Abstract. In this paper, we propose cryptanalytic results on two lightweight stream ciphers: Fountain v1 and Lizard. The main results of this paper are the followings:

- We propose a zero-sum distinguisher on reduced round Fountain v1. In this context, we study the non-randomness of the cipher with a careful selection of cube variables. Our obtained cube provides a zero-sum on Fountain v1 till 188 initialization rounds and significant non-randomness till 189 rounds. This results in a distinguishing attack on Fountain v1 with 189 initialization rounds.

- Further, we find that the same cipher has a weakness against conditional Time-Memory-Data-Tradeoff (TMDTO). We show that TMDTO attack using sampling resistance has online complexity $2^{310}$ and offline complexity $2^{146}$.

- Finally, we revisit the Time-Memory-Data-Tradeoff attack on Lizard by Maitra et al. (IEEE Transactions on Computers, 2018) and provide our observations on their work. We show that instead of choosing any random string, some particular strings would provide better results in their proposed attack technique.

1. Introduction

Design and analysis of lightweight cryptographic algorithms have become one of the most important directions of research in cryptology. In the last two decades, the use of low powered and resource-constrained devices has increased rapidly, which resulted in the requirement of lightweight stream ciphers. The reduction of the state size reduces the power consumption of the cipher. So, designing ciphers with small states has become a challenging task in the cryptographic community. In the last few years, several new stream ciphers with smaller state sizes have been designed. The design principle of these lightweight stream ciphers differs significantly from the design principle of the standard stream ciphers.

2020 Mathematics Subject Classification: 94A60.

Key words and phrases: LWC, fountain v1, lizard, cube, zero-sum distinguisher, TMDTO.

* Corresponding author: Dibyendu Roy.
In 2019, NIST initiated a standardization project on lightweight cryptographic algorithm (LWC [1]). LWC has received 57 submissions among which 56 of them were selected as Round 1 candidates. Among these 32 were selected for Round 2. Fountain v1 [32], designed by Zhang is one such cipher among the 56 candidates of Round 1. This cipher is an authenticated cipher with a state size of 256-bit and a key size of 128-bit. The complete state (i.e., 256-bit) is divided into four LFSRs, where each of the LFSRs is of 64-bit. A more detailed design specification of this cipher is provided in Section 2.6.

Sprout [2] was the first lightweight cipher, which involved a state size equal to the key size. In the design principle of Sprout, the key bits are repeatedly used to update its state. This repeated involvement of the key bits protects the cipher from the birthday attack. However, Sprout was immediately attacked in [21, 5, 23, 11, 33] because of several weaknesses in the design rationale. Later, two more lightweight stream ciphers were designed, namely Plantlet [27] and Fruit [13]. Plantlet [27] is based on a state of size 101-bit and a key of size 80-bit, whereas the state size and the key size of Fruit [13] is 80-bit. Another variant of Fruit with 128-bit state size was also proposed in [14]. A fault attack on Plantlet was proposed by Maitra et al. [24]. The 80-bit version of Fruit was cryptanalyzed by Dey et al. [9], Zhang et al. [34], and Hamann et al. [17].

Lizard is one such lightweight stream cipher designed by Hamann et al. [18] in 2016. The cipher is based on a 120-bit key size and 121-bit state size. The design specification of Lizard is very similar to Grain-like [19] stream cipher although it does not have any LFSR. In the case of standard stream ciphers (such as Grain, Trivium, etc.) the key can be recovered if the state of the cipher can be recovered at any round of the keystream generation phase. This is not the case for Lizard. In the case of Lizard, the key can not be recovered by inverting the cipher even if an attacker can recover the state of the cipher at any round of the keystream generation phase.

The first analysis of Fountain v1 has been presented in [28]. They presented a slide attack by using 32 relations on key bits with time complexity of $17 \times 2^{80}$ and the success rate claimed to be 98%. They have also studied some internal state transition properties of Fountain v1 which allows input data (key-IV-ad) that produce identical ciphertexts with probability $2^{-32}$.

The first analysis of Lizard was done by Banik et al. [6]. Later, a TMDTO attack has been suggested by Maitra et al. [22], where the time, memory and data complexities are less than $2^{60}$.

**Our contribution.** The main contributions of this paper are as follows.

- In this paper we propose a zero-sum distinguishing attack on Fountain v1 with reduced initialization rounds. We show that Fountain v1 with 189 initialization rounds can be distinguished from a random source with a very high confidence level.
- Further, we analyze the security of Fountain v1 under conditional TMDTO attack using sampling resistance. Here we show that the state of the cipher can be recovered with online time, data, memory complexity $2^{110}$, and pre-processing complexity $2^{146}$.
- Finally, we revisit the TMDTO attack on Lizard proposed by Maitra et al [22]. In their attack model, the adversary chooses a random binary sequence of size $\psi$ and searches for it in the output keystream bits. However, since the probability of occurrence of any random binary sequence of length $\psi$ is $\frac{1}{2^\psi}$,
the authors assumed that all patterns would provide the same data and time complexity. As they mention in their paper:

“The probability of getting a ψ-bit keystream pattern is \( \frac{1}{2^\psi} \cdot \cdots \) the data complexity will be \( D = D' \times 2^{\psi + \tau} \).”

However, in this paper, we show that all the patterns will not provide equal results. The expectations of the occurrences of different patterns are different. There are some patterns for which the expected number of keystream bits required is \( 2^{\psi + 1} - 2 \), whereas there are some patterns for which the same expected number is \( 2^\psi \). Therefore, if the patterns are not suitably chosen, the data complexity will become double for some patterns.

Organization of the article.
- Section 2 is a preliminary section where we define terminologies and describe the design specification of Fountain v1.
- In Section 3, we present our distinguisher on reduced round Fountain v1.
- In Section 4, we describe our observations related to TMDTO attack on Fountain v1.
- The design specification of Lizard is described in Section 5.1.
- Section 5.2 revisits the attack of Maitra et al. [22] on Lizard.
- In Section 5.3, we explain our observation on the result proposed by Maitra et al. [22].
- Section 6 concludes the paper.

2. Preliminaries

This section is preliminary, here we describe some definitions, terminologies and design specifications of Fountain v1 [32].

2.1. Boolean function. An \( n \)-variable Boolean function \( f \) is a mapping from \( F_2^n = \{0, 1\}^n \) to \( F_2 = \{0, 1\} \) i.e., \( f : F_2^n \to F_2 \). The set of all Boolean functions involving \( n \) variables is denoted by \( B_n \). A Boolean function \( f \) involving \( n \) variables can be represented in different forms such as truth table, algebraic normal form etc. Here we define only the algebraic normal form of an \( n \)-variable Boolean function. The algebraic normal form (ANF) of \( f \in B_n \) is the following multivariate polynomial over \( F_2 \).

\[
f(x_0, \ldots, x_{n-1}) = \sum_a \lambda_a x_0^{a_0} x_1^{a_1} \cdots x_{n-1}^{a_{n-1}}
\]

Here \( a = (a_0, a_1, \ldots, a_{n-1}) \in F_2^n \) and \( \lambda_a \in F_2 \) are the coefficients of the monomials of the multivariate polynomial. The degree of an \( n \)-variable Boolean function \( f \) is the number of variables present in the highest order monomial with \( \lambda_a \neq 0 \) of the ANF of \( f \) (as in Equation (1)). In case of a random Boolean function \( f \) the ANF will have highest degree with probability \( \frac{1}{2} \).

2.2. Cube variable and superpoly. From the definition of ANF of \( f \in B_n \) (as described in subsection 2.1) it can be seen that \( f \) can be represented as a multivariate polynomial expression (as in Equation (1)). For any index set \( I = \{i_1, i_2, \ldots, i_k\} \subset \{0, 1, \ldots, n - 1\} \) we define cube variables \( C_I \) which is a set of variables \( C_I = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\} \) and term \( t_I \) which is monomial \( t_I = x_{i_1} x_{i_2} \cdots x_{i_k} \). With this \( C_I \) and \( t_I \) the Equation (1) can be rewritten as

\[
f(x_0, x_1, \ldots, x_{n-1}) = t_I P_{s(I)}(v_0, v_1, \ldots, v_{n-1-k}) + Q_I(x_0, x_1, \ldots, x_{n-1}),
\]
where \( v_i \in \{x_0, x_1, \ldots, x_{n-1}\} \setminus C_I \), \( i = 0, 1, \ldots, n - 1 - k \) and \( t_I \) does not divide any monomial of \( Q_I \). The polynomial \( P_{s(I)} \) is known as the superpoly corresponding to the cube variables \( C_I \). We also use \( f_I \) to denote the XOR of all outputs of \( f \) corresponding to all \( 2^k \) inputs for \( C_I \). In 2009, Dinur and Shamir [10] introduced the concept of cube variables and superpoly. In the same paper, they proved the following theorem.

