Some cryptanalytic results on Lizard

Subhadeep Banik[1] and Takanori Isobe[2]

[1] Temasek Labs, Nanyang Technological University, Singapore.
`bsubhadeep@ntu.edu.sg`
[2] Graduate School of Applied Informatics, University of Hyogo, Japan.
`takanori.isobe@ai.u-hyogo.ac.jp`

Abstract. Lizard is a lightweight stream cipher proposed by Hamann, Krause and Meier in `IACR ToSC` 2017. It has a Grain-like structure with two state registers of size 90 and 31 bits. The cipher uses a 120 bit Secret Key and a 64 bit IV. The authors claim that Lizard provides 80 bit security against key recovery attacks and a 60-bit security against distinguishing attacks. In this paper, we present an assortment of results and observations on Lizard. First, we show that by doing 2^{58} random trials it is possible to a set of 2^{64} triplets (K, IV_0, IV_1) such that the Key-IV pairs (K, IV_0) and (K, IV_1) produce identical keystream bits. Second, we show that by performing only around 2^{28} random trials it is possible to obtain 2^{64} Key-IV pairs (K_0, IV_0) and (K_1, IV_1) that produce identical keystream bits. Thereafter, we show that one can construct a distinguisher for Lizard based on IVs that produce shifted keystream sequences. The process takes around $2^{51.5}$ random IV encryptions and around $2^{76.6}$ bits of memory. Finally, we propose a key recovery attack on a version of Lizard with the number of initialization rounds reduced to 223 (out of 256) based on IV collisions.

Keywords: Grain v1, Lizard, Stream Cipher.

1 Introduction

Lightweight stream ciphers have become immensely popular in the cryptological research community, since the advent of the eStream project [1]. The three hardware finalists included in the final portfolio of eStream i.e. Grain v1 [13], Trivium [7] and MICKEY 2.0 [4], all use bitwise shift registers to generate keystream bits. After the design of Grain v1 was proposed, two other members Grain-128 [14] and Grain-128a were added to the Grain family mainly with an objective to provide a larger security margin and include the functionality of message authentication respectively. In FSE 2015, Armknecht and Mikhalev proposed the Grain-like stream cipher Sprout [2] with a startling trend: the size of its internal state of Sprout was equal to the size of its Key. After the publication of [6], it is widely accepted that to be secure against generic Time-Memory-Data tradeoff attacks, the internal state of a stream cipher must be atleast twice the size of the secret key. However the novelty of the Sprout design ensured that the cipher remained secure against generic TMD tradeoffs. The smaller internal state makes the cipher particularly attractive for compact lightweight implementations. However, Sprout has been cryptanalyzed in more ways than one [5,8,12,19] and so naturally there has been a lot of research going into design of secure lightweight stream ciphers.

At the FSE 2017 conference of `IACR ToSC`, two lightweight stream ciphers Lizard [11] and Plantlet [15] were proposed. While Plantlet was a re-design of Sprout after patching some existing weaknesses, Lizard was a new construction. It uses a Grain-like structure with two state registers of size 90 and 31 bits. The cipher uses a 120 bit Secret Key and a 64 bit IV. The authors claim 80 bit security against generic Time Memory Data Tradeoff attacks, and 60 bit security against distinguishing attacks. Unlike members of the Grain family [3,13,14], Sprout [2] and Plantlet [15], the Key-IV mixing in Lizard is not one-to-one and hence not invertible. This guarantees that even if an attacker manages to recover the internal state of Lizard, it does not lead to a key recovery attack. The authors also recommend that not more than 2^{18} keystream bits be generated from one Key-IV pair, and hence Lizard is not suitable for applications requiring encryption of large bulks of data.

1.1 Contribution and Organization of the Paper

We summarize the contributions in this paper as follows. In Section 2, we present the mathematical description of the Lizard stream cipher. In Section 3, we show that by performing around 2^{58} random experiments, it is possible to get 2^{64} triplets (K, IV_1, IV_2) such that the Key-IV pairs (K, IV_1) and (K, IV_2) produce identical

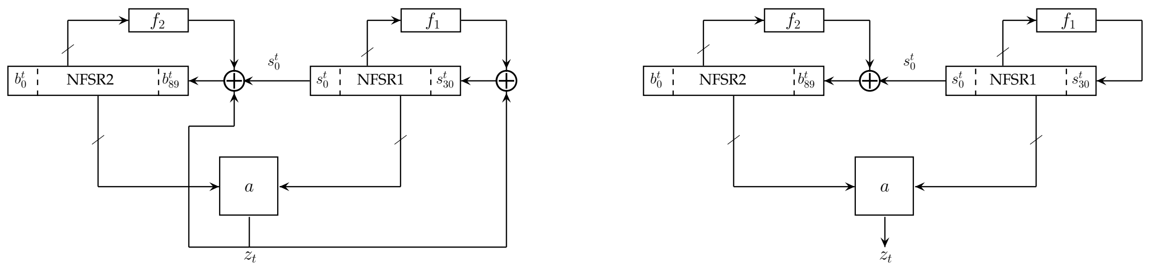


Fig. 1: Block Diagram of Lizard (A) In Phase 2 of Initialization, (B) During keystream generation

keystream bits. In Section 4, we show that by performing only around 2^{28} random trials it is possible to obtain 2^{64} Key-IV pairs (K_1, IV_1) and (K_2, IV_2) that produce identical keystream bits. In Section 5, we show that one can construct a distinguisher for Lizard based on IVs that produce shifted keystream sequences. The process takes around 2^{52} random IV trials and around 2^{69} simple bit operations. Finally in Section 6, we propose a key recovery attack on a version of Lizard with the number of initialization rounds reduced to 223 (out of 256) based on IV collisions. In Section 7, we conclude the paper.

