Floating Fault Analysis of Trivium

Michal Hojsík[1] and Bohuslav Rudolf[2]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway
[2] National Security Authority, Na Popelce 2/16, 150 06 Prague 5, Czech Republic
michal.hojsik@ii.uib.no and b.rudolf@nbu.cz

Abstract. One of the eSTREAM final portfolio ciphers is the hardware-oriented stream cipher Trivium. It is based on 3 nonlinear feedback shift registers with a linear output function. Although Trivium has attached a lot of interest, it remains unbroken by passive attacks.
At FSE 2008 a differential fault analysis of Trivium was presented. It is based on the fact that one-bit fault induction reveals many polynomial equations among which a few are linear and a few quadratic in the inner state bits. The attack needs roughly 43 induced one-bit random faults and uses only linear and quadratic equations.
In this paper we present an improvement of this attack. It requires only 3.2 one-bit fault injections in average to recover the Trivium inner state (and consequently its key) while in the best case it succeeds after 2 fault injections. We termed this attack floating fault analysis since it exploits the floating model of the cipher. The use of this model leads to the transformation of many obtained high-degree equations into linear equations.
The presented work shows how a change of the cipher representation may result in much better attack.

Keywords : Trivium, stream cipher, differential fault analysis

1 Introduction

The year 2008 is the last year of the European project ECRYPT. Within the project a search for new stream ciphers, eSTREAM project, took place. After 3 project phases within 4 years the final results were announced on Eurocrypt 2008. The eSTREAM committee has pointed out four ciphers within each of the two profiles (hardware/software oriented ciphers) and published them as the eSTREAM portfolio.

At the very beginning of eSTREAM, 34 ciphers were proposed and evaluated but only some have made their way up to the final phase or were even included to the final portfolio. Among these (portfolio) ciphers, the stream cipher Trivium is somehow special. First of all its design is indeed very simple. It brings us again to the question how simple can a secure cipher be. Secondly, Trivium is the fastest cipher among all proposals in many tested architectures (see e.g. [15] or [14] for more details) and although Trivium was largely analysed, it remains unbroken by passive attacks.

Earlier this year at FSE 2008, a differential fault attack on Trivium was presented [2]. It is based on the fact, that an injection of a one-bit fault (a bit flip) into a Trivium inner state reveals to an attacker a few linear and a few quadratic equations in the inner state bits. The attack requires roughly 43 fault injections at random positions and it assumes that all fault injections are performed into the same inner state. This can be achieved in the chosen-ciphertext attack scenario assuming that the initialisation vector is a part of the cipher input. In this case, an attacker will always use the same cipher input (cipher text and initialisation vector) which will lead to the same cipher inner state during the decryption. This would allow him to perform the fault injection to the same inner state during the deciphering process. Hence the attack can be described as chosen-ciphertext fault injection attack.

In this paper, we present an essential improvement of this attack and we describe a novel approach to the differential fault analysis of Trivium. Using this approach, we have obtained much better results in the sense of number of fault injections needed. Where the original attack needs 43 fault injections, our approach reveals the secret key after 3.2 fault injections in average (over $10,000$ experiments). In the best cases we have obtained the secret key after only 2 fault injections.

The main idea behind our attack is a simple way of transforming polynomial equations obtained during the attack into linear equations. This procedure is based on what we call *the floating description* of Trivium and on some fault propagation properties of Trivium's inner state evolution.

Formally, we significantly extend the number of variables by denoting every new inner state bit as a new variable, while not forgetting its connections given by the Trivium inner state evolution. Consequently many keystream difference equations become linear or have low degree. We use equations up to degree 4 and we linearise them subsequently using already revealed inner state bits, which are obtained by the use of Gauss-Jordan elimination for the linear equations.