**Theorem 1** (Dinur and Shamir [10]). For any Boolean function \( f \) and an index set \( I \), \( f_I \equiv P_{s(I)} \mod 2 \).

Hence from Theorem 1, one can observe that the XOR of the outputs corresponding to all \( 2^k \) values of cube variables provides the corresponding superpoly. This idea has been exploited to analyze several stream ciphers.

In the case of a stream cipher, a keystream bit can be represented as a function of key and IV. As the IV bits are considered as public parameters, an attacker has the freedom to choose IV according to his/her choice. Among the complete set of IV bits, the attacker carefully selects some of the IV bits as cube variables and (s)he is allowed to get the keystream bits corresponding to all possible 0/1 values of all his/her cube variables for the unknown secret key. Further (s)he computes the sum on these obtained keystream bits to observe the presence of non-randomness in the respective superpoly. If there is significant non-randomness present in the superpoly, then (s)he can distinguish the stream cipher (pseudorandom bit generator) from a random source. If this sum (i.e., superpoly) is highly biased towards zero (or towards one), then the distinguisher is often called a zero-sum distinguisher. Here the prime goal of an attacker is to select cube variables in such a way that the corresponding superpoly becomes nonrandom.

### 2.3. Cube tester.

Cube tester is an algorithm based on careful selection of cube variables which can test the non-randomness of a cipher (or a Boolean function). The concept of cube tester was first introduced by Aumasson et al. [3] in 2009. The main idea behind designing a cube tester on a cipher (or a Boolean function) is to select the cube variables in such a way that if there is any non-randomness in the cipher (or a Boolean function) the non-randomness must be reflected in the corresponding superpoly. By using this kind of testing procedure one can check several properties of a function such as the presence of any monomial in the function, presence of neutral variables in the function, upper bound of the degree of the function, and balancedness of the function, etc. In the following section, we discuss one such cube tester.

### 2.4. Upper bound on the degree of a reduced function.

Let \( f : \{0, 1\}^n \rightarrow \{0, 1\} \) be an \( n \)-variable Boolean function. We first fix a few variables of the function \( f \). Without loss of generality, we assume that the first \( k \) variables \( x_0, x_1, \ldots, x_{k-1} \) are fixed to zero. After fixing the \( k \) variables the reduced function will be a function involving \( n - k \) variables. Hence the maximum possible degree of this reduced function can be \( n - k \). To check whether the degree of the reduced function is \( n - k \) or not, we select \( x_k, x_{k+1}, \ldots, x_{n-1} \) as cube variables. Further, we compute the sum on the output for all possible 0/1 values of \( x_k, x_{k+1}, \ldots, x_{n-1} \). If the cube sum (i.e., the superpoly) is zero, then the degree of the reduced function is strictly less than \( n - k \).
2.5. TMDTO on stream ciphers. Time-Memory-Data tradeoff (TMDTO) attack on stream ciphers was first introduced by Biryukov and Shamir to recover the internal states of a stream cipher [7]. However, the initial idea of TMDTO attack on a block cipher was introduced by Hellman [20]. TMDTO attack is a chosen-plaintext attack where the adversary considers the cipher as a black box without considering the deep level internal architecture of the cipher. The complete attack consists of two phases: (1) preprocessing phase, (2) online phase. In the preprocessing phase, the adversary prepares the processing tables, and later, in the online phase, those tables are used to recover the secret key. The following five parameters determine a complete TMDTO attack: (1) size of the search space \(N\), (2) required preprocessing time \(P\), (3) online time complexity \(T\), (4) space complexity \(M\), and (5) available data in the online phase \(D\). Hellman [20] derived the following tradeoff relation \(TM^2 = N^2\) and \(P = N\), where \(T \geq D^2\) and \(N\) represents the size of the total keyspace for a block cipher. In the case of a stream cipher, Babbage [4] and Golić [15] independently coined the Time-Memory tradeoff attack to recover the internal state by inverting the keystream sequences available in the online phase. This attack is also known as BG attack.

Further, Biryukov and Shamir [7] introduced the Time-Memory-Data tradeoff attack on stream ciphers by combining the ideas of Hellman’s attack on block ciphers and BG attack on stream ciphers.

For a stream cipher, we do not require to cover the whole search space \(N\), as \(D\) data segments are available in the online phase. By using the birthday paradox, only \(\frac{N}{D}\) internal states covered in the preprocessing tables are enough to invert at least one of the \(D\) segments available in the online phase. To ensure that this inversion is successful the dimension of the preprocessing table must be \(k \times t\), such that \(kt^2 = N\), which is also called the matrix stopping rule. Thus, the total number of preprocessing tables required to be constructed is \(t \times \frac{N}{D}\). The preprocessing time will be \(P = \frac{N}{D}\) and the memory requirement is \(M = \frac{kt}{D}\).

One should note that the adversary needs to prepare a single table in case of \(t = D\); otherwise (s)he is required to construct multiple tables. For constructing multiple tables, the adversary needs to modify the random function \(\phi(\cdot)\) as suggested in [7]. Now, the total time required to perform the actual attack is the time required to invert any one of the data segments \(D\). In the worst case scenario, for each table, the adversary needs to undertake \(t\) attempts at inverting \(\phi\). So the total time complexity of an actual attack becomes \(T = t \times \frac{t}{D} \times D = t^2\). In this way, by considering \(kt^2 = N\) the tradeoff curve between \(T, M\) and \(D\) becomes

\[
TM^2D^2 = t^2 \times \left(\frac{kt}{D}\right)^2 \times D^2 = N^2,
\]

where \(t \geq D\), i.e., \(T \geq D^2\).

2.5.1. BSW sampling resistance and TMDTO attack. Biryukov, Shamir, and Wagner introduced the idea of sampling resistance [8] to derive better tradeoff parameters for the TMDTO attack. This approach may have reasonable complexity even for stream ciphers having a large internal state size. They showed that if a stream cipher has sampling resistance \(R = 2^{-l}\), then one can easily enumerate a set of special internal states which generate keystream bits with a particular prefix of length \(l\). In a more simplified sense, we can say that if an adversary has access to a predetermined pattern of keystream bits of length \(l\), then the same adversary can recover \(l\) specific internal state bits of the cipher by using that predetermined prefix.
of keystream of length $l$ and guessing the remaining bits of the internal state. Thus, due to sampling resistance, the size of the search space $N$ is reduced to $NR = N2^{-l}$, and given $D$ bits of the actual keystream, the expected number of special states encountered is $DR = D2^{-l}$.

We attempt to invert only those data segments which are generated from the special internal states. So, only the special states are covered in the preprocessing tables. For this, the random function $\phi$ used to prepare the preprocessing table is required to be changed so that it maps a special state to keystream bits having a particular pattern of length $l$ as a prefix. Here the attacker assumes that during the online phase there exists at least one data segment which has the required prefix, i.e., $DR \geq 1$ in the online phase and only those data segments are attempted to be inverted.

By substituting $DR$ and $NR$ in place of $D$ and $N$ in the tradeoff equation (3), the following tradeoff relation is derived:

$$TM^2(DR)^2 = (NR)^2$$

$$\Rightarrow TM^2D^2 = N^2,$$

where $T \geq (DR)^2$.

Remarkably, the lower bound on the time complexity of the online phase is reduced from $D$ to $DR$ in comparison to the traditional TMDTO attack with the TMDTO attack based on sampling resistance. The computation of sampling resistance of a stream cipher is a cipher specific problem. The tap positions of the filter functions, the feedback functions, and the size of the internal state have a significant influence on the calculation of sampling resistance. Conditional sampling resistance along with the TMDTO attack on several stream ciphers such as Grain-v1, Grain-128, Lizard have been proposed in [25, 26, 22].

2.6. Design specification of Fountain v1. Fountain v1 [32] is a lightweight authenticated stream cipher, designed by Zhang. This cipher is one of the candidates of Round 1 of the NIST competition [1]. Fountain v1 is based on three main components: one key of 128-bit length, one IV of 96-bit length and a variable-length associated piece of data. This 128-bit secret key and 96-bit IV are used to initialize the state of the cipher. After initializing the state by the secret key and IV, the cipher loads the associated data into its state. After that, the cipher generates keystream bits to encrypt plaintext bits.

