ATINER CONFERENCE PRESENTATION SERIES No: MAT2020-0186

ATINER's Conference Paper Proceedings Series
MAT2020-0186
Athens, 8 July 2020

New and Efficient Combined Hard Fault and Algebraic Attack on Full Trivium

Patrice Parraud

Athens Institute for Education and Research
8 Valaoritou Street, Kolonaki, 10683 Athens, Greece

ATINER's Conference Paper Proceedings Series
MAT2020-0186
Athens, 8 July 2020
ISSN: 2529-167X

Patrice Parraud, Assistant Professor, Ecoles de Saint-Cyr Coëtquidan, France

New and Efficient Combined Hard Fault and Algebraic Attack on Full Trivium

ABSTRACT

Hard fault attack is a powerful kind of attack to stream cipher. The major idea is to simplify the stream cipher equations by *injecting* or *creating* some faults to reveal the hidden secret key of the encryption machine. In this paper we present a new and efficient hard fault attack on the Full version of the hardware-oriented synchronous stream cipher Trivium of the European project eSTREAM. This hard fault reset based attack has a complexity less than $\mathcal{O}(2^{48})$ and can be made whenever during the cipher stream generation by finding both the the 80-bit secret key and the 80-bit Initial Values. The main idea of this transient fault attack is to stick to a constant a particular register by targeting its reset wire and to make a cryptanalysis in order to recover the secret. This attack is actually better in terms of complexity, than the best known attacks.

Keywords: Full trivium, cryptanalysis, fault attack, algebraic attack, complexity.

Introduction

A stream cipher is a symmetric encryption algorithm which takes a stream of plaintext, a secret key and initial values as input and then generate a keystream used to encrypt the the plaintext. The secret key and initial values are of fixed length and used to initialize the inner state of keystream generator. In a synchronous stream cipher the keystream and the plaintext are independent. Trivium is a hardware-oriented synchronous stream cipher designed by C. De Cannière and B. Preneel [2, 3] in 2005 for the ECRYPT Stream Cipher Project, abbreviated eSTREAM [1] which is a multi-year effort to identify new stream ciphers potentially suitable for widespread adoption. Since 2004, no less than 34 different stream cipher proposals were submitted in two kind of profile, software oriented and hardware oriented. Trivium has been accepted for the final portfolio of promising new stream ciphers with 7 other candidates among which Grain-v1 and MICKEY-v2 in the same performance profile. Trivium has a simple and elegant structure composed of 3 non-linear feedback shift registers (NFSRs) and a linear output function. Its internal state is initialized with an 80-bit secret key and an 80-bit initial value then rotated over 1,152 clock cycles (called Full Trivium). Trivium has attracted a lot of interest ([4, 5, 10, 11, 12, 13, 16]) with several cryptanalysis on Full Trivium or on reduced version (number of initialization rounds, number of registers) and also with design improvements in terms of security ([20, 21]). Here we only briefly mention some particular classes of attacks. In [6] authors developed statistical tests to show statistical weaknesses of Trivium with up to 736 initialization rounds. In [7] combined statistical tests are used to built an attack on 672-initialization- rounds Trivium with complexity 2^{55}. In [8, 9] A related key differential attack is developed on Full Trivium with complexity 2^{68}. In [22] (then [23, 27]) an algebraic attack called Cube attack is applied on 735-initialization-rounds Trivium with complexity 2^{30}. An another powerful kind of attacks is the hard fault attack which is efficient to stream cipher to reveal the hidden secret key of the encryption machine. The idea is to *inject* or *creat* some soft faults (changing the values of some positions at some moment) or hard faults (setting the values of some positions permanently) to simplify the cipher system. The goal is to obtain a large number of low-degree equations of its initial state from a generated keystream vector. In [17] a soft fault analysis of Trivium is presented as a known-differential attack in which it makes use of the fault injection to obtain the state differential. In [18] same authors present a floating fault analysis of Trivium in which it makes use of the floating fault under two strong assumptions : the fault injection can be made for the state at a fixed time (especially at the initial time) and after it exactly one random bit is changed. In [14] an improvement of the floating fault attack is presented with two practical assumptions : the fault injection can be made for the state at a random time and the positions of the fault bits are from random one of 3 NFSRs, and from a random area within 8 neighboring bits. In [19] a hard fault analysis is presented for breaking block ciphers. In [15] the same kind of attack is applied to Trivium with different result cases. In the best case attacker can obtain all of the 80-bit secret