In the beginning, we start off the equations system by the keystream equations and the inner state bit connections and afterwards we use fault injection to obtain more equations. After each fault injection and the following equations processing, we search the inner state bit sequences for a time t in which the Trivium inner state IS_t would contain a sufficient number of known bits. Since shifting of the Trivium inner state (seen as an interval of the inner state bit sequence) in time reminded us of floating of the inner state, we call this description *the floating description* and the attack *the floating fault analysis*. After finding such a time t, for which the inner state IS_t is known, we clock Trivium backwards until we obtain an initial state from which we can directly read used secret key and IV.

The rest of the paper is organised as follows. In Sec. 2 we review the related work, while Sec. 3 describes Trivium in the floating notation. In Sec. 4 we summarise attack prerequisites, followed by the attack description in Sec. 5. The paper is concluded by Sec. 6.

2 Related Work

As far as we known, the stream cipher Trivium has been, despite its simplicity, secure against all non side-channel attacks. At this point, we would like to recall some related work on Trivium as well as some results on stream cipher side-channel analysis.

New methods of solving systems of sparse quadratic equations are applied by Raddum to Trivium in [4]. The attack has complexity of $O(2^{162})$. Maximov and Biryukov in [5] tried to solve the system of equations given by Trivium keystream by guessing some inner state bits, which would result in a reduction of degrees of obtained equations. The complexity of their attack is $O(c \cdot 2^{83.5})$, where c is the time needed to solve a sparse system of linear equations. In [6], presented at SASC 2007, Babbage pointed out different possible approaches to the analysis of Trivium. Thuran and Kara presented a model of the Trivium initialisation part as an 8-round function in [7]. Their linear approximation of 2-round Trivium has bias 2^{-31}. Differential cryptanalysis is applied to the initialisation part of Trivium in [8]. Recently at SASC 2008, Fisher, Khazaei and Meier presented a new method for key recovery attacks [13]. They successfully applied their attack to Trivium with a reduced number of initialisation steps (672 out of the original 1152 steps). In the same paper they also provide evidence that the proposed attack is not applicable on Trivium with full initialisation.

Since the presented attack is a fault analysis attack, i.e. a side-channel attack, we would also like to mention some previous work on the side-channel analysis of stream ciphers. An overview on passive side-channel attacks on stream ciphers can be found in [9], while fault attacks on stream ciphers are described in [10]. Recently, authors of [12] have theoretically analysed ciphers in phase 3 of the eSTREAM project with respect to many types of side-channel analysis. Early this year at FSE 2008, authors of [2] have described differential fault analysis of Trivium. They have presented an attack that requires 280 keystream bits and in average 43 fault injections to reveal the used secret key. The authors have also designed a simple method for fault position determination. As already mentioned, our attack is an extension of their work which leads to a rapid reduction in the number of fault injections needed.

3 Trivium Description in The Floating Model

The stream cipher Trivium is a bit-oriented additive synchronous stream cipher with 80-bit secret key and 80-bit initialisation vector (IV). Trivium (as other stream ciphers) can be divided into two parts: the *initialisation algorithm*, which turns a secret key and an initialisation vector into the inner state of Trivium, and the *keystream generation algorithm*, which produces the keystream (one bit per step).

Necessarily condition for a simple description of our attack is the use of the following notation (first described in [3]). In this notation the cipher inner state registers are represented by the binary sequences they produce instead of describing them as finite length NLFSRs. We will refer to this notation as *the floating notation* or *the floating description*. We have chosen this name since in this notation a Trivium inner state is an interval which is "floating" on the inner state bit sequences as time goes on.

