Cryptanalysis of the Stream cipher BEAN

Ågren, Martin; Hell, Martin

Published in:
[Host publication title missing]

DOI:
10.1145/2070425.2070432

2011

Cryptanalysis of the Stream Cipher BEAN

Martin Ågren
Dept. of Electrical and Information Technology,
Lund University
P.O. Box 118
SE-221 00 Lund, Sweden
Tel: +46 46 222 38 69
Fax: +46 46 12 99 48
martin.agren@eit.lth.se

Martin Hell
Dept. of Electrical and Information Technology,
Lund University
P.O. Box 118
SE-221 00 Lund, Sweden
Tel: +46 46 222 43 53
Fax: +46 46 12 99 48
martin.hell@eit.lth.se

ABSTRACT

BEAN is a recent stream cipher proposal that uses Feedback with Carry Shift Registers (FCSRs) and an output function. There is a sound motivation behind the use of FCSRs in BEAN as they provide several cryptographically interesting properties. In this paper, we show that the output function is not optimal. We give an efficient distinguisher and a key recovery attack that is slightly better than brute force, requiring no significant memory. We then show how this attack can be made better with access to more keystream. Already with access to 6 KiB, the 80-bit key is recovered in time 2^{73}.

Categories and Subject Descriptors

E.3 [**Data Encryption**]: Code Breaking

General Terms

Algorithms, Security

1. INTRODUCTION

The Linear Feedback Shift Register (LFSR) is a common building block in stream ciphers. This is partly due to its simple implementation, in particular in hardware, but also because of the random-like properties of the produced sequences. However, the inherent linearity in the produced sequence requires the stream cipher to also include one or more nonlinear components. The properties of these components, which can be e.g., S-boxes, Boolean functions or Nonlinear Feedback Shift Registers (NFSRs), are crucial to the security of the cipher. A Feedback with Carry Shift Register (FCSR) can be seen as an alternative to an LFSR. While sharing several cryptographically important properties with LFSRs, as well as being easily implementable, these are updated nonlinearly. This nonlinear update somewhat relaxes the requirements on the other building blocks.

SIN'11, November 14–19, 2011, Sydney, Australia.

There have been several FCSR-based stream ciphers proposed in the literature. One notable design is F-FCSR-H [1, 2] which was selected for the final portfolio in the eSTREAM project. The hardware performance is good and the design is very simple in that it only uses a linear filter together with one FCSR. The performance of, and interest in, the F-FCSR-H stream cipher have made it evident that FCSRs are attractive building blocks for stream ciphers, even though F-FCSR-H was later cryptanalyzed in [7].

Another stream cipher selected for the final portfolio in eSTREAM is Grain [9]. Grain is also very simple and is based on one LFSR, one NFSR and a nonlinear Boolean output function. It can be seen as a variant of a nonlinear filter generator in which the taps to the nonlinear Boolean output function are taken from both shift registers.

BEAN [11] is a more recent design, which is influenced by Grain but replaces both shift registers by FCSRs. There is a sound motivation behind this idea since the LFSR in Grain is used to provide large period and guaranteed random-like properties while the NFSR is used to provide nonlinearity. An FCSR combines both these properties, while still being very efficient in hardware. Thus, BEAN has two shift register components, both providing nonlinearity, large period and random-looking sequences. The attack on F-FCSR-H in [7] took advantage of the fact that 8 keystream bits were produced in each clocking of the register. As BEAN produces 2 keystream bits in 3 FCSR updates, that attack is not applicable to BEAN.

In this paper we give two attacks on the BEAN stream cipher. While the design idea can be well-motivated, as described above, the design of the output function allows for attacks. First, we give a distinguishing attack based on the low correlation immunity order of the output function. This attack is very efficient as it can distinguish the keystream of BEAN from a random sequence using only about 2^{17} keystream bits, or 16 KiB. Second, we give a key recovery attack that is based on information leakage in the output function. By guessing a carefully chosen subset of the state bits, a portion of the keystream can be used to verify the guess, resulting in a divide-and-conquer kind of attack on the state. We give a trade-off between computational complexity and keystream required in the attack. As an example, we can recover the state using 6 KiB of keystream and 2^{73} computations, each computation being as complex as testing one key.

With these attacks as background, we discuss the specific design choices made in the BEAN stream cipher and how

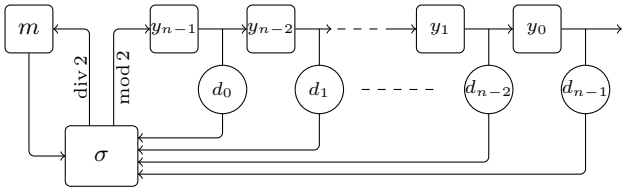

Figure 1: An FCSR in Fibonacci architecture.

the cipher can be improved in the future. Our results can thus be seen as a foundation for studying the security of BEAN-like stream ciphers.

While the BEAN specification [11] is sometimes ambiguous, for example as to whether the FCSRs are really FCSRs or merely LFSRs, the reference implementation can be used to clarify such uncertainties. We have chosen a very conservative approach and always used the more secure interpretation in these cases. It is known that constructing a decent stream cipher using only LFSRs and an output function (a non-linear combiner) is practically impossible [4]. Indeed, if BEAN is implemented with LFSRs instead of FCSRs, independent work [14] has shown that it is susceptible to algebraic attacks. Similarly, correlation attacks would be a natural approach to such weakened versions. This paper deals exclusively with the stronger version of BEAN that results from always making the sane choice in case of ambiguities.

This paper is organized as follows. In Section 2 we describe FCSRs and properties of Boolean functions. Section 3 describes the stream cipher BEAN, before Section 4 gives a distinguishing attack. Section 5 outlines the standard brute force attack so that we have an attack cost to compare our subsequent findings to. Section 6 then describes how to find the key slightly faster than brute force, and Section 7 introduces a time–data trade-off that needs more keystream but is significantly faster. Section 8 outlines what needs to be reconsidered in the BEAN design, before Section 9 concludes the paper.

2. PRELIMINARIES

In this section we give some background theory on FCSRs and Boolean functions. This theory will later be used in our attacks.

