Differential Fault Analysis of Sosemanuk

Yaser Esmaeili Salehani, Aleksandar Kircanski, and Amr Youssef

Concordia Institute for Information Systems Engineering,
Concordia University
Montreal, Quebec, H3G 1M8, Canada

Abstract. Sosemanuk is a software-based stream cipher which supports a variable key length of either 128 or 256 bits and 128-bit initial values. It has passed all three stages of the ECRYPT stream cipher project and is a member of the eSTREAM software portfolio. In this paper, we present a fault analysis attack on Sosemanuk. The fault model in which we analyze the cipher is the one in which the attacker is assumed to be able to fault a random inner state word but cannot control the exact location of injected faults. Our attack, which recovers the secret inner state of the cipher, requires around 6144 faults, work equivalent to around 2^{48} Sosemanuk iterations and a storage of around $2^{38.17}$ bytes.

1 Introduction

The European Network of Excellence of Cryptology (ECRYPT) [12] stream cipher project, also known as eSTREAM [14], is a project that aimed to identify new promising stream ciphers. Sosemanuk [4] is a fast software-oriented stream cipher that has passed all the three phases of the ECRYPT eSTREAM competition and is currently a member of the eSTREAM Profile 1 (software portfolio). It uses a 128-bit initialization vector and allows keys of either 128-bit or 256-bits, whereas the claimed security is always 128-bits. The design of Sosemanuk (See Fig. 1) is based on the SNOW2.0 stream cipher [13] and utilizes elements of the Serpent block cipher [2]. Sosemanuk aims to fix weaknesses of the SNOW 2.0 design and achieves better performance, notably in the ciphers initialization phase. Also, the secret inner state of Sosemanuk is reduced when compared to SNOW 2.0 and amounts to 384 bits.

The preliminary analysis [4], conducted during the Sosemanuk design process, includes the assessment of the cipher with respect to different cryptanalytic attacks such as correlation attacks, distinguishing attacks and algebraic attacks. Public analysis followed and Sosemanuk was assessed in [1] by Ahmadi *et al.* where a guess-and-determine attack requiring 2^{226} operations and 2^4 keystream words was provided. Another improved guess-and-determine attack was presented by Tsunoo *et al.* in [24]. A correlation attack on Sosemanuk was presented by Jung-Keun Lee *et al.* [20] with a computational complexity of $2^{147.88}$ and success probability 99% to recover the initial secret inner state. The data requirement for the attack was relaxed by Cho *et al.* [9]. In 2009, Lin *et al.* [21] improved the guess-and-determine attack, achieving complexity of 2^4

A. Nitaj and D. Pointcheval (Eds.): AFRICACRYPT 2011, LNCS 6737, pp. 316–331, 2011.

word keystream using 2^{192} steps. Another guess-and-determine attack with time complexity 2^{176} was recently presented by Feng *et al.* in Asiacrypt 2010 [15].

In this paper, we present a fault analysis attack on SOSEMANUK. The fault analysis model adopted in the paper is the one in which the attacker is assumed to be able corrupt a random inner state register in between the iterations of the cipher but the attacker has no control or knowledge over which inner state register has been corrupted. Also, the attacker is assumed to be able to reinitialize the cipher with the same key and IV arbitrary number of times. The attack recovers the secret inner state without recovering the key and requires about 6144 faults, 2^{48} operations each equivalent to one SOSEMANUK iteration and the storage of about $2^{38.17}$ bytes.

The rest of the paper is organized as follows. In the next section, we provide a brief overview of fault analysis attacks. In Section 3, relevant details of SOSEMANUK are reviewed. An overview of the proposed attack is provided in Section 4. Details of the attack are described in Section 5 and Section 6. Finally, the conclusion is given in Section 7.

2 Fault Analysis Attacks

In fault analysis attacks, the cryptanalyst applies some kind of physical influence, such as ionizing radiation, on the internal state of the cryptosystem which influence the crypto-primitive execution or memory. By carefully studying the results of computations performed under such faults, an attacker can retrieve information about the secret key. In 1996, Boneh *et al.* [8] introduced fault analysis by describing an attack that targets the RSA public key cryptosystem and exploits a faulty Chinese Remainder Theorem computation to factor the modulus n. Subsequently, fault analysis attacks were extended to symmetric systems such as DES [7] and later to AES [11]. Fault analysis attacks became a more serious threat after cheap and low-tech methods of applying faults were presented [23].

Fault attacks against stream ciphers were introduced by Hoch *et al* [16], where attacks against LILI-128 and SOBER-t32 and RC4 were described. Other stream ciphers that have been analyzed in the fault analysis model include SNOW 3G [10], Trivium [17], HC-128 [19] and Rabbit [18,5]. The number of required faults in the above attacks varies depending on the assumed fault analysis model. In general, all models follow the one given in Armknecht *et al.* [3], which assumes that the attacker has access to the physical device, and that the attacker is able to reset the device to the same unknown initial settings as often as needed. However, different assumptions with respect to the amount of control the attacker has over the induced faults are utilized. For example, the attacker may have control over the location of the faulted memory register, or may be able to restrict to Hamming weight of the induced faults. For instance, Biham *et al.* [6] assumed a model in which the attacker can choose the exact location (register) of the fault which causes RC4 to enter a special inner state and makes its recovery a trivial task. Similarly, Armknecht *et al.* [3] described a fault analysis attack gainst SNOW 2.0 where they assumed that the fault occurs exactly in a particular register of the cipher. On the other hand, in the fault analysis of Trivium [17], it

