depends on $2p + 1$ bits which are all bits from $q_{i-1-p,t}$ to $q_{i-1+p,t}$.
3. $q_{i+1,t+p}$ depends on 2p+1 bits which are all bits from $q_{i+1-p,t}$ to $q_{i+1+p,t}$.

Let us assume further that:

4. $q_{i,t+p}$ depends non-linearly on $2p$ bits which are all bits from $q_{i-p+1,t}$ to $q_{i+p,t}$. This is Theorem 2.

Due to symmetric nature, this assumption implies:

5. $q_{i-1,t+p}$ depends non-linearly on $2p$ bits which are all bits from $q_{i-1-p+1,t}$ to $q_{i-1+p,t}$.

6. $q_{i+1,t+p}$ depends non-linearly on $2p$ bits which are all bits from $q_{i+1-p+1,t}$ to $q_{i+1+p,t}$.

Let us make one more further assumption that:

7. In the expression of $q_{i,t+p}$, the bit $q_{i+p,t}$ has a non-linear association with all the bits from $q_{i-(p-1),t}$ to $q_{i+p-1,t}$.

Again, due to symmetric nature, this assumption implies:

8. In the expression of $q_{i-1,t+p}$, the bit $q_{i-1+p,t}$ has a non-linear association with all the bits from $q_{i-1-(p-1),t}$ to $q_{i-1+p-1,t}$.
9. In the expression of $q_{i+1,t+p}$, the bit $q_{i+1+p,t}$ has a non-linear association with all the bits from $q_{i+1-(p-1),t}$ to $q_{i+1+p-1,t}$.

We need to prove that the assumptions hold good when $p \rightarrow p+1$. Now,

$$q_{i,t+p+1} = q_{i-1,t+p} + q_{i,t+p} + q_{i+1,t+p} + q_{i,t+p} \cdot q_{i+1,t+p} \tag{1}$$

First, $q_{i-1,t+p}$ brings an additional bit $q_{i-1-p,t}$=$q_{i-(p+1),t}$ according to assumption 2 which is not present in any other terms according to assumptions 1 and 3.

Second, the term $q_{i,t+p}$ can be expressed as: $q_{i-p,t} + \cdots$ *non-linear terms involving all the bits from* $q_{i-p+1,t}$ *to* $q_{i+p,t}$ (from assumptions 1 and 4).

The term $q_{i+1,t+p}$ can be expressed as: $q_{i+1-p,t}$ + *some non-linear terms* + *non-linear terms involving* $q_{i+1+p,t}$.

The product term $q_{i,t+p} \cdot q_{i+1,t+p}$ can be expressed as: ($q_{i-p,t} + \cdots$ *non-linear terms*) $\cdot$ ($q_{i+1-p,t}$ + *some non-linear terms* + *non-linear terms involving* $q_{i+1+p,t}$) = (($q_{i-p,t}$) $\cdot$ ($q_{i+1-p,t}$ + *some non-linear terms* + *non-linear terms involving* $q_{i+1+p,t}$)) + ((*some non-linear terms*) $\cdot$ ($q_{i+1-p,t}$ + *some non-linear terms involving bits* $q_{i-p+1,t}$ to $q_{i+p,t}$ + *non-linear terms involving* $q_{i+1+p,t}$)) = $q_{i-p,t} \cdot$ (*non-linear terms involving* $q_{i+1+p,t}$) $+ \cdots$ = $q_{i-p,t} \cdot q_{i+1+p,t} \cdot$ (*terms involving all the bits from* $q_{i+1-(p-1),t}$ *to* $q_{i+1+p-1,t}$ (from assumption 9)) $+ \cdots$.

Hence, (1) can be written as:

$$q_{i,t+p+1} = q_{i-p,t} \cdot q_{i+1+p,t} \cdot (\textit{terms with all the bits from } q_{i+1-(p-1),t} \textit{ to } q_{i+1+p-1,t}) + \cdots \tag{2}$$

None of the terms on the first part will cancel out because this is the only place so far where product $q_{i-p,t} \cdot q_{i+1+p,t}$ appears in the non-linear terms.

Now let us show the assumptions 1–9 are valid when the number of cycles is p+1.

Assumption 1: Note from the above analysis, two new bits are added in the overall expression of $q_{i,t+p+1}$ which are $q_{i-(p+1),t}$ and $q_{i+1+p,t}$. Also from the above expression the product term contain all the bits from $q_{i-p,t}$ to $q_{i+1+p,t}$. Hence, the $q_{i,t+p+1}$ depends on all the bits from $q_{i-(p+1),t}$ to $q_{i+(p+1),t}$ which is $2(p+1)+1$ number of bits.

From symmetry the assumptions 2 and 3 are also valid.

Assumption 4: From the above expression, $q_{i,t+p+1}$ depends non-linearly on bits $q_{i-p,t}$ to $q_{i+1+p,t}$ which is equivalent to bits $q_{i-(p+1)+1,t}$ to $q_{i+p+1,t}$ (total $2 \cdot (p+1)$ bits.)

From symmetry the assumptions 5 and 6 are also valid.

Assumption 7: From the above expression, $q_{i+(p+1),t}$ has a non-linear association with $q_{i-p,t}$ and all the bits from $q_{i-(p-1),t}$ to $q_{i+1+p-1,t}$ i.e. it has non-linear association with $q_{i-((p+1)-1)}$ to $q_{i+(p+1)-1,t}$.

Now, it can be easily seen in Example 1 below, that all the assumptions are valid for $p = 1$ and $p = 2$, hence the assumptions are valid for all $p < n/2$. □

The following example illustrates the above Theorems.

Example 1 The expression for the next state bit of Rule 30 can equivalently be written as,

$$q_{i,t+1} = q_{i-1,t} + q_{i,t} + q_{i+1,t} + q_{i,t}q_{i+1,t}$$

where all the operations are in $GF(2)$ and $q_{i,t}$ denotes the i-th state bit at t-th time instant. Similarly,

$$q_{i,t+2} = q_{i-1,t+1} + q_{i,t+1} + q_{i+1,t+1} + q_{i,t+1}q_{i+1,t+1}.$$

$$q_{i-1,t+1} = q_{i-2,t} + q_{i-1,t} + q_{i,t} + q_{i-1,t}q_{i,t}$$

$$q_{i+1,t+1} = q_{i,t} + q_{i+1,t} + q_{i+2,t} + q_{i+1,t}q_{i+2,t}$$

Combining the above three expressions, we get:

$$q_{i,t+2} = q_{i-2,t} + q_{i+2,t} + q_{i+1,t} + q_{i+1,t}.q_{i+2,t} + q_{i-1,t}.q_{i+1,t} + q_{i-1,t}.q_{i+2,t}$$
$$+ q_{i,t}.q_{i+2,t} + q_{i+1,t}.q_{i,t} + q_{i-1,t}.q_{i+1,t}.q_{i+2,t} + q_{i,t}.q_{i+1,t}.q_{i+2,t}$$

The following facts should be observed from the above equation:

1. The non-linear state bit depends on five bits which conforms with Theorem 1.
2. The non-linear state bit depends non-linearly on four bits conforming Theorem 2.
3. The algebraic degree is three.
4. With each additional cycle run with Rule 30, the output expression becomes much more complex.