The design of Fountain v1 is based on four LFSRs each of length 64-bit, one lightweight 4-bit to 4-bit S-box, one MDS matrix, one nonlinear filter function and one output function. The S-box used in the design of the Fountain v1 will be different for different phases. Here Figure 1 provides a pictorial description of Fountain v1. As our distinguisher works on the first 189 rounds, we only consider the key-IV initialization phase of the cipher. Regarding other phases of the cipher, one can go through the full design specification of Fountain v1 in [32]. In the following subsections, we describe the design specification and key-IV initialization phase of the Fountain v1.

2.7. State update function of key-IV initialization of Fountain v1. Fountain v1 [32] is based on four LFSRs, each LFSR is of length 64-bit. The connection polynomial related to each LFSR is provided below.

LFSR 1: $1 + x^{12} + x^{25} + x^{31} + x^{64}$; LFSR 2: $1 + x^9 + x^{19} + x^{31} + x^{64}$;
LFSR 3 : $1 + x^{14} + x^{20} + x^{31} + x^{64}$; LFSR 4 : $1 + x^6 + x^{10} + x^{31} + x^{64}$.

The complete state of size 256-bit is divided into these four LFSRs. In each round the state bits of the LFSRs are shifted by following the usual procedure and the feedback bit of each LFSR is computed by using their respective linear feedback function and a nonlinear function. In each round one 4-bit to 4-bit S-box operates on the 4-bit of the current state of the cipher. The output of the S-box is further multiplied by an MDS matrix which generates 4 bits $(y_0, y_1, y_2, y_3)$. Further, these bits $y_i$ are XOR’ed with the linear feedback bit of LFSR $i + 1$, where $i = 0, 1, 2, 3$.

The S-box which is used in the key-IV initialization phase is described in Table 1.

Table 1. S-box for key-IV initialization phase

| $x$   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S(x)$ | 1 | A | 4 | C | 6 | F | 3 | 9 | 2 | D | B | 7 | 5 | 0 | 8 | E |

The MDS matrix by which the state is multiplied after the application of the S-box is defined over $GF(2^2)$. For more detail regarding this matrix multiplication one can go through the original article [32]. The expression for the MDS matrix is provided below.

$$B = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

To describe the state update process during the initialization phase we use some notation. Let $s_i^{(j)}$, $i = 0, \ldots, 63$ denote the state bits of the $j$-th LFSR, where $j = 1, 2, 3, 4$. In each round $s_1^{(1)}$, $s_1^{(2)}$, $s_1^{(3)}$ and $s_1^{(4)}$ are taken as an input to the S-box. Let $x_1, x_2, x_3, x_4$ be the output from the S-box, then the input to the MDS matrix multiplication process will be the two elements $x_4 || x_3$ and $x_1 || x_2$. These two elements are considered as elements in the finite field $GF(2^2)$. More detail regarding this finite field can be found in [32]. Let $y_1, y_2, y_3, y_4$ be the output bits after the multiplication by the MDS matrix, then we have the following.
of the four LFSRs. Hence, the feedback of the LFSRs will be as follows.

\[
S(s_1^{(1)}, s_1^{(2)}, s_1^{(3)}, s_1^{(4)}) \rightarrow \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \rightarrow B \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \rightarrow \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}
\]

The complete process i.e., first applying the S-box, then multiplying by the MDS matrix can be represented as an integrated S-box. The description of this integrated S-box is provided in Table 2.

**Table 2. Integrated S-box for key-IV initialization phase**

| \(S(x)\) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-----------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x         | 9 | 5 | 6 | D | 8 | A | 7 | 2 | E | 4 | C | 1 | F | 0 | B | 3 |

The final output bits \(y_1, y_2, y_3, y_4\) are further XOR’ed with the linear feedback of the four LFSRs. Hence, the feedback of the LFSRs will be as follows.

\[
\begin{align*}
LFSR 1 &: \ s_{64+i}^{(1)} = s_{0+i} + s_{12+i} + s_{25+i} + s_{31+i} + y_1 \\
LFSR 2 &: \ s_{64+i}^{(2)} = s_{0+i} + s_{9+i} + s_{19+i} + s_{31+i} + y_2 \\
LFSR 3 &: \ s_{64+i}^{(3)} = s_{0+i} + s_{14+i} + s_{20+i} + s_{31+i} + y_3 \\
LFSR 4 &: \ s_{64+i}^{(4)} = s_{0+i} + s_{6+i} + s_{10+i} + s_{31+i} + y_4
\end{align*}
\]

for \(0 \leq i \leq 63\). The output bit of the Fountain v1 is computed by using a nonlinear filter function \(h\) and a linear function involving seven state bits. The algebraic normal form of the nonlinear filter function \(h(x)\) is defined below.

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

Here \(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\) are \(s_2^{(1)}, s_5^{(1)}, s_4^{(2)}, s_1^{(3)}, s_{11+i}^{(3)}, s_{23+i}^{(3)}, s_{27+i}^{(3)}, s_{24+i}^{(3)}\), \(s_{29+i}, s_{30+i}\) respectively. The \(i\)-th keystream bit of Fountain v1 is calculated by using the following expression.

\[
z_i = s_{3+i}^{(1)} + s_{11+i}^{(1)} + s_{20+i}^{(2)} + s_{5+i}^{(3)} + s_{16+i}^{(3)} + s_{7+i}^{(4)} + s_{29+i}^{(4)} + h.
\]

### 2.8. Key-IV Initialization Phase

In this phase the state of the cipher will be initialized by one 128-bit secret key and one 96-bit IV. The secret key, IV and 32 padding bits are loaded into the state of the cipher in the following way. We denote key bits by \(k_i, i = 0, 1, \ldots, 127\) and IV bits by \(v_i, i = 0, 1, \ldots, 95\).

\[
\begin{align*}
s_1^{(1)} &= k_i; \\
s_{i+8}^{(1)} &= v_i; \\
s_{i+16}^{(1)} &= k_{i+8}; \\
s_{i+24}^{(1)} &= v_{i+8}; \\
s_{i+32}^{(1)} &= k_{i+16}; \\
s_{i+40}^{(1)} &= v_{i+16}; \\
s_{i+48}^{(1)} &= k_{i+24}; \\
s_{i+56}^{(1)} &= v_{i+24}; \\
s_1^{(2)} &= k_{i+32}; \\
s_{i+8}^{(2)} &= v_{i+32}; \\
s_{i+16}^{(2)} &= k_{i+40}; \\
s_{i+40}^{(2)} &= v_{i+40}; \\
s_{i+48}^{(2)} &= k_{i+48}; \\
s_{i+56}^{(2)} &= v_{i+48}; \\
s_1^{(3)} &= k_{i+64}; \\
s_{i+8}^{(3)} &= v_{i+64}; \\
s_{i+16}^{(3)} &= k_{i+72}; \\
s_{i+24}^{(3)} &= v_{i+72};
\end{align*}
\]
output. Instead these bits of the cipher, it will run for 384 rounds without generating any keystream bit as
Here \( 0 \leq i \leq 7 \) and the constants \( C_i, (1 \leq i \leq 4) \) are defined as \( C_1 = 0xff, C_2 = 0x3f, C_3 = 0x00, C_4 = 0x80 \). After loading the key and IV into the state of the cipher, it will run for 384 rounds without generating any keystream bit as output. Instead these bits \( z_i \), for \( i = 0, \ldots, 383 \) are XOR’ed with the feedback bits of the four LFSRs. Hence the feedback functions for the four LFSRs for 384 rounds will be as follows.

\[
\begin{align*}
\text{LFSR 1: } s_{64+i}^{(1)} &= s_{0+i}^{(1)} + s_{12+i}^{(1)} + s_{25+i}^{(1)} + s_{31+i}^{(1)} + y_1 + z_i \\
\text{LFSR 2: } s_{64+i}^{(2)} &= s_{0+i}^{(2)} + s_{12+i}^{(2)} + s_{25+i}^{(2)} + s_{31+i}^{(2)} + y_2 + z_i \\
\text{LFSR 3: } s_{64+i}^{(3)} &= s_{0+i}^{(3)} + s_{12+i}^{(3)} + s_{25+i}^{(3)} + s_{31+i}^{(3)} + y_3 + z_i \\
\text{LFSR 4: } s_{64+i}^{(4)} &= s_{0+i}^{(4)} + s_{12+i}^{(4)} + s_{25+i}^{(4)} + s_{31+i}^{(4)} + y_4 + z_i.
\end{align*}
\]

After the initialization phase the cipher enters into the associated data processing phase. The detailed description of associated data processing phase of Fountain v1 can be found in [32]. As our distinguisher works on reduced round Fountain v1 (specifically \( \leq 384 \)), we assume that the cipher starts generating keystream bits just after \( r \) initialization rounds \((r \leq 384)\). Several cryptanalytic results on reduced round ciphers [12, 30] are also based on similar assumptions.