key with a probability not smaller than 0.2291 or 69 bits of the 80-bit secret key with a probability not smaller than 0.2396. Whereas most of these attacks are black box attacks and work under (weak, strong or practical) assumptions, the purpose of this paper is to present a new and efficient attack on the Full version of Trivium based on a particular kind of hard fault attack. This attack is a two-stage attack with an operational side where Trivium is initialized then used to generate keystream bits during which the reset is made at any time and an offensive side where the effect of the reset is exploited to obtain the secret key bits.This attack allows to find both the secret key bits and the Initial Values bits of the initial internal state of the Full Trivium. It's complexity is less than $\mathcal{O}(2^{48})$. The contents of this paper are organized as follow. In the next Section, we give a complete description of Full Trivium. In Section *Hard Fault Model*, we describe the hard fault model used for this attack. In Section *The Attack*, we present this new and efficient attack emphasizing its different phases and its computational aspect. In Section *The Complexity*, we explain the calculation of its complexity and its improvement using a wise internal bit structure. In Section *Proposed Countermeasures*, we propose some countermeasures to face this kind of attack.

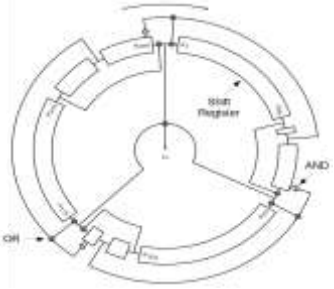
Description of Trivium

The hardware-oriented synchronous stream cipher Trivium has a simple and elegant structure composed by combined shift registers with and and or logic gates shown in Figure 1. Trivium is designed to generate up to 2^{64} bits of keystream from an internal state of 288 bits loaded with an 80-bit secret key and an 80-bit initial value (IV) with 1,152 clock cycles for the Full version sum up in Table 1.

Table 1. *Trivium Parameters*

Parameters		
FULL TRIVIUM cycles	:	1152 rounds
Secret Key size	:	80 bits
Initial Values size	:	80 bits
Internal State size	:	288 bits
Key stream maximal length	:	2^{64} bits

Figure 1. *Internal Structure of Trivium*



It consists of an iterative process, that after been initialized, extracts the values of 15 specific state bits and uses them both to update 3 bits of the state and to compute 1 bit of the keystream. The state bits are then rotated, and the process is repeated. The process consists of two phases: an initialization phase and a generation phase. We have chosen to present Trivium in a chronological operational way while it is usually presented in the counter way.

The initialization internal state phase load the 80-bit secret key denoted by $(K_0, \cdots, K_{79})$, the 80-bit IV denoted by $(IV_0, \cdots, IV_{79})$ and 3 bits to 1 into the 288-bit internal state denoted by $(s_0, \cdots, s_{287})$ according to Table 2.

Table 2. *Secret Key and IV Loading of Trivium*

$$
\begin{array}{lll}
(s_0, \cdots, s_{79}, s_{80}, \cdots s_{92}) & \leftarrow & (K_0, \cdots, K_{79}, 0, \cdots, 0) \\
(s_{93}, \cdots, s_{172}, s_{173}, \cdots s_{176}) & \leftarrow & (IV_0, \cdots, IV_{79}, 0, \cdots, 0) \\
(s_{177}, \cdots, s_{285}, s_{286}, s_{287}) & \leftarrow & (0, \cdots, 0, 1, 1, 1)
\end{array}
$$

Then the state is rotated over 4 Full cycles that is $4 * 288 = 1152$ clock cycles relatively to the *Initialization pseudo-code* of Table 3. The generation keystream phase operates exactly the same as the previous phase except that it generates keystream bits denoted by z_i $(1 \leq i \leq 2^{64})$ relatively to the *Generation pseudo-code* of Table 4.

Table 3. *Initialization Pseudo-Code Equations*

For $(i = 1; i \leq 4 * 288; i \leftarrow i + 1)$ **Do**

$$
\begin{array}{l}
t_1 \leftarrow s_{65} + s_{90}.s_{91} + s_{170} + s_{92} \\
t_2 \leftarrow s_{161} + s_{174}.s_{175} + s_{263} + s_{176} \\
t_3 \leftarrow s_{242} + s_{285}.s_{286} + s_{68} + s_{287} \\
(s_0, s_1, \ldots, s_{92}) \leftarrow (t_3, s_0, \ldots, s_{91}) \\
(s_{93}, s_{94}, \ldots, s_{176}) \leftarrow (t_1, s_{93}, \ldots, s_{175}) \\
(s_{177}, s_{178}, \ldots, s_{287}) \leftarrow (t_2, s_{177}, \ldots, s_{286})
\end{array}
$$