Lemma 1 *The algebraic degree of the state bits with respect to the previous state bits, when the n-cell periodic-boundary CA with Rule 30 runs for p cycles ($p < n/2$), is at least $p + (p/2)$.*

Proof On pth cycle, the $q_{i,t+p}$ gets multiplied by a new bit from right side, which is $q_{i+p,t}$. This new bit increases the algebraic degree by one. This implies that the term with highest algebraic degree will have the bit $q_{i+p,t}$. This in turn implies that all the bits from $q_{i,t}$ to $q_{i+p,t}$ will be contained in the highest algebraic degree term (as we move from the first cycle to pth cycle). Similarly, the bits $q_{i-1,t}$ to $q_{i-1+p-1,t}$ will be contained in the highest algebraic degree term of $q_{i-1,t+p-1}$. However, this term will be linearly added to the expression of $q_{i,t}$. Finally, the bits $q_{i+1,t}$ to $q_{i+1+p,t}$ will be contained in the highest algebraic degree term of $q_{i+1,t+p}$

On the next cycle, the expression $q_{i,t+p+1}$ will have a multiplication between the highest algebraic degree terms of $q_{i-1,t+p-1}$ (which was linearly added in the previous cycle in $q_{i,t+p}$)) and $q_{i+1,t+p}$. This multiplication will add one more bit from the left

hand side to the expression of the term with highest algebraic degree. Hence, at every two cycles one bit from the left hand side is added to the term with highest algebraic degree of $q_{i,t+p}$. Hence, after p cycles, the algebraic degree is at least $p + (p/2)$ □

Lemma 2 *For a linear CA employing rules* 90/150*, the boolean expression of the ith output bit after pth ($p < n/2$) cycle contains $(i + p)$th and $(i - p)$th bits, provided the bits exist.*

Proof Since both Rule 90 and 150 performs XOR with both left and right neighbors, every bit moves one bit to right and left via XOR at every cycle. After p cycles, the bit $(i + p)$th will move p bits to left and the bit $(i - p)$th will move p bits to right where the XOR will be performed only once. Hence these two bits will not cancel out at pth cycle in the boolean expression. □

Property 1 When the non-linear CA transformation, linear CA transformation and XOR between the two are performed and fed back to non-linear state (running for 16 cycles), in an instance of *CAR30*, each non-linear state bit depends on at least 33 non-linear previous state bits and 33 linear state bits.

Property 2 When each CA runs for 16 cycles, the algebraic degree of the key stream bits with respect to the state bits in *CAR30* is at least 24.

Property 3 When each CA runs for 16 cycles, the algebraic degree of the non-linear state bits with respect to the previous state bits in *CAR30* is at least 24.

Finally, let us see the choice of five rounds of initialization.

Theorem 3 *At the time of initialization in* CAR30*, each state bit depends non-linearly on all the Key and the IV bits within* 64 *cycles, i.e two rounds.*

Proof After 32 cycles, each non-linear CA bit depends on 65 neighboring bits. Since the non-linear CA is having periodic boundary condition, every bit in non-linear CA depends on all other bits after two rounds. Again, the non-linear CA are initialized with Key bits. This proves that the complete mixing among all the Key bits in non-linear CA after two rounds, i.e. after 64 cycles.

During initialization, the key-stream bits (which are coming from non-linear transformation) are fed back to linear CA. Also, after two rounds, every non-linear state bit depends on all the Key bits. Hence, the linear CA bits will also depend on all Key bits after the XOR feedback to linear CA at the end of second round. This process takes 64 cycles.

The linear CA bits which are initialized with IV bits are XORed with non-linear CA bits and the XOR output is fed back to non-linear CA bits at the first cycle itself. Hence, all the non-linear CA bits are uploaded with IV bits at the first cycle itself. Going by the logic in the previous two paragraphs, at the end of two rounds (i.e. another 63 cycles after the first cycle) each non-linear state bit will depend on 127 IV bits. The only exclusion is the IV bit at a distance of 64 from particular state bit. However, in the first cycle, the bit at a distance 63 from a particular bit, already has the neighboring bits (through linear CA transformation) which include the bit at a

distance of 64 from that particular bit. Hence, each non-linear state bit will depend on all the IV bits after two rounds. After the XOR feedback the linear CA bits also will depend on all the IV bits.

For simplicity, we have not considered the eight individual linear state bits that are XORed with non-linear state bits. It can be shown that if we consider those bits, this state of dependency will be achieved even earlier. □

So, after two rounds (64 cycles) all the state bits depend on all the Key and the IV bits. Another three rounds (96 cycles) are kept for security margin and to have more non-linear effect on the bits. Hence, 160 cycles are kept for initialization which translates into five rounds.

In the next section, security analysis of *CAR30* is performed.

4 Security analysis

In this section, we analyze security of the cipher and show how it can resist a few of the known attacks. To check the statistical randomness of the proposed stream cipher NIST [21] statistical test suit was run on the key stream bits. First we show that *CAR30* can be reduced to the security model of Grain family of stream ciphers.

Proposition 1 CAR30 *can be modeled equivalently with the stream cipher Grain with respect to Algebraic and Correlation attacks.*

Proof To prove this proposition, we show first how *CAR30* can be reduced to a Grain like structure and then show that the component functions are cryptographically comparable with those of Grain.

As shown in the design rationale section, each cycle in the non-linear CA generates a boolean equation. These boolean equations can be mapped to the NFSR feedback function $g(x)$ and the filter function $h(x)$ of Grain. Similarly, the linear CA transformation function can be mapped to the LFSR feedback polynomial $f(x)$ of Grain. Now, if we observe Fig. 1 carefully, every key-stream bit is generated after a non-linear transformation of a few state bits, a linear transformation of a few more bits and finally filtered by a non-linear transformation of both linearly and non-linearly transformed internal state bits. By the fact that CA is a special type of LFSR, the Linear CA (LCA) can be treated as LFSR and the non-linear CA can be treated as NFSR. The filtering by the non-linear CA (NLCA) is achieved by the automated boolean function generator provided by the non-linear CA. Hence, the whole cipher can be reduced to Grain family of stream cipher with parallel transformation of each state bit.

To show the second part, the function $f(x)$ of Grain is equivalent to linear CA running for 16 cycles. For each individual bits, it can be argued that there is no significant difference between LFSR and CA in terms of cryptographic properties. The difference will be in the number of taps. The output equation of the LFSR chosen in Grain-128 takes 6 LFSR bits as input. The output equation of a particular bit for the maximum length CA running for 16 cycles will depend on 33 bits, but will contain approximately 17 CA bits in the boolean equation as some of the bits will be XORed even number of times. Some more variation can be expected depending on

the rule vector chosen for the maximum length CA and position of the bit (leftmost or rightmost), but in all cases, the number of input state bits in the output equation will be more than 6.