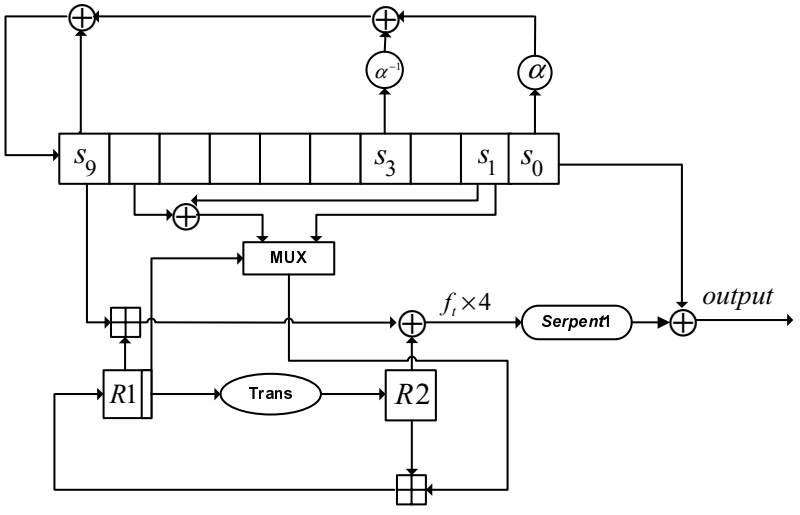

Fig. 1. Overview of the SOSEMANUK stream cipher

is assumed that the attacker has no control or knowledge over the fault position. Different assumptions also exist regarding the Hamming weight of induced faults. For instance, in [19] it is assumed that the fault causes a 1-bit flip in the inner state of the cipher, whereas in [6] it is assumed that the fault is localized in one byte of the inner state.

3 The Sosemanuk Specification

The following notation will be utilized throughout the rest of the paper:

- x^i: i-th bit of an n-bit word x
- $\boxplus, \times$: addition and multiplication modulo 2^{32}, respectively
- $\oplus$: bit-wise XOR
- $\lll$: left rotation defined on 32 bit values
- $|$: concatenation
- $X_i = f^i_{t+3}|f^i_{t+2}|f^i_{t+1}|f^i_t$: input value for i-th S-box applied in the Serpent1 function at some step t (the t value will be clear from the context). The Serpent1 function, shown in Fig. 2, is defined by 32 applications of S in the bit-slice mode, where

$$S = [8, 6, 7, 9, 3, 12, 10, 15, 13, 1, 14, 4, 0, 11, 5, 2]$$

  is the S-box used in the third S-box layer of the Serpent block cipher [2].
- $'$: Sign for denoting faulty cipher registers or output. For example s'_0 will denote the LFSR register s_0 in the faulty instance of the cipher.

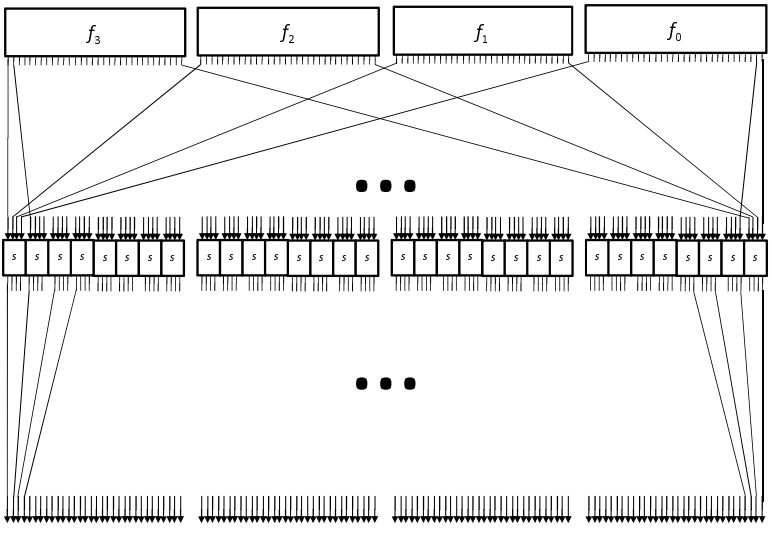

Fig. 2. The Serpent1 function

While the claimed security level of SOSEMANUK is 128 bits, it supports a variable key length of 128 or 256 bits and 128 bit initialization value. As depicted in Fig. 1, the secret inner state of SOSEMANUK consists of 12 32-bit words $(s_0, \ldots, s_9, R1, R2)$ and utilizes three main components to generate the keystream output: a linear feedback shift register (LFSR), a finite state machine (FSM) and an S-box-like function, Serpent1. To update the LFSR, the following recurrent relation is applied:

$$s_{t+10} = s_{t+9} \oplus \alpha^{-1} s_{t+3} \oplus \alpha s_t \tag{1}$$

where α is a root of the primitive polynomial $P(X) = X^4 + \beta^{23}X^3 + \beta^{245}X^2 + \beta^{48}X + \beta^{239}$ over GF(2^8) and β ia a root of the primitive polynomial $Q(X) = X^8 + X^7 + X^5 + X^3 + 1$ over GF(2).

The FSM update procedure is defined as follows:

$$R1_{t+1} = (R2_t \boxplus mux(lsb(R1_t), s_{t+1}, s_{t+1} \oplus s_{t+8})) \tag{2}$$

$$R2_{t+1} = (Trans(R1_t)) \tag{3}$$

where $mux(c, x, y) = \begin{cases} x \; if \; c = 0 \\ y \; if \; c = 1 \end{cases}$, $Trans(x) = (\mathrm{M} \times x) \lll 7$ and $M = 0x54655307$.

The FSM output at each step is defined by

$$f_t = (s_{t+9} \boxplus R1_{t+1}) \oplus R2_{t+1} \tag{4}$$

The inner state right after the initialization is denoted by $(s_0, \ldots, s_9, R1_0, R2_0)$. At each step, first the FSM is updated and the f_t and s_t values are preserved in

the internal buffer, then the LFSR is updated. Once every four steps, a 128-bit word is generated by

$$z_t|z_{t+1}|z_{t+2}|z_{t+3} = Serpent1(f_t|f_{t+1}|f_{t+2}|f_{t+3}) \oplus s_t|s_{t+1}|s_{t+2}|s_{t+3}. \quad (5)$$

For a more detailed description of SOSEMANUK, the reader is referred to [4].

4 The Attack Overview

In this section, we provide a high level overview of the proposed attack. According to our fault analysis model, the attacker is assumed to be able to re-initialize the cipher an arbitrary number of times. Furthermore, while we assume that each induced fault corrupts only one of the 12 inner state registers, the attacker does not know, and cannot control the position or the new value of the faulted register.