2.1 Feedback with Carry Shift Registers

A Feedback with Carry Shift Register (FCSR) is a device that computes the 2-adic expansion of the rational number p/q, with q odd. FCSRs were proposed by Klapper and Goresky and their cryptographic properties were thoroughly examined and determined in [10]. We refer to that paper for details on p-adic numbers.

An FCSR can be realized using either a Galois or a Fibonacci architecture. While most previous stream cipher designs use a Galois architecture, BEAN implements the FCSRs using a Fibonacci architecture. A relation between states in the two architectures can be found in [6]. A general Fibonacci FCSR is given in Figure 1 and we will refer to this implementation type in the sequel.

We will implicitly assume that any vector $(v_0, v_1, \ldots, v_{k-1}) \in \{0,1\}^k$ is also identified by the integer $\sum_{i=0}^{k-1} v_i 2^i$. The state consists of two parts, one main register

$$\mathbf{y} = (y_0, y_1, \ldots, y_{n-2}, y_{n-1})$$

and one memory m. The feedback is given by $(d_0, d_1, \ldots, d_{n-1})$.

In each update of the FCSR, the sum

$$\sigma = m + \sum_{i=0}^{n-1} y_i d_{n-i-1}$$

is computed and the state is updated as

$$m = \sigma \text{ div } 2, \quad (1)$$

$$\mathbf{y} = (y_1, y_2, \ldots, y_{n-1}, \sigma \bmod 2). \quad (2)$$

The FCSR automaton is completely determined by the connection integer $q = 1 - 2d$. The size of the main register is given by $n = \lfloor \log(|q| + 1) \rfloor$, i.e., the bit length of d. The state of the FCSR is associated with the integer p, which is given by

$$p = m2^n - \sum_{k=0}^{n-1} \sum_{j=0}^{k} d_{j-1} y_{k-j} 2^k,$$

where we define $d_{-1} = -1$. The output of the FCSR is then the 2-adic expansion of p/q and it can be shown, see e.g., [10], that the output is strictly periodic if and only if $0 \le p \le |q|$. Let the initial state be given by p. If the state at time t corresponds to the integer p (≥ 0), then the state p' at time $t+1$ corresponds to the integer

$$p' = 2^{-1} p \bmod q.$$

The output of the FCSR is given by $p \bmod 2$, and thus the tth output is given by $(2^{-t} p \bmod q) \bmod 2$ if p corresponds to the initial state. If $0 < p < |q|$, q odd, and q and p are coprime, then the period of the output sequence equals $\mathrm{ord}_q(2)$. Thus, the optimal choice of q is a negative prime with 2 being a primitive root. The FCSR automaton will then produce a maximum length sequence, also known as an l-sequence. The following result [3] will be useful to us:

LEMMA 1. *Using any FCSR of length n and starting in any state, it takes at most $n + 4$ FCSR updates to reach a strictly periodic state.*

Let $\mathcal{T}_y$ denote the set of register taps used when computing σ, i.e., $i \in \mathcal{T}_y$ if and only if $d_i = 1$, and let $|\mathcal{T}_y|$ be the cardinality of the set, i.e., the number of register taps used. It is straightforward to see, and it was formally shown in [10], that

$$m \le |\mathcal{T}_y| - 1$$

if the register is in a periodic state. Due to this, the carry can be realized using $\lceil \log |\mathcal{T}_y| \rceil$ bits. When we refer to an FCSR of length n, we implicitly consider its state to consist of in total $n + \lceil \log |\mathcal{T}_y| \rceil$ bits.

2.2 Boolean Function Properties

We briefly review some important properties of Boolean functions. We can represent an n-variable Boolean function $f(x_1, \ldots, x_n)$ by its *truth table*, i.e., a binary string of length 2^n, $f = [f(0,0,\ldots,0),\ f(0,\ldots,0,1),\ f(0,\ldots,1,0),\ \ldots,\ f(1,1,\ldots,1)]$. A Boolean function f is *balanced* if the truth table contains an equal number of 1's and 0's.

The *Hamming weight* of a binary string S is the number of ones in the string and is denoted $wt(S)$. The *Hamming distance* between two strings, S_1 and S_2, is denoted $d_H(S_1, S_2)$ and is the number of places where S_1 and S_2 differ.

A Boolean function can be represented as a polynomial over F_2, called the *algebraic normal form* (ANF),

$$f(x_1, \ldots, x_n) =$$
$$a_0 \oplus \bigoplus_{1 \le i \le n} a_i x_i \oplus \bigoplus_{1 \le i < j \le n} a_{ij} x_i x_j \oplus \ldots \oplus a_{12\ldots n} x_1 x_2 \ldots x_n,$$

where the coefficients $a_0, a_{ij}, \ldots, a_{12\ldots n} \in \{0, 1\}$. The *algebraic degree*, denoted $\deg(f)$, is the number of variables in the highest order term with non-zero coefficient. *Affine* Boolean functions are those with no terms of degree > 1 in the ANF. The set of all affine n-variable functions is denoted $A(n)$. The *nonlinearity* of an n-variable Boolean function is the minimum distance from the set of all n-variable affine functions,

$$nl(f) = \min_{g \in A(n)} (d_H(f, g)).$$

The correlation immunity of a Boolean function is a measure of to which degree its output is correlated to a subset of its inputs. For an m^{th} order correlation immune Boolean function, the mutual information between the output and a subset of at most m inputs is zero. If the function is balanced, it is called m-resilient.

The *Walsh transform* can be used to describe many properties of a Boolean function. Let $x = (x_1, \ldots, x_n)$ and $\omega = (\omega_1, \ldots, \omega_n)$ both belonging to $\{0, 1\}^n$ and $x \cdot \omega = x_1\omega_1 \oplus \ldots \oplus x_n\omega_n$. The *Walsh transform* of $f(x)$ is a real valued function over $\{0, 1\}^n$ which is defined as

$$W_f(\omega) = \sum_{x \in \{0,1\}^n} (-1)^{f(x) \oplus x \cdot \omega}. \quad (3)$$

A function f is balanced if and only if $W_f(0) = 0$. The nonlinearity of f is given by

$$nl(f) = 2^{n-1} - \frac{1}{2} \max_{\omega \in \{0,1\}^n} |W_f(\omega)|.$$

A function is m-resilient if and only if its Walsh transform satisfies $W_f(\omega) = 0$, for $0 \le wt(\omega) \le m$.