Table 4. *Generation Pseudo-Code Equations*

For $(i = 1; i \leq N; i \leftarrow i + 1)$ **Do**

$$
\begin{array}{l}
t_1 \leftarrow s_{65} + s_{92} \\
t_2 \leftarrow s_{161} + s_{176} \\
t_3 \leftarrow s_{242} + s_{287} \\
\mathbf{z_i \leftarrow t_1 + t_2 + t_3} \\
t_1 \leftarrow t_1 + s_{90}.s_{91} + s_{170} \\
t_2 \leftarrow t_2 + s_{174}.s_{175} + s_{263} \\
t_3 \leftarrow t_3 + s_{285}.s_{286} + s_{68} \\
(s_0, s_1, \ldots, s_{92}) \leftarrow (t_3, s_0, \ldots, s_{91}) \\
(s_{93}, s_{94}, \ldots, s_{176}) \leftarrow (t_1, s_{93}, \ldots, s_{175}) \\
(s_{177}, s_{178}, \ldots, s_{287}) \leftarrow (t_2, s_{177}, \ldots, s_{286})
\end{array}
$$

Hard Fault Model

The fault effect on a circuit has to be clearly modeled for it to be useful for cryptanalysis. Two criteria can be exhibited in order to classify a fault: the introduction difficulty and the repetitively. The easiest fault is to perturb the circuit on several clock cycles. A more powerful fault is to choose a cryptographic object such as a register and to randomly modify it or to stick it to a constant value (resetting it for example). The register size depends on the technology targeted but also the capability of the attacker to localize in the space the sensitive logic. The attack presented in this paper lies within the scope of this kind of powerful fault. The main idea is to stick to a constant a register by targeting its reset wire but not the register itself (Figure 2). In this paper, we interest in the case of transient fault which need a cryptanalysis in order to recover a secret. This attack is a hard fault reset based attack in which the reset action can be made whenever during the cipher stream generation. This attack exploits the 80-bit initial value (IV) which is used to load freely the internal state of Trivium. The attack consists in resetting these IV bits register after the initialization phase of Trivium and whenever during the generation phase of the cipher stream. This hard fault attack does not rest on any particular assumption, excepted to have an encryption machine equipped with Trivium, and it works on a public (e.g., free, accessible) area of the internal state of Trivium and it does not give to the opponent the control of the reset time. This is a realistic attack which can be built using laser for example.

Figure 2. *Fault Effect on a Register by Targeting the Reset Wire*

The Attack

This attack focus on Full Trivium and manage to obtain both the 80-bit secret key and the 80-bit IV. Equations of Table 4 show that generated keystream bits can be expressed in terms of polynomial functions of internal state bits. Unfortunately their lengths increase exponentially according to the number of rounds and their degrees become very high. The main idea of this attack is to reset the IV-bit register by targetting its reset wire whenever during the cipher stream generation and keep the generated keystream bits. The interest of this reset action is that the degree of these polynomial functions is kept low.

Description of the Attack

In order to avoid any possible confusion, the notations used in this section

are the following. The *Internal State* of Full Trivium at time k is noted E_k and its 288 internal bits $s_0^k \dots s_{287}^k$ while the first one are simply noted $s_0 \dots s_{287}$.

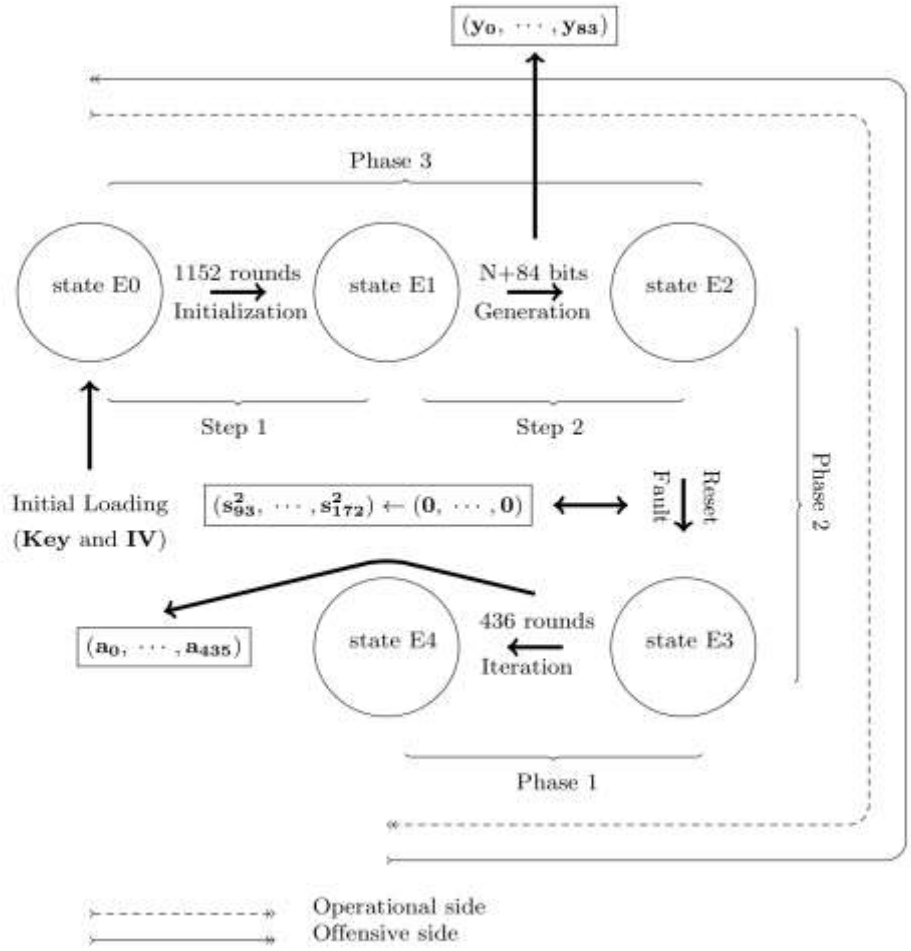
This attack has three distinct phases counter numbered to distinguish the operational aspect (dashed way in Figure 2) of the attack to the offensive one (continuous way in Figure 2). The first phase denoted by **Phase 3** with two steps, **Step 1** and **Step 2**, is a *preparation* part. The second phase denoted by **Phase 2** is typically the *reset* part. The last phase denoted by **Phase 1** is a particular *iteration* part. The general description of the attack is shown in Figure 3.