3. Our Distinguisher on Fountain v1

In this section, we describe our zero-sum distinguisher on Fountain v1 with 189 initialization rounds. Our distinguisher is based on the careful selection of the cube variables (which are IV variables). We perform a cube sum on the output bits corresponding to all possible values of the selected cube variables for randomly many secret keys. This process checks the non-randomness in the corresponding superpoly. So our experimental work is based on two phases: (1) selection of the cube variables, \(2\) check of the randomness of the corresponding superpoly.

3.1. Experiments on Fountain v1. To select the cube variables we have followed a very similar technique as in [29]. Here our goal is to select the cube variables (i.e., IV variables) in such a way that the cube sum on the output bits corresponding to all possible values of the cube variables is zero for the maximal number of initialization rounds. We start with an empty cube set say \( C_I \) (i.e., \( C_I = \emptyset \)). Further, we add one good cube bit into the set. A cube bit will be called a good cube bit among all the IV bits when it produces a zero-sum for the maximum number of initialization rounds among the other IV bits. Algorithm 1 describes the cube variable selection process. This selection approach is also known as the Greedy approach.

Now if we run the Algorithm 1 with an empty set \( C_I = \emptyset \), then the first cube bit that is added into the set is \( v_{95} \). As for this cube bit \( v_{95} \) we achieve zero cube sum for the highest number of rounds among other IV bits. The main reason behind this is that \( v_{95} \) will appear in the keystream generation function much later than the other IV bits (see Equation (8)). If Algorithm 1 produces a zero-sum for the same number of rounds for all the IV bits, then we modify the \( C_I \) updation process. In this kind of scenario we select the two best cube bits at a time to update \( C_I \) i.e., \( C_I = C_I \cup \{\text{temp}_1, \text{temp}_2\} \). During this cube selection process, several good
cube bits (one or two bits at a time) will produce a zero-sum for the same number of rounds. In that case, we select the cube bit randomly (similarly for two bits at a time). We continue this process of adding good cube bits by using Algorithm 1. For cubes of size 26 we obtain two cubes $C_{I_1}$ and $C_{I_2}$; whose indices are $I_1 = \{95, 94, 93, 92, 91, 90, 29, 28, 26, 25, 24, 23, 21, 20, 59, 58, 57, 63, 89, 37, 88, 32, 78, 60\}$, $I_2 = \{95, 94, 93, 92, 91, 90, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 59, 58, 57, 63, 89, 37, 88, 32, 81, 60\}$. For these two cubes $C_{I_1}$ and $C_{I_2}$ we obtain a zero-cube sum for up to 185 initialization rounds. Among these two cubes, we select $C_{I_2}$ and randomly includes $\{54, 86, 18\}$ into $I_2$. Hence the updated index set will be $I_3 = I_2 \cup \{54, 86, 18\} = \{95, 94, 93, 92, 91, 90, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 59, 58, 57, 63, 89, 37, 88, 32, 81, 60, 54, 86, 18\}$. For this cube we obtain a zero-sum for up to 186 initialization rounds. Further, we include two good bits at a time in $C_{I_3}$ to construct a cube of size 31 and we have observed that including $\{87, 62\}$ into $I_3$ improves the maximal zero-sum round. Hence the updated index set will be $I_4 = \{95, 94, 93, 92, 91, 90, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 59, 58, 57, 63, 89, 37, 88, 32, 81, 60, 54, 86, 18, 87, 62\}$. For this cube, we perform a cube sum on the keystream bits for all possible values of the variables of $C_{I_4}$. We repeat this experiment for 256 random keys. The observed biases in the corresponding superpoly from this experiment are described in Table 3.

From Table 3 it can be observed that cube sum is zero for up to 188 initialization rounds and there is a significant bias in the superpoly at 189 rounds (i.e., $Pr[\text{superpoly} = 0] = 0.93$). These biases are computed using 256 random keys. One may note that the number of used random keys is sufficient for our distinguishing attack (see Remark 1).

Remark 1. Let $A$ and $B$ be two distributions and an event $E$ happens in $A$ with probability $p$ and the same event happens in $B$ with probability $p(1 + q)$. Then to distinguish these two distributions one will require $O\left(\frac{1}{pq^2}\right)$ random samples. In fact
Some results on lightweight stream ciphers Fountain v1 & Lizard

Table 3. Distinguishing attack on Fountain v1

| Cube Size | Cube variable indices | Probability of superpoly = 0 |
|-----------|-----------------------|-----------------------------|
| $[I_4] = 31$ | $I_4$ | 187 188 189 190 191 192 193 194 195 |

$I_4 = \{95, 94, 93, 92, 91, 90, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 59, 58, 57, 63, 89, 37, 88, 32, 81, 60, 54, 86, 18, 87, 62\},$

one will require $\frac{32}{pq^2}$ random samples for a 99.7% confidence level (see [29] for more detail).

To achieve a significant confidence level for our distinguisher, we have performed our experiment for 256 random keys. In case of our distinguisher for 189 initialization rounds, we have $p = 0.5$ and $p(1 + q) = 0.93$. Hence $\frac{32}{pq^2} = 87$. Since we have repeated our experiment for 256 random keys, the confidence level of our distinguisher is more than 99.7%.

Description of our distinguisher.

Here we provide a compact description of our distinguisher on Fountain-v1 with 189 initialization rounds.

1. Select the IV variables whose indices belong to $I_4 = \{95, 94, 93, 92, 91, 90, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 59, 58, 57, 63, 89, 37, 88, 32, 81, 60, 54, 86, 18, 87, 62\}$.
2. Consider all possible values of IV bits whose index belongs to $I_4$ for any random key (which is unknown). So, we will have $2^{31}$ keystream bits for each random key.
3. XOR all $2^{31}$ keystream bits for any random key and observe if the value obtained is 0 or not.
4. Repeat the Step 2 and Step 3 for sufficiently many random keys.
5. If the value obtained in Step 3 is zero for a high proportion of random keys ($\approx 0.93$), then the source of the keystream bits is Fountain-v1 with 189 initialization rounds.
6. Else the source is not Fountain-v1 with 189 initialization rounds.

4. Sampling resistance of Fountain v1

In this section, we present an analysis of Fountain v1 using conditional BSW sampling. We first describe how we can recover some state bits of Fountain v1 after the complete initialization phase by fixing certain state bits. Then using the process described in [22] we find the best parameters for TMDTO attack on Fountain v1.

It must be noted that the designer of Fountain v1 [32] has stated the following in the design specification of the cipher:
“the 256-bit size internal state also eliminates the threat of the known form of the time/memory/data tradeoff attacks with respect to 112-bit security, when taking into account the pre-computation/memory/time/data complexities.”

Fountain v1 has a state size of 256 bits which is more than twice the 112-bit security claimed by the author. The output bit of Fountain v1 is computed by using a nonlinear filter function \( h \) and a linear function involving seven state bits (see Equation (8)). The nonlinear filter function of Fountain v1 is described in Equation (7). We wish to exploit the distance between the two tap bits \( s^{(1)}_{11}, s^{(1)}_{29} \).

At any particular time the state bits \( s^{(1)}_{11+i}, i = 0, \ldots, 17 \) can be recovered using the equation

\[
s^{(1)}_{11+i} = z_i + s^{(1)}_{3+i} + s^{(2)}_{20+i} + s^{(3)}_{5+i} + s^{(3)}_{16+i} + s^{(4)}_{7+i} + s^{(4)}_{29+i} + h, \ i = 0, \ldots, 17.
\]

One can observe that we cannot proceed further, as \( s^{(1)}_{29} \) needs to be guessed to recover \( s^{(1)}_{11} \). The remaining 18 state bits, \( s^{(1)}_{11+i}, i = 0, \ldots, 17 \), are recovered from 18 keystream bits.

We can see that if we fix \( s^{(2)}_{24+i} \) to 0, we do not need to guess \( s^{(1)}_{29+i} \). Hence we can recover more state bits. But it needs to be taken note of that after 34 bits are recovered, the next bit recovery involves the feedback bit of LFSR4 so we need to guess some more state bits which are involved in the feedback bit. We can use Equation (10) to recover 36 state bits by fixing 36 state bits and guessing 158 state bits. All the equations which are required to recover the state bits are mentioned in Appendix A.