The stream cipher Trivium consists of 3 non-linear feedback shift registers. The sequences produced by the first, the second and the third register will be denoted by $\{x_n\}$, $\{y_n\}$ and $\{z_n\}$ respectively. Since Trivium registers are of lengths 93, 84 and 111 these sequences will be indexed from -93, -84 and -111 onwards. So at time t the Trivium inner state is equal to

$$IS_t = (x_{t-1}, \dots, x_{t-93}, y_{t-1}, \dots, y_{t-84}, z_{t-1}, \dots, z_{t-111}).$$

At the beginning of the initialisation part of Trivium, an 80-bit secret key $K = (k_1, \dots, k_{80})$ is used to initialise the sequence $\{x_n\}_{n=-93}^{\infty}$ so that $x_{-i} = k_i,\ i = 1, \dots, 80$ and an 80-bit initialisation vector $IV = (u_1, \dots, u_{80})$ is used to initialise the sequence $\{y_n\}_{n=-84}^{\infty}$ so that $y_{-i} = u_i,\ i = 1, \dots, 80$. Finally $z_{-109},\ z_{-110}$ and z_{-111} are set to one and all previously unset $x_n,\ y_n$ and z_n are set to zero for all $n < 0$. We will refer to this state as to the *initial state.*

For $n \geq 0$, the following recursions are used to compute new inner state bits x_n, y_n and z_n : [3]

$$\begin{aligned} x_n &= x_{n-69} + z_{n-66} + z_{n-111} + z_{n-110}z_{n-109}, \\ y_n &= y_{n-78} + x_{n-66} + x_{n-93} + x_{n-92}x_{n-91}, \\ z_n &= z_{n-87} + y_{n-69} + y_{n-84} + y_{n-83}y_{n-82}. \end{aligned} \tag{1}$$

The only difference between the initialisation part and the keystream generation part of Trivium is the keystream production function present in the latter one, while the former consists of 1152 recursion steps. For the sake of notation simplicity, we will denote the keystream sequence by $\{o_n\}_{n=1152}^{\infty}$ (i.e. it will be indexed from 1152 onwards). It is computed as

$$o_n = x_{n-66} + x_{n-93} + y_{n-69} + y_{n-84} + z_{n-66} + z_{n-111},\ n \geq 1152. \tag{2}$$

In the classical description, the cipher is represented by its 288 bit inner state $IS = (s_1, \dots, s_{288})$ which is updated each cipher clock. This description can be found e.g. in the cipher specification [1] and we will refer to this as the *static notation* or *static description.*

In the rest of the paper we will start our investigations at some random, but fixed time t. For simplicity, we will denote this time as time $t = 0$. So the inner state at this time will be $IS_0 = (x_{-1}, \dots, x_{-93}, y_{-1}, \dots, y_{-84}, z_{-1}, \dots, z_{-111})$. (This was denoted by $IS_{t_0} = (s_1, \dots, s_{288})$ by authors of [2].) From now on, by x_{-1} we mean the value of x_n for $n = t-1$ for the fixed but unknown time t and not the value k_1 as set in the initialisation part of Trivium.

[3] All additions in this paper are carried modulo 2

4 Attack Prerequisites

The attack presented in this paper is a differential fault analysis attack, meaning that an attacker has to be able to (repeatedly) insert a fault into a cipher inner state IS_0. As recently mentioned in Sect. 3, this is an arbitrary but fixed Trivium inner state. In our case, inserting a fault means a bit flip on an unknown random position. The inner state after the fault injection will be denoted by IS'_0.

We also assume that an attacker is able to obtain N consecutive keystream bits after the fault injection, where in our implementation we have successfully used $N = 800$. This keystream will be referred to as the *faulty keystream* and will be denoted by $\{o'_n\}$. Further we assume that the attacker has also access to the first N consecutive bits of so called *proper* keystream, $\{o_n\}$, which is generated from the proper inner state IS_0.

The last assumption we make is that the attacker is able to repeat the fault injection to the same inner state IS_0 (each time inserting a fault into a new random position). This means, that the attacker has to run Trivium more than once with the same secret key and IV to reach the same inner state. This can be achieved in the chosen-ciphertext scenario, assuming that the initialisation vector is a part of the cipher input. In this case, the attacker will always use the same cipher input (cipher text and initialisation vector) which will lead to the same cipher inner state during the decryption. This would allow him to perform the fault injection to the same inner state during the deciphering process.