The below are some observations on the equivalence of $g(x)$ and $h(x)$ of Grain which are the basis of cryptographic strengths of Grain:

- *Balancedness*: Due to the linear relation in Rule 30, after sixteen cycles there exists one state bit which propagates linearly in the output expression of each bit. But, the initial state of the NLCA is the XOR of LCA and NLCA state bits. The LCA bit depends on 33 LCA state bits. Hence, the number of state bits linearly affecting output expression is much more than Grain which translates into more resiliency degree and balancedness compared to Grain functions.
- *Non-linearity and Bias*: Since the output expression of rule 30 after 16 cycles depends on 33 input bits, it is difficult to evaluate the non-linearity and bias for such an expression. It is also difficult to predict the bias theoretically. This is because at every cycle after the first cycle, the involved bits in execution of *CAR30* are not independently updated, hence the piling up lemma is not quite applicable here. So, we evaluated them for rule 30 running for 16 cycles experimentally. Again, due to the exponential nature of the algorithms to find the bias and the non-linearity, it is not possible to evaluate the bias and the non-linearity in a reasonable amount of time when the number of input bits are more than 17. Hence, the non-linearity was evaluated only with the self and the right side 16 bits after running 16 cycles. The non-linearity was 63,216 and the bias was approximately 2^{-6}. The additional dependencies with the left hand side bits will only add to the security of the resulting functions after running rule 30 for 16 cycles. Several experiments were performed with varying the number of cycles and number of bits involved in the calculation of non-linearity. The conclusion was that the non-linearity would be in the order of 2^{30} after running 16 cycles based on extrapolation. This is much more than the non-linearity of the Grain constituent functions. The bias is also comparable (for Grain, the bias of $g(x)$ is 2^{-8} and the bias of $h(x)$ is 2^{-5}). In *CAR30*, there are 2^{25} linear approximation functions with such a bias (for Grain there are 2^{14} such linear functions for $g(x)$ and 2^8 such linear functions for $h(x)$). The number of equations and bias together will make security against linear approximations much better than Grain.
- *Security Enhancement CAR30*'s generic design helps to enhance the security. In the present description of *CAR30*, sixteen cycles are used to generate the functions $f(x)$, $g(x)$ and $h(x)$. This will provide good enough security with sufficient margins. But, the security can be increased to much higher if thirty-two cycles are run to generate the above functions. This would increase the hardware marginally as the only addition in hardware implementation is that the counter bits are increased by one. The hardware and software speed would be about half of what is reported in this paper. But still the hardware throughput remains competitive with Grain and Trivium and software speed remains competitive with Rabbit. The main advantage in that case is, it can be shown easily that if the non-linear CA is run for thirty-two cycles then even for finding out a linear expression and calculating the bias is more complex than brute force. This provides a new dimension to the security of *CAR30* against correlation where finding out the correlation coefficient itself is NP-Complete. □

4.1 Statistical tests

For any new cipher it is necessary to perform statistical tests. NIST [21] statistical test suit was used for this purpose with 12.8 million key stream bits. NIST tool takes the test parameters as input. After choosing appropriate parameters the statistical tests were run. The tests run were Frequency, Runs, Serial, Rank, FFT, Cumulative Sums, Block Frequency, Universal, Longest Run of Ones, Random Excursion, Overlapping templates, Linear Complexity, Approximate Entropy, Non-Overlapping Templates and Random Excursion Variant. All the tests passed with appropriate parameters.

After evaluating the statistical tests, we analyze how *CAR30* can resist a few of the classical attacks on stream ciphers. We start with an estimation of the period of the cipher.

4.2 Period

In this cipher, the linear layer which is a maximum length hybrid CA guarantees large period. The linear CA are run for thirty-two cycles. Hence the period is at least $(2^{128} - 1)/32$. The non-linear CA do not have maximum length period. Since, the linear CA and the non-linear CAs are XORed to feed the non-linear block before producing the key stream, the actual period will be more and depends on the Key-IV pair used. The Key and the IV will determine the initial state of the non-linear blocks which in turn will determine the actual period. To check this we evaluated the cipher with a smaller version of 12-bits, no period was seen in the first 2^{12} blocks of key streams. Since, both the 12-bit version and the 128-bit version are identically updated, the observed period will extend to 128-bit version of the cipher.

It can be argued that in a filter or a combiner model, the linear block and the non-linear block are built on so different principles that there is no real risk that the nonlinear block can reduce the period achieved by the linear block; but here the two blocks are both built on CA. However, even if both the linear blocks and non-linear blocks are CA, after running a few cycles the boolean equation for each bit of the non-linear block will be far more complex than the linear part as shown in Example 1. This argument is also supported by the number of terms found in the ANF of the simplified cipher, which is mostly contributed by the non-linear block. Hence, we state that the period is always at least $(2^{128} - 1)/32$ independently of the nonlinear layer.

4.3 Correlations

Since the cipher can be modeled to a Grain like structure most of the arguments for prevention of correlation attack against Grain are applicable here [16]. The strength of this model comes from general decoding problem [19]. Let, $A_g(x)$ and $A_h(x)$ be two linear approximation functions for $g(x)$ and $h(x)$, with biases ϵ_g and ϵ_h respectively. It was shown in [19] that there exists a time invariant linear equation with key-stream bits and the LFSR bits in a Grain like structure (for *CAR30*, we should consider the linear CA bits) with a bias:

$$\epsilon = 2^{\eta(A_g)+\eta(A_h)-1} \cdot \epsilon_g^{\eta(A_g)} \cdot \epsilon_h^{\eta(A_h)} \tag{3}$$

where $\eta(A_g)$ and $\eta(A_h)$ denote the number of NFSR (non-linear CA for *CAR30*) bits involved.

As shown in Proposition 1 *CAR30* can be modeled as a Grain like structure, hence *CAR30* will also have the existence of such an equation concerning the key bits and the linear CA bits with such a bias. So like Grain, it is important to design the functions $g(x)$ and $h(x)$ with proper security properties, so that the bias of such an equation is extremely small. It was also shown in Proposition 1 that for *CAR30*, these functions possess such security properties that correlation attacks are not applicable for *CAR30*. Also, it was shown theoretically in [24], CA based mappings can provide resistance to classical Inversion Attacks and Anderson Leakage, which aid to the above arguments.

4.4 Algebraic attack

In *CAR30* a non-linear state update function and a non-linear filter is used to produce the key-streams. The algebraic degree of each of these non-linear functions is $16 + 16/2 = 24$ from Lemma 1. As in Grain, the algebraic degree of the output expressions expressed in terms of initial state bits of *CAR30* is time varying as well. Even though the nonlinearity being based on the iteration of the same nonlinear operation, additional bits are involved with every operation which themselves have undergone non-linear operations making the algebraic relationship complex.

Another way of looking at the immunity against algebraic attack is computing the number of equations required to approximate the output expression in linear equations, i.e. over-defined systems of equations. Although, this is not strictly applicable for a Grain like structure since the algebraic degree is time varying, the attacker can make an attempt by having a large set of keys and observing the first key-stream bits. However, in *CAR30* this approach will also not be possible. The number of linear equations of 256 state bit variables with degree 24 is, $\sum_{i=0}^{24} \binom{256}{i} \simeq 2^{110}$. These equations cannot be solved with complexity less that 2^{128}. Sometimes, it is possible to find a low degree equivalent of a boolean function which is characterized by algebraic immunity. However, if the number of input bits that constitute the boolean function is high the algebraic immunity also becomes high. In this case, each output bit depends on 33 input bits which will prevent the existence of such low degree equivalent of the output equations. Given the above facts, the algebraic attack [8] on this stream cipher is going to be difficult.