4.1. Trade-off parameters for Fountain v1. In Section 4 of [22] it has been mentioned that the best parameters for TMDTO under conditional BSW sampling are obtained when \( 5\psi + 2\tau = n \), where \( \psi \) is the number of bits recovered, \( \tau \) is the number of bits fixed and \( n \) is the state size. In case of Fountain v1 the state size is \( n = 256 \). The online complexity of the attack will be \( T = M = D = 2^{\frac{n+\psi}{2}} \) and the pre-processing table will be of size \( P = \frac{N}{D} = 2^{\frac{n}{2+\psi}} \).

For Fountain v1, we find the best values of \( \psi \) and \( \tau \) for achieving the best online complexity. Here \( \psi = 36, \tau = 38 \). It can be noted that we can recover 36 state bits by fixing 36 state bits, however we need to fix two more state bits to meet \((5 \cdot 36 + 2 \cdot 38) = 256 = n\). Hence the best possible parameters for TMDTO attack on Fountain v1 will be those described in Table 4.

Table 4. TMDTO parameters for Fountain v1

| Time | Memory | Data | Pre-processing |
|------|--------|------|----------------|
| 110  | 110    | 110  | 146            |

From the above discussion it can be seen that the online complexity of our attack is \( 2^{110} \) and the pre-processing complexity is \( 2^{146} \). Here the online complexity is less than the complexity of exhaustive key search but the offline complexity is much higher than the complexity of exhaustive key search.

In this analysis we show that sampling is possible on the cipher and that if the assumptions on the precomputation cost is relaxed the online time of the TMDTO attack reduces to \( 2^{110} \). We do not view this to be a break of Fountain, but we point out that the cipher has a weakness against conditional TMDTO attack.
5. A NOTE ON A TMDTO ATTACK ON LIZARD

In this section we revisit the TMDTO attack on Lizard proposed by Maitra et al. [22]. Further, we describe our observation on the same attack. For a better understanding of the TMDTO attack and our observation we first look into the design specification of Lizard [18], which is described in Section 5.1.

5.1. SHORT DESCRIPTION OF LIZARD.

Structure: Unlike several popular stream ciphers Lizard doesn’t have any LFSR. It has two NFSRs, namely NFSR1 and NFSR2 and the total state size is 121. The NFSR1 is of 31-bit, and the state of that NFSR is denoted by \((S^t_0, \ldots, S^t_{30})\). The size of the second NFSR, i.e., NFSR2 is 90-bit, and the state is denoted by \((B^t_0, \ldots, B^t_{89})\).

Feedback functions of the NFSRs: The feedback function of NFSR1 is described in Equation (11).

\[
S^{t+1}_{30} = S^t_0 + S^t_1 + S^t_5 + S^t_6 + S^t_{15} + S^t_{17} + S^t_{18} + S^t_{20} + S^t_{25} + S^t_8S^t_{18} + S^t_8S^t_{20}
+ S^t_{12}S^t_{21} + S^t_{14}S^t_{19} + S^t_{17}S^t_{21} + S^t_{20}S^t_{22} + S^t_{14}S^t_{12}S^t_{22} + S^t_{4}S^t_{19}S^t_{22}
+ S^t_{7}S^t_{20}S^t_{21} + S^t_{8}S^t_{18}S^t_{22} + S^t_{8}S^t_{20}S^t_{22} + S^t_{13}S^t_{19}S^t_{22} + S^t_{20}S^t_{21}S^t_{22}
+ S^t_{4}S^t_{7}S^t_{12}S^t_{21} + S^t_{4}S^t_{7}S^t_{19}S^t_{21} + S^t_{4}S^t_{12}S^t_{21}S^t_{22} + S^t_{4}S^t_{19}S^t_{21}S^t_{22}
+ S^t_{7}S^t_{8}S^t_{18}S^t_{21} + S^t_{7}S^t_{8}S^t_{20}S^t_{21} + S^t_{7}S^t_{12}S^t_{19}S^t_{21} + S^t_{8}S^t_{18}S^t_{21}S^t_{22}
+ S^t_{8}S^t_{20}S^t_{21}S^t_{22} + S^t_{12}S^t_{19}S^t_{21}S^t_{22}.
\]

(11)

The feedback function of the second register NFSR2 is described in Equation (12).

\[
B^{t+1}_{89} = S^t_0 + B^t_0 + B^t_{24} + B^t_{40} + B^t_{70} + B^t_{84} + B^t_{10}B^t_{59} + B^t_{10}B^t_{12} + B^t_{15}B^t_{16}
+ B^t_{25}B^t_{53} + B^t_{35}B^t_{32} + B^t_{55}B^t_{58} + B^t_{60}B^t_{74} + B^t_{20}B^t_{22}B^t_{23} + B^t_{62}B^t_{68}B^t_{72}
+ B^t_{77}B^t_{80}B^t_{83}.
\]

(12)

Output Function: The output bit \(z_t\) is computed as

\[
z_t = \mathcal{L}_t + \mathcal{Q}_t + \mathcal{T}_t + \mathcal{T}'_t,
\]

where

\[
\mathcal{L}_t = B^t_7 + B^t_{11} + B^t_{30} + B^t_{40} + B^t_{45} + B^t_{54} + B^t_{71},
\mathcal{Q}_t = B^t_3B^t_{21} + B^t_3B^t_{52} + B^t_{18}B^t_{37} + B^t_{44}B^t_{16},
\mathcal{T}_t = B^t_5 + B^t_8B^t_{62} + B^t_{34}B^t_{73} + B^t_{35}B^t_{32} + B^t_{54}B^t_{58} + B^t_{60}B^t_{74} + B^t_{27}B^t_{68}B^t_{72}
+ B^t_6B^t_{42}B^t_{61}B^t_{61} + B^t_1B^t_{19}B^t_{27}B^t_{43}B^t_{57}B^t_{66}B^t_{75},
\mathcal{T}'_t = S^t_{23} + S^t_{13}S^t_{16} + S^t_{19}S^t_{13}B^t_{48} + S^t_{12}S^t_{21}B^t_{38}B^t_{63}.
\]

Initialization: The initialization process is done in four phases. Below we describe each of these phases one by one.

1. Key-IV loading: The key and IV size of Lizard are 120-bit and 64-bit respectively. The key and IV are denoted by \(K = (K_0, \ldots, K_{119})\) and \(IV = (IV_0, \ldots, IV_{63})\) respectively. The initial state of NFSR1 is as follows.

\[
B^0_j = \begin{cases} 
K_j + IV_j, & \text{for } 0 \leq j \leq 63 \\
K_j, & \text{for } 64 \leq j \leq 89
\end{cases}
\]

The 30 bits of NFSR2 are initialized as follows.

\[
S^0_j = \begin{cases} 
K_{j+90}, & \text{for } 0 \leq j \leq 28 \\
K_{119} + 1, & \text{for } j = 29 \\
1, & \text{for } j = 30
\end{cases}
\]
2. Grain like mixing: In this phase the output bit $z_t$ is computed, but not produced as output. Rather it is fed back into both the NFSRs for $0 \leq t \leq 127$ rounds i.e., $z_t$ is XOR’ed with the feedback bits of both the NFSRs. The complete process of this mixing phase for $0 \leq t \leq 127$ rounds is described below.

$$B_{i+1}^t = B_i^t + K_j, \quad \text{for } 0 \leq j \leq 89,$$

$$S_{i+1}^t = S_i^t \quad \text{for } i \in \{0, \ldots, 29\},$$

$$S_{i+1}^t = S_i^t \quad \text{for } 0 \leq j \leq 29$$

$$S_{i+1}^t = S_i^t \quad \text{for } j = 30$$

3. Second key addition: In this phase, the key is XOR’ed bitwise with the feedback bits of both the NFSRs. The process of key addition is described below.

$$B_j^{129} = B_j^{128} + K_j, \quad \text{for } 0 \leq j \leq 89$$

$$S_j^{129} = \begin{cases} S_j^{128} + K_j + 90, & \text{for } 0 \leq j \leq 29 \\ 1, & \text{for } j = 30 \end{cases}$$

4. Final diffusion: In this final phase, both the NFSRs are clocked for 128 rounds without computing any keystream bit. The complete procedure which needs to be followed for $t = 129, \ldots, 256$ rounds is described below.

$$B_{i+1}^t = B_{i+1}^t \quad \text{for } i \in \{0, \ldots, 88\},$$

$$B_{i+1}^t = B_{i+1}^t \quad \text{for } i \in \{0, \ldots, 29\},$$

$$S_i^{1+1} = S_i^{1+1} \quad \text{for } i \in \{0, \ldots, 29\}$$

The final states of the registers NFSR1, NFSR2 after finishing all these phases will be $(S_0^{257}, \ldots, S_{89}^{257})$ and $(B_0^{257}, \ldots, B_{89}^{257})$ respectively.