All together, these are the prerequisites of our attack:

1. Attacker is able to obtain the first N consecutive bits of the keystream $\{o_n\}$ produced out of the inner state IS_0.
2. Attacker is able to inject exactly one fault (a bit flip) into the inner state IS_0 into an unknown random position.
3. Attacker is able to obtain the first N consecutive bits of the keystream $\{o'_n\}$ produced out of the faulty inner state IS'_0.
4. Attacker is able to repeat the fault injection into a random position of IS_0 M times.

In our implementation of the attack, we have used $N = 800$ and the attack reveals the secret key in average after 3.2 fault injections. During our experiments (we have run the attack 10000 times) the attack succeeded with the probability 2% for $M = 2$, 78.5% for $M = 3$, 99.8% for $M = 4$ and for $M = 5$ the attack always revealed the secret key.

5 Floating Fault Analysis of Trivium

The authors of [2] described a differential fault analysis of Trivium based on the static model. In this paper we take the advantage of the floating model and we show that this model leads to much better results.

Before describing the attack itself, let us introduce some more notation. For each of the sequences $\{x_n\}$, $\{y_n\}$, $\{z_n\}$ and $\{o_n\}$ we define a *delta sequence*

$\{\delta x_n\}$, $\{\delta y_n\}$, $\{\delta z_n\}$ and $\{\delta o_n\}$ respectively as a difference between the proper sequence (a sequence without the fault injection) and the faulty sequence (the sequence after the fault injection). All faulty variables are marked by a prime. Using (1), (2) and the following equation describing the difference induced by multiplication

$$\delta(ab) = a'b' + ab = \delta a \cdot b + a \cdot \delta b + \delta a \cdot \delta b$$

we obtain the following equations describing the delta sequences:

$$\begin{aligned} \delta o_n &= \delta x_{n-66} + \delta x_{n-93} + \delta y_{n-69} + \delta y_{n-84} + \delta z_{n-66} + \delta z_{n-111} && (3) \\ \delta x_n &= \delta x_{n-69} + \delta z_{n-66} + \delta z_{n-111} + \delta z_{n-109} \cdot z_{n-110} + && (4) \\ &\quad + z_{n-109} \cdot \delta z_{n-110} + \delta z_{n-109} \cdot \delta z_{n-110} \\ \delta y_n &= \delta y_{n-78} + \delta x_{n-66} + \delta x_{n-93} + \delta x_{n-91} \cdot x_{n-92} + && (5) \\ &\quad + x_{n-91} \cdot \delta x_{n-92} + \delta x_{n-91} \cdot \delta x_{n-92} \\ \delta z_n &= \delta z_{n-87} + \delta y_{n-69} + \delta y_{n-84} + \delta y_{n-82} \cdot y_{n-83} + && (6) \\ &\quad + y_{n-82} \cdot \delta y_{n-83} + \delta y_{n-82} \cdot \delta y_{n-83}. \end{aligned}$$

According to the assumptions 1 and 3 in Sec. 4 an attacker is able to obtain the proper keystream $\{o_n\}$ as well as the faulty keystream $\{o'_n\}$. During the attack he will compute the delta keystream $\{\delta o_n\}$ for each fault injection and express its bits as expressions in variables $\{x_n\}, \{y_n\}$ and $\{z_n\}$ using equations (3), (4), (5) and (6). We will refer to this equations as the delta keystream equations. Afterwards he will try to solve these equations.