4.1 The Main Idea

The main idea of the attack can be explained as follows. In every SOSEMANUK iteration, 32 S-boxes are applied in the bit-slice mode as a part of the Serpent1 function. The first part of the attack restricts the input for each of the S-boxes by considering faults that occur at s_5 and s_4. Consider the case where the fault has been injected right after the SOSEMANUK initialization step and that it occurred in the register s_5. During the next cipher iteration in which the $z_0|z_1|z_2|z_3$ 128-bit keystream word is produced, the fault moves in the right-hand direction as the LFSR is clocked for 4 times. In particular, no faulty values participate in generation of f_0. Furthermore, since in every step, first the FSM is updated and then the f_t value is computed and finally the LFSR is clocked, f_1 and f_2 are computed without error and the fault affects only f_3. Now the non-faulty f_0, f_1, f_2 and the faulty f_3 enter the Serpent1 function. In the bit-slice mode, the Serpent1 function applies 32 S-boxes 4-bit inputs, where i-th bit comes from register f_i, $i = 0, \ldots 3$ (See Fig. 2). Thus, the input difference of all activated S-boxes will be equal to $0x8$ (1000 in binary). The attacker can then retrieve the corresponding S-box output difference and restrict the set of candidates for the S-box input-output values. When the fault occurs at register s_5, each S-box output will be faulted with probability $\frac{1}{2}$, which allows us to establish a criterion to recognize faults in register s_5. Similarly, in the case where the fault occurs at s_4, it propagates as shown in Figure 3 potentially affecting only f_2 and f_3. In other words, only the two most significant bits of every S-box input might be affected. Since a criterion for recognizing faults at s_4 can also be established, observing the output S-box differences for such faults also reduces the set of candidates for the S-box input-output values.

After the candidates for the S-box input-output values have been restricted, equation (5) is used to provide a restriction on the LFSR registers. From (1), it follows that the LFSR registers are not independent and restrictions on the LFSR registers can be coupled with the dependence of the LFSR registers to further prune the candidates for the s_t values. Finally, a guess and determine attack is used to find the rest of the inner state.

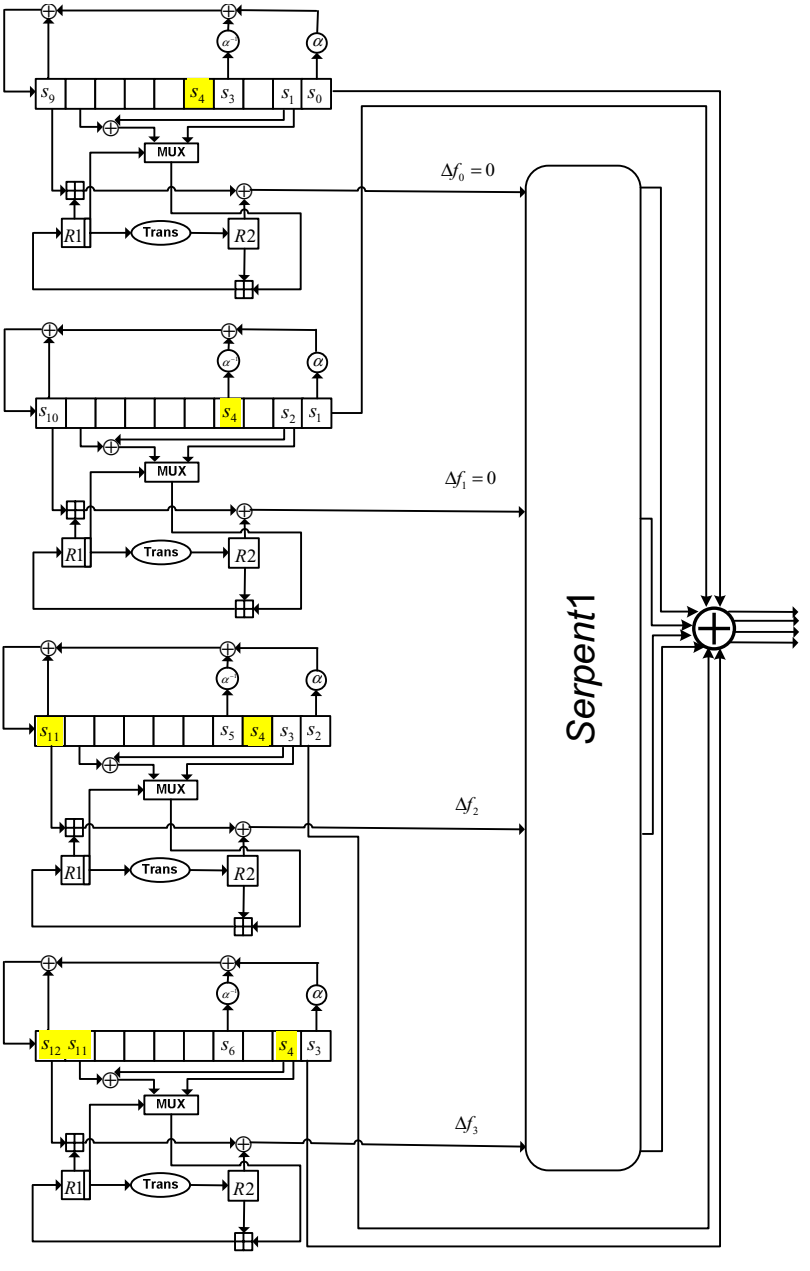

Fig. 3. The Δf values corresponding to the case where s_4 is faulted

4.2 The Steps of the Attack

The attack can be divided into two phases. The first phase collects faulty output in four different steps of the cipher execution and can be summarized as follows:

- For $l \in \{0, 1, 2, 4\}$
  - Repeat the steps below for m times
    - Reinitialize the cipher
    - Iterate for l times
    - Induce a fault, corrupting a random inner state register
    - Collect and store the keystream output word $z'_{4l}|z'_{4l+1}|z'_{4l+2}|z'_{4l+3}$

The second phase, which uses the collected information to uniquely determine the secret inner state, can be summarized as follows:

(1) Use the faulty outputs gathered in the first phase of the attack for $l \in \{0, 2, 4\}$ to reduce the number of candidates for (s_0, s_1, s_2, s_3), $(s_8, s_9, s_{10}, s_{11})$ and $(s_{16}, s_{17}, s_{18}, s_{19})$ to 2^{32} each. Then, use dependencies between the three four-plets imposed by relation (1) to further reduce the corresponding numbers of candidates (details are explained in Section 5)
(2) Similar to the previous step, using the information collected in the first phase of the attack for $l = 1$, reduce the number of candidates for (s_4, s_5, s_6, s_7) to 2^{32} (details are explained in Section 5)
(3) Apply the guess-and-determine strategy through the space reduced sets of candidates obtained by previous two steps to recover the complete inner state (details are explained in Section 6)

In the first phase of attack, data is collected for $l = 4$ and not for $l = 3$ since the LFSR registers candidate sets due to $l = 0$, $l = 2$ and $l = 4$ are correlated and allow further reduction. The reduction due to $l = 1$ is used later in the guess-and-determine attack.

5 Reducing the Number of Candidates for LFSR Registers (s_0, s_1, s_2, s_3) and $(s_8, s_9, s_{10}, s_{11})$

The starting number of candidates for the LFSR registers (s_0, s_1, s_2, s_3) and $(s_8, s_9, s_{10}, s_{11})$ is 2^{128} each. In this section, first we show how to reduce this number to 2^{32} and then, by exploiting the fact that the two register components are linked by relation (1), reduce it further to 2^{16}, each.

5.1 Recovering the S-Box Differences

Let SOSEMANUK be in state $t = 0$. From (5) and since $z_0|z_1|z_2|z_3$ is accessible to the attacker, it is evident that reducing the uncertainty for $f_0|f_1|f_2|f_3$ leads to reducing the uncertainty of $s_0|s_1|s_2|s_3$. In this subsection, the $f_0|f_1|f_2|f_3$ value is constrained by calculating the S-box input-output differences using the faulty information. Since the algorithms below are also applied to constraint $f_4|f_5|f_6|f_7$,

$f_8|f_9|f_{10}|f_{11}$ and $f_{16}|f_{17}|f_{18}|f_{19}$, these algorithms are specified for general time t and will be used for $t \in \{0, 4, 8, 16\}$.

Define δ_i and Δ_i by

$$\delta_i = S(X_i \oplus 0\text{x}8) \oplus S(X_i),$$
$$\Delta_i = \{S(X_i \oplus 0\text{x}4) \oplus S(X_i), S(X_i \oplus 0\text{x}c) \oplus S(X_i)\}$$

for every $i = 0, \ldots 31$. Algorithm 1 and Algorithm 2, described below, are used to recover δ_i and Δ_i, respectively, for each $i = 0, \ldots 31$.

In what follows, the probability distribution of the number of non-activated S-boxes in the SOSEMANUK output is analyzed. In particular, probabilities of the event that there will be more than 16 non-activated S-boxes are estimated under different assumptions about the location of the fault. For that purpose, let $0 \leq n \leq 32$ be a random variable which denotes the number of S-boxes that are *not* active in the application of the 32 S-boxes of Serpent1 in some steps of a faulty SOSEMANUK instance. Consider for example the probability that a particular S-box will not be activated given that the fault has occurred at s_0. In that case, only the 3 most significant bits of the S-box input may be corrupted. Note that, due to (5) by which the corrupted s_0 is XOR-ed to the least significant bits of each S-box, it may also happen that the difference in the S-box output caused by the S-box input cancels out. However, such a possibility has been ruled out by exhaustively checking that for each S-box input value it is not possible to cause a difference only in the least significant bit of the S-box output by any of the differences in the 3 most significant bits of the input. Thus, the probability that the particular S-box has not been activated is 2^{-3}. Now, it is clear that variable $n \sim B(2^{-3}, 32)$, i.e., n follows binomial distribution with parameters $p = 2^{-3}$ and $n = 32$. According to the binomial distribution, $P[16 \leq n \leq 31] = \sum_{i=16}^{31} \binom{32}{i} p^i (1-p)^{32-i} \approx 2^{-21}$. More generally, the distribution of n in terms of the fault position is given follows:

- $\{s_0\}$: $P[16 \leq n \leq 31] \approx 2^{-21}$ as explained above.
- $\{s_1, s_9, R1, R2\}$: all four S-box input bits may be corrupted. Hence, $n \sim B(2^{-4}, 32)$. For the fault position s_1, the possibility of cancelling out the S-box output difference has been ruled out the same way as in the case of s_0. Using the binomial distribution, it follows that $P[16 \leq n \leq 31]$ is negligible
- $\{s_8\}$: if $R1_0^0 = 0$, then, $n = 0$ with probability 1. Otherwise, all four S-box input bits may be corrupted and $n \sim B(2^{-4}, 32)$ and as for the previous case, $P[16 \leq n \leq 31]$ is negligible.
- $\{s_2, s_3\}$: only the least significant bit will certainly not be corrupted. For s_3, the cancellation of the S-box output difference is ruled out as in the case of s_0. In case of s_2, there exists one S-box input such that the S-box output difference can be cancelled out by inverting the second most significant bit ($S(1111) = S(1111 \oplus 1110) \oplus 0100$). Approximating $n \sim B(2^{-3}, 32)$ gives $P[16 \leq n \leq 31] \approx 2^{-21}$.
- $\{s_4\}$: the most significant two bits may be corrupted, from which it follows that $n \sim B(2^{-2}, 32)$. So, $P[16 \leq n \leq 31] \approx 0.002$

- $\{s_6, s_7\}$: no S-box input bits can be corrupted and thus $n = 32$ with probability 1
- $\{s_5\}$: Only the most significant bit of every S-box input may be corrupted. Thus $n \sim B(\frac{1}{2}, 32)$ and $P[16 \leq n \leq 31] = \sum_{i=16}^{31} \binom{32}{i} \frac{1}{2^i} \frac{1}{2^{(32-i)}} = 0.569$

From the above reasoning, it follows that when the fault does not occur at s_5, $P[16 \leq n \leq 31] \approx \frac{1}{11} \times 0.02 \approx 0.0018$, where $\frac{1}{11}$ is the probability that the fault occurred at s_4, given that it did not occur at s_5. On the other hand, if the fault occurred at s_5, the probability of event $16 \leq n \leq 31$ is equal to 0.569. This analysis indicates that one can decide whether the fault occurred at s_5 or not by verifying whether $16 \leq n \leq 31$, or not, respectively.