The operational aspect of the attack is the following. In **Step 1** of **Phase 3** we start with the initial internal state E_0 filled with the 80-bit secret key, the 80-bit IV according to the initial load of Table 2. After 1,152 initialization rounds we obtain the first internal state E_1 used to generate the keystream. In **Step 2** of **Phase 3** we generate $N + 84$ bits of keystream from this state E_1. The parameter N is a natural integer uncontrolled by the attacker (pointing out the fact that this attack can be made whenever during the cipher stream generation). We obtain the second internal state E_2. The last 84 bits of the keystream are noted $y_0 \dots y_{83}$. In **Phase 2** we apply the reset action to the 80-bit IV positions $s_{93}^2 \dots s_{172}^2$ of the internal state E_2 and obtain the third internal state E_3 to generate one keystream bit. In **Phase 1** we repeat this procedure 436 times with the new obtained internal state E_3. That is, one keystream bit is generated from the internal state E_3 then the 80-bit IV positions of the internal state E_3 are reseted and so on. We finish with the last internal state E_4 and note $a_0 \dots a_{435}$ the 436 bits of the obtained keystream.

The offensive aspect of the attack is the following. We start with **Phase 1** and find all the 288-bit internal state E_3 from the last 436-bit generated keystream $a_0 \dots a_{435}$. We go on with **Phase 2** and determine all the 288-bit internal state E_2 from the last 84-bit generated keystream $y_0 \dots y_{83}$. We achieve the attack with **Phase 3** by counter round Trivium from the 288-bit internal state E_2 to the entire initial internal state E_0. In this way, we have managed to guess both the 80-bit secret key and the 80-bit Initial Values of the Full Trivium.

We now describe in detail each phase of this attack relatively to the offensive way.

Figure 3. *Complete Scheme Attack on Full Trivium*



Phase 1

In this first phase, the goal is to find all the 288 bits $s_0^3 \dots s_{287}^3$ of the internal state E_3 from the last 436 bits $a_0 \dots a_{435}$ of generated keystream.

The Linear System

The last 436-bit generated keystream $a_0 \dots a_{435}$ could be expressed in terms of polynomial functions of the 288 bits s_i^3 of the internal state E_3. The main interest of the reset action made during **Phase 2** is that the degree of these polynomials will be kept low. In a first time, we keep only polynomials for which their degree is less or equal to two. Then we consider all binomial of type $s_i^3 s_{i+1}^3$ like variables similarly as the s_i^3. We obtain an linear system in which we add the condition that all equations have to be linearly independent in $\mathbb{R}$. After deleting all unused variables we obtain an linear system of 257 equations with 319 unknowns.

$$Z = AX \qquad (1)$$

Where Z and X are the column vectors

$$Z = [a_0, \cdots, a_{174}, a_{180}, \cdots, a_{201}, \cdots, a_{243}, \cdots, a_{249}, a_{393}, \cdots, a_{399}, a_{405}, \cdots, a_{408}, a_{414}, \cdots, a_{417}, a_{423}, \cdots, a_{426}, a_{432}, \cdots, a_{435}]$$

$$X = [s_0^3, \cdots, s_{92}^3, s_{173}^3, \cdots, s_{287}^3, s_{173}^3 s_{174}^3, s_{174}^3 s_{175}^3, s_{177}^3 s_{178}^3, \cdots, s_{285}^3 s_{286}^3]$$

and $A \in \mathcal{M}_{258,319}(\mathbb{F}_2)$ (binary 258×319 matrix).