2 Description of Lizard

The exact structure of Lizard is explained in Figure 1. It consists of two NFSRs of size 90 and 31 bits each. Certain bits of both the shift registers are taken as inputs to a combining Boolean function, whence the keystream is produced. The 121-bit inner state of Lizard is distributed over the two NFSRs, NFSR1 and NFSR2, whose contents at time $t = 0, 1, \ldots$ is denoted by $S^t = (s_0^t, s_1^t, \ldots, s_{30}^t)$ and $B^t = (b_0^t, b_1^t, \ldots, b_{89}^t)$ respectively. We also have, for $t \in \mathbb{N} \setminus \{0, 128\}$, $s_i^{t+1} = s_{i+1}^t$ for $i = 0$ to 29 and $b_i^{t+1} = b_{i+1}^t$ for $i = 0$ to 88. The keystream is produced after performing the following steps:

Phase 1: Key-IV loading: Let $K = (k_0, k_1, \ldots, k_{119})$ denote the 120-bit key and $IV = (v_0, v_1, \ldots, v_{63})$ be the 64-bit public IV. The registers of the keystream generator are initialized as follows:

$$b_j^0 = \begin{cases} k_j \oplus v_j, & \text{for } j \in \{0,1,2,\ldots,63\} \\ k_j, & \text{for } j \in \{64,65,66,\ldots,89\} \end{cases} \qquad s_i^0 = \begin{cases} k_{i+90}, & \text{for } i \in \{0,1,2,\ldots,28\} \\ k_{119} \oplus 1, & \text{for } i = 29 \\ 1, & \text{for } i = 30 \end{cases}$$

Phase 2: Mixing: During this phase the cipher is clocked for 128 cycles without producing any keystream bits. During this phase the registers are updated as follows. For $t = 0, 1, 2, \ldots, 127$, we compute:

$$\begin{aligned} b_i^{t+1} &= b_{i+1}^t, \quad \text{for} \quad i \in \{0,1,\ldots,88\} \\ b_{89}^{t+1} &= z_t \oplus s_0^t \oplus f_2(B^t) \end{aligned}$$

$$\begin{aligned} s_i^{t+1} &= s_{i+1}^t, \quad \text{for} \quad i \in \{0,1,\ldots,29\} \\ s_{30}^{t+1} &= z_t \oplus f_1(S^t) \end{aligned}$$

where $f_1(S^t), f_2(B^t)$ and z_t are computed as follows:

$$\begin{aligned}
f_1(S^t) = {} & s_0^t \oplus s_2^t \oplus s_5^t \oplus s_6^t \oplus s_{15}^t \oplus s_{17}^t \oplus s_{18}^t \oplus s_{20}^t \oplus s_{25}^t \oplus s_8^t \cdot s_{18}^t \oplus s_8^t \cdot s_{20}^t \oplus s_{12}^t \cdot s_{21}^t \oplus s_{14}^t \cdot s_{19}^t \oplus \\
& s_{17}^t \cdot s_{21}^t \oplus s_{20}^t \cdot s_{22}^t \oplus s_4^t \cdot s_{12}^t \cdot s_{22}^t \oplus s_4^t \cdot s_{19}^t \cdot s_{22}^t \oplus s_7^t \cdot s_{20}^t \cdot s_{21}^t \oplus s_8^t \cdot s_{18}^t \cdot s_{22}^t \oplus s_8^t \cdot s_{20}^t \cdot s_{22}^t \oplus \\
& s_{12}^t \cdot s_{19}^t \cdot s_{22}^t \oplus s_{20}^t \cdot s_{21}^t \cdot s_{22}^t \oplus s_4^t \cdot s_7^t \cdot s_{12}^t \cdot s_{21}^t \oplus s_4^t \cdot s_7^t \cdot s_{19}^t \cdot s_{21}^t \oplus s_4^t \cdot s_{12}^t \cdot s_{21}^t \cdot s_{22}^t \oplus \\
& s_4^t \cdot s_{19}^t \cdot s_{21}^t \cdot s_{22}^t \oplus s_7^t \cdot s_8^t \cdot s_{18}^t \cdot s_{21}^t \oplus s_7^t \cdot s_8^t \cdot s_{20}^t \cdot s_{21}^t \oplus s_7^t \cdot s_{12}^t \cdot s_{19}^t \cdot s_{21}^t \oplus s_8^t \cdot s_{18}^t \cdot s_{21}^t \cdot s_{22}^t \oplus \\
& s_8^t \cdot s_{20}^t \cdot s_{21}^t \cdot s_{22}^t \oplus s_{12}^t \cdot s_{19}^t \cdot s_{21}^t \cdot s_{22}^t
\end{aligned}$$

$$\begin{aligned}
f_2(B^t) = {} & b_0^t \oplus b_{24}^t \oplus b_{49}^t \oplus b_{79}^t \oplus b_{84}^t \oplus b_3^t \cdot b_{59}^t \oplus b_{10}^t \cdot b_{12}^t \oplus b_{15}^t \cdot b_{16}^t \oplus b_{25}^t \cdot b_{53}^t \oplus b_{35}^t \cdot b_{42}^t \oplus b_{55}^t \cdot b_{58}^t \oplus \\
& b_{60}^t \cdot b_{74}^t \oplus b_{20}^t \cdot b_{22}^t \cdot b_{23}^t \oplus b_{62}^t \cdot b_{68}^t \cdot b_{72}^t \oplus b_{77}^t \cdot b_{80}^t \cdot b_{81}^t \cdot b_{83}^t
\end{aligned}$$

$$L_t = b_7^t \oplus b_{11}^t \oplus b_{30}^t \oplus b_{40}^t \oplus b_{45}^t \oplus b_{54}^t \oplus b_{71}^t$$

$$Q_t = b_4^t \cdot b_{21}^t \oplus b_9^t \cdot b_{52}^t \oplus b_{18}^t \cdot b_{37}^t \oplus b_{44}^t \cdot b_{76}^t$$

$$\begin{aligned}
T_t = {} & b_5^t \oplus b_8^t \cdot b_{82}^t \oplus b_{34}^t \cdot b_{67}^t \cdot b_{73}^t \oplus b_2^t \cdot b_{28}^t \cdot b_{41}^t \cdot b_{65}^t \oplus b_{13}^t \cdot b_{29}^t \cdot b_{50}^t \cdot b_{64}^t \cdot b_{75}^t \oplus \\
& b_6^t \cdot b_{14}^t \cdot b_{26}^t \cdot b_{32}^t \cdot b_{47}^t \cdot b_{61}^t \oplus b_1^t \cdot b_{19}^t \cdot b_{27}^t \cdot b_{43}^t \cdot b_{57}^t \cdot b_{66}^t \cdot b_{78}^t
\end{aligned}$$

$$\tilde{T}_t = s_{23}^t \oplus s_3^t \cdot s_{16}^t \oplus s_9^t \cdot s_{13}^t \cdot b_{48}^t \oplus s_1^t \cdot s_{24}^t \cdot b_{38}^t \cdot b_{63}^t$$

$$z_t = L_t \oplus Q_t \oplus T_t \oplus \tilde{T}_t$$

Phase 3: Second Key Addition: After this the 120 bit key is added to the state as follows:

$$b_j^{129} = b_j^{128} \oplus k_j, \qquad \text{for } j \in \{0,1,2,\ldots,89\} \;\; \text{and} \;\; s_i^{129} = \begin{cases} s_i^{128} \oplus k_{i+90}, & \text{for } i \in \{0,1,2,\ldots,29\} \\ 1, & \text{for } i = 30 \end{cases}$$