In Algorithm 1, keystream words for which $16 \leq n \leq 31$ are considered. Namely, once such a keystream word have been found, the values of *activated* S-boxes are used to learn about the corresponding δ_i values. According to the discussion above, if the fault indeed occurred at s_5, such differences necessarily represent the S-box output difference for the input difference equal to $0x8$. To diminish the possibility of false positives (event $16 \leq n \leq 31$ takes place, but the fault does not occur at s_5), the final output difference value is taken as the most frequent difference candidate taken over different faulty keystream words at the (fixed) SOSEMANUK step in question, for which $16 \leq n \leq 31$ holds.

Algorithm 1

- Initialize 32 multisets: $Cand_1(k) = \emptyset$, $k = 0, \ldots, 31$.
- For each faulty keystream word $z'_t|z'_{t+1}|z'_{t+2}|z'_{t+3}$, such that

$$16 \leq \#\{z_t'^i|z_{t+1}'^i|z_{t+2}'^i|z_{t+3}'^i = z_t^i|z_{t+1}^i|z_{t+2}^i|z_{t+3}^i : i = 0, \ldots 31\} \leq 31 \qquad (6)$$

  do:
  - For each $0 \leq k \leq 31$, if $d = z_t'^k|z_{t+1}'^k|z_{t+2}'^k|z_{t+3}'^k \oplus z_t^k|z_{t+1}^k|z_{t+2}^k|z_{t+3}^k$ is different than 0, add d to $Cand_1(k)$.
- Return the most frequent element in the multiset $Cand_1(i)$ as $\delta_i = S(X_i \oplus 0x8) \oplus S(X_i)$, for each $0 \leq i \leq 31$.

The overall number of required fault injections $m = 1536$ has been determined by incrementing m in steps of 128 and experimentally verifying that Algorithm 1 always recovers the correct $\delta_i = S(X_i \oplus 0x8) \oplus S(X_i)$, $i = 0, \ldots 31$ for 1000 randomly initialized instants of SOSEMANUK.

Algorithm 2 uses δ_i recovered by Algorithm 1 to find the sets Δ_i, $i = 0, \ldots 31$. In particular, the algorithm recognizes faulty keystream words that correspond to an error in register s_4 and then uses the S-box output differences in such keystream words to deduce Δ_i for $i = 0, \ldots 31$.

The criterion for recognizing faults in register s_4 is similar to the previously stated criterion for recognizing faults in s_5. However, instead of asking for 16 or more unactivated S-boxes, we expect to have more than 16 S-boxes which are either unactivated or with output difference equals to δ_i. Namely, let v be the number of S-boxes in one step of SOSEMANUK which are either not activated,

or activated by an input difference of $0x8$. The probability of the event that one S-box is either not activated, or activated by an input difference of $0x8$ depends on the location where the fault occurred. In case the error is in register s_4, the probability in question will be $\frac{1}{2}$ since in that case only the 2 most significant bits of the S-box input may be faulted and the input difference has to among $0x0$, $0x8$, $0xc$ and $0x4$ values. Thus, if the fault is in s_4, $v \sim B(\frac{1}{2}, 32)$, and $P[16 \leq v \leq 31] = 0.569$. On the other hand, if the fault occurs at some other register, say at $R1$, all four S-box input bits may be corrupted and the probability that the input difference will be either $0x8$ or $0x0$ is significantly smaller. Again, this gives a methodology to decided whether the fault occurred at s_4 or not by counting the number of S-boxes which reacted with difference of either δ_i (using the corresponding i) or 0. Once the faults due to an error in register s_4 are recognized, finding the sets Δ_i proceeds with the following logic. When a keystream word for which the event $16 \leq v \leq 31$ took place has been found, the output S-box differences which are not due to input difference of $0x8$ or $0x0$ have to be due to difference $0xc$ or $0x4$. Again, to diminish the possibility of false positives (i.e., $16 \leq v \leq 31$ but the fault does not occur at s_4), the final output set is taken as the set with two most frequent difference candidates for the difference taken over different faulty keystream words at the SOSEMANUK step in question for which $16 \leq v \leq 31$ holds.

Algorithm 2

- Initialize 32 multisets: $Cand_{2,3}(k) = \emptyset$, $k = 0, \ldots, 31$.
- For each faulty keystream output word $z'_t|z'_{t+1}|z'_{t+2}|z'_{t+3}$, such that

$$\begin{array}{c} 16 \leq \#\{z_t^{'i}|z_{t+1}^{'i}|z_{t+2}^{'i}|z_{t+3}^{'i} = z_t^i|z_{t+1}^i|z_{t+2}^i|z_{t+3}^i | i = 0, \ldots 31\} + \\ \#\{z_t^{'i}|z_{t+1}^{'i}|z_{t+2}^{'i}|z_{t+3}^{'i} \oplus z_t^i|z_{t+1}^i|z_{t+2}^i|z_{t+3}^i = \delta_i | i = 0, \ldots 31\} \leq 31 \end{array} \quad (7)$$

  where δ_i, $0 \leq i \leq 31$ has been recovered by Algorithm 1, do:
  - For each $0 \leq k \leq 31$, add each $d = z_t^{'k}|z_{t+1}^{'k}|z_{t+2}^{'k}|z_{t+3}^{'k} \oplus z_t^k|z_{t+1}^k|z_{t+2}^k|z_{t+3}^k$ such that $d \notin \{0, \delta_k\}$ to the multiset $Cand_{2,3}(k)$.
- Return the two highest occurring elements in the multiset $Cand_{2,3}(i)$ as the required two-element set Δ_i, for each i.