4.5 Cube attack

In this type of attack [10], the attacker uses the stream cipher as tweakable polynomials by choosing the public variables (IV and key stream) and solve the resultant system of polynomial equations in terms of the Key bits. The attacker requires only black box access of the function and needs negligible memory. This attack has a preprocessing phase with chosen Keys and chosen IVs and an online phase with unknown Keys and chosen IVs. The factor polynomials are evaluated where the linear terms are called superpoly and the factors are called cube.

It is known that if the cipher has a very high algebraic degree then the cube attacks are difficult. In CAR30, the algebraic degree is at least 24. With such a high algebraic

degree it is difficult to perform cube attacks. To verify our assertion, we have performed cube attack (unsuccessfully) in CAR30. We have used cube dimension of 4, 6 and 8, with equivalent number of computations, with the first key stream bit. Finally, we attempted with a cube size 12 and tried upto 500,000 computations which took three days. However, we did not find any linear superpoly with the experiments that we have performed.

4.6 Inversion attack

The inversion attack works on a non-linear filter generator and attempts to find out the input sequence from the knowledge of an output sequence. Golic first proposed this attack [13] and shown that such an attack is effective when the first or the last bit of the filter input is linear. Later, Golic et al. generalized this attack [14] without such an assumption. They showed that if the length of the LFSR is N and M bits are used as an input to the filter function, then the inversion attack has a complexity of 2^M. As a countermeasure of this attack, the input memory M of the filter function should be close to the number of LFSR bits N.

For *CAR30*, the first type of attack [13] is applicable as the bits coming from left side of Rule 30 has a linear relationship. However, we show that such an attack has a complexity of more than brute-force. Let us consider a typical round and see how state bits are filtered to produce the key stream bits.

As shown in Theorem 1, after 32 cycles, the non-linear CA will filter 65 non-linear state bits into a single key stream bit. Also, due to the XOR between linear state and the non-linear state, each non-linear state is affected by 33 linear state bits after 16 cycles. Even if the linear CA bits are null-boundary, the periodic boundary condition of non-linear CA will ensure that 33 linear state bits affect each non-linear state bit for all bit positions. Now, let us see the effect of the bits numbers 8, 24, 40, 56, 72, 88, 104 and 120 of linear state affecting bit numbers 72, 88, 104, 120, 8, 24, 40 and 56 of non-linear state bits, respectively. The selection of these bit positions will ensure that at every cycle there are two new linear bits added to the corresponding non-linear state bits. Now, for these particular bit numbers of non-linear CA, these bits will bring dependency with 65 additional linear bits in 32 cycles. Hence, for these key-stream bits the non-linear filter will have $65 + 33 + 65 = 163$ internal state bits as input. Clearly, the inversion attack is infeasible for these bits in less than brute force. Now let us see the effect of the linear state bits on other bits. After p cycles, each non-linear bit will affect p bits on the left and on the right side. Again at each cycle, the non-linear state bits at the positions of XOR are affected by two additional linear bits. Now, for any bit at a distance $i < p$ from the above bits, it can be shown that it depends on $2 \cdot (p - i)$ additional linear state bits. If $p > 8$, due to the positions of the above bits, the same state bit depends on $2 \cdot (p - (16 - i))$ linear state bits from the other side. So, in total the non-linear state bit will depend on $2 \cdot (p - i) + 2 \cdot (p - (16 - i)) = 2 \cdot (2 \cdot p - 16)$. This is irrespective of the position of the non-linear state bit. Since, $p = 32$, every non-linear state bit depends on at least 96 additional state bits. Hence any key-stream bit is filtered by $65 + 33 + 96 = 194$ state bits. There could be some fine tuning on the above analysis for some particular bit positions, but, nevertheless we can conclude safely that each key-stream bit will have dependency on at least 190 bits. This makes inversion attack (with complexity 2^{190}) infeasible for any bit position.

The above analysis was performed for each individual bits on overall cipher. However, an attacker can go into details of each stage and work on the entire state and entire 128-bit key-stream to perform the attack. In this attack, it is assumed that the state of the nonlinear CA is known at a given time and, then, the objective is to find its state at a time in the past. To this end, the unknown refreshing bits from the linear CA need to be guessed. In particular, if we also guess the state of the linear CA, then the objective would be to invert the next-state function. Can the next-state function be effectively inverted by the inversion attack? If yes, then the effect of the first 5 discarded cycles for the key-stream generation is negligible. Also, by guessing linear CA bits, a given 128-bit keystream segment (a) can be inverted into the internal state (c) of the nonlinear CA used for generating the next keystream segment. If this state is known, then the generator becomes linear and so on.

We show now that an appropriate choice of the positions of the eight refreshing bits from linear CA prevent this attack in $CAR30$. Let us exactly discuss how the inversion attack applies to particular stages of the key-stream generator and the initialization scheme. Since both the state update and filtering functions are similar, we check how the non-linear filter function is affected by previous state bits. After 16 cycles, each non-linear state bit depends on 33 previous non-linear state bits. Hence, the inversion attack has a complexity of 2^{33} when inverting from key-stream bits to the state bits. We need to guess the linear bits as well to perform the attack. How many linear bits affect the non-linear state bits in 16 cycles? Since the linear CA transformation is performed just before the non-linear transformation, each linear CA bit that refreshes the non-linear state brings forth at least two linear state bits (due to the fact that both Rule 90 and Rule 150 that the linear CA are comprised of perform XOR with neighboring bits—see Lemma 2). So after 16 cycles, each linear CA bit will refresh the non-linear state with 16 bits from left and 16 bits from right of the linear CA state bits provided the bits exist. Now consider the bit positions from linear CA that are sent to non-linear CA. They are $(16 \cdot i + 8)(i = 0 \cdots 7)$. So it will put all 128 linear state bits in the non-linear CA. So the attacker needs to guess all the 128 linear state bits in order to get to the state bits from key-stream bits. The overall complexity is $2^{33+128} = 2^{161}$ which is clearly much more than the brute force. It can be shown in a similar manner that for the next state function (i.e state update function) the complexity of inversion attack is much more than the brute force. Since for both the filter and state update functions the complexity is more than brute force, it is not possible to perform this attack during initialization.

Finally, the attacker may perform the inversion attack cycle by cycle on the non-linear CA. In this case, in addition to the inversion complexity of non-linear CA, the attacker has to guess eight refreshing linear CA bits per cycle. In each stage, there are 16 cycles of the non-linear CA. Hence the attacker has to guess 128 linear CA bits. This makes the inversion attack performed this way beyond the scope of cryptanalysis.

The initialization follows the same method as key-stream generation. Note that the initialization scheme should satisfy the criterion that it should be infeasible to recover the key from the already recovered initial state of the key-stream generator. But as shown in different cases of inversion attack above, the inversion attack in $CAR30$ is always more complex than brute-force. Moreover, the initialization process runs for five rounds without producing any key stream. Hence, the inversion attack will be infeasible to find out the Key and the IV bits for $CAR30$.