Phase 4: Diffusion: During this phase the cipher is again clocked for 128 cycles without producing any keystream bit. However the feedback of the keystream bit is discontinued. Thus for $t = 129, 130, 131, \ldots, 256$, we compute:

$$\begin{aligned}
b_i^{t+1} &= b_{i+1}^t, \quad \text{for} \quad i \in \{0,1,\ldots,88\} \\
b_{89}^{t+1} &= s_0^t \oplus f_2(B^t) \\
\\
s_i^{t+1} &= s_{i+1}^t, \quad \text{for} \quad i \in \{0,1,\ldots,29\} \\
s_{30}^{t+1} &= f_1(S^t)
\end{aligned}$$

After this the cipher starts producing the keystream bit z_t while following the update rule in Phase 4.

3 Finding IV collisions for the same Key

Phase 2 of the initialization process, essentially clocks the two NFSRs for 128 cycles without producing keystream. Since the update functions of both the shift registers are of the form $x_0^t \oplus f(x_1^t, x_2^t, \ldots, x_{n-1}^t)$, the update function is one-to-one and efficiently invertible [9]. As such the function F which maps the 121-bit input of Phase 2 to its output is essentially a permutation on $\mathbb{F}_2^{121}$. Since the same is true for Phase 4, the function map for this phase is also a permutation over the same domain. In fact, we present explicitly the process to invert one round of the state updates in Phases 2 and 4 (see Algorithms 1 and 2). The algorithms when iterated 128 times will invert the function maps of Phases 2 and 4 respectively. Before we present the algorithms, let us define $f_1(S^t) = s_0^t \oplus f_1'(s_1^t, s_2^t, \ldots, s_{30}^t)$ and $f_2(B^t) = b_0^t \oplus f_2'(b_1^t, b_2^t, \ldots, b_{89}^t)$ and the function $z(S^t, B^t) = z_t$.

```
Input: $S^t, B^t$: The NFSR states at time $t$;
Output: $S^{t-1}, B^{t-1}$: The NFSR states at time
        $t-1$;
----------------------------------------
$s \leftarrow s^t_{30}, \quad b \leftarrow b^t_{89}$;
$B' = (b^t_0, b^t_1 \ldots, b^t_{88}), \quad S' = (s^t_0, s^t_1 \ldots, s^t_{29})$;
$\hat{z} = z(S', B')$;
$\hat{s} = s \oplus f'_1(S') \oplus \hat{z}, \quad \hat{b} = b \oplus f'_2(B') \oplus \hat{s} \oplus \hat{z}$
$S^{t-1} \leftarrow (\hat{s}, s^t_0, s^t_1 \ldots, s^t_{29})$;
$B^{t-1} \leftarrow (\hat{b}, b^t_0, b^t_1 \ldots, b^t_{88})$;
Return $S^{t-1}, B^{t-1}$
```

Algorithm 1: Algorithm $P2^{-1}$

```
Input: $S^t, B^t$: The NFSR states at time $t$;
Output: $S^{t-1}, B^{t-1}$: The NFSR states at time
        $t-1$;
----------------------------------------
$s \leftarrow s^t_{30}, \quad b \leftarrow b^t_{89}$;
$B' = (b^t_0, b^t_1 \ldots, b^t_{88}), \quad S' = (s^t_0, s^t_1 \ldots, s^t_{29})$;

$\hat{s} = s \oplus f'_1(S'), \quad \hat{b} = b \oplus f'_2(B') \oplus \hat{s}$
$S^{t-1} \leftarrow (\hat{s}, s^t_0, s^t_1 \ldots, s^t_{29})$;
$B^{t-1} \leftarrow (\hat{b}, b^t_0, b^t_1 \ldots, b^t_{88})$;
Return $S^{t-1}, B^{t-1}$
```

Algorithm 2: Algorithm $P4^{-1}$

In Phase 3 of the initialization process, the designers set the last bit of NFSR2 i.e. s^{129}_{30} to 1. This makes the initialization process a non-injective function, which that there may be two different initial states that leads to the same 120 bit state after Phase 3. That is to say, it is possible to get a triplet K, IV_1, IV_2 so that after completion of Phase 2, the system initialized with K, IV_1 and the system initialized with K, IV_2 differ only in the last bit. Since Phase 3, adds the key to the first 120 bits and forces the last bits of both systems to 1, the internal states of both systems thereafter will be identical and they would obviously produce the same keystream bits. We call this event an IV collision. In the original Lizard paper [11], the authors had proven that, if the key K is unknown, then it would take around $2^{60.5}$ random IV trials with K to find an IV collision for K. What we show in this section is not opposed to the findings of [11]. We show that by performing around 2^{58} random experiments it is possible to tabulate a set of 2^{64} IV collisions for 2^{64} specific different keys. We do not assume the unknown key setting as in [11].

Our algorithm can be described as follows. Let $F : \mathbb{F}_2^{121} \rightarrow \mathbb{F}_2^{121}$ be the function which maps the 121-bit input of Phase 2 to its output. Since F is a permutation, so is F^{-1}. Let R be any 120 bit string. Consider the two 121 bit strings $R_0 = R||0$ and $R_1 = R||1$. Applying F^{-1} on each of these gives us $T_0 = F^{-1}(R_0)$ and $T_1 = F^{-1}(R_1)$. Now if there exists a a triplet K, IV_0, IV_1 such that $T_0 = K[0 \text{ to } 63] \oplus IV_0 \ || \ K[64 \text{ to } 119] \ || \ 1$ and $T_1 = K[0 \text{ to } 63] \oplus IV_1 \ || \ K[64 \text{ to } 119] \ || \ 1$, then the systems initialized with K, IV_0 and K, IV_1 after Phase 3 will both lead to the internal state $R \oplus K||1$. The states for both systems are identical hereafter and so they produce identical keystream bits. We put the above ideas into the form of an algorithm as follows:

```
Algorithm to generate IV Collision
----------------------------------------
1. Set Success $\leftarrow 0$

2. Do the following till Success =1

    • Select $R \overset{\mathsf{R}}{\longleftarrow} \{0,1\}^{120}$ randomly.
    • Define $R_0 := R||0$ and $R_1 := R||1$
    • Compute $T_0 = F^{-1}(R_0)$ and $T_1 = F^{-1}(R_1)$
    • If $T_0[64 \text{ to } 119] = T_1[64 \text{ to } 119]$ and $T_0[120] = T_1[120] = 1$ then set Success =1
    • If Success =1 then exit from loop else continue.

3. Select $\alpha \overset{\mathsf{R}}{\longleftarrow} \{0,1\}^{64}$ randomly.

4. Set $K = \alpha \ || \ T_0[64 \text{ to } 118] \ || \ T[119] \oplus 1$, Set $IV_0 = \alpha \oplus T_0[0 \text{ to } 63]$ and $IV_1 = \alpha \oplus T_1[0 \text{ to } 63]$

5. Return $K, IV_0, IV_1$.
```

