RELATED-KEY DIFFERENTIAL SLIDE ATTACK AGAINST FOUNTAIN V1

Raluca POSTEUCA*

* COSIC, KU Leuven, Belgium
E-mail: raluca.posteuca@esat.kuleuven.be

Abstract. The stream cipher FOUNTAIN was introduced in April 2019 as one of the candidates in the NIST lightweight crypto standardization process. In this paper we introduce a slide attack that leads to the construction of 32 relations on key bits, with time complexity around 17×2^{80}. The success of the attack is around 98%. We also present some properties of the internal state transitions that allow the identification of (key-iv-ad) input data that produce identical ciphertexts, with probability of 2^{-32}.

Key words: lightweight cryptography, Fountain, slide attacks, internal states collisions, invertible states transition.

1. INTRODUCTION

Lightweight cryptography represents nowadays a very popular research area due to the necessity of designing and implementing efficient and secure cryptographic primitives for devices with constrained resources. During the last years numerous primitives were designed, having improved properties regarding the suitability for software and hardware implementation, performance and efficiency.

Some of the proposals became part of European or international standards, such as ISO/IEC standards. One of the standards relevant for lightweight cryptography is the 29192 standard, dedicated to lightweight cryptography, including the block ciphers PRESENT[3] and CLEFIA[14] and the stream ciphers Trivium[5] and Enocoro[7]. The 18033 standard describes encryption algorithms, some of them being lightweight, such as MISTY1[11], SNOW 2.0 [12] and Rabbit [10].

The European Network of Excellence for Cryptology (ECRYPT) funded a 4-year project, eSTREAM, dedicated to the identification of new stream ciphers dedicated to software applications with high throughput (Profile I) and to hardware applications with restricted resources (Profile II). Grain 128A [9], Trivium and Mickey[13] algorithms were selected for the Profile II portfolio.

CAESAR, the Competition for Authenticated Encryption: Security, Applicability and Robustness, took place between 2014 and 2018; in the section dedicated to lightweight applications two stream ciphers were selected as winners – ACORN[8] and Ascon[6].

More details regarding the State of the Art in Lightweight Symmetric Cryptography can be found in [1].

The lightweight cryptography became even more attractive since NIST initiated a process meant to lead to the standardization of lightweight algorithms that are suitable for usage in constrained environments. The first round of this competition is on-going, having 56 submissions.

Our contribution. In order to contribute to the public research efforts dedicated to the analysis of NIST Round 1 candidates, we focused on one of the proposals, the Fountain stream cipher [4]. In this paper we introduce a slide attack with time complexity around 17×2^{80}. The goal of this attack is to construct a system of 32 low degree equations using the key bits, this being equivalent to recovering 32 bits of the key (by solving this equation system, the complexity of the exhaustive search is decreased from 2^{128} to 2^{96}). We have also identified some properties of the internal states transitions that allow the identification of (key, IV, AD) input data that produce identical ciphertexts, with probability of 2^{-32}.

Organization of the paper. The paper is organized as follows: the Fountain cipher is briefly described in Section 2; in Section 3 we present a set of properties of the state update function and present an attack on a

slightly modified version of the cipher Fountain, in the hypothesis that the associated data is null. The attack leads to the identification of input data which determines collisions in the internal state and, thus, to the generation of identical ciphertexts (when using the same plaintext); in Section 4 we present a manner in which the previous attack can be extended to the original version of Fountain; Section 5 presents the method in which we were able to identify (key, IV, AD) input data that produce identical ciphertexts; the last section concludes our paper.

2. DESCRIPTION OF FOUNTAIN CIPHER

2.1 Parameters

Fountain is a lightweight stream cipher with 16-byte secret key K and 12-byte initialization vector IV. The input data of the cipher is represented by a quartet (K, IV, AD, M) where AD represents the associated data, M the plaintext and K, IV are the key and the initialization vector, respectively. The length of AD can range from 0 (no associated data) to $2^{50} - 1$ bytes. The length of M can be less than or equal to 2^{61} bytes. The output of the authenticated encryption is a pair (C, T) where C represents the encryption of M and T is the authentication tag of AD and M. The length of the tag T can be 64 or 128 bits. The verification and decryption process takes as input the tuple (K, IV, AD, C, T) and outputs the plaintext M only if the tag verification succeeds. If the tag verification fails, no plaintext is returned.

2.2 Component functions

The state update function of Fountain operates on 256-bits internal states, using 4 LFSRs, 4×4 S-boxes, an MDS matrix and a Boolean (filter) function used for the generation of keystream bits.

The 4 LFSRs used in Fountain have the same length, but different feedback polynomials, as follows:

$$\text{LFSR1: } 1 + x^{12} + x^{25} + x^{31} + x^{64}$$

$$\text{LFSR2: } 1 + x^{9} + x^{19} + x^{31} + x^{64}$$

$$\text{LFSR3: } 1 + x^{14} + x^{20} + x^{31} + x^{64}$$

$$\text{LFSR4: } 1 + x^{6} + x^{10} + x^{31} + x^{64}$$

The initial values of the four LFSRs are denoted by:

$$LFSR1 = (\alpha_0, \dots, \alpha_{63})$$

$$LFSR2 = (\beta_0, \dots, \beta_{63})$$

$$LFSR3 = (\gamma_0, \dots, \gamma_{63})$$

$$LFSR4 = (\zeta_0, \dots, \zeta_{63})$$

The linear recursions are:

$$LFSR1\text{: } \alpha_{64+i} = \alpha_{31+i} + \alpha_{25+i} + \alpha_{12+i} + \alpha_i$$

$$LFSR2\text{: } \beta_{64+i} = \beta_{31+i} + \beta_{19+i} + \beta_{9+i} + \beta_i$$

$$LFSR3\text{: } \gamma_{64+i} = \gamma_{31+i} + \gamma_{20+i} + \gamma_{14+i} + \gamma_i$$

$$LFSR4\colon \zeta_{64+i} = \zeta_{31+i} + \zeta_{10+i} + \zeta_{6+i} + \zeta_i$$