5.1 Faults in the Floating Model and the Corresponding Delta-Equations

We claim that using the floating notation, an attacker will obtain many more linear and quadratic equations than with the static notation, using the same attack model and having the same assumptions. Why is there a difference between the static model and the floating model? In fact there is no difference in the obtained equations - they are equivalent. The difference is in the representation, in our viewpoint. In the static model, at time t for some $t > 0$, the floating model variable x_t (or y_t or z_t equivalently) would be expressed as a polynomial in bits of the fixed initial state $IS_0 = (x_{-1}, \ldots, x_{-93}, y_{-1}, \ldots, y_{-84}, z_{-1}, \ldots, z_{-111}) = (s_1, \ldots, s_{288})$. Hence some of the obtained delta keystream equations which are in fact linear in the floating variables ($\{x_n\}$, $\{y_n\}$, $\{z_n\}$), are polynomial in the static initial state variables ($(s_1, \ldots, s_{288})$). In other words, δo_n is in the static model often a linear combination of nonlinear terms $\{\delta x_n\}$, $\{\delta y_n\}$, $\{\delta z_n\}$ which are expressed in the variables $(s_1, \ldots, s_{288})$. On the other hand, in the floating model we have clearly many more variables. Instead of 288 variables of the fixed inner state in the static model, we have $3N + 288$ variables in the floating model, which gives us 2688 variables for $N = 800$ as we have used in our implementation (288 initial state variables plus 3 new variables for each Trivium step). Since we do not forget about the connections between variables, the floating model can be seen as a useful extension of the static model.

Another important difference between these two models is the following: In the static model we fix the set of variables as bits of the inner state into which the fault injections are performed. So before the single injected fault spreads over the inner state and produces many linear equations, the expressions for the new inner state bits become polynomial and therefore in fact linear delta keystream equations contain high-degree polynomials. In the floating model we decide which inner state we would like to compute according to the obtained equations. So we wait until the fault spreads over the inner state and only then do we try to compute the inner state for the best possible time. More precisely, during the attack we try to determine values of all the variables $\{x_n\}$, $\{y_n\}$ and $\{z_n\}$ and we wait until there are enough known variables in an interval of a single inner state, regardless of the actual position of this state in time. For illustration, in our experiments during the attack we have obtained enough equations in the bits of inner state IS_t which is reached after approximately 300 steps of Trivium, i.e. we have obtained a Trivium inner state that appears roughly 300 steps after the fault injection. Afterwards we clock Trivium backwards until we reach a state IS_u similar to the initial state. Then the secret key equals to $(x_{u-1}, \ldots, x_{u-80})$ and $IV = (y_{u-1}, \ldots, y_{u-80})$.

In the floating model, the attack starts with the initial delta inner state $\{\delta x_n\}_{n=-93}^{-1}$, $\{\delta y_n\}_{n=-84}^{-1}$, $\{\delta z_n\}_{n=-111}^{-1}$, where all the bits are equal to zero except one (the bit where the fault injection occurred). So for example if the fault was injected into $x_i, i \in \{-93, \ldots, -1\}$ (a bit of the first register), the initial delta state would be

$$\begin{array}{ccc} (\delta x_{-93}, \ldots, \delta x_{i-1}, \delta x_i, \delta x_{i+1}, \ldots, \delta x_{-1}) & = & (0, \ldots, 0, 1, 0, \ldots, 0) \\ (\delta y_{-84}, \ldots, \delta y_{-1}) & = & (0, \ldots, 0) \\ (\delta z_{-111}, \ldots, \delta z_{-1}) & = & (0, \ldots, 0). \end{array}$$

Afterwards we inductively use equations (4), (5) and (6) to express $\{\delta x_n\}, \{\delta y_n\}$ and $\{\delta z_n\}$ for $n \geq 0$ as polynomials in variables $\{x_n\}$, $\{y_n\}$, $\{z_n\}$. These are afterwards used to represent bits of known sequence $\{\delta o_n\}$ as terms in the variables $\{x_n\}$, $\{y_n\}$, $\{z_n\}$, i.e. they are used to create the delta keystream equations.

Now let's look closer on these equations. When do they contain non-linear dependencies? From (3) it follows, that δo_n is a linear combination of values $\{\delta x_n\}, \{\delta y_n\}$ and $\{\delta z_n\}$. So it will contain non-linear terms if and only if any of these values will be non-linear. Let's examine e.g. the case of δx_n. For some $j > 0$, δx_j depends non-linearly on $\{x_n\}, \{y_n\}, \{z_n\}$ if and only if at least one of the following conditions is satisfied:

1. at least one of the values δx_{j-69}, δz_{j-66} or δz_{j-111} is non-linear in $\{x_n\}$, $\{y_n\}$, $\{z_n\}$,
2. δz_{j-109} or δz_{j-110} (or both of them) has degree at least 1 as a polynomial in $\{x_n\}$, $\{y_n\}$ and $\{z_n\}$.