The above subroutine can be briefly described as follows: we select a 120 bit string R and run the F^{-1} function on $R||0$ and $R||1$ (note that F^{-1} may be computed efficiently by invoking algorithm $P2^{-1}$ a total of 128 times) to get the 121 bit strings T_0 and T_1 respectively. We stop only if

A. The 64th to 119th bits of T_0 and T_1 are identical. This is because these bits of initial state are composed with the last 56 bits of the secret key. So if T_0 and T_1 are to come from the initialization with the same key, the 64th to 119th bits need to be identical.

B. The last bit of both T_0 and T_1 is equal to 1. This is because the in Phase 1, the starting state is initialized with the last bit equal to 1.

Both these events would be satisfied with probability 2^{-58} for a random R, and so the loop needs to be iterated around 2^{58} times before Success. Once the algorithm has the required pair T_0, T_1, we can make not one but 2^{64} triplets K, IV_1, IV_2 such that K, IV_1 and K_IV_2 will lead to an IV Collision. This is because the first 64 bits of the initial state is the bitwise xor of the IV and first 64 keybits. So we take any random 64-bit string α and set $K = \alpha \mid\mid T_0[64 \text{ to } 118] \mid\mid T[119] \oplus 1$ (the last bit of is inverted because the specifications of Phase 1 indicate that the last key bit inverted during the initialization). Then by setting $IV_0 = \alpha \oplus T_0[0 \text{ to } 63]$ and $IV_1 = \alpha \oplus T_1[0 \text{ to } 63]$ we ensure that after Phase 1, we have the required values of the initial states equal to T_0 and T_1. Since any value of α can be used, this gives us a set of 2^{64} triplets.

4 Finding Key-IV pairs K_0, IV_0 and K_1, IV_1 that produce same keystream

Since Lizard uses an internal state of 121 bits and the Key and IV in total is 184 bits long, it seems inevitable that there would exist two Key-IV pairs K_0, IV_0 and K_1, IV_1 that would lead to identical internal states after Phase 3, and hence produce exactly the same keystream bits. We call this event a Key-IV Collision. In this section, we will show that it is possible to find a Key-IV Collision after performing around 2^{28} random experiments. We will use the following subroutine to find Key-IV pairs that generate identical keystream sequences.

```
Algorithm to generate Key-IV Collision

1. Set Success $\leftarrow 0$

2. Fix a value of $L \overset{R}{\longleftarrow} \{0,1\}^{56}$.
3. Do the following till Success =1

    • Select $M \overset{R}{\longleftarrow} \{0,1\}^{64}$ randomly.
    • Define $R := M \mid\mid L \mid\mid 1$
    • Compute $S = F(R)$

    • Let $\hat{S} = S[64 \text{ to } 119]$, store $\hat{S}$ in a hash table along with current value of $M$.
    • If there is a collision in the hash table then Success =1.
    • If Success =1 then exit from loop else continue.

4. Let $M_0$ and $M_1$ are the values of $M$ which result in collision.

5. That is, 64th to 119th bits of $S_0 = F(M_0||L)$ and $S_1 = F(M_1||L)$ are equal.

6. Select $\alpha \overset{R}{\longleftarrow} \{0,1\}^{64}$ randomly and define $\Delta := S_0[0 \text{ to } 63] \oplus S_1[0 \text{ to } 63]$

7. Set $K_0 = \alpha \mid\mid L[0 \text{ to } 54] \mid\mid L[55] \oplus 1$, Set $IV_0 = \alpha \oplus M_0$ .

8. Set $K_1 = \alpha \oplus \Delta \mid\mid L[0 \text{ to } 54] \mid\mid L[55] \oplus 1$, Set $IV_1 = \alpha \oplus \Delta \oplus M_1$

9. Return $K_0, IV_0$ and $K_1, IV_1$.
```

The above algorithm can be described thus. We fix a 56 bit constant L which we will use to construct the 64th to 119th bits of the initial state. Then we choose a 64 bit constant M randomly and use it to construct the 1st 64 bits of the internal state. We run the state update function F of Phase 2 on $M \,||L\, ||\, 1$ and store the result in the variable S. We take the 56 bit value $\hat{S}$ which is the 64th to 119th bit of S and store it in a hash table along with the value of M. We keep doing this until we find a collision. Since we are looking for a collision over a 56 bit space, by birthday arguments this part of the algorithm should yield Success in around $\sqrt{2^{56}} = 2^{28}$ trials.

Once we have a collision we proceed as follows. Let M_0 and M_1 be the values of M that produce a collision. Then we will have the 64th to 119th bits of $S_0 = F(M_0||L)$ and $S_1 = F(M_1||L)$ equal. Phase 3 will set the 120th bit of both systems to 1, and so it is the first 120 bits we need to concentrate on. For identical keystream bits, we need that the states of both systems after the Key addition of Phase 3 be equal. The 64th to 119th bit of S_0 and S_1 are already equal, so we need that the difference of the two Keys K_0 and K_1 (in bits 0 to 63) that are to be added to S_0 and S_1 be equal to $\Delta = S_0[0 \text{ to } 63] \oplus S_1[0 \text{ to } 63]$. This ensures that both systems have identical internal states after Phase 3. So again, we take any 64-bit constant α and set $K_0 = \alpha \,||\, L[0 \text{ to } 54] \,||\, L[55] \oplus 1$ and $K_1 = \alpha \oplus \Delta \,||\, L[0 \text{ to } 54] \,||\, L[55] \oplus 1$. We must now ensure that the IVs be chosen so that the 2 systems start with the initial states $M_0 \,||\, L \,||\, 1$ and $M_1 \,||\, L \,||\, 1$ respectively. This can be done by setting $IV_0 = \alpha \oplus M_0$ and $IV_1 = \alpha \oplus \Delta \oplus M_1$. In Table 1 we tabulate a class of Key-IVs that produce the same keystream bits, that were found using the procedure listed above. Note that we can take any 64-bit constant α and add it to the first 64 bits of both the Keys and the IVs to get another set of Key-IV pairs that produce the same keystream bits. Thus we have 2^{64} such pairs from one run of the above algorithm.