The Reorganization of the Matrix A

There are 111 variables in X that can be expressed as the product of two other variables.

We reorganize the matrix A into two parts. We choose in $\mathbb{F}_2$ the bits $s_{226}^3, \cdots, s_{287}^3$ from which we obtain directly the bits $s_{226}^3 s_{227}^3, \cdots, s_{286}^3 s_{287}^3$. Then we have to calculate the bits $s_0^3, \cdots, s_{225}^3$ and the other products. We reorganize the columns of A and therefore the vector X in order to have the assumption bits and their products on the right part of A and the unknown bits on the left part of A. We note $A_{ord} \in \mathcal{M}_{258,319}(\mathbb{F}_2)$ and $X_{ord} \in \mathcal{M}_{258,1}(\mathbb{F}_2)$ the obtained arranged matrix and vector.

$$Z = A_{ord}.X_{ord} \qquad (2)$$

Where X_{ord} is the column vector

$$X_{ord} = [s_0^3, \cdots, s_{92}^3, s_{173}^3, \cdots, s_{225}^3, s_{173}^3 s_{174}^3, s_{174}^3 s_{175}^3, s_{177}^3 s_{178}^3, \cdots, s_{225}^3 s_{226}^3, s_{226}^3, \cdots, s_{287}^3, s_{226}^3 s_{227}^3, \cdots, s_{285}^3 s_{286}^3]$$

and $A_{ord} \in \mathcal{M}_{258,319}(\mathbb{F}_2)$ (binary 258×319 matrix).

The Triangulation of the Matrix A_{ord}

Similarly to the Gaussian elimination method we find a square and invertible binary matrix $U \in \mathcal{M}_{258,258}(\mathbb{F}_2)$ such that

$$U.Z = T.X_{ord} \qquad (3)$$

Where $T = U.A_{ord}$ with $T \in \mathcal{M}_{258,319}(\mathbb{F}_2)$ an upper triangular matrix of rank 198.

As there are many "1" on the diagonal we find almost unknown left bits $X_{ord,i}$.

The Final Linear System

We calculate $Z_U = U.Z$ with Z_U a binary vector of length 258 ($Z_U \in \mathcal{M}_{258,1}(\mathbb{F}_2)$) and write obtained equations from (3) beginning with the last lines. We consider the i^{th} line i. If there is an "1" on the diagonal (which is always true excepted for the three bits $s_{90}^3, s_{91}^3, s_{92}^3$ and some other variables like $s_i^3 s_{i+1}^3$) we get out the variable and obtain a linear equation of type

$$X_{ord,i} = L_{ord,i}(X_{ord,i+1}, \cdots, X_{ord,319}) \quad (4)$$

where $L_{ord,i}$ is a linear function. In the counter case, we can always either exploit the equation to get out an unknown variable or use this equation as a test. If the test is true we go on solving the system else we stop the calculus as there is no solution for these taken $X_{ord,i}$ on hypothesis. Therefore, we can express bits $s_{280}^3, \cdots, s_{287}^3$ in term of bits $s_{226}^3, \cdots, s_{279}^3$ which allows us to reduce the number of hypothesis variables to $s_{226}^3, \cdots, s_{279}^3$. We have added two more conditions by taking some a_i which are polynomial of degree greater than two in order to obtain at least the same number of conditions than assumption bits.

Particular Case of $s_{90}^3, s_{91}^3, s_{92}^3$

There is no equation from (3) that allow to determine the bits $s_{90}^3, s_{91}^3, s_{92}^3$. But these bits are used only after all tests in the system from (3). Therefore, it's possible to not consider them as the bits $s_{226}^3, \cdots, s_{279}^3$ taking on hypothesis and fix them if we are sure that bits $s_{226}^3, \cdots, s_{279}^3$ are good and that only bits $s_0^3, \cdots, s_{89}^3$ remain to determine. The system is shown on Table 5.