4.7 Meier's attack

Wolfram designed the Rule 30 based stream cipher [26] by having a CA of length $2 \cdot N + 1$ employing Rule 30 and taking out the temporal bit (which is the $(N + 1)$th bit) sequence produced at every cycle. Meier and Staffelbach attacked this cipher in the following way. They generated a random seed (initial state) for all the bits on the right half of the CA and then calculated the right adjacent sequence which is the sequence on the right side of the temporal sequence. With the right adjacent sequence, they move backwards to find out the left half of the seed as the left side is linear with the temporal sequence. Then they load the computed seed on the CA to see if there is match with the actual key-stream. There are many seeds leading to the correct adjacent sequence which by moving backwards determine the correct seed.

Now let us see if such an attack is applicable to *CAR30*. First of all, note that the main reasons of this attack are that the uncertainty of the right adjacent sequence given the temporal sequence is very small and that the whole seed can be directly recovered from short segments of the two sequences, by moving backwards, which is possible due to the periodic boundary conditions. As a result, from a small number of (wrong) guesses about the right half of the seed, the correct segment of the right adjacent sequence can be obtained and then tested on a longer temporal sequence, by first recovering the candidate seed as a whole, by moving backwards. To prevent this attack, the crucial point is the decimation of CA output sequences by 16/32 cycles. After running the CA for 16/32 cycles, the uncertainty of the right adjacent sequence given the temporal sequence is large. Hence, this type of attack is not possible in *CAR30*.

4.8 Time/Memory/Data tradeoff attack

The complexity of time/memory/data tradeoff attacks on stream ciphers is $O(2^{n/2})$, where n is the number of inner states of the stream cipher. Since the total number of internal states are 256 bits, it is difficult to perform time/memory/data tradeoff. It is known that stream ciphers with low sampling resistance have tradeoff attacks with fewer table lookups and a wider choice of parameters [2, 3].

One such attack is proposed on Grain in [3]. We show here that such an attack is not applicable for *CAR30*. In the attack on Grain, the authors perform BSW sampling [2] on the cipher and calculate the "sampling resistance" of Grain. If the attacker can reach a particular state of the cipher from which l subsequent cipher bits are a fixed string by means of BSW sampling, then the "sampling resistance" is defined as $R = 2^{-l}$. The smaller is the value of l, the better is the cipher. The authors prove that if the attacker knows 133 particular state bits and 18 key stream bits, then another 18 internal state bits can be deduced efficiently and hence the "sampling resistance" of Grain is 2^{-18}. Finally, with the help of Biryukov-Shamir [2] tradeoff curve and the sampling resistance, they derive some possible time and memory complexity values that are better than the exhaustive search.

Prevention of Time/Memory/Data tradeoff attack The sampling resistance of Grain is low because some of the LFSR and NFSR bits of Grain do not vary in each successive step. With the use of Cellular Automata instead of NFSR/LFSR, this assumption is no longer true. Since Cellular Automata apply local transformations in

each bit in each run, it is extremely difficult to fix them in each of these steps. So the aforementioned state bits of Grain cannot be determined with reduced complexity in *CAR30*. In the description of the attack, the authors have derived the sampling resistance as 2^{-18} by finding out the fixed NFSR bits in a particular round. With the use of Cellular Automata, this process of deriving the sampling resistance will not be possible. Consequently, the sampling resistance is no longer 2^{-18} and will be much larger that will not be a matter of concern. Hence, Time/Memory/Data Tradeoff attack cannot be performed with complexity less than brute-force attack in *CAR30*.

4.9 Re-synchronization and chosen IV attacks

In this type of attack, the attacker manipulates the IV and extracts the Key bits by observing the key-stream bits [6]. A complete non-linear mixing among the Key bits and the IV bits is achieved in the initialization of the cipher. As shown in Theorem 3, all the state bits are affected by all the Key bits and all the IV bits after 64 cycles of initialization (i.e. after second round). Another 96 cycles are run for security margin during initialization. This way the IV bits also will have non-linear dependencies along with the Key bits; a fact that makes such an attack impossible.

The chosen IV attack sometimes takes place by keeping the same Key and choosing the IVs that varies only with one byte. As stated above, due to strong non-linear, statistically random dependency among all the Key and the IV bits, this type of attack is not possible. The maximum entry in the difference distribution table of the 12-bit version of the cipher reinforces this assertion.

One such attack was reported against Grain in [6]. This is an attack on the initialization of the Grain and is based on the facts that only the last bit is updated in both NFSR and LFSR of a LFSR/NFSR based stream cipher (like Grain) at each round and the other bits are just shifted bit stream of the previous clock cycle. Now, the Key-IV pair is loaded to NFSR and LFSR during initialization. After first round of initialization, one of the state bits is updated for each LFSR/NFSR and the rest of the state bits are shifted Key-IV bits. If the last bit of the LFSR is initialized with one (to prevent LFSR to go into all zero state) and the updated bit is also one, then it corresponds to another known Key-IV pair when the original key-IV pair is known. For a NFSR/LFSR based stream cipher like Grain, the state of the LFSR and NFSR at the start of key-stream generation using the new Key-IV pair above is same as the state of the LFSR and NFSR after producing one key-stream bit using the original key-IV pair if the zeroth bit of the NFSR after initialization process of the original key-IV pair is zero. The conditions above happen with a probability of 1/4 assuming both the conditions are independent.

Prevention of slide re-synchronization attack Now, let us try to apply the same attack on *CAR30*. When we use Cellular Automata instead of NFSR/LFSR, at every step all the bits in the Cellular Automata get updated. Hence in the initialization phase, the content of the Cellular Automata is never a shifted initial value and neither has it corresponded to an IV. Hence it is not possible to find a related key. Similarly, in order that the key stream generated with the original key-IV pair becomes a shifted key-stream generated from the related key-IV pair, the assumption is the content of state bits is the shifted content of the previous state bits, except the last bit. This assumption is no longer true when we use Cellular Automata. In

Cellular Automata the next state of each bit is the XOR of its neighboring bits and also XOR with itself depending on the rule vector. So this attack is not possible when we replace LFSR with Cellular Automata in *CAR30*.

4.10 Guess and determine attack

In each iteration of the cipher, all the state bits undergo parallel transformations. It is an advantage of CA over LFSR that all the bits in a CA undergo transformation in each cycle. As shown previously, each output bit depends on 160 previous state bits. Hence guessing a few bits in the state and then determining other state bits, as this attack employs [15], is not going to be an easy task.

4.11 Fault attack

Here the attacker can introduce a fault in any of the state bits with partial control on timings and the position of the fault. It is assumed that the attacker can reset the device and re-introduce the fault to observe the behavior of the cipher. The CA will make such a fault introduced impossible to track. This is because both for the non-linear CA and the linear CA performs XOR with the neighbors and after running for few cycles the sequences will spread statistically very well. The excellent values of SAC obtained in the 12-bit version of the cipher support this argument.

4.12 Side channel attack

In this type of attack, the attacker observes the number of cycles or the power consumed for different keys. Based on the power characteristics, the attacker tries to determine the Key. One of the most important side channel attack is differential power analysis (DPA) [17]. DPA has been performed mainly in public key cryptography and in block cipher in the first or the last round of iteration. Highly non-linear boolean functions and S-boxes that are used in block ciphers are particularly vulnerable [7, 23]. There has been some work reported for stream ciphers in the literature [11]. To prevent DPA, one of the methods is to apply linear masking during implementation [18]. In *CAR30*, such a linear masking can be performed in both linear CA and non-linear CA at the implementation level to prevent DPA attack.