The following 4×4 S-boxes are used in Fountain:

$$SR\_kg = \{9,5,6,D,8,A,7,2,E,4,C,1,F,0,B,3\};$$

$$SR\_ad = \{9,D,E,5,8,A,F,2,6,C,4,1,7,0,B,3\};$$

$$SR\_tag = \{B,F,E,8,7,A,2,D,9,3,4,C,5,0,6,1\};$$

The S-boxes are always applied to the nibbles formed by the second bit from the 4 LFSRs. More precisely, at step i, the input of the Sbox is the nibble $\zeta_{i+1}\gamma_{i+1}\beta_{i+1}\alpha_{i+1}$, where α_{i+1} is the LSB of the nibble. The output nibble and the component bits are denoted by $f^i = f_1 f_2 f_3 f_4$.

The filter function is given by

$$z_i = \alpha_{i+3} + \alpha_{i+11} + \beta_{i+20} + \gamma_{i+5} + \gamma_{i+16} + \zeta_{i+7} + \zeta_{i+29} + h(x),$$

where z_i represents the keystream bits, and h is the Boolean function

$$h(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = x_0x_1 \oplus x_2x_3 \oplus x_4x_5 \oplus x_6x_7 \oplus x_0x_4x_8,$$

where

$$(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (\zeta_{i+2},\ \alpha_{i+5},\ \beta_{i+4},\ \gamma_{i+11},\ \zeta_{i+23},\ \gamma_{i+27},\ \beta_{i+24},\ \alpha_{i+29},\ \zeta_{i+30})$$

2.3 Phases

In order to produce an authenticated ciphertext using Fountain, the following phases are performed.

The Loading Phase

Let $K = (K_0, K_1, \dots, K_{15})$ and $IV = (IV_0, IV_1, \dots, IV_{11})$ denote the key and the initialization vector, respectively and let $rev(x)$ denote the function that reverses the bits' order in the byte x (for example, the least significant bit becomes the most significant one).

The cipher's state will be initialized using the loading function $load$ defined as:

$$(LFSR1, LFSR2, LFSR3, LFSR4) = load(K, IV)$$

$$(\alpha_0, \dots, \alpha_{63}) = (rev(K_0), rev(IV_0), rev(K_1), rev(IV_1), rev(K_2), rev(IV_2), rev(K_3), rev(IV_3))$$

$$(\beta_0, \dots, \beta_{63}) = (rev(K_4), rev(IV_4), rev(K_5), rev(IV_5), rev(K_6), rev(IV_6), rev(K_7), rev(IV_7))$$

$$(\gamma_0, \dots, \gamma_{63}) = (rev(K_8), rev(IV_8), rev(K_9), rev(IV_9), rev(K_{10}), rev(IV_{10}), rev(K_{11}), rev(IV_{11}))$$

$$(\zeta_0, \dots, \zeta_{63}) = (rev(K_{12}), rev(K_{13}), 0x\text{FF}, rev(K_{14}), rev(K_{15}), 0x\text{FC}, 0x00, 0x01).$$

Initialization Phase

The initialization phase consists of 384 rounds. In each round i, the Boolean function is applied in order to compute the keystream bit z_i. Afterwards, the nibble f^i is computed by applying the Sbox SR_kg. In the end, each of the 4 LFSRs are clocked one step, with $z_i \oplus f_j$ being feedback to the j[th] LFSR, $j = \overline{1,4}$. Note that the keystream bit is used to update the internal state.

Associated Data Processing

The associated data processing consists of $adLen$ rounds, where $adLen$ denotes the number of bits in AD. In each round i, the Boolean function is applied in order to compute the keystream bit z_{i+384}. Afterwards, the nibble f^i is computed by applying the Sbox SR_ad. In the end, each of the 4 LFSRs are clocked one step, with $z_{i+384} \oplus f_j \oplus ad_i$ being feedback to the j[th] LFSR, $j = \overline{1,4}$. Note that during this phase the keystream bit xored with the AD bit is used to update the internal state.

Middle Separation Phase

After the associated data processing, the middle separation phase represents the application of 64 rounds, consisting of the same operations as during the initialization phase (again using the SR_kg S-box).

Plaintext processing

Before starting processing the plaintext, the first bit of the second LFSR is xored with 1:

$$\beta_{448+adLen} = \beta_{448+adLen} \oplus 1 \quad (1)$$

For each bit m_i of the plaintext, the keystream bit $z_{i+448+adLen}$ is computed; the corresponding ciphertext bit c_i is computed by xoring the keystream bit and the plaintext bit. Each of the 4 LFSRs are clocked one step, with $m_i \oplus f_j$ being feedback to the LFSRs (in this case, the keystream bit is not used to update the internal state, instead it is xored to the plaintext bit, in order to generate the keystream bit). During the rounds, the SR_kg S-box is used. This phase consists of $pLen$ rounds, where $pLen$ denotes the length of the plaintext.