Clearly, case 1 is only a transition of a non-linearity from other terms so a new non-linearity is created only in the case 2. As described in the above example, the starting delta inner state is all zero except one bit and consequently there are only few non-linearity creations and transitions for small values of j. Although afterwards both cases occur more often, we are still able to obtain low degree equations thanks to the simple substitution we use to eliminate non-linearities (we substitute already known variables into the higher degree equations). Moreover many variables are known directly from the delta keystream equations, since they appear as a linear equation with only one term.

Tab. 1 shows the average number of obtained equations of degree up to four after one, two, tree and four fault injections. Each table entry contains two values, where the first one stands for the number of equations right after the fault injection while the second one stands for the number of equations after they were processed by methods described in Sect. 5.3. The average is computed over 10000 experiments.

Table 1. The average number of equations after different numbers of fault injections. FI stands for fault injection(s).

	# equations before/after eq. processing			
	degree 1	degree 2	degree 3	degree 4
Before FI	800/800	2400/2400	0	0
After 1 FI	825/992	2466/2350	35/2	57/1
After 2 FI	1017/1232	2419/2236	36/3	57/1
After 3 FI	1258/2396	2298/484	37/1	56/0
After 4 FI	2402/2685	498/6	8/0	12/0

5.2 Fault Position Determination

During the fault injection phase of the attack, a fault (a bit flip) is injected at a random position of the actual Trivium inner state. In order to be able to proceed with the attack, we need to know this position. Authors of [2] have described a very simple fault position determination technique. In this paper, we will use their technique as described in section 5.3. of [2]. Briefly, the technique is based on the fact that the distribution of ones in the delta keystream uniquely determines the position of the induced fault within the inner state. After the fault injection, an attacker computes the delta keystream $\{\delta o_n\}$ and determines the fault position (by a single table look-up) according to the distance between the first occurrences of non-zero bits in $\{\delta o_n\}$.

This technique is deterministic and leads to the right result for all possible fault positions, assuming that exactly one fault was injected. In the attack description, we will refer to this technique as the *fault_position_determination()*.

5.3 The Equations System and Its Processing

All the equations used during the attack are in variables $\{x_n\}$, $\{y_n\}$ and $\{z_n\}$. The corresponding system will contain equations from 3 different sources:

- The first set of equations is given by the proper keystream $\{o_n\}$ for $n = 0, \ldots, N-1$. In the floating description, all keystream equations are linear and have the form of Eq. (2). These are the 800 linear equations in the first line of Tab. 1.
- The second set of equations comes from the Eq. (1) which describes the connections between the variables $\{x_n\}$, $\{y_n\}$ and $\{z_n\}$ and these are the 2400 quadratic equations in the first line of Tab. 1.
- The last set of equations are those coming from the fault injections and we term them delta keystream equations.

During the attack we try to solve the actual equation system by the use of two simple methods. The first one is the Gauss-Jordan elimination which we apply to the system of obtained linear equations and we term this procedure *Gauss()*. The second used technique is the substitution. By the term *Substitution()* we will denote a procedure which substitutes values of already known variables into the equations of higher degree (in our implementation we use equations up to degree 4). Since the number of revealed variables after each fault injection is fairly high, the substitution reduces the degree of many non-linear equations. Hereby we utilise the higher degree equations as a potential source of new linear equations and this is the only way how we use them.

5.4 The Attack Algorithm

The attack algorithm is described by Alg. 1. We have used equations up to degree 4 and $N = 800$. Afterwards the attack has revealed the secret key after 3.2 fault injections in average.