After the final diffusion phase, Lizard becomes ready to produce keystream bits as output and enters into the keystream generation phase. During the keystream
generation phase, the cipher updates its state by following the usual shifting and feedback mechanism and the keystream bit will be computed by using Equation (13). For more detailed design specifications of Lizard, one may look into the original article [18].

5.2. THE ATTACK IDEA PROPOSED IN [22]. Here, in short, we explain the attack idea proposed in [22]. For a more detailed explanation, one can go through that paper.

Consider a cipher with state size $\nu$, i.e., the total search space is $N = 2^\nu$. We then try to deduce some bits from the secret state by fixing certain key stream bit pattern. So, fixing
- $\tau$ bits of the state to a specific pattern
- assuming a specific pattern of $\psi$ keystream bits, and
- assigning values to the rest of the $\nu - \tau - \psi$ state bits

we try to deduce $\psi$ bits of the state. The total search space now reduces to $N' = 2^{\nu - \tau - \psi}$.

Preprocessing Stage: In the pre-processing step, we generate the table to be used in the online phase. First we take a random string of length $\nu - \tau - \psi$, let us call it $\omega$. Then we fix $\tau$ bits of the state to a specific pattern decided previously, this gives $\nu - \tau - \psi + \tau = \nu - \psi$ of the state. Now we obtain the remaining $\psi$ bits of the state using the $\psi$ bits of the keystream bits which are of a specific pattern. So now we have the information of the complete state. If this state is clocked, the first $\psi$ bits of the keystream is the pattern we had fixed earlier. We run this cipher for $\nu - \tau - \psi$ more steps, and this pseudo-random string will be considered as the next element of the table, referred as $f(\omega)$. We repeat this $t$ times to obtain a row. This whole process is repeated for $m$ randomly chosen $(\nu - \tau - \psi)$ bit strings to obtain $m$ such rows. Thus this table contains $mt$ elements of which only the first (SP) and the last element (EP) of the row is stored. According to the birthday paradox, with proper parameters, this table will have negligible collisions. At the same time, the online data (here keystream) should be of such an amount so that the attack becomes successful, i.e., we obtain the intended pattern in the table.

Processing Stage: In the output keystream, the attacker searches for a specific pattern of $\psi$ bits in the (EPs) of the table. Upon a hit, the next immediate $(\nu - \tau - \psi)$ bits (name this $\xi$) are considered to be the string to be searched in the offline table. If a match is found, it means the required $(\nu - \tau - \psi)$-bit state is stored in the $(t - 1)^{th}$ position of that row for some $t$. To obtain this state the adversary must operate the cipher $(t - 1)$ times on the SP of the same row where the match was found. If a match is not found then an $f$ operation is applied to $\xi$ to obtain $f(\xi)$ and this string is searched in the table. This process is repeated until a hit is obtained. In the worst case, the adversary has to apply $f$ to $\xi$ for $t$ times. If no match is found even after applying $f$ for $t$ times, we reject this attempt and search for another block of the specified $\psi$ bits in the output keystream.

5.3. OUR OBSERVATION. The authors in [22] suggested to fix any random string of size $\psi$ and create the table in the preprocessing stage based on that fixed string to be the output keystream. Also, in the processing stage, we look for the same fixed pattern of the output keystream $z$. They have not mentioned any particular pattern for those $\psi$ bits that might yield a better result than other possible strings.
Since the probability of the occurrence of any pattern is $\frac{1}{2^\psi}$, apparently it seems that all patterns will have an equal effect on the final complexity. The authors have also used the expression $2^\psi$ in their calculation of this complexity. But we observe that some particular patterns provide better complexity in this attack compared to other strings. Let us explain this with a simple example of tossing an unbiased coin.

Suppose we are tossing a coin repeatedly. We are looking for a particular chosen pattern as output for any two consecutive tosses. Two consecutive tosses can have four possible outcomes: $HH$, $TT$, $HT$, $TH$. Since the coin is unbiased, the probability of the occurrence of any of these four outcomes in two consecutive tosses is $\frac{1}{4}$. But if we look at the expected number of tosses required to achieve our desired pattern, we can see that all four patterns do not give the same results. With very simple calculations we can check that the expected number of tosses required to achieve $HT$ or $TH$ as output is lower than that of $HH$ or $TT$. This kind of problem can be treated using the idea of Absorbing Markov Chain [16]. Here we give a short description of the procedure. A detailed explanation can be found in [16].

We consider all the possible intermediate states that occur to achieve our desired pattern. For example, if we want to achieve the pattern $HTHT$, the intermediate states are $H$, $HT$, $HTH$. The initial state is denoted as $\phi$ and the final state is $HTHT$. From the initial state to reach the final state, each of the intermediate states should be achieved.

$\phi \rightarrow H \rightarrow HT \rightarrow HTH \rightarrow HTHT$

Now, from any state, the next state can be attained with probability $\frac{1}{2}$. It can come back to any of the previous states or can stay at the same state with probability $\frac{1}{2}$. For example, from the state $HTH$, the next state $HTHT$ can be attained if the next toss gives tail (probability is $\frac{1}{2}$). Otherwise, it comes back to state $H$.

Now, we construct a $5 \times 5$ matrix $M$ where the $r$-th row and column correspond to the $r$-th state from the end. The entry $m_{i,j}$ is the probability of attaining the state corresponding to the column from the state corresponding to the row. Now, we ignore the first row and column and call the remaining matrix $Q$. Then we compute $(I - Q)^{-1}$. The sum of the entries of the last row of $(I - Q)^{-1}$ gives the expectation of the pattern. For a detailed explanation, one can go through Chapter 11 of [16].

When we are looking for some fixed pattern in the output keystream, all possible patterns of size $\psi$ do not have the same expected value for the number of keystream bits required to achieve such a pattern. Usually, the patterns involving many 0’s and 1’s or vice versa have a higher expected value for the number of keystream bits required, compared to the strings where the total number of 0’s and 1’s is balanced.

Experimentally we have observed that the maximum expectation for a $\psi$-bit pattern can be observed for the patterns of all zeros or all ones. This expectation is $2^{\psi+1} - 2$, we prove the same in Lemma 2.

The minimum expectation can be observed for the patterns 1 followed by all zeros or zero followed by all ones. This value is $2^\psi$. We provide the theoretical justification of this in Lemma 1. These kinds of problems can be treated by using the idea of Absorbing Markov Chain [16].

Lemma 1. For the patterns of the form,

1. $1$ followed by all $0$’s
2. 0 followed by all 1’s

the expected number of trials is $2^\psi$.

**Proof.** Since the two patterns are symmetric, we prove only our assertion for 100 \cdots 0. The other result can be similarly proved.

So, the states in this case are 100 \cdots 0, 100, 10, 1, \phi. The matrix corresponding to this string is $M = (m_{i,j})$, where

$$
\begin{align*}
m_{1,1} &= 1 \\
nm_{i+1,i} &= \frac{1}{2} \\
nm_{i,\phi} &= \frac{1}{2} \\
nm_{\psi+1,\psi+1} &= \frac{1}{2} \\
nm_{i,j} &= 0 \\
n\end{align*}
$$

Now, ignoring the first row and first column, we obtain the matrix $Q$ of size $\psi \times \psi$. Subtracting it from the identity matrix $I_{\psi,\psi}$, we have $X = I - Q$ as follows,

$$
\begin{align*}
x_{i,i} &= 1 & \text{for } 1 \leq i \leq (\psi - 2) \\
x_{i,i} &= \frac{1}{2} & \text{for } i = (\psi - 1), (\psi) \\
x_{i+1,i} &= -\frac{1}{2} & \text{for } 1 \leq i \leq (\psi - 1) \\
x_{i,\psi-1} &= -\frac{1}{2} & \text{for } 1 \leq i \leq (\psi - 2) \\
x_{i,j} &= 0 & \text{otherwise.}
\end{align*}
$$

Now, we compute the sum of the last row of $X^{-1}$. Suppose, $X^{-1} = Y$.

Let us focus on the last row of $Y X = I$. Suppose, the last row of $Y$ is $(y_1, y_2 \cdots y_\psi)$. The $r$-th element of the last row of $YX$ is given by $\sum y_i x_{i,r}$. Since the last element of the last row of $YX = I$ is 1, we have $y_\psi = 2$.