The *best* linear approximation, $\ell(f)$, of f produces a sequence that is correlated to the output of the function as

$$\Pr(f(\cdot) = \ell(f)) = \frac{1}{2}(1 + \varepsilon), \qquad \varepsilon = 1 - \frac{nl(f)}{2^{n-1}}. \quad (4)$$

In a distinguishing attack, nonlinear building blocks, e.g., Boolean functions, are approximated by linear blocks and some added noise. The nonlinearity $nl(f)$ is then a measure of how good the best approximation is. However, linear approximations other than the best ones could be more favorable to consider due to the cipher's internal structure. Typically, approximations with as few terms as possible might allow an attack that is much more successful than using the best approximation. Generalizing (4), ε for a linear approximation is given by

$$\varepsilon = \frac{W_f(\omega)}{2^n}$$

where ω corresponds to the linear approximation, see (3). The least number of terms in a linear approximation with $|\varepsilon| > 0$ is $m + 1$ where m is the resiliency of the Boolean function.

Table 1: Truth table of the output Boolean function used in BEAN.

0	0	0	1	1	0	1	0	0	1	1	1	0	1	0	1
0	1	1	0	1	0	1	1	0	1	1	0	0	0	1	0
1	1	0	0	1	0	1	0	0	0	0	1	0	1	1	1
0	0	1	1	0	1	0	0	1	1	1	0	1	1	0	0

3. BEAN SPECIFICATION

BEAN is very similar to Grain in that it consists of two shift registers and one output function, taking input from both registers. The size of the secret key is 80 bits. While Grain uses one LFSR and one NFSR, BEAN instead has two FCSRs, both implemented in Fibonacci architecture. These are denoted FCSR-I and FCSR-II, see Figure 2. An overview of the design of BEAN is given in the design document [11]. In order to avoid any ambiguity or misinterpretation of the specification, we have also studied the implementation provided by the designers.

3.1 Keystream Generation

Both FCSRs are 80 bits in size, i.e., the same as the key size. We denote the state of FCSR-I at time instance i by $\mathbf{B}^i$ and correspondingly the state of FCSR-II at time instance i by $\mathbf{S}^i$. Thus we have

$$\begin{aligned} \mathbf{B}^i &= (\mathbf{b}^i, m_b^i) = (b_i, \ldots, b_{i+79}, m_b^i), \\ \mathbf{S}^i &= (\mathbf{s}^i, m_s^i) = (s_i, \ldots, s_{i+79}, m_s^i), \end{aligned}$$

and FCSR-I is updated according to

$$\begin{aligned} \sigma_b &= b_{i+62} + b_{i+51} + b_{i+38} + \\ &\quad b_{i+23} + b_{i+13} + b_i + m_b^i, \\ b_{i+80} &= \sigma_b \bmod 2, \\ m_b^{i+1} &= \left\lfloor \frac{\sigma_b}{2} \right\rfloor. \end{aligned}$$

FCSR-II is updated in a similar way, but using other tap positions. The set of tap positions for the feedback for FCSR-I are given by $\mathcal{T}_b = \{17, 28, 41, 56, 66, 79\}$ and for FCSR-II they are given by $\mathcal{T}_s = \{1, 2, 3, 78\}$. Thus, m_b and m_s are realized using 3 and 2 bits respectively.

A Boolean function $f(x_1, \ldots, x_6)$ is used to produce the keystream. This is given as a 6-to-4-bit S-box in the original description [11] but as only one bit from each word is taken as output, it is easier to analyze it if we see it as a 6-to-1 Boolean function. The truth table of f is given in Table 1. The keystream bits $z_0, z_1, \ldots$ are then given by

$$z_{2i} = f(b_{i+23}, b_{i+73}, s_{2i+5}, s_{2i+9}, s_{2i+29}, b_{i+51}), \quad (5)$$
$$z_{2i+1} = f(b_{i+23}, b_{i+73}, s_{2i+6}, s_{2i+10}, s_{2i+30}, s_{2i+68}). \quad (6)$$

The algebraic normal form of f is

$$\begin{aligned} f(\cdot) = {} & x_1 \oplus x_4 \oplus x_1x_2 \oplus x_1x_3 \oplus x_1x_4 \oplus x_1x_5 \oplus x_2x_5 \oplus x_2x_6 \oplus \\ & x_3x_4 \oplus x_3x_5 \oplus x_3x_6 \oplus x_4x_6 \oplus x_5x_6 \oplus x_1x_2x_5 \oplus \\ & x_1x_2x_6 \oplus x_1x_3x_4 \oplus x_1x_3x_6 \oplus x_1x_4x_5 \oplus x_1x_5x_6 \oplus \\ & x_2x_3x_5 \oplus x_2x_3x_6 \oplus x_2x_4x_5 \oplus x_2x_4x_6 \oplus x_2x_5x_6 \oplus \\ & x_3x_4x_5 \oplus x_3x_4x_6 \oplus x_4x_5x_6 \oplus x_1x_2x_3x_5 \oplus x_1x_2x_3x_6 \oplus \\ & x_1x_2x_4x_5 \oplus x_1x_2x_4x_6 \oplus x_1x_2x_5x_6 \oplus x_1x_3x_4x_5 \oplus \\ & x_1x_3x_4x_6 \oplus x_1x_3x_5x_6 \oplus x_1x_4x_5x_6 \oplus \\ & x_1x_2x_3x_4x_6 \oplus x_1x_2x_4x_5x_6. \end{aligned}$$