Since Eq. (3), (4), (5) and (6) are simple, we do not need any precomputations compared to the attack from [2]. After we determine the fault position at step 10 of Alg. 1 using the function *fault_position_determination()*, we compute symbolic delta equations (described by Eq. 3) for the actual fault position. Afterwards we insert these equations into our equations system using the values of the actual delta-keystream $\{\delta o_n\}$ computed at step 9 of Alg. 1 as the right-hand-side.

Table 2 shows the average maximal number of known variables in a single inner state IS_t (maximum over all possible inner states, i.e. over all positions of the inner state in time t) after $1,\ 2,\ 3,\ 4$ and 5 fault injections as well as the estimated probability that the attack will succeed after a given number of fault injections. We see that the attack will succeed after 2 fault injections only with probability 2%, while after 4 fault injections the attack had revealed the secret key in 99.8% of all the cases.

5.5 Implementation and Complexity

In the floating model, the number of variables depends on the number of Trivium steps performed. In our implementation we have used $N = 800$ which gives us 2400+288 (initial state) = 2688 variables. Clearly, all obtained equations are very sparse. However, we have used the straight-forward implementation of the

Algorithm 1 Floating Fault Attack

```
1: get Trivium in an unknown fixed inner state $IS_0$
2: obtain the first $N$ consecutive bits of $\{o_n\}$, starting from inner state $IS_0$
3: insert keystream equations (Eq. 2 in Sect. 3) into the eq. system
4: insert connection equations (Eq. 1 in Sect. 3) into the eq. system
5: while for all $t$ $IS_t$ not known do
6:    reset Trivium to the state $IS_0$
7:    insert a fault into $IS_0$
8:    obtain the first $N$ consecutive bits of the faulty keystream $\{o'_n\}$
9:    $\delta o_n \leftarrow o_n + o'_n,\ n = 0, \ldots, N-1$
10:   $e \leftarrow fault\_position\_determination(\{\delta o_n\})$
11:   compute delta keystream equations for $e$ and insert them into the eq. system
12:   repeat
13:      do Substitution()
14:   until it keeps changing the equation system
15:   do Gauss()
16:   if new variables obtained by Gauss() then
17:      goto 12
18:   end if
19: end while
20: // at this point we have a known inner state $IS_t$
21: run Trivium backwards starting with $IS_t$ until an inner state similar to the initial
    state is reached
22: read the secret key from the reached initial state
```

Table 2. The average maximal number of known variables in a single inner state and the estimated probability of success after different numbers of fault injection. Average made over $10,000$ experiments with random key, IV and fault position.

# fault injections	1	2	3	4	5
# known variables in a single state	30	70.4	245.5	287.5	288
probability of success	0	0.02	0.79	0.99	1

Gauss-Jordan elimination. Due to the implementation complexity, we use only equations up to degree 4.

According to the nature of the proposed algorithm, it is indeed very hard to do any theoretical complexity analysis. Hence we have used the average number of procedure calls to do the following complexity estimate. The most complex procedure used in the attack algorithm is the Gauss-Jordan elimination. We suppose that for a $m \times n$ matrix it has the complexity of $\mathrm{O}(nm^2)$ operations. Since the number of columns equals 2688 and if we suppose that the number of linear equations is in average lower than 1024, we gain the running time of roughly $2^{11.4} \cdot 2^{20} = 2^{31.4}$ simple operations per an average Gauss-Jordan elimination used in the attack. We have performed 10000 runs of the attack and in the average one attack makes $80 \doteq 2^{6.3}$ calls of the *Gauss()* procedure. All together we can retrieve the secret key in the time of $2^{31.4} \cdot 2^{6.3} = 2^{37.7}$ simple operations. Since the average running time of the attack was only 40.3 seconds on our desktop computer with AMD Athlon 64 X2 Dual-Core 3800+ processor, the complexity estimate can be seen as the upper limit.