#	$Key - IV$	Keystream
1	K_0: `0000 0000 0000 0000 6850 8c64 c649 74` IV_0: `724b b286 2f5c f1b2`	`23f4 9770 0a91 3089 d800 5513 58e1 6352 ...`
2	K_1: `1e45 1adc 2ad8 3124 6850 8c64 c649 74` IV_1: `3e18 82d1 d5ac 0376`	`23f4 9770 0a91 3089 d800 5513 58e1 6352 ...`

Table 1: Key-IV pairs that produce identical keystream bits

5 Distinguisher based on Shifted keystream bits

In [11], the authors had proven that, if the key K is unknown, then it would take around $2^{60.5}$ random IV trials with K to find an IV collision for K. In this section, we show that even if the key is secret, (as is the setting followed in a chosen-IV distinguisher) then it takes much lesser number of trials to find IVs which produce shifted keystream bits when used with the given secret key. Before we outline our algorithm, let us look to the following theorem concerning shifted keystream bits in Lizard.

Theorem 1. *Let p be an integer greater than zero. Then, for every 120-bit secret key K in* Lizard*,*

1. *There exists around 2^6* IV Collisions *on average,*
2. *There exists around 2^7 IV pairs (IV_0, IV_1) on average, such that the key-IV pairs K, IV_0 and K, IV_1 produce exactly p-bit shifted keystream sequences.*

Proof. The proof is by construction. Let us define $G : \mathbb{F}_2^{121} \to \mathbb{F}_2^{121}$ to be the function that maps the input of Phase 4 of Lizard to its output (note that G^{-1} can be computed efficiently by iterating the Algorithm $P4^{-1}$ a total of 128 times). Consider the following subroutine:

Input: A 121 bit string U, a 120-bit key K, Output: The values 0/1/2.
Subroutine $\theta(U, K)$

```
1. Compute $\hat{U} = K \oplus G^{-1}(U)$.
2. If $\hat{U}[120] = 0$ then abort and return 0.

3. Compute $U'_0 = F^{-1}(\hat{U}[0 \text{ to } 119] \mid\mid 0)$
4. Compute $U'_1 = F^{-1}(\hat{U}[0 \text{ to } 119] \mid\mid 1)$

5. Set $r \leftarrow 0$.

6. If $U'_0[64 \text{ to } 120] = K[64 \text{ to } 118] \mid\mid K[119] \oplus 1 \mid\mid 1$, increment $r \leftarrow r + 1$.
7. If $U'_1[64 \text{ to } 120] = K[64 \text{ to } 118] \mid\mid K[119] \oplus 1 \mid\mid 1$, increment $r \leftarrow r + 1$.

8. Return $r$.
```

The above subroutine θ takes as input a 121 bit string U, a 120-bit key K and finds if the string U is a valid internal state at the beginning of the keystream generation phase (i.e. end of Phase 4) when used with the Key K. In other words it finds out if there exists an IV such that K, IV leads to the internal state U after the four phases of initialization. The subroutine first peels off the effect of Phase 4 and 3 by applying G^{-1} and adding K to obtain $\hat{U}$. Since Phase 3 of the forward initialization process sets the last bit to 1, the last bit of a valid initialization must result in $\hat{U}[120] = 1$, failing which the subroutine returns 0. After this, the algorithm runs F^{-1} on both $\hat{U}[0 \text{ to } 119] \mid\mid 0$ and $\hat{U}[0 \text{ to } 119] \mid\mid 0$ to get U'_0 and U'_1 respectively (since Phase 3 sets the last bit automatically to 1, both $\hat{U}[0 \text{ to } 119] \mid\mid 0$ and $\hat{U}[0 \text{ to } 119] \mid\mid 0$ are candidates for valid states at this point). The last 57 bits of either U'_0 and U'_1 has to be equal to $K[64 \text{ to } 118] \mid\mid K[119] \oplus 1 \mid\mid 1$ for a valid initialization, and the subroutine returns 2 if both the conditions in lines 6, 7 are met, and 1 if only one condition is met. Otherwise the subroutine returns 0. Note that we do not need to impose a similar condition on the first 64 bits, since these are supposed to be the bitwise xor sum of the key and the IV. So whenever either one or both conditions in Lines 6 or 7 are satisfied, $U'_i[0 \text{ to } 63] \oplus K[0 \text{ to } 63]$ (for $i = 0$ or 1 or both), gives us the value of the IV, that along with K leads to the state U after Phase 4.

One can use the above subroutine to estimate the number of IV Collisions for a single key K. It is given as the number of times $\theta(U, K)$ returns 2, when U is iterated over all the possible 2^{121} values. Note that for the subroutine to return 2, a total of 115 bit conditions need to be satisfied, one in Line 2 and 57 each in Lines 6, 7. Assuming that these bit conditions are satisfied according to *i.i.d* uniform distributions, the total number of times the subroutine returns 2 can be estimated as $2^{121-115} = 2^6$. Thus on average, for each K there exist 2^6 IV pairs that collide.