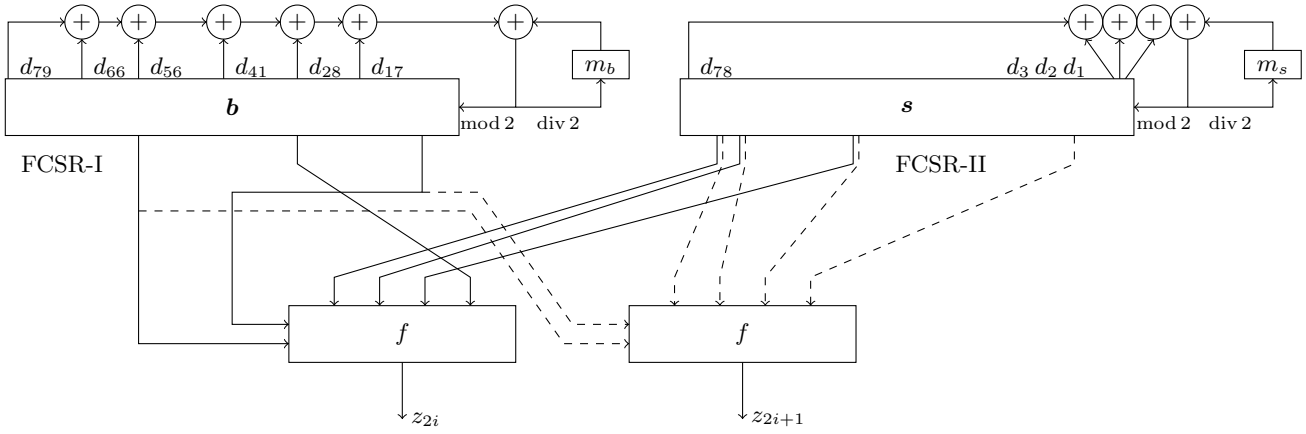

Figure 2: An overview of BEAN. Non-zero d_i's are named next to the corresponding taps.

Algorithm 1 — BEAN Keystream Generation

Input: initialized FCSR-I and FCSR-II
Output: 2-bit keystream word

Repeat until enough keystream bits are generated {
    Output an even numbered keystream bit
    Update FCSR-II
    Output an odd numbered keystream bit
    Update FCSR-I and FCSR-II
}

Figure 3: BEAN keystream generation.

Referring to Table 1, the first two bits determine the row, while the last four bits determine the column from which the keystream bit is taken (the top left bit corresponds to the all-zero input). The keystream generation algorithm is given in Figure 3. We will sometimes refer to the function f in the representation in Table 1 by referring to "rows" and "columns". This approach arises naturally as the row is always selected by **B**, while the column is (almost) exclusively selected by **S**.

3.2 BEAN Initialization

BEAN is initialized by loading the key $k = (k_0, k_1, \ldots, k_{79})$ into the registers as

$$b_i = k_{i+81}, \quad i = -81, \ldots, -2$$
$$s_i = k_{i+81}, \quad i = -81, \ldots, -2$$

The carries are set to $m_s^{i-81} = m_b^{i-81} = 0$. Then, both FCSRs are initialized by updating them 81 times. After initialization, the registers contain $b_0, b_1, \ldots, b_{79}$ and $s_0, s_1, \ldots, s_{79}$ respectively and keystream is ready to be produced according to Algorithm 1.

4. A DISTINGUISHING ATTACK ON BEAN

In this section we give a very efficient distinguishing attack on BEAN. A distinguisher takes a stream of bits as input and decides whether it is most likely to have been generated by the cipher or if it looks random. Thus, it will output either BEAN or RANDOM. In [11], the designers performed several statistical tests, provided by NIST [13], on the keystream. The keystream showed no deviation from random behaviour when the tests were applied to sequences of about 2^{23} bits. However, these tests are generic and do not take the internal structure of the stream cipher into account. Taking the structure into account, we show that with as few as 2^{17} keystream bits, a deviation from random can be observed.

Recall the truth table of the Boolean output function given in Table 1. We write it as $f(x_1, x_2, x_3, x_4, x_5, x_6)$ where the input variables can be immediately translated to register bits for z_{2i} and z_{2i+1} in (5) and (6), respectively. Analysing the Boolean function and its Walsh transform, the following properties can be found.

Balanced	Yes
Algebraic Degree	5
Resiliency	0
Nonlinearity	22
Best Linear Approximations	$\ell_1(f)$: $1 + x_1 + x_4 + x_5$, $\ell_2(f)$: $x_2 + x_3 + x_5 + x_6$

Using $\ell_1(f)$ we can e.g., write

$$\Pr(z_{2i} = b_{i+23} \oplus s_{2i+9} \oplus s_{2i+29}) = \frac{1}{2}(1 - \frac{10}{32}). \quad (7)$$

The common approach in a linear distinguishing attack is then to find a relation in b_i and s_i variables that sum to zero, leaving only an expression involving keystream bits, see e.g., [5, 8]. However, the FCSRs are nonlinearly updated and it is very difficult to find such a relation unless the full period is considered. As the period is large, this attack is not applicable to BEAN. However, noticing that the resiliency of the Boolean function is 0, we know that there are biased linear approximations of weight 1. Studying the Walsh transform of the Boolean function we find that

$$\Pr(f(\cdot) = x_3 \oplus 1) = \Pr(z_{2i} = s_{2i+5} \oplus 1) = \frac{1}{2}(1 + 2^{-4}) \quad (8)$$

and, similarly,

$$\Pr(f(\cdot) = x_5 \oplus 1) = \Pr(z_{2i} = s_{2i+29} \oplus 1) = \frac{1}{2}(1 + 2^{-4}). \quad (9)$$

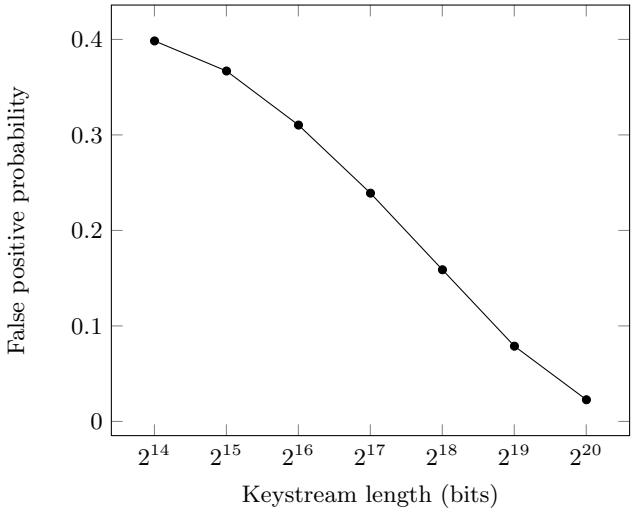

Figure 4: Using the distinguisher on more amounts of keystream material results in smaller probabilities of false positives. The decision thresholds were chosen such that false positives and false negatives were equally likely. Each trial was conducted over 2^{15} different keys.