Table 5. *Phase 2 - Particular Case of* s_{90}^3 ,s_{91}^3 ,s_{92}^3

Particular case Pseudo code
Input : $s_{226}^3, \cdots, s_{279}^3$
`First part of the system`
Calculation of $s_{226}^3 s_{227}^3, \cdots, s_{278}^3 s_{279}^3$ by the definition of $s_i s_{i+1}$
Calculation of $s_{280}^3, \cdots, s_{287}^3$ with obtain equation from (3)
Calculation of $s_{279}^3 s_{280}^3, \cdots, s_{286}^3 s_{287}^3$
`First tests`
If *false* **Then** *exit*
Calculation of $s_{225}^3, \cdots, s_{220}^3$ and $s_{225}^3 s_{226}^3, \cdots, s_{220}^3 s_{221}^3$ with equations from (3)
Calculation of $s_{219}^3, \cdots, s_{173}^3$ and $s_{219}^3 s_{220}^3, \cdots, s_{173}^3 s_{174}^3$ with equations from (3).
(*each calculus of* $s_i s_{i+1}$ *is followed by several tests*)
`Second part of the system`
(*From here, if all tests have been satisfied then the Input bits are good and we know bits* $s_{173}^3, \cdots, s_{287}^3$)
For $((s_{90}^3, s_{91}^3, s_{92}^3) = (0,0,0), \cdots, (1,1,1))$ **Do**
Calculation of $s_{89}^3, \cdots, s_0^3$ from $s_{173}^3, \cdots, s_{287}^3$ previously obtained from (3)
`Second tests`
If *true* **Then** we have find E_3 and stop
EndFor

The test condition of the second part of the system are derived from the polynomial a_i of degree greater than two which have not been satisfied during the construction of the matrix A and the tests on $y_0 \cdots y_{14}$ which are common with the **Phase 2** of the attack.

Solving the System

All we have to do is to solve the system for all the possible binary 54-tuple $(s_{226}^3, \cdots, s_{279}^3)$. If all test have been verified during the solution of the system then assumption bits $s_{226}^3, \cdots, s_{279}^3$ are right. We have found the entire internal state E_3. Else, the solution of the system stops at the first false test condition.

Phase 2

We have determined all the 288 bits $s_0^3, \cdots, s_{287}^3$ of the internal state E_3. In this second phase, the goal is to find all the 288 bits $s_0^2, \cdots, s_{287}^2$ of the internal state E_2 from the last 84 bits $y_0, \cdots, y_{83}$ of generated keystream.

The $s_{93}^2, \cdots, s_{161}^2$ *Bits*

Finding the 288 bits $s_0^2, \cdots, s_{287}^2$ of the internal state E_2 is equivalent to determine the 80 bits of IV which have been reseted at the beginning of **Phase 2**. We use the 84 bits $y_0, \cdots, y_{83}$ of the obtained keystream before the reset. We counter write Trivium generation equations and write the y_i in terms of internal state bits of E_2. But as the size of these obtained equations increase exponentially and their degree goes up linearly it is hard full to write them or

simplify them after 20 counter rounds. Fortunately the last y_i have the following form.

$$\begin{aligned}
y_{83} &= s_{93}^2 + P_{93}(s_0^2, \cdots, s_{92}^2, s_{162}^2, \cdots, s_{287}^2)\\
y_{82} &= s_{94}^2 + P_{94}(s_0^2, \cdots, s_{93}^2, s_{162}^2, \cdots, s_{287}^2)\\
y_{81} &= s_{95}^2 + P_{95}(s_0^2, \cdots, s_{94}^2, s_{162}^2, \cdots, s_{287}^2)\\
y_{80} &= s_{96}^2 + P_{96}(s_0^2, \cdots, s_{95}^2, s_{162}^2, \cdots, s_{287}^2)\\
&\vdots = \vdots\\
y_{15} &= s_{161}^2 + P_{161}(s_0^2, \cdots, s_{160}^2, s_{162}^2, \cdots, s_{287}^2)
\end{aligned}$$

Where P_i is a polynomial function . If we know bits $s_{162}^2, \cdots, s_{172}^2$ and E_3 (it means bits $s_0^2, \cdots, s_{92}^2, s_{173}^2, \cdots, s_{287}^2$) we can solve the system. For $i = 93$ to $i = 161$ we can determine $P_i(s_0^2, \cdots, s_{i-1}^2, s_{162}^2, \cdots, s_{287}^2)$ with $94 - i$ counter rounds iteration of Trivium from the internal state E_3 in which we add the bits $s_{162}^2, \cdots, s_{172}^2$ and $s_{92}^2, \cdots, s_{i-1}^2$ of the internal state E_2. Therefore we can evaluate the bits s_i^2 of the internal state E_2. We repeat 69 times this procedure and obtain all 288 bits $s_0^2, \cdots, s_{287}^2$ of the internal state E_2 from the internal state E_3 and the bits $s_{162}^2, \cdots, s_{172}^2$.

$$\begin{aligned}
s_{93}^2 &\leftarrow y_{83} + P_{93}(s_0^2 \ldots s_{92}^2, s_{162}^2 \ldots s_{287}^2)\\
s_{94}^2 &\leftarrow y_{82} + P_{94}(s_0^2 \ldots s_{93}^2, s_{162}^2 \ldots s_{287}^2)\\
&\vdots \leftarrow \vdots\\
s_{161}^2 &\leftarrow y_{15} + P_{161}(s_0^2 \ldots s_{160}^2, s_{162}^2 \ldots s_{287}^2)
\end{aligned}$$

The Internal State E_2

To find the bits $s_{93}^2, \cdots, s_{172}^2$ we proceed as the following.