Finalization and tag generation

Before starting the tag generation, the first bit of the fourth LFSR is xored with 1:

$$\zeta_{448+adLen+pLen} = \zeta_{448+adLen+pLen} \oplus 1 \quad (2)$$

For 384 rounds, the keystream bit $z_{i+448+adLen+pLen}$ is computed, and the SR_tag S-box is applied on the states bits mentioned previously; each of the 4 LFSRs are clocked one step, with $z_i \oplus f_j$ being feedback to the LFSRs (the keystream bit is again used to update the internal state).

The authentication tag T is computed by xoring the secret key K with the last 128 keystream bits. For the 64-bit tag version, the last 64 keystream bits are xored with the least significant 64 bits of the key K, in order to compute the tag.

The state update functions corresponding to each phase of the cipher can be viewed as a function parametrized by the used Sbox and by the function used to compute the feedback of the 4 LFSRs. We denote the state update function by $Round_{x,S}$. The formal description of this functions is presented in Fig.1.

Let us denote the composition of the function $Round_{x,S}$ by

$$Round_{x,S}^n = \underbrace{Round_{x,S} \circ \ldots \circ Round_{x,S}}_{n\ times}$$

$Round_{x,S}()$
Compute the keystream bit z_i
Compute $f_1 f_2 f_3 f_4 = \mathrm{S}[\xi_1 \gamma_1 \beta_1 \alpha_1]$
Run the 4 LFSRs 1 step with $z_i \oplus x \oplus f_i$ as feedback

Fig. 1 One round of the cipher, parametrized by the bit x and the Sbox S

Using this notation, the cipher's phases can be written as depicted in Table 1.

Table 1 Fountain phases as a composition of the function $Round_{x,S}$ and the associated parameters

Phase	Description
Initialization	$Round_{0,SR\_kg}^{384}$
Associated Data Processing	$Round_{ad_i,SR\_ad}^{ADlen}$
Middle Separation	$Round_{0,SR\_kg}^{64}$
Plaintext Processing	$Round_{c_i,SR\_kg}^{Plen}$
Finalization	$Round_{0,SR\_tag}^{384}$

Observations. The initialization and the middle separation phases use the same state transition function, the only difference being given by the number of iterations performed (in the initialization phase are performed 384 iterations, while in the middle separation phase are performed only 64 iterations). Moreover, the initialization, the middle separation and plaintext processing phases use the same Sbox. By getting $p_i = z_i$, where p_i is a plaintext bit and z_i is a keystream bit, and a null associated data, then the encryption process is composed by $384 + 64 + pLen$ identical steps, where $pLen$ denotes the length of the plaintext. In this case, the ciphertext will be represented by the string 0^{pLen}.

3. PROPERTIES OF THE STATE UPDATE FUNCTION

3.1 Slide-based property of $Round_{x,S}^{n}$

Slide attacks were introduced by Alex Biryukov and David Wagner in 1999 [2]. This type of attacks were proved to be successful in breaking iterative ciphers with high-self similarity, in terms of component operations, usually independently of the number of rounds. Slide attacks were applied to various families of ciphers, for example Feistel and Even-Mansour schemes.

In general, in order to achieve efficiency and to prove the security against general known cryptanalytic techniques, lightweight ciphers are using a large number of similar rounds. This approach may lead to potential weaknesses in the context of applying slide attacks.

Slide attacks are exploiting key schedule weaknesses or general structural properties, depending of the cipher's design. A very well-known method to prevent slide attacks is to avoid self-similarity of the iterative process (using fixed random constants or iterative counters).

A slide attack is based on the identification of a relation between two inputs that also holds for the corresponding outputs. This relation could be the composition of a fixed number of rounds, in the case of the block ciphers, or a set of state transitions, for stream ciphers. The identification of this type of relation may lead to the recovery of some secret data (key or plaintext bits).

In this section we introduce a slide-based property of the state update function $Round_{x,S}^{n}$.

Definition. Let (K_1, IV_1) and (K_2, IV_2) be two key-IV pairs. The pairs are called *n-slid pairs* if, for $n > 0$,

$$Round_{0,SR\_kg}^{n}\big(load(K_1, IV_1)\big) = load(K_2, IV_2)$$

Observation. For a pair (K_1, IV_1) and a value *n*, there exists an *n-slid pair* if and only if, after computing $Round^{n}_{0,SR_{kg}}(load(K_1, IV_1))$, the internal state satisfies the same property as a loaded state, i.e. the 4th LFSR satisfies the property:

$$\begin{aligned} \zeta_{n+16} \dots \zeta_{n+23} &= 0xFF \\ \zeta_{n+40} \dots \zeta_{n+47} &= 0xFC \\ \zeta_{n+48} \dots \zeta_{n+55} &= 0x00 \\ \zeta_{n+56} \dots \zeta_{n+63} &= 0x01 \end{aligned} \tag{3}$$

Lemma 1. The minimum value of n for which there exist *n-slid pairs* is 47.