We can rewrite (8) as

$$\Pr(z_{2i+24} = s_{2i+29} \oplus 1) = \frac{1}{2}(1 + 2^{-4}) \tag{10}$$

and combine this with (9) to get

$$\Pr(z_{2i} = z_{2i+24}) = \frac{1}{2}(1 + 2^{-8}),$$

using the piling-up lemma [12].

A common rule of thumb, see e.g., [8], is that approximately ε^{-2} samples are needed in order to detect this non-randomness and to correctly decide whether a given sequence is drawn from the cipher distribution or a uniform distribution. Thus, our distinguishing attack requires only about 2^{16} samples, or 2^{17} bits, to succeed. The attack has been simulated and the results have been verified, see Fig. 4.

5. A STANDARD BRUTE FORCE KEY RECOVERY

In the remainder of the paper, we will derive several known-plaintext key-recovery attacks.

Before giving the details of the attacks, we need to know the cost of the generic attack, which does not exploit the internals of BEAN: the exhaustive search. For this reason, we will first give a reasonable measure on the time (computations) spent in a brute force approach to find the secret key behind a given keystream.

For this, we will consider the initialization to consist of 80 clockings. This is simply because 80 is a slightly easier number to work with than 81. This simplification has virtually no effect on any of the measurements in this paper.

From the specification in Section 3, it can be seen that one bit of key never affects the contents of $\mathbf{S}$ as $d_{79} = 0$: k_0 is thrown out of the state in the very first update without affecting the feedback. Furthermore, $\mathbf{S}$ never affects $\mathbf{B}$. These properties will we used in our "trivial" brute force attack. We do not claim that these are flaws in and of themselves, and we suspect that any attacker attempting a standard brute force would use this property: to test two keys that only differ in k_0, we need to initialize $\mathbf{B}$ twice, but $\mathbf{S}$ only once.

Thus, during a standard brute force attack on BEAN, one expects to perform $2^{79} \times 80 + 2^{78} \times 80 = 2^{80} \times 60$ FCSR updates. This will be the reference point in this paper: when we claim that an attack requires e.g., time 2^{79}, we mean that it requires $2^{79} \times 60$ FCSR updates (and some additional work that is negligible in comparison).

5.1 On Data Requirements

We assume that the key is drawn from a uniformly random distribution. Looking at only the first bit of the keystream, half the keys will be valid candidates. Only after looking at 80 bits of keystream can the attacker expect to find a unique key. This, therefore, is the fundamental data requirement of any brute force attack.

We will give several attacks which require more data. The reasons why a cryptanalytic attack can sometimes require a considerable amount of data are many, but one common example is that the attacker has to wait for some special condition to arise in the internals of the cipher. It could be, as in the F-FCSR-H analysis [7], that the FCSR has to be in a special state, so that it behaves very nicely to the attacker. If such an event happens with probability $2^{-\gamma}$, the attacker can expect to look at 2^{γ} different "windows" of keystream before the event occurs. Note that, asymptotically, the size of the window has no effect on the data requirements. However, the attack cost has to incorporate actually looking at all parts of the keystream, checking if we are in such a special state. For this reason it is important to find an efficient abort rule that allows the attacker to discard a non-special state.

In this paper, we will exploit that the keystream leaks information about the state of the FCSRs (specifically, $\mathbf{S}$). The attacker's strategy will be to look at a small portion of keystream and make an informed guess on the state of BEAN. This can be done using some very small lookup tables. The guess will be correct with some probability $2^{-\gamma}$ so, similar to above, we will require data 2^{γ}, so that we can expect the attack to succeed.

6. A KEY RECOVERY ATTACK

In this section, we describe how structural flaws allow for a key recovery faster than brute force, i.e., cheaper than $2^{80} \times 60$ FCSR updates. The fundamental observation is that we can brute force the 79-bit key for $\mathbf{S}$ before taking care of $\mathbf{B}$. To see this, let us assume that we have a guess for $\mathbf{S}^j$, meaning that we can track this FCSR for all future time. We can then observe (z_{2i}, z_{2i+1}) and find contradictions.

EXAMPLE 1. Without loss of generality, we consider $2i = 0$. Let $(s_5, s_6, s_9, s_{10}, s_{29}, s_{30}, s_{68}) = (1, 0, 0, 0, 0, 0, 0)$. This means that the column for z_0 is picked as $(1, 0, 0, \cdot)$ where the last bit is unknown, while z_1 is picked from column $(0, 0, 0, 0)$. If we observe $z_1 = 1$, we can conclude that the row is $(1, 0)$. If we also see $z_0 = 1$, the row *cannot* be $(1, 0)$ meaning we have a contradiction.

To summarize this example, by looking at nine bits, we can conclude whether or not they pose a contradiction. We have implemented a simple scheme such as this. It uses a small lookup table of 2^9 bits and at average only needs to perform

Table 2: Integer representations of nine-bit vectors of state and output that cannot appear in BEAN.

9	13	24	34	41	45	55
56	66	87	94	136	156	157
163	168	170	178	188	189	192
195	207	210	256	259	263	267
271	274	277	282	329	333	344
380	385	386	394	400	406	407
414	415	456	476	477	480	493

15 FCSR updates before rejecting a bad guess. Table 2 lists the impossible state–output combinations that can be used to reject guesses of $\mathbf{S}$. Using the notation of Example 1, we have vectors $(s_5, s_6, s_9, s_{10}, s_{29}, s_{30}, s_{68}, z_0, z_1)$. E.g., the number $385 = (1, 0, 0, 0, 0, 0, 0, 1, 1)$ corresponds to the specific numerical values studied in Example 1.