5 Implementation

In this section, we describe the specifics of the implementation. *CAR30* can be implemented easily both in hardware and software. The following subsections describe the hardware architecture followed by the software implementation suggestions.

5.1 Hardware architecture

The hardware architecture of the proposed stream cipher is shown in Fig. 3. This architecture uses one module each for Rule 30 CA implementation and Rule 90/150 maximum length linear CA. These two CAs run continuously in every cycle using two flip-flop banks of 128 bits. For the linear CA, the output is fed back to the input. An

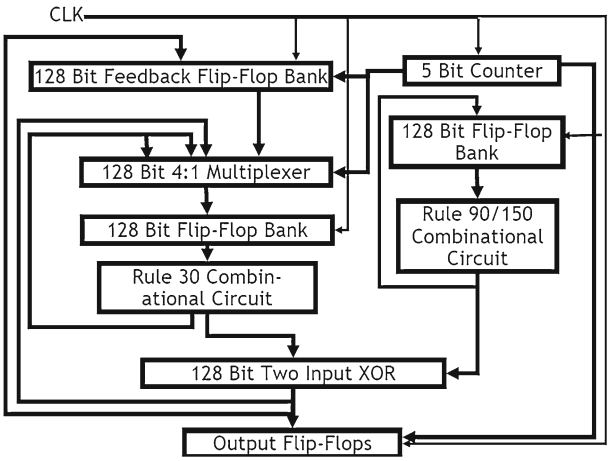

Fig. 3 The cipher hardware architecture

128-bit series of 4:1 multiplexer controls the input of the Rule 30 CA. The output of the non-linear CA is fed back to the multiplexer. The outputs of both the CAs are fed to a module of 128-bit two input XOR gates. There is another 128-bit flip-flop bank (feedback) that stores the output of the XOR layer of the Fig. 1. The output of this flip-flop bank is connected to the input of the multiplexer. The output of the XOR is connected to the feedback flip-flops and the multiplexer input. The whole circuit is controlled by a 5-bit counter which functions as follows. After the linear CA and the non-linear CA are run for 16 cycles, it enables the feedback flip-flop bank to store the output of the XOR for the next iteration. After 16 more cycles, it enables the output flip-flops to store the key-streams. In every clock cycle, it selects the required input from the multiplexer to feed to the non-linear CA. On sixteenth cycle, the multiplexer circuit outputs the XOR output, on thirty-second cycle it outputs the feedback flip-flop contents and for the rest of the cycles it outputs the non-linear CA output.

The architecture above for the key stream generation was synthesized with Xilinx ise v7.1i, on spartan 3 FPGA device with speed as optimization goal. In synthesis results, the total number of slices required was 499, the speed estimate was 185.05 MHz (running the CAs for 32 cyles was included in synthesis). Hence, the throughput of the cipher is 23.7 GBPS and the throughput to area ratio is 47.5 MHz per slice. When post place and route static analysis was performed, the area was 547 slices, the speed was 63.5 MHz and the throughput was 8.1 GBPS. Hence, the throughput to area ratio was 14.9 MHz per slice. The other hardware parameters on post place and route results are number of LUTs 936, number of bonded IOBs 258, minimum period 15.7 ns.

5.2 Software implementation

CAR30 can be efficiently implemented in 8-bit, 16-bit, 32-bit and 64-bit architecture. Thus, it is suitable for both small devices as well as large devices. We show here how it is implemented in 32-bit architecture. Similar logic can be used in other architectures for implementation. The implementation has a small foot-print with less code size.

The cipher code when compiled and executed in Microsoft Visual Studio 2008 Version 9.0.30729.1 SP, on a IBM Lenovo T60 laptop with Intel Core Duo CPU T2500 @2GHz, it took 9 s to generate 1.28×10^9 bits of key streams. This provides us a very good indication on the software speed of the cipher. This was also run on HP rp3440-4core server with OS HP-UX B.11.23 and PA8900 CPU (999MHz). There it took 22.35 s to generate 1.28×10^9 bits of key-stream with a clock speed of 10^6 and the total number of clocks elapsed was 22,350,000. To compare, we used the C-code given for Rabbit and Grain in eSTREAM and generated the same number of bits. Rabbit took 30 s to generate same number of key stream bits and the total number of clocks elapsed was 30,560,000. On the other hand, Grain was much slower in software in the same platform and it took 3,300 s to generate same number of bits with total clocks was 3,242,000,000. We can conclude that *CAR30* is faster in software than the software oriented stream cipher Rabbit.

6 Comparison with grain and trivium

In this section we provide comparison of CAR30 with two of the eSTREAM finalists Grain and Trivium. The comparisons are shown in Table 1. The hardware implementation figures are taken from [12], for the basic mode of Grain and Trivium. We acknowledge that hardware implementation figures depend on various factors like expertise, synthesis tools etc. But, nevertheless the table points out some distinct advantages of *CAR30* over the state of the art stream ciphers.

Firstly, on synthesis results the hardware estimation for throughput, throughput to area ratio and FOM of *CAR30* is much above both Grain and Trivium. The raw hardware speed on the synthesis result is also quite at par with the state of the art due to parallel transformation. As expected, the area is more again due to parallel transformation. Normally, the hardware results go down on post place and route results and so is the case with *CAR30*. Even then, the hardware throughput is four times more than the best possible results achieved with Grain. Throughput to area ratio and the FOM achieved with post place and route results of *CAR30* is more than Grain and is comparable with Trivium. The main advantage of *CAR30* over Trivium is the number of initialization cycles which is 160 for *CAR30* and 1,152 for Trivium.

Table 1 Comparison of *CAR30* with grain and trivium

Parameter	*CAR30*	Grain	Trivium
Key length	128	80/128	80
IV length	120	64/96	80
State size	256	160/256	288
Hardware area	499/547	122–356	188–388
HW speed (MHz)	185.1/63.5 (Synth/PPR)	193–155	201–190
HW throughput (Mbps)	23,692/8136 (Synth/PPR)	193–2480	201–12160
Throughput to area	47.5/14.9 (Synth/PPR)	1.58–6.97	1.07–31.34
Figure of merit [1] (Throughput/Area^2)	0.095/0.027	0.013–0.019	0.006–0.080
S/W efficiency	1.5 × *Rabbit* speed	0.01 × *Rabbit* speed	–
Extensibility	Easy	Redesign	Redesign
Initialization cycles	160	160/256	1,152

Another advantage of *CAR30* over any other hardware oriented stream cipher is the software speed. The software speed of *CAR30* is even better than software oriented stream cipher Rabbit. We have arbitrarily chosen Rabbit for comparison from the ESTREAM finalist software oriented stream ciphers. In this respect *CAR30* is indeed a unique stream cipher that can be compared with both hardware oriented and software oriented stream ciphers. Finally, the generic design of *CAR30* helps us to extend the cipher for a larger Key and IV size, which is another unique advantage over any other stream ciphers.