For the above choice of total number of faults $m = 1536$, Algorithm 2 always succeeded in recovering the sets Δ_i, $i = 0, \ldots 31$, for 1000 randomly initialized instants of SOSEMANUK.

5.2 Restricting the Number of Candidates for the LFSR Registers

In each SOSEMANUK step, in which a 128-bit keystream word is produced, according to (5), 32 4×4 S-boxes are applied. In the previous subsection, it has been shown how to use the faulty information to deduce the S-box output differences for certain input S-box differences. Naturally, these evaluated input-output differences impose a constraint on the actual input-output values. In this subsection, the sets of possible S-box input-output values are deduced and the effect of

Table 1. Determining the S-box input-output values based on sets δ_i and Δ_i

δ_i,Δ_i	i-th S-box input	i-th S-box output
5,{8,B}	0	8
9, {2,D}	2	7
3,{B,E}	4	3
F,{4,D}	6	A
5,{D,E}	8	D
9,{4,B}	A	E
3,{8,D}	C	0
F,{2,B}	E	5
7,{A,D}	{1,5,9,D}	{6,C,1,B}
D,{6,B}	{3,7,B,F}	{9,F,4,2}

the deduced input-output S-box values constraints on the number of candidates for the LFSR registers (s_0, s_1, s_2, s_3) is presented.

Having determined the δ_i value and the two-element set Δ_i by Algorithms 1 and 2, for each $0 \leq i \leq 31$, the actual input-output values for the S-box are deduced according to Table 1. As can be noted from the table, in case the S-box input is even, the input-output value can be deduced uniquely. On the other hand, in case when the S-box input value is odd, there exist four candidates for the S-box input-output.

Assuming a uniform distribution on the S-box input values, it is expected that the attacker will deduce 64 out of 128 output bits. For the remaining 64 bits, it will be composed out of 16 4-bit values, each restricted to 4 candidates. The overall number of candidates for the 128-bit value $Serpent1(f_0|f_1|f_2|f_3)$ is then $4^{16} = 2^{32}$. Since we have

$$z_0|z_1|z_2|z_3 = Serpent1(f_0|f_1|f_2|f_3) \oplus s_0|s_1|s_2|s_3 \tag{8}$$

and $z_0|z_1|z_2|z_3$ is known, it follows that there will be 2^{32} candidates for $s_0|s_1|s_2|s_3$.

The number of candidates for $s_4|s_5|s_6|s_7$, $s_8|s_9|s_{10}|s_{11}$ and $s_{16}|s_{17}|s_{18}|s_{19}$ can be restricted in a similar way. Namely, for that purpose, Algorithms 1 and 2 need to be applied using $z_4|z_5|z_6|z_7$, $z_8|z_9|z_{10}|z_{11}$ and $z_{16}|z_{17}|z_{18}|z_{19}$ and the faulty values obtained by the first phase of the attack described in Section 4 for $l = 1$, $l = 2$ and $l = 4$, respectively. Then, Table 1 is utilized to restrict the S-box input-output values occurring in steps $t = 1$, $t = 2$ and $t = 4$. Following the procedure explained in this section, it follows that $s_4|s_5|s_6|s_7$, $s_8|s_9|s_{10}|s_{11}$ and $s_{16}|s_{17}|s_{18}|s_{19}$ are expected to be restricted to 2^{32} candidates each.

5.3 Further Pruning of the LFSR Registers Candidates

In the previous subsection, the uncertainty for (s_0, s_1, s_2, s_3), $(s_8, s_9, s_{10}, s_{11})$ and $(s_{16}, s_{17}, s_{18}, s_{19})$ values has been reduced. In this subsection, we note that these three four-tuples of 32-bit values are not independent. Namely, according to (1), we have $s_{10} = s_9 \oplus \alpha^{-1} s_3 \oplus \alpha s_0$ and $s_{18} = s_{17} \oplus \alpha^{-1} s_{11} \oplus \alpha s_8$. These two relations

are used to further prune candidates for (s_0, s_1, s_2, s_3) and $(s_8, s_9, s_{10}, s_{11})$. More precisely, after the end of the process, the attacker is left with 2^{16} candidates for

$$(f_0, f_1, f_2, f_3, s_0, s_1, s_2, s_3, f_8, f_9, f_{10}, f_{11}, s_8, s_9, s_{10}, s_{11}) \tag{9}$$

The two relations from the previous paragraph can be rewritten as

$$\alpha^{-1} s_3 \oplus \alpha s_0 = s_{10} \oplus s_9 \tag{10}$$

$$\alpha^{-1} s_{11} \oplus \alpha s_8 = s_{18} \oplus s_{17} \tag{11}$$

Before stating the candidate reduction procedure, we note that the candidates for (s_0, s_1, s_2, s_3) are specified in a way which allows listing them in a table efficiently. In particular, the candidate set for (s_0, s_1, s_2, s_3) is specified by sets B_i, $i = 0, \ldots 31$, such that $s_0^i | s_1^i | s_2^i | s_3^i \in B_i$. Then, each element of the set $B_0 \times B_1 \times \ldots \times B_{31}$ specifies one (s_0, s_1, s_2, s_3) value. The sets of candidates for $(s_8, s_9, s_{10}, s_{11})$ and $(s_{16}, s_{17}, s_{18}, s_{19})$ can be transformed to a list in the same way and this property is used in step (1) and step (5) of the procedure below.

1. List all of the (s_0, s_1, s_2, s_3) and $(s_{16}, s_{17}, s_{18}, s_{19})$ candidates and call the two generated tables T_1 and T_3, respectively. Include also the columns containing (f_0, f_1, f_2, f_3) and $(f_{16}, f_{17}, f_{18}, f_{19})$ in T_1 and T_3, respectively. Create an empty table T.
2. Extend T_1 by adding a column with the left-hand side of equation (10).
3. Extend T_3 by adding a column with the right-hand side of equation (11).
4. Sort T_1 and T_3 by columns added in steps (2) and (3).
5. For each candidate for $(s_8, s_9, s_{10}, s_{11})$
   - 5.1. Calculate the left-hand side of equation (11). If there does not exists an element in T_3 such that (11) holds, go to the next $(s_8, s_9, s_{10}, s_{11})$ candidate (step (5)).
   - 5.2. Otherwise, calculate the right-hand side of equation (10) and find rows of T_1 for which (10) holds. For each such row, add the complete row of the form (9) to table T.