We summarize the above as it is a central discovery relating to BEAN and will be used later in the paper.

CLAIM 1. *Given keystream and a guess for $\mathbf{S}^j$, the expected number of FCSR updates needed before rejecting this guess is* 20. *The additional work is negligible, as it only amounts to a small number of table lookups.*

In Claim 1 we have exaggerated the expected cost of the verification. This is for the benefit of even numbers, but can also be seen as a buffer for implementation overhead.

There is obviously one guess which will never be rejected, no matter how many times we update the FCSR: the correct guess. The strategy of the attack is now clear: first guess 79 bits of $\mathbf{S}$. Each individual guess is cheap to verify and once the correct 79 bits are found, the remaining key bit is trivial to brute force through $\mathbf{B}$ with additive, i.e., negligible cost. The cost is $2^{78} \times (80 + 20) = 2^{80} \times 25 \approx 2^{78.7} \times 60$ FCSR updates. This attack takes less than half the time of a brute force and requires no significant amount of memory.

Since $d_{79} = 0$ in $\mathbf{S}$, the remainder of this paper will use a slightly modified definition of $\mathbf{s}^i$:

$$\mathbf{s}^i = (s_{i+1}, \ldots, s_{i+79}).$$

We redefine $\mathbf{S}^i$ correspondingly. The FCSR is still the same in a mathematical sense, only the state is one bit smaller.

7. A TIME–DATA TRADE-OFF

The major cost of the above key recovery lies in the guess of key for $\mathbf{S}$: this requires us to perform 2^{79} initializations of $\mathbf{S}$ and some clockings in order to determine whether the state is correct or not. We would like to be able to make better guesswork. That is, rather than each guess succeeding with probability 2^{-79}, we would like to do significantly better.

This requires us to look at keystream and make informed guesses from it. Note that we want to guess $\mathbf{S}$ in the middle of operations and cannot assume $m_s = 0$ but have to find these two bits as well. Thus, it appears as if we need to guess 81 bits. All in all, we aim to recover a 164-bit state faster than 2^{79}. To succeed with this, we need several observations which we describe below.

7.1 Guessing S

In Section 7.3, we will show that it is straightforward to recover $\mathbf{B}$ once $\mathbf{S}$ is known. Thus, we first investigate how to recover $\mathbf{S}$.

First, we study how to verify a guess of $\boldsymbol{s}$ and recover m_s. We simply try all four possibilities for the two bits of carry, which allows us to track $\mathbf{S}$ for all future time. We can then use Claim 1 to reject three or four guesses. Note in particular how rejecting all four guesses costs $4 \times 20 = 80$ FCSR updates. The rest of this section will investigate how we can come up with a good guess for $\boldsymbol{s}$.

Assume we have a guess on the left-most bit of $\boldsymbol{s}$ and the carry m_s. There are in total eight such guesses. However, as we calculate the new state using $\sigma_s = s_{i+1} + s_{i+76} + s_{i+77} + s_{i+78} + m_s^i$, knowing $(s_{i+76}, s_{i+77}, s_{i+78})$, we can only produce $|\mathcal{T}_s| + 1 = 5$ distinct $\mathbf{S}$ in the next time step. Thus, rather than guessing 79 bits and guessing the carry 4 times, we can guess 78 bits and then the carry (rather, the rest of the information) 5 times. As an example of this, note that $(s_{i+1}, m_s^i) = (1, 1)$ produces the same new state as $(s_{i+1}, m_s^i) = (0, 2)$, where we have moved the 1 from s_{i+1} into the carry.

We can generalize the above as the following lemma. Its precise formulation is specific to the special nature of $\mathbf{S}$, where s_{i+0} is included neither in the update rule nor the redefined state, but it can be easily stated for the more common type of FCSR.

LEMMA 2. *Let $\mathbf{S}$ be an FCSR of length n using taps $\mathcal{T}_s$ and with state $\mathbf{S}^i = (s_{i+1}, \ldots, s_{i+n-1}, m_s^i)$. Let $l = \max \mathcal{T}_s = n - 2$ and $k = \max \mathcal{T}_s \backslash \{l\}$. For each j, $0 < j \leq l - k$ and each state $(s_{i+1}, \ldots, s_{i+j}, s_{i+j+1}, \ldots, s_{i+n-1}, m_s^i)$, there is an equivalent state $(0, \ldots, 0, s_{i+j+1}, \ldots, s_{i+n-1}, m_s^i + \sum_{k=0,\ldots,j-1} 2^k s_{i+1+k})$. The states are equivalent in the sense that they produce the same future state $(s_{i+j+1}, \ldots, s_{i+j+n-1}, m_s^{i+j})$.*

Thus, having guessed $(s_{i+j+1}, \ldots, s_{i+n-1})$, we can assume $s_{i+1} = \ldots = s_{i+j} = 0$ and only need to find the carry $m_s^i \in \{0, \ldots, 2 + 2^j\}$ where we have extended the set of possible m_s^i.

This lemma is highly applicable to $\mathbf{S}$ in BEAN as the first two taps are $l - k = 78 - 3 = 75$ bits apart. This means we can guess $(s_{i+76}, \ldots, s_{i+79})$, assume $(s_{i+1}, \ldots, s_{i+75}) = \mathbf{0}$ and then recover the equivalent carry at cost $3 + 2^{75} \approx 2^{75}$. In total, this attack would require work $2^4(3 + 2^{75}) \approx 2^{79}$. Obviously, the gain of applying this trick to yet another bit quickly becomes unimpressive. Certainly, one can also claim that $2^{54}(3+2^{25}) \approx 2^{79}$, so the marginal gain of applying this trick to 75 rather than 25 bits is negligible.