We can also use the algorithm, to estimate the number of IV pairs that result in exactly p-bit shifted keystream sequences (for $p > 0$). Let $g : \mathbb{F}_2^{121} \to \mathbb{F}_2^{121}$ that maps the transition resulting from one clock cycle in Phase 4 (which is also the state update during the keystream generation phase). Note that therefore $G = g^{128}$. To estimate the number of such pairs we need to find the number of times $\theta(U, K)$ and $\theta(g^p(U), K)$ both return non-zero values. The probability that $\theta(U, K)$ gives a non zero value is given as (we denote by A the event the condition in Line 2 is satisfied and the routine returns 0, B by the event when the condition in Line 6 is satisfied and C by the even when the condition in Line 7 is satisfied)

$$\begin{aligned}
\mathsf{Pr}[\theta(U,K) = 2] &= \mathsf{Pr}[\theta(U,K) = 2 \mid A] \cdot \mathsf{Pr}[A] + \mathsf{Pr}[\theta(U,K) = 2 \mid A^c] \cdot \mathsf{Pr}[A^c] \\
&= 0 \cdot \frac{1}{2} + \mathsf{Pr}[B \vee C \mid A^c] \cdot \frac{1}{2} \\
&= \frac{1}{2} \cdot (\mathsf{Pr}[B \mid A^c] + \mathsf{Pr}[C \mid A^c]) \\
&= \frac{1}{2} \cdot (2^{-57} + 2^{-57}) = 2^{-57}
\end{aligned}$$

Assuming that $\theta(U, K)$ and $\theta(g^p(U), K)$ are identically and uniformly distributed, the probability that both return non-zero is $2^{-2\cdot 57} = 2^{-114}$, and so the number of IV pairs that result in p-bit shifted keystream sequences, for a given K, is $2^{121-114} = 2^7$ on average. $\square$

The authors of Lizard recommend that a single Key-IV pair be used to generate not more than 2^{18} keystream bits. For any fixed K, imagine the space of Initial Vectors as an undirected Graph $G = (W, E)$, where $W = \{0,1\}^{64}$ is the Vertex set which contains all the possible 64-bit IVs as nodes. An edge $(IV_1, IV_2) \in E$ if and only if (K, IV_1) and (K, IV_2) produce either an IV collision or p-bit shifted keystream sequence (for $1 \le p \le 2^{18} - 80$). From the above discussion, it is clear that the cardinality of edge-set E is expected to be $(2^{18} - 80)\cdot 2^7 + 2^6 \approx 2^{25}$. So we can formulate a distinguisher as follows

1. Generate 2^{18} keystream bits $[z_0,\ z_1, \ldots, z_{2^18-1}]$ for the unknown Key K and some randomly generated Initial Vector IV.

2. For $i = 0$ to $2^{18} - 80$
   - Store $[z_i, z_{i+1}, \ldots, z_{i+79}]$ in a Hash table along with the IV that generated it.

3. Continue the above steps with more randomly generated IVs till we obtain two Initial Vectors for K that generate either IV Collision or p-bit shifted keystream for some (for $1 \le p \le 2^{18} - 80$).

The question now remains how many random Initial Vectors do we need to try before we get a match. When we run the Distinguisher algorithm for N different Initial Vectors, we effectively add $\binom{N}{2}$ edges to the coverage and a match occurs when one of these edges is actually a member of the Edge-set E. Since there are potentially $\binom{2^{64}}{2}$ edges in the IV space, by the Birthday bound, a match will occur when the product of $\binom{N}{2}$ and the cardinality of E which is around 2^{25} is equal to $\binom{2^{64}}{2}$. From this equation solving for N, we get $N \approx 2^{51.5}$. This gives a bound for the time and memory complexity of the Distinguisher. The time complexity is around $2^{51.5}$ encryptions with different IVs, and the memory required is $2^{51.5} \cdot (2^{18} - 80) \cdot 144 \approx 2^{76.6}$ bits.

Decreasing the Memory Complexity: We can obtain better bounds on memory if we restrict the range of p. Suppose the distinguisher uses an upper bound P. In that case, cardinality of E is around $P \cdot 2^7$. The equation we need to solve to get N becomes

$$\binom{N}{2} \cdot P \cdot 2^7 = \binom{2^{64}}{2} \ \Rightarrow N \approx \sqrt{\frac{2^{121}}{P}}$$

Thus the time complexity is $\sqrt{\frac{2^{121}}{P}}$ encryptions and memory required is $\sqrt{\frac{2^{121}}{P}} \cdot (P - 80) \cdot 144$ bits. For example for $P = 2^{11}$, the time complexity is 2^{55} and memory complexity is $2^{73.1}$ bits. Note that when $P = 0$, i.e. when we only consider IV collisions, this reduces to the chosen-IV distinguisher already mentioned by the authors of Lizard in [11].

6 Impossible Collision attack

In this Section, we present an attack on round reduced Lizard stream cipher in which Phase 2 is reduced to 95 (out of 128) rounds, and Phase 4 is run for the full 128 rounds. The attack is similar to Impossible Differential attacks in the context of block ciphers. In impossible differential attack on a block cipher, the attacker uses an input and output differential which never occurs in the plaintext-ciphertext pairs produced by the cipher. If the impossible differential characteristic involves only a fraction of the keybits, the attacker can discard all those candidate keys that result in the characteristic, and hence reduce the size of the keyspace. An impossible collision attack follows roughly the same idea. From Theorem 1, we know that for any key K, there exists on average 2^6 pairs of IVs that produce identical keystream bits. This should also hold in the round reduced version of Lizard in which Phase 2 is reduced to 95 rounds. The attacker first exhausts the entire codebook of the 64-bits IVs to obtain 2^{64} sets of keystream sequences generated by the secret key and each of the IVs. This, therefore, takes time equivalent to 2^{64} encryptions. On average, he is expected to find 2^6 pairs of IVs that generate identical keystream bits.

Let $IV_0 = [v_0, v_1, v_2, \ldots, v_{63}]$ and $IV_1 = [\hat{v}_0, \hat{v}_1, \hat{v}_2, \ldots, \hat{v}_{63}]$ be one of the IV-pairs that result in a IV collision for the given secret key $K = [k_0, k_1, k_2, \ldots, k_{119}]$ in round reduced Lizard. Then we know that after 95 rounds of Phase 2, the key-IV pairs K, IV_0 and K, IV_1 will lead respectively to the internal states S_{95} and $\hat{S}_{95}$, which would differ only in the 120th bit. Using a computer algebra software like SAGE [18], we can compute the algebraic expression for $S_{95}[0]$, i.e. the 0th bits of S_{95}. It is given as :