Proof. For $n < 47$, the system of equations (3) is incompatible, i.e. the initial values of the constants directly influence the values of the bits $\zeta_{n+16} \dots \zeta_{n+23}, \zeta_{n+40} \dots \zeta_{n+63}$, making the constants pattern impossible. □

For $n = 47$, we conjecture that the system of equations corresponding to the bits $\zeta_{n+16} \dots \zeta_{n+23}, \zeta_{n+40} \dots \zeta_{n+63}$ is incompatible or that the last 47 bits of the 4th LFSR are not uniformly distributed, resulting in a very low probability of finding the correct values of the constants.

For $n = 48$, we have experimentally found *n-slid pairs;* one such example is:

$$K_1 = \{8D, 41, 3F, A4, 29, 58, 52, E9, 64, B4, E6, 1C, 68, 03, 64, D1\}$$

$$IV_1 = \{A4, D8, A2,25,0A, 0D, 11, D8,52, F8,47, B1\}$$

$$K_2 = \{A4, D2,7A, 4C, E9,5E, D8,88,1C, A1, B6,6F, 00,80,24,30\}$$

$$IV_2 = \{25,94,31, F4, D8, B3, B4, E8, B1, AF, FC, 8C\}$$

Observation. Since the pair (K_2, IV_2) is obtained after 48 rounds, then the following relations will always hold for a *48-slid pair*:

$$K_2[4 \times i] = K_1[4 \times i + 3], \forall i \in \{0,1,2\};$$

$$IV_2[4 \times i] = IV_1[4 \times i + 3], \forall i \in \{0,1,2\};$$

$$K_2[12] = 0x00; K_2[13] = 0x80;$$

Assuming that, after 48 rounds, the last 48 bits of the 4th LFSR are uniformly distributed over $\mathbb{Z}_{2^{48}}$, then the probability of finding *48-slid pairs* is 2^{-32}. We have also experimentally verified this probability.

Definition. Let f: $\mathbb{F}_q^n \to \mathbb{F}_q^n$ and g: $\mathbb{F}_q^n \to \mathbb{F}_q^n$. We say that two pairs $(a, f(a))$ and $(b, f(b))$ have a slide-based property with respect to the function g if $a = g(b)$ and $f(a) = g(f(b))$.

Lemma 2. Two *n-slid pairs* (K_1, IV_1) and (K_2, IV_2) define a slide-based property with respect to the state update function $Round^{m}_{0,SR\_kg}, \forall m > 0$.

Proof. The proof is straightforward for $g = Round^{n}_{0,SR\_kg}$, $a = load(K_2, IV_2)$ and $b = load(K_1, IV_1)$. □

3.2 Slide attack on a modified version of Fountain

In this section we present a slide attack on a modified version of the stream cipher Fountain, the attack being the starting point to the attack that will be presented in the sequel. The scope of the attack (on the modified version of the cipher) is to prove the existence of collisions of the keystream, independent of the length of the plaintexts.

Let us denote by Fountain_{wx} the cipher Fountain **without** the following xor operation:

$$\beta_{448+adLen} = \beta_{448+adLen} \oplus 1$$

In this case, two *48-slid pairs* (K_1, IV_1) and (K_2, IV_2) will generate a slide behavior as follows:

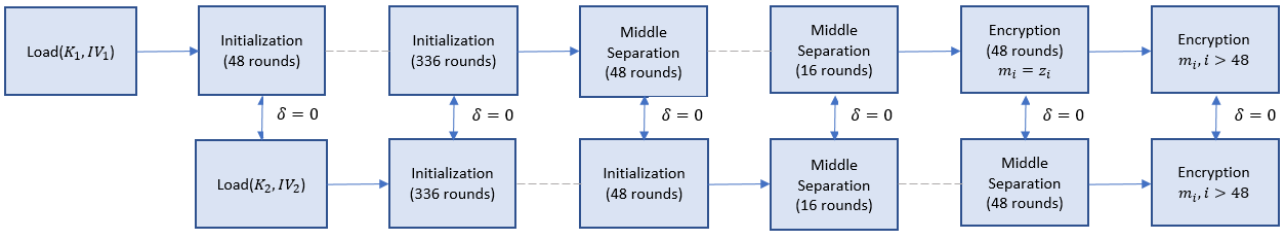

Fig. 2 Slide behavior of Fountain_{wx}

The first process will perform the encryption of the plaintext $p_1 = z_{448}, .. z_{495}, m_{49}, ... m_{pLen-1}$ using the key K_1 and the initialization vector IV_1. The second process will encrypt the plaintext $p_2 = m_{49}, ... m_{pLen-1}$, using the pair (K_2, IV_2).

Since the Initialization and the Middle Separation phases use the same parameters of the state update function, the internal states will satisfy the slide property for 448 rounds, until the first pair finishes the Middle Separation phase and begins the processing of the plaintext. Since the first 48 bits of the plaintext are the keystream bits, then, after the encryption of the first 48 bits, the two processes are perfectly synchronized (have the same internal state). Therefore, after this step, if the two processes are fed with the same plaintext, no matter the length of it, the corresponding ciphertexts will be equal.

Since the attack assumes the black-box hypothesis, an attacker does not have access to the correct values of the keystream bits $z_{448}, .. z_{495}$. Assuming that these 48 bits are uniformly distributed, the probability of correctly guessing them is 2^{-48}.

Since the complexity of finding a *48-slid pair* is 2^{-32}, the probability of finding this slide behavior of Fountain_{wx} is 2^{-80}. So, using approximately 2^{80} data of the type $(K_1, IV_1, z_{448}, .. z_{495})$, we will be able to find a collision on the ciphertexts.