In the following attack, we will guess 19 bits from $\mathcal{A} = \{s_{2i+30}, \ldots, s_{2i+104}\}$ with probability $> 2^{-14.5}$, brute force the rest of that set, assume all bits in $\mathcal{C} = \{s_{2i+26}, \ldots, s_{2i+29}\}$ are zero, and recover m_s^{2i+25} at cost $3 + 2^4$ where each cost unit is 20 FCSR updates. Thus, we recover the entire $\mathbf{S}^{2i+25}$ using time $2^{14.5}2^{56}(3 + 2^4) \times \frac{20}{60} \approx 2^{73}$ and data $\approx 2^{14.5} \times 2$ which is approximately 6 KiB. (The factor 2 shows up because we apply our strategy to windows of $\boldsymbol{s}$ beginning with even numbered bits.)

7.2 Guessing 19 Bits of s

In the above, we have given all the characteristics of the recovery of $\mathbf{S}$, except a very crucial part: how to guess 19 bits of $\boldsymbol{s}$ with probability at least $2^{-14.5}$. For this, we will use that

$$\Pr(f(\cdot) = x_5 \oplus x_6) = \frac{38}{64} = \frac{1}{2}(1 + \frac{6}{32}), \qquad (11)$$

as this linear approximation allows us to only involve bits of $\boldsymbol{s}$ (cf. (7)).

Now, each time we observe a bit $z_{2(i+n)+1}$, we can make an educated guess as to the value of $s_{2(i+n)+30} \oplus s_{2(i+n)+68}$, thanks to (11). Due to this, we can make a correct guess on $\mathcal{B} = \{s_{2i+30+2n} | n = 0, 1, \ldots, 18\}$ with probability

$$\left(\frac{38}{64}\right)^{19} \approx 2^{-14.3}.$$

Indeed, the probability of making a correct guess has been found experimentally as approximately $2^{-14.4}$. Observe that we brute force 56 bits from $\mathcal{A}$ while the other 19 bits follow directly from these together with the keystream $\boldsymbol{z}$.

7.3 Find B Given S

Assume that we have the correct $\mathbf{S}$, so that we can track this FCSR for all future time. Our aim is to recover $\mathbf{B}$. The strategy is to first derive about 20 bits in a window of eighty bits of $\boldsymbol{b}$. A concluding brute force will require guessing 63 bits, so that the total cost is negligible to the previous parts of the attack.

Note that for z_{2i+1}, we know precisely which *column* is used in f. There are 10 unbalanced columns in Table 1 and with probability $\frac{1}{4}$, one of these columns uniquely identifies a row, i.e., two bits of $\boldsymbol{b}$. Thus we recover two bits of $\boldsymbol{b}$ with probability

$$\frac{10}{16} \cdot \frac{1}{4} = \frac{5}{32}.$$

More likely, with probability $\frac{6}{16}$, we can learn one bit of $\boldsymbol{b}$, either as a "real" bit b_j, or as a parity bit $b_j \oplus b_k$. While we can also get non-linear equations, we ignore this here.

We have implemented this part, and have found that with probability .5, at least 20 bits can be recovered. Similarly, with probability .022, we recover at least 30 bits. If we are unlucky and only recover some small number of bits from $\boldsymbol{b}$, we can fast-forward in $\mathbf{S}$ and $\boldsymbol{z}$ and make another try. Within just a few trials, we can find enough bits to make the total recovery cost of $\mathbf{B}$ negligible.

7.4 A General Conclusion

Let us use Lemma 2 on j bits. One can see that $(3 + 2^j)\frac{20}{60} < 2^{j-1}$ for $j > 2$. If we are able to guess k of the remaining bits with probability 2^{-p}, we can construct an attack that requires $2^p \times 2$ data and time

$$2^p 2^{79-j-k}(3 + 2^j)\frac{20}{60} < 2^{78+p-k}.$$

As an example, above we found 19 bits with probability $2^{-14.5}$. If we could similarly guess 30 bits with probability 2^{-23}, there would be an attack requiring 2 MiB of data and time 2^{71}. In the same fashion, there would be an attack requiring 2 GiB and time 2^{67}.

7.5 Recovering the Key

The obvious approach is to reverse the cipher from the recovered state. By reversing both FCSRs to their respective states $n + 4 = 84$ states after key loading, we know from Lemma 1 that these particular states, $\mathbf{B}^3$ and $\mathbf{S}^3$, were visited by the BEAN FCSRs shortly after initialization finished.

We can then revert $\mathbf{B}$ all the way back to $\mathbf{B}^{-81}$. The state we then reach was not necessarily (in fact, not likely) the state that the FCSR was put into at key loading. We are on the cycle, while the state we are looking for is at some tail leading into it. Nonetheless, we can expect the absolute majority of bits to be correct. A first approach is to use Lemma 2 to try to get the carry to be 0, but this only gives the correct key with a low probability. We will refer to the state of $\boldsymbol{b}$ that we reach in this way, as the key stub.

We can use this key stub as the basis for a brute force through the register, creating key candidates: by changing more and more bits, from left to right, we eventually reach the true key and will find that both registers, when loaded with this key candidate, reach $\mathbf{B}^3$ and $\mathbf{S}^3$, respectively. While the worst case scenario is that we must brute force the entire key, simulations have shown that we can expect to find the correct key after trying $\approx 2^4$ key candidates. Lemma 1 really describes the worst case behaviour [3]: despite performing 2^{35} tests, we never found a situation where we needed to try more than 2^{25} key candidates, which suggests that there is only some very small probability that this part of the attack becomes non-marginal.

There are several ways of improving this step, but the cost is already negligible compared to the state recovery.

Summarizing, we outline the complete attack in Figure 5. We use three sets of bits,

$$\mathcal{A} = \{s_{2i+30}, \ldots, s_{2i+104}\},$$
$$\mathcal{B} = \{s_{2i+30+2n} | n = 0, 1, \ldots, 18\},$$
$$\mathcal{C} = \{s_{2i+26}, \ldots, s_{2i+29}\}.$$

In order to completely formalize the attack, we would need to introduce several indices and variables, so instead we only give the broad picture.