$$\begin{aligned} S_{95}[0] = \bigoplus_{i\in A} x_i \oplus \quad & x_6 \cdot x_{24} \cdot x_{32} \cdot x_{48} \cdot x_{62} \cdot x_{71} \cdot x_{83} \oplus x_7 \cdot x_{33} \cdot x_{46} \cdot x_{70} \oplus x_8 \cdot x_{64} \oplus x_9 \cdot x_{26} \oplus \\ & x_{11} \cdot x_{19} \cdot x_{31} \cdot x_{37} \cdot x_{52} \cdot x_{66} \oplus x_{13} \cdot x_{87} \oplus x_{14} \cdot x_{57} \oplus x_{15} \cdot x_{17} \oplus x_{18} \cdot x_{34} \cdot x_{55} \cdot x_{69} \cdot x_{80} \oplus \\ & x_{20} \cdot x_{21} \oplus x_{23} \cdot x_{42} \oplus x_{25} \cdot x_{27} \cdot x_{28} \oplus x_{30} \cdot x_{58} \oplus x_{39} \cdot x_{72} \cdot x_{78} \oplus x_{40} \cdot x_{47} \oplus x_{43} \cdot x_{68} \cdot x_{96} \cdot x_{119} \oplus \\ & x_{49} \cdot x_{81} \oplus x_{53} \cdot x_{104} \cdot x_{108} \oplus x_{60} \cdot x_{63} \oplus x_{65} \cdot x_{79} \oplus x_{67} \cdot x_{73} \cdot x_{77} \oplus x_{82} \cdot x_{85} \cdot x_{86} \cdot x_{88} \oplus x_{98} \cdot x_{111} \end{aligned}$$

where $A = \{5, 10, 12, 16, 29, 35, 45, 50, 54, 59, 76, 84, 89, 95, 118\}$ and the x_i's are defined as:

$$x_i = \begin{cases} k_i \oplus v_i, & \text{for } i \in \{0, 1, 2, \ldots, 63\} \\ k_i, & \text{for } i \in \{64, 65, 66, \ldots, 118\} \\ k_i \oplus 1, & \text{for } i = 119 \end{cases}$$

The expression consists of 38 monomials and involves 83 bits of the secret key and 50 bits of the IV. Let us now look at the algebraic expression for $S_{95}[0] \oplus \hat{S}_{95}[0]$.

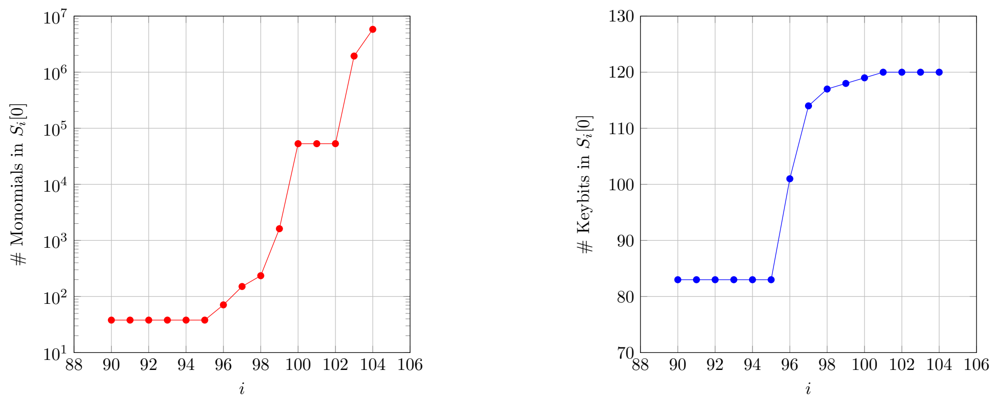
$$\begin{aligned} S_{95}[0] \oplus \hat{S}_{95}[0] = & \bigoplus_{i\in B} (v_i \oplus \hat{v}_i) \oplus (x_6 \cdot x_{24} \cdot x_{32} \cdot x_{48} \cdot x_{62} \oplus \hat{x}_6 \cdot \hat{x}_{24} \cdot \hat{x}_{32} \cdot \hat{x}_{48} \cdot \hat{x}_{62}) * x_{71} \cdot x_{83} \oplus \\ & (x_7 \cdot x_{33} \cdot x_{46} \oplus \hat{x}_7 \cdot \hat{x}_{33} \cdot \hat{x}_{46}) * x_{70} \;\oplus\; (v_8 \oplus \hat{v}_8) * x_{64} \;\oplus\; (x_9 \cdot x_{26} \oplus \hat{x}_9 \cdot \hat{x}_{26}) \oplus \\ & (x_{11} \cdot x_{19} \cdot x_{31} \cdot x_{37} \cdot x_{52} \oplus \hat{x}_{11} \cdot \hat{x}_{19} \cdot \hat{x}_{31} \cdot \hat{x}_{37} \cdot \hat{x}_{52}) * x_{66} \;\oplus\; (v_{13} \oplus \hat{v}_{13}) * x_{87} \oplus \\ & (x_{14} \cdot x_{57} \oplus \hat{x}_{14} \cdot \hat{x}_{57}) \;\oplus\; (x_{15} \cdot x_{17} \oplus \hat{x}_{15} \cdot \hat{x}_{17}) \;\oplus\; (x_{18} \cdot x_{34} \cdot x_{55} \oplus \hat{x}_{18} \cdot \hat{x}_{34} \cdot \hat{x}_{55}) * x_{69} \cdot x_{80} \oplus \\ & (x_{20} \cdot x_{21} \oplus \hat{x}_{20} \cdot \hat{x}_{21}) \;\oplus\; (x_{23} \cdot x_{42} \oplus \hat{x}_{23} \cdot \hat{x}_{42}) \;\oplus\; (x_{25} \cdot x_{27} \cdot x_{28} \oplus \hat{x}_{25} \cdot \hat{x}_{27} \cdot \hat{x}_{28}) \oplus \\ & (x_{30} \cdot x_{58} \oplus \hat{x}_{30} \cdot \hat{x}_{58}) \;\oplus\; (v_{39} \oplus \hat{v}_{39}) * x_{72} \cdot x_{78} \;\oplus\; (x_{40} \cdot x_{47} \oplus \hat{x}_{40} \cdot \hat{x}_{47}) \;\oplus \\ & (v_{43} \oplus \hat{v}_{43}) * x_{68} \cdot x_{96} \cdot x_{119} \;\oplus\; (v_{49} \oplus \hat{v}_{49}) * x_{81} \;\oplus\; (v_{53} \oplus \hat{v}_{53}) * x_{104} \cdot x_{108} \oplus \\ & (x_{60} \cdot x_{63} \oplus \hat{x}_{60} \cdot \hat{x}_{63}). \end{aligned}$$

where $B = \{5, 10, 12, 16, 29, 35, 45, 50, 54, 59\}$ and the $\hat{x}_i := k_i \oplus \hat{v}_i$ for $i \in \{0, 1, 2, \ldots, 63\}$. We can see from the above equation, that $S_{95}[0] \oplus \hat{S}_{95}[0]$ is a function of 51 keybits only. This gives us an opportunity to reduce the keyspace. We start with any colliding pair of IVs. We know that for the correct key K, the first 120 bits of S_{95} and $\hat{S}_{95}$ are equal. In particular, we concentrate our attention on $\delta := S_{95}[0] \oplus \hat{S}_{95}[0]$. For the correct guess of key, δ must be zero. So if any candidate key results in $\delta = 1$, we can immediately discard it, since collision is impossible for this candidate key. Hence the name Impossible Collision. Moreover, δ depends on only 51 key bits, so we have the added advantage of searching over only a limited keyspace. Our algorithm is as follows:

Impossible Collision Attack

1. Given around 2^6 colliding pair of IVs.
2. For each guess of the 51-bit key
   - Compute $\delta = S_{95}[0] \oplus \hat{S}_{95}[0]$ for the next colliding IV pair.
   - If $\delta = 1$, reject the key and start with another key guess **else** go to the previous step and try out another colliding IV pair.