> We assume the bits $s_{162}^2, \cdots, s_{172}^2$.
> We calculate the bits $s_{93}^2, \cdots, s_{161}^2$ from the bits $y_{15}, \cdots, y_{83}$ by the previous method. We will have the internal state E_2 if the previous assume bits are good.
> We verify that we obtain the bits $y_0, \cdots, y_{14}$ by counter round the previous state from 70 to 84 rounds. If it's the case we have find the internal state E_2 else we have to change the assume bits of the first step and replay the procedure.

To verify the 11 assume bits $s_{162}^2, \cdots, s_{172}^2$ with the 15 bits $y_0, \cdots, y_{14}$ of keystream is typically an empirical result. Some experimental tests have shown us that if we take less than 11 bits we could find several states E_2 from bits y_i

and from E_3. We don't prove this result as bits $y_0, \cdots, y_{14}$ are polynomial functions of bits of E_2 for which the degree is around 70.

We have guessed the internal state E_3 during **Phase 1** then the internal state E_2 in **Phase 2**, we now have all the necessary information to determine for example the keystream bits generated before the reset by counter rounding Trivium. As the main goal of this paper is to attack Trivium, it means guess both the secret key bits and the IV bits, we go on with the last phase.

1. Phase 3

We have guessed all the 288 bits $s_0^2, \cdots, s_{287}^2$ of the internal state E_2. In this last phase, it remains to find all the 288 bits $s_0, \cdots, s_{287}$ of the first internal state E_0.

We counter round Trivium (Table 6) from the 288-bit internal state E_2 until all the bits $s_{80}, \cdots, s_{92}, s_{173}, \cdots, s_{284}$ are equal to "0" and the three bits $s_{285}, s_{286}, s_{287}$ are equal to "1" ($1152 + N + 84$ counter rounds in total). At the end we obtain both the 80-bit secret key and the 80-bit Initial Values of the Full Trivium.

Table 6. *Phase 3 - Pseudo-Code*

Counter round equations of TRIVIUM
$t_1 \leftarrow s_{93} + s_{91}.s_{92} + s_{171}$
$t_2 \leftarrow s_{177} + s_{175}.s_{176} + s_{264}$
$t_3 \leftarrow s_0 + s_{286}.s_{287} + s_{69}$
$z_i \leftarrow t_1 + t_2 + t_3$
$t_1 \leftarrow t_1 + s_{66}$
$t_2 \leftarrow t_2 + s_{162}$
$t_3 \leftarrow t_3 + s_{243}$
$(s_0, s_1, \cdots, s_{287}) \leftarrow (s_1, \cdots, s_{287}, s_0)$
$s_{92} \leftarrow t_1$
$s_{176} \leftarrow t_2$
$s_{287} \leftarrow t_3$

The Complexity

In **Phase 1** we have to test all the possible binary 54-tuple $(s_{226}^3, \cdots, s_{279}^3)$ of the system done by (4). With an optimized implementation using 64-bit type structure in which we fix the values of $(s_{226}^3, \cdots, s_{231}^3)$ relatively to Table 7 in order to obtain all the possible 6-tuple bitwise these 6 variables. Therefore, testing all the 54-tuple $(s_{226}^3, \cdots, s_{279}^3)$ is equivalent to test all the possible 48-tuple $0xFFFFFFFF.(s_{232}^3, \cdots, s_{279}^3)$. Finally, **Phase 1** has a maximal complexity equal to 2^{48} (with a complexity mean equal to 2^{47}). We have to try

out all the possible 48-tuple $(s^3_{232}, \cdots, s^3_{279})$ in the worst case.

Table 7. *Choices of* $s^3_{226}, \cdots, s^3_{231}$

6-tuple	distribution
s^3_{226}	$\leftarrow$ 000000000000000000000000000000011111111111111111111111111111111
s^3_{227}	$\leftarrow$ 000000000000000011111111111111110000000000000000111111111111111
s^3_{228}	$\leftarrow$ 000000001111111100000000111111110000000011111111000000001111111
s^3_{229}	$\leftarrow$ 000011110000111100001111000011110000111100001111000011110000111
s^3_{230}	$\leftarrow$ 001100110011001100110011001100110011001100110011001100110011001
s^3_{231}	$\leftarrow$ 010101010101010101010101010101010101010101010101010101010101010

Phase 2 has a maximal complexity equal to 2^{11} (with a complexity mean equal to 2^{10}). We have to try out all the possible 11-tuple $(s^2_{162}, \cdots, s^2_{172})$ in the worst case which represents only 2,048 cases to test.