Algorithm 2 — Time–Data Trade-Off Attack

Input: keystream
Output: corresponding key

```
Until satisfied {
    Look at a window of keystream z_{2i}, ...
    For each guess of bits in A\B
        Guess bits in B using (11)
        Assume all bits in C are zero
        For each m_s ∈ {0, ..., 2 + 2^4}
            Clock S, comparing to z,
                    until contradiction or satisfied
}
Until satisfied {
    Derive bits from B, using S and z
    If at least 20 bits found: satisfied
    Else: fast-forward a few clocks
}
Brute force the rest of B
Recover key as in Section 7.5
```

Figure 5: The complete attack.

8. PROTECTING AGAINST THE ATTACKS

We believe that a crucial part in strengthening BEAN is selecting a better output function f. If the resiliency had been at least one, we could not have found the very straightforward distinguisher in Section 4. Furthermore, with resiliency at least two, it would not have been possible to

involve only two bits of $\boldsymbol{s}$ in each guess as we did in (11). It might also be necessary to let f depend on more than six variables. A third factor to consider might be suitability for hardware implementation.

One notable difference between BEAN and the various incarnations of Grain is that the latter use a feed-forward from the NFSR to the LFSR. One idea might thus be to strengthen BEAN by adding feed-forward from **S** to **B**. However, this does not protect from any of our attacks as they first derive **S** completely before turning to **B**. Also, doing this would alter the behaviour of **B** from well-known to unknown.

We suggest that any future design includes an IV, that can be transmitted in the clear and significantly simplifies key management. This approach is taken in most modern stream ciphers.

Also, one should reselect the feedback taps as there are some unfortunate properties of the current choices:

- One initial state bit of **S** is completely disregarded as $79 \notin \mathcal{T}_s$.
- The taps in **S** are clustered, as opposed to spread out somewhat evenly (cf. **B**).
- The numbers q are not optimal; as indicated in Section 2.1, q should ideally have certain number theoretic properties.

While we only exploited the first of these observations, a serious redesign should address these potential weaknesses.

8.1 Improving the Attack

We do not claim that our approach for deriving e.g., $\boldsymbol{s}$ is optimal. One can easily make assumptions on some bits of $\boldsymbol{s}$, increasing the probability of guessing correctly. By assuming some specific configuration of d bits, which should occur with probability 2^{-d}, one can make more advanced guesswork recovering the remaining bits with probability 2^{-e}, where e should be "small". The data complexity would then be 2^{d+e}, and we suspect that the time complexity can be lower than what we have found in this paper. However, to find the "optimal" attack, one needs to consider several properties of f, bits that reoccur in the equations, and build decision trees that allow the implementation to adapt its behaviour to the guesses already made.

9. CONCLUSION

We have seen that the non-linear function in BEAN, combined with other properties of the construction, allows for an efficient distinguisher and a key-recovery faster than brute force. We also show how access to more keystream allows for a faster key recovery. Already at a very modest 6 KiB, the 80-bit key is recovered in time 2^{73}. While the distinguisher requires slightly more data, and thus is information theoretically inferior to the distinguisher inherent in the key-recovery attack, it is of practical time complexity, making it interesting in its own right.

10. ACKNOWLEDGMENT

This work was supported by the Swedish Foundation for Strategic Research (SSF) through its Strategic Center for High Speed Wireless Communication at Lund. The authors are grateful for the insightful comments received from the anonymous reviewers.

11. REFERENCES

[1] F. Arnault and T. Berger. Design of new pseudo random generators based on filtered FCSR automaton. The State of the Art of Stream Ciphers, Workshop Record, SASC 2004, Brugge, Belgium, October 2004.

[2] F. Arnault, T. Berger, and C. Lauradoux. Update on F-FCSR stream cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/025, 2006. *http://www.ecrypt.eu.org/stream.*

[3] F. Arnault, T. Berger, and M. Minier. Some results on FCSR automata with applications to the security of FCSR-based pseudorandom generators. *IEEE Transactions on Information Theory*, 54(2):836–840, February 2008.

[4] A. Braeken and J. Lano. On the (im)possibility of practical and secure nonlinear filters and combiners. In B. Preneel and S. Tavares, editors, *Selected Areas in Cryptography—SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 159–174. Springer-Verlag, 2005.

[5] D. Coppersmith, S. Halevi, and C.S. Jutla. Cryptanalysis of stream ciphers with linear masking. In M. Yung, editor, *Advances in Cryptology—CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 515–532. Springer-Verlag, 2002.

[6] S. Fischer, W. Meier, and D. Stegemann. Equivalent representations of the F-FCSR keystream generator. The State of the Art of Stream Ciphers, Workshop Record, SASC 2008, Lausanne, Switzerland, February 2008.

[7] M. Hell and T. Johansson. Breaking the F-FCSR-H stream cipher in real time. In *Advances in Cryptology—ASIACRYPT 2008*, volume 5350/2008 of *Lecture Notes in Computer Science*, pages 557–569. Springer-Verlag, 2008.

[8] M. Hell, T. Johansson, and L. Brynielsson. An overview of distinguishing attacks on stream ciphers. *Cryptography and Communications*, 1(1):71–94, 2008.

[9] M. Hell, T. Johansson, and W. Meier. Grain - a stream cipher for constrained environments. *International Journal of Wireless and Mobile Computing, Special Issue on Security of Computer Network and Mobile Systems.*, 2(1):86–93, 2006.

[10] A. Klapper and M. Goresky. Feedback shift registers, 2-adic span, and combiners with memory. *Journal of Cryptology*, 10(2):111–147, 1997.

[11] N. Kumar, S. Ojha, K. Jain, and S. Lal. BEAN: a lightweight stream cipher. In *Proceedings of the 2nd international conference on Security of information and networks*, SIN '09, pages 168–171, New York, NY, USA, 2009. ACM.

[12] M. Matsui. Linear cryptanalysis method for DES cipher. In T. Helleseth, editor, *Advances in Cryptology—EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer-Verlag, 1994.

[13] NIST. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST Special Publication 800-22b, 2010.

[14] N. R. Pillai and Y. K. Lather. Algebraic attack on BEAN a lightweight stream cipher. To appear.