So for each guess of the key, we compute δ for each of the 64 colliding IV pairs, and reject immediately if



(A) (B)

Fig. 2: Plot of (A) # Monomials, (B) # Keybits in $S_i[0]$

$\delta = 1$ for any pair. The correct key guess will give $\delta = 0$ for all colliding pairs, whereas any incorrect keybit survives all the 64 filters with a probability of 2^{-64}. And since the keyspace we are searching in has only 2^{51} candidates, it is very likely tat any incorrect guess gets rejected in the process.

Note that it may be possible, that for certain values of IV_0, IV_1, δ is identically 0, which makes these IV pairs unusable for key filtering. This happens when the difference between IV_0, IV_1 is zero in all the 41 bit locations that nonlinearly affect the expression for δ. Assuming these variables follow *i.i.d* uniform distributions, this event is likely to occur with a very low probability 2^{-41}. The probability that it happens for more than three colliding pairs is less than 2^{-120}. So we are always likely to have enough colliding pairs to perform the attack. It is also very difficult to extend the attack to more number of rounds because the algebraic complexity of $S_i[0]$ both in terms of the number of monomials and the number of keybits involved rises very quickly after $i = 95$, as is shown graphically in Figure 2. For $i = 96$, $S_i[0]$ is a function of 101 keybits and so any attack under 2^{80} computations seems infeasible.

6.1 Complexity of the attack

We begin with 2^{64} encryptions with all the possible IVs to find all the colliding pairs. The filtering algorithm for 2^{51} keys takes at most 2^6 computations of delta for each key guess and so for this part of the algorithm the complexity is bounded by 2^{57} calculations of δ. We need to do a brute force search over the remaining 69 keybits which would take another 2^{69} encryptions. The total complexity is the sum of the above terms and so is dominated by the 2^{69} term.

7 Conclusion

In this paper we present a study of the stream cipher Lizard. In the first part we show that it is possible, with some effort, to find distinct key-IV pairs that produce identical keystream bits. Thereafter we construct a distinguisher for Lizard based on IVs that produce shifted keystream sequences. Finally we propose a key recovery attack on Lizard with 223 initialization rounds. The attack is similar to impossible differential attacks on block ciphers, and makes use of sparse key-IV mixing upto 95 rounds of the Phase 2 initialization in the cipher.

References

1. The ECRYPT Stream Cipher Project. eSTREAM Portfolio of Stream Ciphers. Revised on September 8, 2008.
2. F. Armknecht and V. Mikhalev. On Lightweight Stream Ciphers with Shorter Internal States. To appear in FSE 2015. Preprint available at `http://eprint.iacr.org/2015/131.pdf`.

3. M. Ågren, M. Hell, T. Johansson and W. Meier. A New Version of Grain-128 with Authentication. Symmetric Key Encryption Workshop 2011, DTU, Denmark, February 2011.
4. S. Babbage and M. Dodd. The stream cipher MICKEY 2.0. ECRYPT Stream Cipher Project Report. Available at `http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf`.
5. S. Banik. Some Results on Sprout. In Indocrypt 2015, LNCS, Vol. 9462, pp. 124–129, 2015.
6. A. Biryukov, A. Shamir Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In Asiacrypt 2000, LNCS, Vol. 1976, pp. 1–13, 2000.
7. C. De Cannière and B. Preneel. TRIVIUM -Specifications. ECRYPT Stream Cipher Project Report. Available at `http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf`.
8. M. F. Esgin and O. Kara. Practical Cryptanalysis of Full Sprout with TMD Tradeoff Attacks. In Selected Areas in Cryptography, LNCS, Vol. 9566, pp. 67-85, 2015.
9. H. Fredricksen. A survey of full length nonlinear shift register cycle algorithms, SIAM Rev., 24 (1982), pp. 195–221, 1982.
10. S. W. Golomb. Shift Register Sequences. Holden-Day, Inc., Laguna Hills, CA, USA, 1967.
11. M. Hamann, M. Krause and W. Meier. LIZARD - A Lightweight Stream Cipher for Power-constrained Devices. In IACR Transactions of Symmetric Cryptology. Volume 2017, Issue 1, pp. 45-79.
12. V. Lallemand and M. Naya-Plasencia. Cryptanalysis of Full Sprout. In CRYPTO 2015, LNCS, Vol. 9215, pp. 663–682, 2015.
13. M. Hell, T. Johansson and W. Meier. Grain - A Stream Cipher for Constrained Environments. ECRYPT Stream Cipher Project Report 2005/001, 2005. Available at `http://www.ecrypt.eu.org/stream`.
14. M. Hell, T. Johansson and W. Meier. A Stream Cipher Proposal: Grain-128. In IEEE International Symposium on Information Theory (ISIT 2006), 2006.
15. V. Mikhalev, F. Armknecht, and C. Müller. On Ciphers that Continuously Access the Non-Volatile Key. In IACR Transactions of Symmetric Cryptology. Volume 2016, Issue 2, pp. 52-79.
16. S. Sarkar, S. Banik and S. Maitra. Differential Fault Attack against Grain family with very few faults and minimal assumptions. Available at `http://eprint.iacr.org/2013/494.pdf`.
17. M. Soos. CryptoMiniSat-2.9.5. `http://www.msoos.org/cryptominisat2/`.
18. W. Stein. Sage Mathematics Software. Free Software Foundation, Inc., 2009. Available at `http://www.sagemath.org`. (Open source project initiated by W. Stein and contributed by many).
19. B. Zhang, X. Gong. Another Tradeoff Attack on Sprout-Like Stream Ciphers. In Asiacrypt 2015, LNCS, Vol. 9453, pp. 561–585, 2015.