Since all the elements of the last row of $YX = I$ are zeros, except the last element, we have the following equations.

1. $y_2 = 2y_1, y_3 = 2y_2, y_4 = 2y_3, \cdots y_{\psi-1} = 2y_{\psi-2}$
2. $-y_1 - y_2 - \cdots - y_{\psi-2} + y_{\psi-1} - y_\psi = 0$.

From the first $(\psi - 1)$ equations, we derive $y_k = 2^{k-1}y_1$ for $k \leq (\psi - 1)$. Substituting these values and the value of $x_\psi$ into $-y_1 - y_2 - \cdots - y_{\psi-2} + y_{\psi-1} - y_\psi = 0$, we have, $y_1[-(1 + 2 + 4 \cdots + 2^{\psi-3}) + 2^{\psi-2}] = 2$. Solving, we get $y_1 = 2$. So, $y_1 + y_2 + \cdots + y_\psi = 2 + 4 + 8 + \cdots + 2^{\psi-1} + 2 = 2^\psi$.

**Lemma 2.** For the patterns consisting of all 0’s or all 1’s, the expected number of trial is $2^{\psi+1} - 2$.

**Proof.** We prove this only for the pattern 00 \cdots 0. In this case, the state matrix is,

$$
\begin{align*}
m_{1,1} &= 1 \\
m_{i+1,i} &= \frac{1}{2} & \text{for } 1 \leq i \leq \psi \\
nm_{i,\psi+1} &= \frac{1}{2} & \text{for } 2 \leq i \leq \psi + 1 \\
m_{i,j} &= 0 & \text{otherwise.}
\end{align*}
$$

Removing the first row and column and then subtracting from $I$, we have $X = I - Q$ as follows,

$$
\begin{align*}
x_{i,i} &= 1 & \text{for } 1 \leq i \leq \psi - 1 \\
x_{\psi,\psi} &= \frac{1}{2} \\
x_{i+1,i} &= -\frac{1}{2} & \text{for } 1 \leq i \leq \psi - 1 \\
x_{i,\psi} &= -\frac{1}{2} & \text{for } 1 \leq i \leq \psi - 1 \\
x_{i,j} &= 0 & \text{otherwise.}
\end{align*}
$$
As above, from the entries of the last row of $YX = I$, we obtain the following equations,

1. $y^k = 2^{k-1}y_1$ for $1 < k < \psi$
2. $y_1 - y_2 - \cdots - y_{\psi-1} + y_{\psi} = 1.$

Solving, we get $y_1 = 2$. So, $y_1 + y_2 + \cdots + y_{\psi} = 2 + 4 + 8 + \cdots + 2^\psi = 2^{\psi+1} - 2$.

From [31], it is clear that this expectation can not be less than $2^\psi$. So, $2^{\psi}$ is the minimum possible expectation, which is achieved by the patterns $100 \cdots 0$ or $011 \cdots 11$. Hence it can be observed that selection of the above mentioned patterned string will provide improved result.

6. Conclusion

In this paper, we have studied two lightweight stream ciphers: Fountain v1 and Lizard. We have proposed a zero-sum distinguishing attack on Fountain v1 with 189 initialization rounds. Further, we have observed a weakness in Fountain v1 under TMDTO attack. The TMDTO attack on Fountain v1 has online complexity $2^{110}$ and offline complexity $2^{146}$. Although the online complexity of our TMDTO attack is less than the security parameters claimed by the designer (designer’s claim $2^{112}$), its offline complexity is quite high. Hence our observation does not create a major security threat to Fountain v1. However, we believe that Fountain v1 requires some modifications to provide a full level of security. Finally, we have revisited the TMDTO attack on Lizard by Maitra et al. (IEEE Transactions on Computers, 2018) and proposed that some particular string will provide better results over the selection of any random string.

Acknowledgment

We would like to thank the reviewers for their valuable suggestions and comments, which considerably improved the quality of our paper.

Appendix A. State bits recovery

In Table 5, we summarize the equations and guessed bits which are used in our TMDTO attack on Fountain v1. The state bits $s_{i+24}^{(2)}$ are fixed as 0 for $0 \leq i \leq 35$, so the term $s_{i+24}^{(2)}s_{i+29}^{(1)}$ (as $s_{i+24}^{(2)}s_{i+29}^{(1)} = 0$) is not included in the equations given.

References

[1] Lightweight Cryptography Standardization, Available from: https://csrc.nist.gov/projects/lightweight-cryptography.
[2] F. Armknecht and V. Mikhalev, On lightweight stream ciphers with shorter internal states, International Workshop on Fast Software Encryption (FSE), LNCS, Springer, 9054 (2015), 451–470.
[3] J.-P. Aumasson, I. Dinur, W. Meier and A. Shamir, Cube testers and key recovery attacks on reduced-round MD6 and Trivium, International Workshop on Fast Software Encryption (FSE), LNCS, Springer, 5665 (2009), 1–22.
[4] S. H. Babbage, Improved “exhaustive search” attacks on stream ciphers, European Convention on Security and Detection, IET, (1995), 161–166.
[5] S. Banik, Some results on Sprout, International Conference on Cryptology in India (Indocrypt), LNCS, Springer, 9462 (2015), 124–139.
[6] S. Banik, T. Isobe, T. Cui and J. Guo, Some cryptanalytic results on Lizard, IACR Transactions on Symmetric Cryptology, 2017 (2017), 82–98.
### Table: Stream Cipher Performance Evaluation

| Keystream bit | Equation | Guessed bits | Recovered bit |
|---------------|----------|--------------|---------------|
| 20            | $4^{11} = 29 + 4^{14} + 4^{5} + 4^{7} + 4^{9} + 4^{12} + 4^{13} + 4^{14} + 4^{15} + 4^{16}$ | $4^{11}, 4^{14}, 4^{12}, 4^{15}$ | $4^{11}$ |
| 21            | $4^{12} = 4^{12} + 4^{14} + 4^{14} + 4^{15} + 4^{16} + 4^{17} + 4^{17} + 4^{18} + 4^{19} + 4^{20}$ | $4^{12}, 4^{14}, 4^{15}, 4^{17}, 4^{18}, 4^{19}, 4^{20}$ | $4^{12}$ |
| 22            | $4^{13} = 4^{13} + 4^{14} + 4^{16} + 4^{18} + 4^{19}$ | $4^{13}, 4^{14}, 4^{16}, 4^{18}, 4^{19}$ | $4^{13}$ |
| 23            | $4^{14} = 4^{14} + 4^{15} + 4^{17} + 4^{18} + 4^{19}$ | $4^{14}, 4^{15}, 4^{17}, 4^{18}, 4^{19}$ | $4^{14}$ |
| 24            | $4^{15} = 4^{15} + 4^{16} + 4^{17} + 4^{18} + 4^{19}$ | $4^{15}, 4^{16}, 4^{17}, 4^{18}, 4^{19}$ | $4^{15}$ |
| 25            | $4^{16} = 4^{16} + 4^{17} + 4^{18} + 4^{19}$ | $4^{16}, 4^{17}, 4^{18}, 4^{19}$ | $4^{16}$ |
| 26            | $4^{17} = 4^{17} + 4^{18} + 4^{19}$ | $4^{17}, 4^{18}, 4^{19}$ | $4^{17}$ |
| 27            | $4^{18} = 4^{18} + 4^{19}$ | $4^{18}, 4^{19}$ | $4^{18}$ |
| 28            | $4^{19} = 4^{19}$ | $4^{19}$ | $4^{19}$ |
| 29            | $4^{20} = 4^{20}$ | $4^{20}$ | $4^{20}$ |
| 30            | $4^{21} = 4^{21} + 4^{22} + 4^{23} + 4^{24} + 4^{25}$ | $4^{21}, 4^{22}, 4^{23}, 4^{24}, 4^{25}$ | $4^{21}$ |
| 31            | $4^{22} = 4^{22} + 4^{23} + 4^{24} + 4^{25} + 4^{26}$ | $4^{22}, 4^{23}, 4^{24}, 4^{25}, 4^{26}$ | $4^{22}$ |

[7] A. Biryukov and A. Shamir, Cryptanalytic time/memory/data tradeoffs for stream ciphers, *International Conference on the Theory and Application of Cryptology and Information Security (Asiacrypt)*, LNCS, Springer, 1976 (2000), 1–13.

[8] A. Biryukov, A. Shamir and D. Wagner, Real time cryptanalysis of A5/1 on a PC, *International Workshop on Fast Software Encryption (FSE)*, LNCS, Springer, 1978 (2000), 37–44.