To find the expected size of table T, note that it is expected that 16 bits of the T_3 table column containing $s_{18} \oplus s_{17}$ value are constant, due to the fact that 16 out of 32 S-box inputs corresponding to $(s_{16}^i, s_{17}^i, s_{18}^i, s_{19}^i)$ have been recovered uniquely by the procedure in the previous subsection. On the other hand, no constant bits are expected to exist in $\alpha^{-1} s_{11} \oplus \alpha s_8$ values due to randomization resulting from multiplying by α and α^{-1}. Thus, about 2^{16} candidates for $(s_8, s_9, s_{10}, s_{11})$, with the corresponding $(f_8, f_9, f_{10}, f_{11})$, will pass the elimination step (5.1).

In step (5.2), the remaining 2^{16} candidates are joined with T_1, which contains 2^{32} rows, according to (10). Since there exists no fixed bits in the $\alpha^{-1} s_3 \oplus \alpha s_0$ column of T_1, it is expected that around 2^{16} will be present in the output of the join step, i.e., in table T. Since both T_1 and T_3 contain 9 32-bit words in each row and table T contains 16 32-bit words in each row, the required memory space for the previous procedure is $2 \times 2^{32} \times 9 \times 4 + 2^{16} \times 16 \times 4 = 2^{38.17}$ bytes. The computational cost is equal to sorting two tables of 2^{32} rows, executing a search

in a sorted table of length 2^{32} for 2^{32} times and finally executing a search for 2^{16} times in the sorted table of 2^{32} entries. By noting that sorting tables of length n takes $O(nlog(n))$ steps and that a binary search in the sorted table requires $O(log(n))$ steps, the overall cost is about $2^{32} \times 32 \times 2 + 2^{32} \times 32 + 2^{16} \times 32 = 2^{38.585}$ operations.

6 Recovering the Rest of the Inner State

In the previous subsections, we have reduced the LFSR complexity to 2^{32} candidates for (s_4, s_5, s_6, s_7) and 2^{16} candidates for the registers present in (9). In this subsection, a guess-and-determine like procedure that completes the secret inner state recovery is provided.

Let $R1_t^0$ denote the least significant bit of register $R1_t$. To recover s_4, s_5, $R1_0$ and $R2_0$, the following steps are applied:

- Pick a row from table T as a guess for (9)
- Determine s_4 from $s_4 = \alpha(\alpha s_1) \oplus \alpha(s_{10} \oplus s_{11})$ which holds due to (1) since s_1, s_{10} and s_{11} are known
- Guess $R1_0$ by fixing the register to one of the 2^{32} possible values.
- Determine:
  - $R2_0$, from $f_0 = (R1_0 \boxplus s_9) \oplus R2_0$
  - $R2_1$, from $R2_1 = Trans(R1_0)$
  - $R1_1$, from $R1_1 = R2_0 \boxplus (s_2 \oplus R1_0^0 \cdot s_9)$, which is another way to formulate (2)
  - $R2_2$, from $R2_2 = Trans(R1_1)$
  - $R1_2$, from $R1_2 = R2_1 \boxplus (s_3 \oplus R1_1^0 \cdot s_{10})$, which follows from (2)
  - $R2_3$, from $R2_3 = Trans(R1_2)$
  - $R1_3$, from $R1_3 = R2_2 \boxplus (s_4 \oplus R1_2^0 \cdot s_{11})$, which follows from (2)
  - s_{12}, from $f_3 = (R1_3 \boxplus s_{12}) \oplus R2_3$
  - s_5, from $s_{12} = s_{11} \oplus \alpha^{-1} s_5 \oplus \alpha s_2$

With a guess for (9) from the first step of the procedure above and having recovered s_4, s_5, $R1_0$ and $R2_0$, the only left unknown inner state registers are s_6 and s_7. To recover the remaining two registers, the table of 2^{32} candidates for (s_4, s_5, s_6, s_7) obtained in Section 5.2 is matched with newly found value for s_4, s_5, as follows. Consider the S-box input-output in the second iteration of SOSEMANUK, for which the input-output has not been recovered uniquely. For some $0 \leq i \leq 31$, $f_7^i|f_6^i|f_5^i|f_4^i$ and consequently, $S(f_7^i|f_6^i|f_5^i|f_4^i)$ can take 4 values as specified by Table 1. More precisely, rewriting (5) while isolating i-th S-box

$$z_7^i|z_6^i|z_5^i|z_4^i = S(f_7^i|f_6^i|f_5^i|f_4^i) \oplus s_7^i|s_6^i|s_5^i|s_4^i, \tag{12}$$

we have two options regarding the possible candidates. In other words, from the last two rows of Table 1, we have either

$$S(f_7^i|f_6^i|f_5^i|f_4^i) \in \{0110, 1100, 0001, 1011\} \tag{13}$$

or

$$S(f_7^i|f_6^i|f_5^i|f_4^i) \in \{1001, 1111, 0100, 0010\}. \tag{14}$$

Moreover, according to the procedure given in this subsection, the value of bits s_4^i, s_5^i has been determined uniquely. Since s_4^i and s_5^i are known, according to (12), the two least significant bits of $S(f_7^i|f_6^i|f_5^i|f_4^i)$ can be determined uniquely. Finally, due to the structure of sets (13) or (14), given information on the two least significant bits, all the 4 bits of $S(f_7^i|f_6^i|f_5^i|f_4^i)$ are uniquely determined. Presented reasoning uniquely determines the input-output for every S-box, from which, according to (12), s_7 and s_6 are determined uniquely, which completes the recovery of the whole secret inner state.

Now, the found secret inner state can be verified by comparing the actual SOSEMANUK output with the output produced by the recovered inner state. If a difference registered, the next guess for (9) and $R1_0$ is made and the procedure is repeated.