4. DIFFERENTIAL SLIDE ATTACK ON FOUNTAIN

As stated before, the difference between the original description of the Fountain cipher and the cipher Fountain_{wx} is Operation (1) performed after the end of the Middle Separation phase. In this case, two *48-slid pairs* (K_1, IV_1) and (K_2, IV_2) will generate the behavior depicted in Fig. 3.

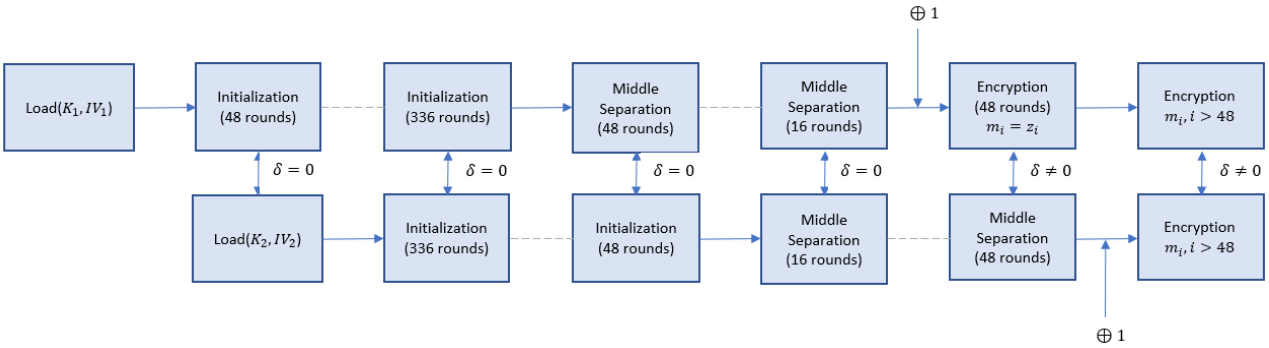

Fig. 3 Slide behavior of Fountain

Therefore, using the approach described in the section 3.2 for the original description of Fountain, the behavior of the internal states for the first 448 rounds will remain unchanged. After the first process finishes the Middle Separation phase, Operation (1) is applied, inducing an 1-bit difference between the corresponding internal states.

In order to attack the Fountain cipher, we have analyzed the propagation of the differences induced after applying Operation (1) is both processes.

4.1 Differential characteristics of Fountain

In this section we will describe the manner in which we analyzed the distribution of the differences through the cipher Fountain.

One difference can be involved either in a linear manner (in the computation of a LFSR feedback or the linear part of the keystream bit generation function) or in a non-linear manner (in the non-linear part of the keystream bit generation function or in the Sbox).

1. **Linear behavior of the differences**

   Let ℓ be a linear function and let δ be an input difference. The output difference Δ is computed by applying the linear function ℓ over the input difference:

   $$\Delta = \ell(x) \oplus \ell(x \oplus \delta) = \ell(\delta).$$

   In this case, the probability of computing Δ knowing δ is $p = 1$.

2. **Non-linear behavior of the differences through the Sbox SR_kg**

   The behavior of differences propagation through an Sbox are usually analyzed using the *Differential Distribution Table* (DDT). The DDT of SR_{kg} is presented in Table 2. For the input difference δ, the output difference is Δ, with a probability of $p = \frac{DDT[\delta,\Delta]}{16}$.

3. **Non-linear behavior of the differences through the h function**

   The function h is a Boolean function of degree 3, having an input of 9 bits:

   $$h(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = x_0x_1 \oplus x_2x_3 \oplus x_4x_5 \oplus x_6x_7 \oplus x_0x_4x_8.$$

Table 2 The DDT of SR_kg

$\delta\backslash\Delta$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	0	0	2	0	0	2	0	2	2	2	2	0	2
2	0	0	2	2	2	2	0	0	4	0	0	0	0	0	0	4
3	0	0	0	2	2	0	0	0	2	0	2	2	2	2	0	2
4	0	6	2	0	2	0	0	2	0	0	0	0	0	0	0	4
5	0	0	0	2	2	0	0	0	0	0	4	2	0	2	2	2
6	0	2	0	2	0	2	0	6	0	0	0	0	0	0	4	0
7	0	0	2	0	0	2	0	0	0	0	0	2	4	2	2	2
8	0	4	0	0	0	0	0	4	0	0	4	0	4	0	0	0
9	0	2	0	0	2	2	0	2	2	2	0	2	0	2	0	0
10	0	0	2	2	2	2	0	0	4	4	0	0	0	0	0	0
11	0	2	2	2	0	0	0	2	2	2	0	2	0	2	0	0
12	0	0	0	2	0	2	4	0	0	0	0	2	0	2	4	0
13	0	0	0	0	2	2	4	0	0	2	2	0	2	0	2	0
14	0	0	2	0	2	0	4	0	0	4	0	2	0	2	0	0
15	0	0	2	2	0	0	4	0	0	2	2	0	2	0	2	0

Let $\delta = (\delta_0, \dots, \delta_8)$ be the input difference of h. The value of the output difference is computed as follows:

$$\Delta = h(\{x_i\}_i) \oplus h(\{x_i \oplus \delta_i\}_i)$$

Therefore, for a fixed input difference, the output difference Δ can be described as a Boolean function f_δ of the input bits $\{x_i\}_i$, the degree of this function being at most 2. Moreover, by observing the value of Δ, the attacker can learn a relation between the input bits $\{x_i\}_i$. In some cases, the attacker learns a linear combination of the input bits or, even more, the exact value of one input bit. The probability of having an output difference $\Delta = 0$ is equal to the probability that the equation $f_\delta(\{x_{i\}_i}) = 0$.