[9] S. Dey, T. Roy and S. Sarkar, Some results on Fruit, Designs, Codes and Cryptography, Springer, 87 (2019), 349–364.

[10] I. Dinur and A. Shamir, Cube attacks on tweakable black box polynomials, *Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, LNCS, Springer, 5479 (2009), 278–299.

[11] M. F. Esgin and O. Kara, Practical cryptanalysis of full Sprout with TMD tradeoff attacks, *International Conference on Selected Areas in Cryptography (SAC)*, LNCS, Springer, 9566 (2015), 67–85.

[12] V. A. Ghafari and H. Hu, A new chosen IV statistical distinguishing framework to attack symmetric ciphers, and its application to ACORN-v3 and Grain-128a, *Journal of Ambient Intelligence and Humanized Computing*, Springer, 10 (2019), 2393–2400.

[13] V. A. Ghafari, H. Hu and Y. Chen, Fruit-80: A secure ultra-lightweight stream cipher for constrained environments, *Entropy, Multidisciplinary Digital Publishing Institute*, 20 (2018), 180.

[14] V. A. Ghafari, H. Hu and M. Alizadeh, Necessary conditions for designing secure stream ciphers with the minimal internal states, *IACR Cryptol. ePrint Arch.*, (2017), 765.
| Keystream bit | Equation | Guessed bits | Recovered bit |
|---------------|----------|--------------|---------------|
| z12           | s_{12} = s_{12} + s_{12} + s_{12} + s_{12} + s_{12} + s_{12} + s_{12} | s_{12} = s_{12} + s_{12} + s_{12} + s_{12} + s_{12} + s_{12} + s_{12} | s_{12} = s_{12} + s_{12} + s_{12} + s_{12} + s_{12} + s_{12} + s_{12} |
|               | z13      |              |               |
|               | z14      |              |               |
|               | z15      |              |               |
|               | z16      |              |               |
|               | z17      |              |               |
|               | z18      |              |               |
|               | z19      |              |               |
|               | z20      |              |               |
|               | z21      |              |               |
|               | z22      |              |               |
|               | z23      |              |               |
|               | z24      |              |               |

[15] J. Golić, Cryptanalysis of alleged A5 stream cipher, *International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, LNCS, Springer, 1233 (1997), 239–255.

[16] C. M. Grinstead and J. L. Snell, *Introduction to Probability*, American Mathematical Society, 2012.

[17] M. Hamann, M. Krause, W. Meier and B. Zhang, Design and analysis of small-state Grain-like stream ciphers, *Cryptography and Communications*, Springer, 10 (2018), 803–834.

[18] M. Hamann, M. Krause and W. Meier, LIZARD - A lightweight stream cipher for power-constrained devices, *IACR Transactions on Symmetric Cryptology*, 2017 (2017), 45–79.

[19] M. Hell, T. Johansson and W. Meier, Grain: A stream cipher for constrained environments, *International Journal of Wireless and Mobile Computing*, 2 (2007), 86–93.

[20] M. E. Hellman, A cryptanalytic time-memory trade-off, *IEEE Transactions on Information Theory*, 26 (1980), 401–406.

[21] V. Lallemand and M. N. Plasencia, Cryptanalysis of full Sprout, *Annual Cryptology Conference (Crypto)*, LNCS, Springer, 9215 (2015), 663–682.

[22] S. Maitra, N. Sinha, A. Siddhanti, R. Anand and S. Gangopadhyay, A TMDTO attack against Lizard, *IEEE Transactions on Computers*, 67 (2017), 733–739.
Some results on lightweight stream ciphers Fountain v1 & Lizard

Table 5. State bits recovery

| Keystream bit | Equation | Guessed bits | Recovered bit |
|---------------|----------|--------------|---------------|
| $z_{25}$      | $z_{17} = z_{25} + z_{41} + z_{30} + z_{38}$ | (3) (4) | $y_{36}$ |
|               | $+ z_{42} + z_{1} + z_{37}$ | $a_{1}$ |              |
| $z_{26}$      | $z_{37} = z_{26} + z_{42} + z_{39}$ | (3) (4) | $y_{37}$ |
|               | $+ z_{32} + z_{40} + z_{38}$ | $a_{2}$ |              |
| $z_{27}$      | $z_{17} = z_{27} + z_{41} + z_{30} + z_{38}$ | (3) (4) | $y_{38}$ |
|               | $+ z_{42} + z_{1} + z_{37}$ | $a_{3}$ |              |
| $z_{28}$      | $z_{37} = z_{28} + z_{41} + z_{30} + z_{38}$ | (3) (4) | $y_{39}$ |
|               | $+ z_{40} + z_{37}$ | $a_{4}$ |              |
| $z_{29}$      | $z_{40} = z_{29} + z_{42} + z_{39}$ | (3) (4) | $y_{40}$ |
|               | $+ z_{32} + z_{40} + z_{38}$ | $a_{5}$ |              |
| $z_{30}$      | $z_{17} = z_{30} + z_{41} + z_{30} + z_{38}$ | (3) (4) | $y_{41}$ |
|               | $+ z_{42} + z_{1} + z_{37}$ | $a_{6}$ |              |
| $z_{31}$      | $z_{17} = z_{31} + z_{41} + z_{30} + z_{38}$ | (3) (4) | $y_{42}$ |
|               | $+ z_{40} + z_{37}$ | $a_{7}$ |              |
| $z_{32}$      | $z_{40} = z_{32} + z_{42} + z_{39}$ | (3) (4) | $y_{43}$ |
|               | $+ z_{33} + z_{40} + z_{38}$ | $a_{8}$ |              |
| $z_{33}$      | $z_{40} = z_{33} + z_{42} + z_{39}$ | (3) (4) | $y_{44}$ |
|               | $+ z_{34} + z_{40} + z_{38}$ | $a_{9}$ |              |

[23] S. Maitra, S. Sarkar, A. Baksi and P. Dey, Key recovery from state information of Sprout: Application to cryptanalysis and fault attack, *IPSI Transactions on Advanced Research*, 12 (2016).
[24] S. Maitra, A. Siddhanti and S. Sarkar, A differential fault attack on Plantlet, *IEEE Transactions on Computers*, 66 (2017), 1804–1808.
[25] M. J. Mihaljević, S. Gangopadhyay, G. Paul and H. Imai, Internal state recovery of Grain-v1 employing normality order of the filter function, *IET Information Security*, 6 (2012), 55–64.
[26] M. J. Mihaljević, S. Gangopadhyay, G. Paul and H. Imai, Generic cryptographic weakness of k-normal Boolean functions in certain stream ciphers and cryptanalysis of Grain-128, *Periodica Mathematica Hungarica*, 65 (2012), 205–227.
[27] V. Mikhailiev, F. Armknecht and C. Müller, On ciphers that continuously access the non-volatile key, *IACR Transactions on Symmetric Cryptology*, 2016 (2016), 52–79.
[28] R. Posteaua, Related-key differential slide attack against Fountain V1, *Proceedings of the Romanian Academy*, Series A, 21 (2020), 61–68.
[29] S. Sarkar, S. Maitra and A. Baksi, Observing biases in the state: Case studies with Trivium and Trivias-e, *Designs, Codes and Cryptography*, 82 (2017), 351–375.
[30] Q. Wang, Y. Hao, Y. Todo, C. Li, T. Isobe and W. Meier, Improved division property based cube attacks exploiting algebraic properties of superpoly, *International Cryptology Conference (Crypto)*, LNCS, Springer, 10991 (2018), 275–305.
[31] D. Williams, *Probability with Martingales*, Cambridge Mathematical Textbooks, 1st Edition, Cambridge University Press, 1991.

[32] B. Zhang, Fountain: A lightweight authenticated cipher (v1), *NIST Lightweight Cryptography Competition*, (2019), 1, https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/fountain-spec.pdf.

[33] B. Zhang and X. Gong, Another tradeoff attack on Sprout-like stream ciphers, *International Conference on the Theory and Application of Cryptology and Information Security (Asiacrypt)*, LNCS, Springer, 9453 (2015), 561–585.

[34] B. Zhang, X. Gong and W. Meier, Fast correlation attacks on Grain-like small state stream ciphers, *IACR Transactions on Symmetric Cryptology*, 2017 (2017), 58–81.

Received September 2019; 1st revision March 2020; final revision September 2020.

E-mail address: ravianandsps@gmail.com
E-mail address: roydibyendu.rd@gmail.com
E-mail address: sarkar.santanu.bir@gmail.com