5.6 Effort for Further Improvements

During our experiments, we have tried to stop the attack at the point when we have found an inner state IS_u (for any u) with at least $288 - G$ known bits for a given G. Then we could use the brute force to determine the rest of the inner state bits. We have tried $G = 20$ and $G = 30$ but in the average over 10000 experiments, these attacks led to very similar results.

Another way to improve our attack could be the trick described in [5]. Namely, we tried to do an educated guess of up to 30 variables in the following way: after the second fault injection when we have already gathered many high-degree equations, we have guessed those variables which maximised the number of higher-degree terms eliminated by this guess. In this way we obtain not only 30 new known variables, but also more low-degree equations. However experiments have shown that the contribution of this smart guess to the final number of fault injections needed is in fact neglectable (even if applied on different places of the attack algorithm).

6 Conclusion

The paper describes a new approach to the fault analysis of Trivium. We have taken advantage of the so called floating model and have proposed and implemented an attack which can reveal the secret key after 3.2 fault injections in average using 800 keystream bits. In the best cases, we were able to reconstruct the secret key only after 2 fault injections.

Our results show how important it is for cryptanalysis to choose the right model for cipher representation.

References

1. De Cannière, C., Preneel, B.: Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/30, http://www.ecrypt.eu.org/stream (2005)
2. Hojsik, M., Rudolf, B.: Differential Fault Analysis of Trivium. In: Nyberg, K. (Ed.) FSE 2008. LNCS, vol. 5086, pp. 158-172. Springer (2008)
3. ECRYPT discussion forum, http://www.ecrypt.eu.org/stream/phorum/read.php?1,448
4. Raddum, H.: Cryptanalytic Results on Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/039, http://www.ecrypt.eu.org/stream (2006)
5. Maximov, A., Biryukov, A.: Two Trivial Attacks on Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/006, http://www.ecrypt.eu.org/stream (2007)
6. Babbage, S.: Some Thoughts on Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/007, http://www.ecrypt.eu.org/stream (2007)
7. Turan, M.S., Kara, O.: Linear Approximations for 2-round Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/008, http://www.ecrypt.eu.org/stream (2007)
8. Biham, E., Dunkelman, O.: Differential Cryptanalysis in Stream Ciphers. COSIC internal report (2007)
9. Rechberger, Ch., Oswald, E.: Stream Ciphers and Side-Channel Analysis. SASC 2004 - The State of the Art of Stream Ciphers, Workshop Record, pp. 320-326. http://www.ecrypt.eu.org/stream (2004)
10. Hoch, J. J., Shamir, A.: Fault Analysis of Stream Ciphers. In: Joye, M., Quisquater, J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 240-253. Springer (2004)
11. Biham, E., Granboulan, L., Nguyen, Ph.: Impossible Fault Analysis of RC4 and Differential Fault Analysis of RC4. SASC 2004 - The State of the Art of Stream Ciphers, Workshop Record, pp. 147-155. http://www.ecrypt.eu.org/stream (2004)
12. Gierlichs, B., et al.: Susceptibility of eSTREAM Candidates towards Side Channel Analysis. SASC 2008 - The State of the Art of Stream Ciphers, Workshop Record, pp. 123-150. http://www.ecrypt.eu.org/stream (2008)
13. Fisher, S., Khazaei, S., Meier, W.: Chosen IV Statistical Analysis for key Recovery Attacks on Stream Cipher. SASC 2008 - The State of the Art of Stream Ciphers, Workshop Record, pp. 33-41. http://www.ecrypt.eu.org/stream (2008)
14. Hwang, D., et al.: Comparison of FPGA - Targeted Hardware Implementations of eSTREAM Stream Cipher Candidates. SASC 2008 - The State of the Art of Stream Ciphers, Workshop Record, pp. 151-162. http://www.ecrypt.eu.org/stream (2008)
15. Good, T., Benaissa, M.: Hardware Performance of eSTREAM Phase-III Stream Cipher Candidates. SASC 2008 - The State of the Art of Stream Ciphers, Workshop Record, pp. 163-174. http://www.ecrypt.eu.org/stream (2008)