For example, let's assume that only one $\delta_i = 1$:

- if $i \in \{1, 3, 5, 7\}$, then $\Delta = x_{i-1}$; $P(\Delta = 0) = P(x_{i-1} = 0) = 0.5$;
- if $i \in \{2, 6\}$, then $\Delta = x_{i+1}$; $P(\Delta = 0) = P(x_{i+1} = 0) = 0.5$;
- if $i \in \{0, 4\}$, then $\Delta = x_{i+1} \oplus x_{i-4}x_{i+4}$; $P(\Delta = 0) = P(x_{i+1} \oplus x_{i-4}x_{i+4} = 0) = 0.5$;
- if $i = 8$, then $\Delta = x_0x_4$; $P(\Delta = 0) = P(x_0x_4 = 0) = 0.75$;

From the analysis of the difference propagation of Fountain, we remark the following:

The initial difference pattern on the internal state, induced by applying (1) is described in Fig. 4. We denote the 1 difference by "X" and the 0 difference by "-".

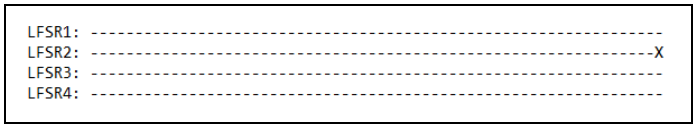

Fig. 4 Initial difference pattern on the 4 LFSRs

For 33 rounds, the difference will not be involved in any operation. At Round 34, the difference will be involved, in a linear manner, in the computation of the feedback of the 2nd LFSR, resulting in the state difference depicted in Fig. 5.

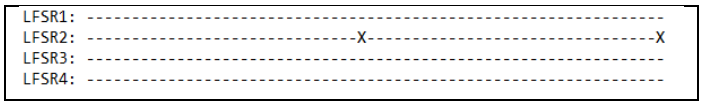

Fig. 5 The state difference pattern at round 34

At the 41st round, the difference will be involved in the computation of the $\boldsymbol{h}$ function. The two possible output differences after 41 rounds are depicted in Fig. 6. The actual difference pattern will be fixed by the value of the 20th position of the second LFSR. If that value is 0, then the difference pattern will be the first one, thus there will only be two differences between the internal states.

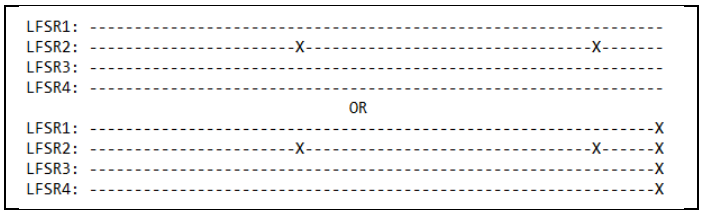

Fig. 6 The state difference patterns at round 41

After performing 45 rounds, in both cases, one difference is involved in a linear manner in the computation of the keystream bit, thus, with probability 1, the last bits of every LFSR will contain a nonzero difference. In order to better control the difference propagation through the following rounds and since this property will be satisfied with probability 1, we will cancel these differences by "guessing" the value of $z_{45} \oplus 1$ instead of the value of z_{45}. The same property will appear at step 90. In this case, we will cancel the difference by choosing two plaintexts with a difference in the 42nd position.

The difference will influence the application of the Sbox for the first time at the 63rd round, the input difference of the Sbox being $\delta = 2$. After this step, there will be 24 possible output difference patterns, having different appearance probabilities.

Observation. After computing one output difference pattern after *n* rounds, we have also generated the "observable" difference pattern, i.e. the keystream difference pattern used in the common part of the ciphertexts. The number of such patterns is smaller than the number of state difference patterns since not all the differences in the state are involved in the computation of the keystream bit difference.

After computing all possible output differences patterns for 102 steps, we obtain 2095680 difference patterns on the internal state and 83200 difference pattern on the observable part of the keystream. We store the 83200 difference patterns in a sorted manner. Since one entrance in this list has 54 bits, the memory needed for this is approximately 4.4 MB.

We have experimentally verified that, for the *48-slid pairs* that we found, the difference patterns obtained after 102 rounds belong to this list.

4.2 Differential slide attack on Fountain

Using the observations and properties described in the previous section, we have designed a related-key differential slide attack on the Fountain cipher.

The data needed for this attack is represented by 2^{80} random IV_1^is and arbitrary 102-bit plaintexts p. The first 48 bits of p are used as the keystream bits $z_{448} \dots z_{495}$.