7 Conclusions

This paper proposes a new stream cipher based on Rule 30 and Maximum Length Cellular Automata. The cipher is shown to be scalable to any key length. This cipher is secure and can be implemented easily in both hardware and software. The speed and the resources required for the cipher are comparable to the existing standards.

Appendices

Experimental evaluation of security on simplified *CAR30*

Since, *CAR30* can be scaled up and scaled down easily, we have performed an experimental evaluation on simplified version of *CAR30* with 12 bit Key size. This is another advantage of the cipher that it keeps nothing obscure in the design. It can be analyzed with a smaller Key size and the analysis can be extended for a larger Key size. In this experiment, the key-streams were generated and the cipher was initialized in the same way as shown in Figs. 1 and 2. We have used a 12-bit maximum length CA with rule 150 on position 3 and 7 and rule 90 on the rest of the bits. Rule 30 was used for the non-linear CA. We have fixed an IV (1234 in decimal) arbitrarily and checked the first 12-bit key stream block for each of the possible 12-bit Keys. Each of the CA blocks (both linear CA and non-linear CA) were run for 4 cycles. Then we defined an input/output relationship between the Key and the first key-stream generated and analyzed this input/output with respect to different security properties like algebraic normal form (ANF), algebraic degree (Alg Deg), linear and differential (Diff) cryptanalysis, strict avalanche criterion (SAC) and non-linearity (NL) etc. Since evaluation of these properties require exponential time, we evaluate it for 12 bits only.

We select 12-bit version for evaluation because the complexity of determining the above properties becomes high for higher number of input/output bits. We have evaluated the algebraic normal form for the 12-bit version of cipher generated in the above way. We found that all the output bits have the algebraic degree as eleven or twelve which is the maximum possible for a 12-bit boolean function. The number of terms was large with a very good distribution of the degrees. The maximum number of terms in ANF was 2,108 for bit number 6 and the minimum number of terms was 1,969 for the bit number 12. Note that, for a random boolean function it should be

2,048. The interaction of the algebraic normal form among the output bits was also checked. The number of terms not occurring at all in the algebraic normal form of all output bits was one. The number of terms occurring only once in the algebraic normal form of all the output bits was 10.

The check the immunity against differential attacks, difference distribution table was generated and the maximum value of in the difference distribution table was found to be 14, showing its immunity against differential attacks. To measure the resistance against fault attack or bit flipping, we have also checked the strict avalanche criterion for the cipher. The maximum value of SAC was 2,150 and the minimum value was 1,904, that proves that any bit flipping will spread well in this cipher. Finally, to check how much non-linearity the rule 30 based cipher can provide, we determined the non-linearity of the cipher. The maximum non-linearity was found to be 1,950 for bit number 8 and the minimum non-linearity was found to be 1,920 for bit number 12. This provides us an indication that the *CAR30* generates highly non-linear key-stream bits.

Similarly, to evaluate the interactions among the key stream bits, we fixed the Key and the IV and produced 2^{12} blocks of key stream bits. The input/output relationship is formed with iteration number and the key-stream of that iteration. We could see excellent algebraic properties of the key stream bits. The number of terms not occurring at all in the ANF of all the bits was 5. The number of terms occurring only once is 13. The maximum differential value was 16. We also got good non-linearity values and SAC values. The Table 2 summarizes the results obtained.

Some of the bits in this analysis had the algebraic degree as 12. Hence, the relationship of Key and the first 12 bits of the cipher is not bijective. However, in a stream cipher the bijective property is not necessary with some particular bits of key stream because it is supposed to be random. But, the sequences of key stream produced for each Key when the IV is fixed, must be unique. We have checked each sequence with each Key and found that it is indeed unique. Period is another aspect that we verified in this experimental evaluation. We checked the period in the second experiment and there was no periodic sequences in the first 2^{12} key stream bits.

It can be seen that excellent values of cryptographic properties were achieved in the simplified version of the cipher with 12-bit Key size. Since the cipher has a regular structure, the cipher can be extended by merely adding more cells to the linear and non-linear CA. The security properties found in this section will also be automatically extended to a larger version of the cipher if the number of cycles is also increased linearly. It is difficult to perform experimental evaluation for the larger version of the cipher, but still this evaluation along with the statistical tests on the 128-bit version of the cipher provide us the confidence on the security of the larger versions.

Table 2 Security properties for 12-bit version of the cipher

Input/Output	Max ANF term	Min ANF term	Alg deg	Max diff	Max SAC	Min SAC	Max NL	Min NL
Key/1st KS	2,108	1,969	11/12	14	2,150	1,904	1,950	1,920
i-th Iteration/*i*-th KS	2,085	1,993	11/12	16	2,150	1,932	1,942	1,919

Pseudo-code

The pseudo-code for the cipher given in Algorithm 1. The 128-Bit non-linear states and the 128-Bit linear states can be stored in $(4 * 2 =)$ eight 32-Bit integers,

Algorithm 1 The CAR30 Software Algorithm

Require: Input Key, IV, Linear Rule and Number of Cycles

1: $NLCA0 = KEY0$; $NLCA1 = KEY1$; $NLCA2 = KEY2$; $NLCA3 = KEY3$; /*Initialization*/
2: $LCA0 = IV0$; $LCA1 = IV1$; $LCA2 = IV2$; $LCA3 = IV3$; /*Include 1 in every 16th bit.*/
3: **for** j=1 to 5 **do**
4: **for** i=1 to 16 **do**
5: $TEMP0 = (LCA0 << 1) \oplus (LCA1 >> 31) \oplus (RULE0 \& LCA0) \oplus (LCA0 >> 1)$;
6: /* TEMP1, TEMP2 and TEMP3 are assigned in a similar way*/
7: $LCA0 \cdots LCA3 = TEMP0 \cdots TEMP3$;
8: $TEMP0 = (((NLCA0 << 1) \oplus (NLCA1 >> 31))|(NLCA0)) \oplus ((NLCA0 >> 1) \oplus (NLCA3 << 31)) \oplus (LCA2 \& 0x800080)$;
9: $TEMP1 = (((NLCA1 << 1) \oplus (NLCA2 >> 31))|(NLCA1)) \oplus ((NLCA1 >> 1) \oplus (NLCA0 << 31)) \oplus (LCA3 \& 0x800080)$;
10: $TEMP2 = (((NLCA2 << 1) \oplus (NLCA3 >> 31))|(NLCA2)) \oplus ((NLCA2 >> 1) \oplus (NLCA1 << 31)) \oplus (LCA0 \& 0x800080)$;
11: $TEMP3 = (((NLCA3 << 1) \oplus (NLCA0 >> 31))|(NLCA3)) \oplus ((NLCA3 >> 1) \oplus (NLCA2 << 31)) \oplus (LCA1 \& 0x800080)$;
12: $NLCA0 \cdots NLCA3 = TEMP0 \cdots TEMP3$;
13: $NLCA0 \cdots NLCA3 = LCA0 \oplus NLCA0 \cdots LCA3 \oplus NLCA3$;
14: **end for**
15: $FEEDBACK0 \cdots FEEDBACK3 = NLCA0 \cdots NLCA3$;
16: **for** i=1 to 16 **do**
17: /*Repeat the statements of lines 5 to 13*/
18: **end for**
19: $LCA0 \cdots LCA3 = LCA0 \oplus NLCA0 \cdots LCA3 \oplus NLCA3$;
20: $NLCA0 \cdots NLCA3 = FEEDBACK0 \cdots FEEDBACK3$;
21: **end for**/* Initialization Complete.*/
22: **while** enough key stream is not generated **do**
23: **for** i=1 to 16 **do**
24: /* Repeat the statements line 5 to 13*/
25: $NLCA0 \cdots NLCA3 = LCA0 \oplus NLCA0 \cdots LCA3 \oplus NLCA3$;
26: **end for**
27: $FEEDBACK0 \cdots FEEDBACK3 = NLCA0 \cdots NLCA3$;
28: **for** i=1 to 16 **do**
29: /*Repeat the statements of lines 5 to 13*/
30: **end for**
31: $KS0 \cdots KS3 = NLCA0 \cdots NLCA3$;
32: $NLCA0 \cdots NLCA3 = FEEDBACK0 \cdots FEEDBACK3$;
33: **end while**