7 Summary and Conclusions

In this paper, a differential fault analysis attack on SOSEMANUK has been presented. The overall attack complexity can be summarized as follows:

- The average number of faults required to perform the attack is $4 \times 1536 = 6144$. These 1536 transient faults are introduced in steps $t = 0$, $t = 1$, $t = 2$ and $t = 4$. This fault injection phase requires the attacker to reinitialize the cipher for 6144 times
- The number of operations required for the attack is dominated by the guess-and-determine part of the analysis. Namely, as concluded in Section 5.3, table T has 2^{16} rows and thus there exists 2^{16} possible guesses for (9). Since register $R1_0$ is a 32-bit value, the number of guesses that need to be checked is $2^{16} \times 2^{32} = 2^{48}$. Verifying each guess according to the procedure in Section 6 is equivalent to one SOSEMANUK iteration and thus the attack requires work equivalent to around 2^{48} iterations.
- The storage amount required for the attack is equal to the size of the tables T_1, T_3 and T which amounts to $2^{38.17}$ bytes.

It should be noted that, when compared to other stream ciphers in the equivalent fault analysis model, DFA of SOSEMANUK requires a relatively smaller number of faults. For example, the DFA attack on RC4 given in [16] requires 2^{16} faults in random locations of the RC4 inner state. Another DFA attack on HC-128 [19] requires around 2^{13} faults in random locations. In future work, it will be interesting to see whether the number of faults for the DFA of SOSEMANUK and other stream ciphers can be drastically decreased in the assumed fault model.

A naive approach to prevent our attack is to use algorithm level redundancy and disable the device output if the two produced key stream values do not match. Another more efficient approach, which partially protects against fault

attacks, is to add parity bits to all the inner state registers and disable the device output if any of these parity checks is violated. Efficient fault analysis resistant implementations for SOSEMANUK, as well as for other stream ciphers, need to be addressed in future research.

References

1. Ahmadi, H., Eghlidos, T., Khazaei, S.: Improved guess and determine Attack on SOSEMANUK (2006), http://www.ecrypt.eu.org/stream/sosemanukp3.html
2. Anderson, R., Biham, E., Knudsen, L.R.: Serpent: A proposal for the advanced encryption standard, NIST AES Proposal (1998)
3. Armknecht, F., Meier, W.: Fault Attacks on Combiners with Memory. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 36–50. Springer, Heidelberg (2006)
4. Berbain, C., Billet, O., Canteaut, A., Courtois, N., Gilbert, H., Goubin, L., Gouget, A., Granboulan, L., Lauradoux, C., Minier, M., Pornin, T., Sibert, H.: Sosemanuk, a fast software-oriented stream cipher. eSTREAM, the ECRYPT Stream Cipher Project, Report 2005/027 (2005)
5. Berzati, A., Canovas-Dumas, C., Goubin, L.: Fault Analysis of Rabbit: Toward a Secret Key Leakage. In: Roy, B., Sendrier, N. (eds.) INDOCRYPT 2009. LNCS, vol. 5922, pp. 72–87. Springer, Heidelberg (2009)
6. Biham, E., Granboulan, L., Nguyen, P.Q.: Impossible Fault Analysis of RC4 and Differential Fault Analysis of RC4. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 359–367. Springer, Heidelberg (2005)
7. Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)
8. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the Importance of Checking Cryptographic Protocols for Faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997)
9. Cho, J.Y., Hermelin, M.: Improved linear cryptanalysis of SOSEMANUK. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 101–117. Springer, Heidelberg (2010)
10. Debraize, B., Corbella, I.M.: Fault analysis of the stream cipher Snow 3G. In: Workshop on Fault Diagnosis and Tolerance in Cryptography 2009, pp. 103–110 (2009)
11. Dusart, P., Letourneux, G., Vivolo, O.: Differential fault analysis on A.E.S. In: Zhou, J., Yung, M., Han, Y. (eds.) ACNS 2003. LNCS, vol. 2846, pp. 293–306. Springer, Heidelberg (2003)
12. ECRYPT, the European Network of Excellence for Cryptology, http://www.ecrypt.eu.org/ecrypt1/
13. Ekdahl, P., Johansson, T.: A New Version of the Stream Cipher SNOW. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 47–61. Springer, Heidelberg (2003)
14. eSTREAM, the ECRYPT Stream Cipher Project, http://www.ecrypt.eu.org/stream/
15. Feng, X., Liu, J., Zhou, Z., Wu, C., Feng, D.: A Byte-Based Guess and Determine Attack on SOSEMANUK. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 146–157. Springer, Heidelberg (2010)

16. Hoch, J., Shamir, A.: Fault Analysis of Stream Ciphers. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 240–253. Springer, Heidelberg (2004)
17. Hojsík, M., Rudolf, B.: Floating fault analysis of Trivium. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 239–250. Springer, Heidelberg (2008)
18. Kircanski, A., Youssef, A.M.: Differential Fault Analysis of Rabbit. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 197–214. Springer, Heidelberg (2009)
19. Kircanski, A., Youssef, A.M.: Differential Fault Analysis of HC-128. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 261–278. Springer, Heidelberg (2010)
20. Lee, J.-K., Lee, D.-H., Park, S.: Cryptanalysis of sosemanuk and SNOW 2.0 using linear masks. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 524–538. Springer, Heidelberg (2008)
21. Lin, D., Jie, G.: Guess and Determine Attack on Sosemanuk. In: Proceedings of the 2009 Fifth International Conference on Information Assurance and Security, vol. 01, pp. 658–661. IEEE, Los Alamitos (2009)
22. Messerges, T.S., Dabbish, E.A., Sloan, H.R.: Examining Smart-Card Security under the Threat of Power Analysis Attacks. IEEE Trans. on Computers 51(4), 541–552 (2002)
23. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 2–12. Springer, Heidelberg (2003)
24. Tsunoo, Y., Saito, T., Shigeri, M., Suzaki, T., Ahmadi, H., Eghlidos, T., Khazaei, S.: Evaluation of Sosemanuk with regard to guess-and-determine attacks (2006), http://www.ecrypt.eu.org/stream/sosemanukp3.html