According to the previous section, after performing 45 rounds, the last bit of every LFSR will contain a nonzero difference, with probability 1. In order to cancel these differences, the bit $p[44]$ should introduce a difference in the internal states, so this value should be different from the keystream bit used in the second process, i.e. $p[44] = z_{492} \oplus 1$. If the first 48 bits of p are correctly guessed, then the first 48 bits of the ciphertext will satisfy the following conditions:

$$c_1[i] = 0, \forall i \neq 45, i < 48 \text{ and } c_{1[44]} = 1. \quad (4)$$

The same behavior of differences will also appear after performing 90 rounds. In order to cancel these differences, we will introduce a difference in the second plaintext in the 42nd position. More precisely, the second plaintext is defined as

$$p_2[i] = p[i + 48], \forall\, i \in \{0, 53\} \setminus \{41\}.$$

The hypothesis of this attack is that the attacker has access to an encryption oracle. The oracle will answer two types of challenges:

- Given an initialization vector IV and an 102-bit plaintext p, the oracle will return the associated ciphertext c, under the secret key K and the input IV;
- Given an initialization vector IV_1 and a 54-bit plaintext p, the oracle will do the following computations
  - compute $s = Round_{0,SR\_kg}^{48}\big(load(K, IV_1)\big)$
  - extract (K_2^i, IV_2^i) from s
  - return the encryption of p under the input pair (K_2^i, IV_2^i)

The attack works as follows. The attacker generates the list of all possible ciphertext differences, as described in the previous section. Then, he randomly generates 2^{80} initialization vectors IV_1^i and asks for the encryption of random 102-bit plaintext p. If the first 48 bits of the ciphertext satisfy (4), then he asks for the encryption of the corresponding p_2. He computes the difference pattern $\{t_i = c_1[i] \oplus c_2[i + 48]\}_i$ and checks if the pattern is contained in the precomputed list of differences. If the constraint holds, then the slide property holds.

The pseudocode of the attack is presented in Fig. 7.

The data complexity of the attack is 2^{80}, while the time complexity is around 17×2^{80}. The difference between the two complexities is explained by the time complexity of the search in the precomputed sorted list. Since the list contains $83200 < 2^{17}$ and the search in a sorted list can be performed in logarithmic time, the time complexity of the search is around 17.

For $i = 0$ to $2^{80} - 1$
  Randomly generate an initialization vector IV_1^i and the arbitrary plaintexts p, p_2
  Ask for the encryption $c_1 = \text{Fountain}(K, IV_1^i, p)$
  If $c_1[i] = 0, \forall i \neq 45, i < 48$ and $c_{1[44]} = 1$
    Ask for the encryption $c_2 = \text{Fountain}(K_2^i, IV_2^i, p_2)$
    Compute $\{t_i = c_1[i] \oplus c_2[i + 48]\}_i$
    If $\{t_i\}_i \in precomputed\ list$
      Return IV_1^i;

Key-recovery
The pairs (K, IV_1^i) and (K_2^i, IV_2^i) are *48-slid pairs* ⇒ 32 equations using the 128 bits of *K*

Fig. 7 The pseudocode of the attack on Fountain

Success probability.

In our attack, in order to filter the right value of IV_1^i, we apply two filters. The first one is based on the values of $c_1[i]$, for $i < 48$. The probability that the condition of this filter is accomplished is 2^{-48}. The second filter is based on the identification of a "good" differential pattern on $\{t_i\}_i$. The probability of finding such a difference, for a *48-slid pair* (if the values of $z_{448} \dots z_{495}$ were correctly guessed) is 1. The probability that such a difference is obtained for random pairs (K_1, IV_1) and (K_2, IV_2) is $2^{-37.66}$. Therefore, the probability that the algorithm described above outputs a IV_1^i that was generated by random input key-IV pairs is $2^{-5.66}$. Thus, the probability of success of our attack is 0.98.

5 OTHER OBSERVATIONS ON FOUNTAIN

The attack presented above is performed under the hypothesis that the associated data is null. We have analyzed the cipher also in the hypothesis that the additional data is not null. In this scenario, if the length of AD is higher than 4 bytes, then the input space of the input of the cipher (until the Middle Separation phase) is $len = 128 + 96 + adLen$, where $adLen$ defines the length of the associated data. If $adLen > 32$, then $len > 256$ (the length of the internal state length). So, mathematically, this means that there will be collisions on the internal state.

We have performed a series of experiments with the scope of finding two input pairs (K_1, IV_1, AD_1) and (K_2, IV_2, AD_2) for which the internal state is the same. Such pairs can be found, with probability 2^{-32}, using the algorithm described in Fig. 8. Our algorithm is based on the fact that the state update function of Fountain is invertible.

An example of input data pairs that lead to state collision are the following:

$K_1 = \{\text{FF}, \text{FF}, \text{FF}, \text{FF}, \text{FF}, \text{FF}, \text{FF}, \text{FF}, \text{FF}, \text{FF}, \text{FF}, \text{FF}, \text{FF}, \text{FF}, \text{FF}, \text{FF}\}$
$IV_1 = \{\text{F0}, \text{F0}, \text{F0}, \text{F0}, \text{F0}, \text{F0}, \text{F0}, \text{F0}, \text{F0}, \text{F0}, \text{F0}, \text{F0}\}$
$AD_1 = \{00,01,02,03,04,05,06\}$
$K_2 = \{\text{DA}, 7\text{F}, \text{A4}, 1\text{B}, \text{D3}, 0\text{E}, 1\text{D}, \text{EA}, 9\text{B}, \text{CC}, \text{C7}, \text{AF}, \text{E3}, 3\text{E}, 83, 11\}$
$IV_2 = \{\text{FF}, 6\text{F}, \text{A7}, 00, 57, \text{AF}, \text{EE}, \text{A0}, 94, 19, 91, \text{CC}\}$
$AD_2 = \{7\text{F}, \text{C1}, \text{A5}, 67, 27\}$