called $NLCA0 \cdots NLCA3$ and $LCA0 \cdots LCA3$. Four temporary variables, called $TEMP0 \cdots TEMP3$, store the state bits temporarily for processing. The 128-bit Key ($KEY0 \cdots KEY3$) and the extended IV ($IV0 \cdots IV3$) are taken as inputs. Finally, the $FEEDBACK0 \cdots FEEDBACK3$ variables store the feedback for the non-linear block. The integers $RULE0 \cdots RULE3$ contains the rule vector of the 128-bit maximum length CA.

Test vectors

1. Key=0x00000000 00000000 00000000 00000000, IV=0x00010001 00010001 00010001 00010001 (including eight fixed 1s), KS= 0x256606be 816e094b 911084a3 a8b7015f
2. Key=0xa9b8c92d 56cad670 05ae2175 56d347a9, IV=0x56b17787 5331bd01 6391ac65 42619871 (including eight fixed 1s), KS= 0x7da160b6 1ef7b7f9 e419dc38 331f5531

References

1. Badel, S., et al.: ARMADILLO: a multi-purpose cryptographic primitive dedicated to hardware. In: Proceedings of Workshop on Cryptographic Hardware and Embedded Systems 2010 (CHES 2010). LNCS, vol. 6225, pp. 398–412. Springer (2010)
2. Biryukov, A., Shamir, A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. In: Proceedings of ASIACRYPT 2000. LNCS, vol. 1976, pp. 1–13. Springer-Verlag (2000)
3. Bjøstad, T.E.: Cryptanalysis of Grain using Time/Memory/Data Tradeoffs. http://www.ecrypt.eu.org/stream/papersdir/2008/012.pdf. Accessed 29 Jan 2013
4. Boesgaard, M., Vesterager, M., Pedersen, T., Christiansen, J., Scavenius, O.: Rabbit: a high-performance stream cipher. In: Proc. FSE 2003. LNCS, vol. 2887, pp. 307–329. Springer (2003)
5. De Canniere, C., Preneel, B.: Trivium Specification. http://www.ecrypt.eu.org/stream/triviump3.html. Accessed 29 Jan 2013
6. De Canniere, C., Kücük, O., Preneel, B.: Analysis of Grain's Initialization Algorithm. Presented in SAC2008. http://www.ecrypt.eu.org/stvl/sasc2008/. Accessed 29 Jan 2013
7. Carlet, C.: On highly nonlinear S-boxes and their inability to thwart DPA attacks. Indocrypt 2005. LNCS, vol. 3797, pp. 49–62 (2005)
8. Courtois, N., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: Biham, E. (ed.) Advances in Cryptology-EUROCRYPT 2003, LNCS, vol. 2656, pp. 345–359. Springer-Verlag (2003)
9. Diffie, W.: The first ten years of public key cryptography. Proc. I.E.E.E. **76**, 5 (1988)
10. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials, EUROCRYPT 2009. Also on Cryptology ePrint Archive, Report 2008/385.
11. Fischer, W., Gammel, B.M., O. Kniffler, Velten, J.: Differential power analysis of stream ciphers. Lect. Notes Comput. Sci. **4377**, 257–270 (2006). doi:10.1007/11967668_17
12. Gaj, K., Southern, G., Bachimanchi, R.: Comparison of hardware performance of selected Phase II eSTREAM candidates. http://www.ecrypt.eu.org/stream/papersdir/2007/026.pdf. Accessed 29 Jan 2013
13. Golic, J.D.: On the security of nonlinear filter generators. In: Gollmann, D. (ed.) FSE '96, LNCS, vol. 1039, pp. 173–188 (1996)
14. Golic, J.D., Clark, A.J., Dawson, E.P.: Generalized inversion attack on nonlinear filter generators. IEEE Trans. Comput. **49**(10), 1100–1109 (2000)
15. Hawkes, P., Rose, G.: Guess-and-determine attacks on SNOW. In: Nyberg, K., Heys, H. (eds.) Selected Areas in Cryptography, SAC 2002, LNCS, vol. 2595, pp. 37–46. Springer-Verlag (2002)
16. Hell, M., Johansson, T., Maximov, A., Meier, W.: A stream cipher proposal: grain-128. In: IEEE International Symposium on Information Theory, pp. 1614–1618 (2006). doi:10.1109/ISIT.2006.261549
17. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) Advances in Cryptology—CRYPTO 1999, LNCS, vol. 1666, pp. 388–397. Springer-Verlag (1999)

18. Akkar, M., Giraud, C.: An implementation of DES and AES, secure against some attacks. In: Koc, C., Naccache, D., Paar, C. (eds.) Proceedings of CHES'01, LNCS, vol. 2162, pp. 309–318. Springer-Verlag (2001)
19. Maximov, A.: Cryptanalysis of the Grain family of stream ciphers. In: ACM Symposium on Information, Computer and Communications Security (ASIACCS'06), pp. 283–288 (2006)
20. Meier, W., Staffelbach, O.: Analysis of pseudo random sequences generated by cellular automata. In: Advances in Cryptology EUROCRYPT-91, LNCS, vol. 547, pp. 186–199. Springer (1991)
21. NIST Statistical Test Suit. http://csrc.nist.gov/rng/. Accessed 29 Jan 2013
22. PalChaudhury, P., RoyChowdhury, D., Nandi, S., Chattopadhay, S.: Additive Cellular Automata Theory and Application, vol. 1. IEEE Computer Society Press, Los Alamitos, CA (1997)
23. Prouff, E.: DPA attacks and S-boxes. In: FSE 2005, LNCS vol. 3557, pp. 424–441 (2005)
24. Sarkar, P.: The filter-combiner model for memoryless synchronous stream ciphers. In: Yung, M. (ed.) Crypto, 2002, LNCS, vol. 2442, pp. 533–548. Springer (2002)
25. The Estream Project. http://www.ecrypt.eu.org/stream/. Accessed 29 Jan 2013
26. Wolfram, S.: Random sequence generation by cellular automata. Adv. Appl. Math. **7**, 123–169 (1986)
27. Wolfram, S.: Cryptography with cellular automata. In: Advances in Cryptology, Crypto-85, Proceedings, LNCS, vol. 218, pp. 429–432. Springer-Verlag (1986)