Randomly generate a fixed key K, IV and AD
Compute the internal state s obtained after the loading, initialization and associated data processing phases
For $i = 0$ to $2^{32} - 1$
    Generate at random a pair (IV_i, AD_i)
    Apply the inverse of the Associated Data Processing phase
    If the internal state is a valid loading state (the 32 constant bits are in the correct place)
        Extract and return (K_i, IV_i)

Fig. 8 The pseudocode of the algorithm for finding pairs (K_1, IV_1, AD_1) and (K_2, IV_2, AD_2) that lead to the same internal state

6 CONCLUSION AND FUTURE WORK

In this paper we introduce a slide attack on full Fountain cipher. The attack may concern question regarding the security margin of the cipher in the related-key scenario. Although the attack involves the identification of (key, IV) pairs with a particular property, concerns about the structural properties and component operations arises. We also present (key - IV - associated data) input data pairs that lead to the same ciphertexts, in the case of enciphering the same message (the xor sum of the associated tags being equal to the xor sum of the initial keys).

The work presented can be extended in different directions. For example, it remains to be investigated if and how the attack presented in this paper can be improved in term of both data and time complexity. It will also be interesting to identify an attack scenario using the property regarding the internal state collison. Further research should also consider the analyze of Fountain in the single-key scenario.

ACKNOWLEDGEMENT

The author would like to thank Vincent Rijmen, Tomer Ashur and all others for all the fruitful discussions and ideas regarding the cryptanalysis of Fountain.

REFERENCES

1. A. BIRYUKOV and L. PERRIN, *State of the Art in Lightweight Symmetric Cryptography*, https://eprint.iacr.org/2017/511.pdf, 2017.
2. A. BIRYUKOV and D. WAGNER, *Slide Attacks*, Proceeding of FSE'99, LNCS 1636, pp.245-259, Springer Verlag, 1999.
3. A. BOGDANOV, L. R. KNUDSEN, G. LEANDER, C. PAAR, A. POSCHMANN, M. J. B. ROBSHAW, Y. SEURIN, and C. VIKKELSOE. *PRESENT: An ultra-lightweight block cipher*. In Cryptographic Hardware and Embedded Systems – CHES 2007, volume **4727** of Lecture Notes in Computer Science, pages 450–466. Springer, Heidelberg, September 2007.
4. B. ZHANG, *Fountain: A Lighweight Authenticated Cipher (v1)*, NIST Information Technology Laboratory, CSRC, Lightweight Cryptography, Round 1 Candidates, https://csrc.nist.gov/projects/lightweight-cryptography/round-1-candidates, June 2019.
5. C. DE CANNIÈRE. *Trivium: A stream cipher construction inspired byblock cipher design principles*. In ISC 2006: 9th International Conference on Information Security, volume **4176** of Lecture Notes in Computer Science, pages 171–186. Springer, Heidelberg, August / September 2006.
6. C. DOBRAUNIG, M. EICHLSEDER, F. MENDEL, and M SCHLÄFFER. *Ascon v1.2*.Candidate for the CAESAR Competition. See also http://ascon.iaik.tugraz.at/, 2016.
7. D. WATANABE, K. IDEGUCHI, J. KITAHAR, K. MUTO, H. FURUICHI, and T. KANEKO. *Enocoro-80: A hardware oriented stream cipher*. In The Third International Conference on Availability, Reliability and Security— ARES 08, pages 1294–1300, 2008.
8. H. WU. *ACORN: A lighweight authenticated cipher (v3)*. Candidate for the CAESAR Competition. See also https://competitions.cr.yp.to/round3/acornv3.pdf, 2016.

9. M. ÅGREN, M. HELL, T. JOHANSSON, and W. MEIER. *Grain-128a: A New Version of Grain-128 with Authentication*. International Journal of Wireless and Mobile Computing 5(1):48-59 · December 2011
10. M. BOESGAARD, M. VESTERAGER, T. CHRISTENSEN, and E. ZENNER. *The stream cipher Rabbit*. Available in the eSTREAM portfolio, a descriptionis available at http://www.ecrypt.eu.org/stream/p3ciphers/rabbit/rabbit_p3.pdf, 2008.
11. M. MATSUI. *New block encryption algorithm MISTY*. In Fast Software Encryption – FSE'97, volume **1267** of Lecture Notes in Computer Science, pages 54–68, Springer, Heidelberg, January 1997.
12. P. EKDAHL and T. JOHANSSON. *A new version of the stream cipher SNOW*. In SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography, volume **2595** of Lecture Notes in Computer Science, pages 47–61. Springer, Heidelberg, August 2003.
13. S. BABBAGE and M. DODD. *The MICKEY stream ciphers*. In New Stream Cipher Designs, volume **4986** of Lecture Notes in Computer Science, pages 191–209. Springer Berlin Heidelberg, 2008
14. T. SHIRAI, K. SHIBUTANI, T. AKISHITA, S. MORIAI, and T. IWATA. *The 128-bit blockcipher CLEFIA* (extended abstract). In Fast Software Encryption – FSE 2007, volume **4593** of Lecture Notes in Computer Science, pages 181–195, Springer, Heidelberg, March 2007.