Phase 3 has a complexity equal to N which could appear very high if we build our attack with a large number of generated keystream bits. But as the maximal keystream length of Trivium is 2^{64} then the maximal complexity of this phase is 2^{64} in the worst case.

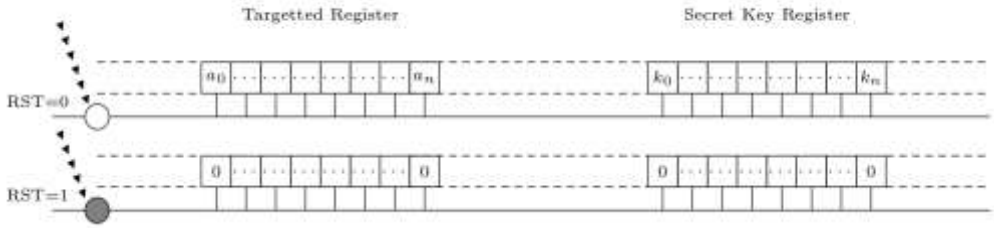
The three phases of our attack are totally independent and are executed sequentially and only once. Therefore the complexity C of the entire attack is the sum of the complexity of each of its phases: $C = 2^{48} + 2^{11} + N$. We assume that $N < 2^{47}$ which means that we build our attack after generating 2^{47} bits of keystream. This assumption is highly credible relatively to this quantity of generated bits (128 terabytes). Therefore we can establish the complexity of the complete attack.

$$C = O(2^{48})$$

Proposed Countermeasures

We have shown that Trivium is weak under hard fault attack. A hardware countermeasure could help the encryption machine equipped with Trivium to avoid and/or detect faults. This attack targets the reset wire of the 80-bit initial value (IV) register in order to stick it to a constant. A suitable countermeasure should be an efficient protection with light detectors, all over the reset wire (high cost) or around the targeted register (low cost) in order to increase the introduction difficulty and to make impossible the repetitively. This protection could be active and react to the reset fault by sticking the secret key bits register to a constant, loosing definitively its original value (Figure 4).

Figure 4. *Countermeasure on the Reset Wire*



Conclusion

This paper presents a new and efficient attack on the hardware-oriented synchronous stream cipher Full Trivium. This original hard fault reset based attack allows to recover both the 80-bit secret key and the 80-bit IV with a complexity less than $\mathcal{O}(2^{48})$. The main idea of this attack is to exploit the 80-bit initial value (IV) register by targeting its reset wire in order to stick it to a constant whenever during the cipher stream generation. Although this attack does not rest on any particular assumption, it needs to have a full version of Trivium implemented on an encryption machine. This is a realistic two-stage attack with an operational side and an offensive side which can be made in a hardware context. We have proposed a suitable countermeasure using an efficient active protection with light detectors to face this attack. An other effective idea of this attack is the particular bits structure which considers the product of two consecutive variables $s_i s_{i+1}$ like a simple variable s_i and the implementation improvement with a 64-bit type structure. As Trivium allows for highly parallelized implementations, this attack should succeed rapidly with a fast and powerful clustered computers architecture. A further research investigation could be to implement this attack on the escargot ASIC design [28] developed at the University of sheffield where development and performance assessment of suitable hardware designs which implement the candidate cipher primitives have been made. This attack seems to offer great investments opportunities and reinforce our motivation to go on with this new kind of approach.

References

[1] eSTREAM Project, *http://www.ecrypt.eu.org/stream/*
[2] C. De Cannière, B. Preneel, A stream cipher construction inspired by block cipher design principles, *http://www.ecrypt.eu.org/stream/triviump3.html*, (2006).
[3] C. De Cannière, B. Preneel, Trivium Specs., h*ttp://www.ecrypt.eu.org/stream/triviump3.html*, (2005).
[4] H. Raddum, Cryptanalytic Results on Trivium, *eprint.iacr.org/2006/039*, (2006).
[5] S. Babbage, Some thoughts on Trivium, *eprint.iacr.org/2007/007*, (2007).
[6] H. Englund, T. Johansson, M. S. Turan, A Framework for Chosen IV Statistical

Analysis of Stream Ciphers, INDOCRYPT, (2007).
[7] S. Fischer, S. Khazaei, W. Meier, Chosen IV Statistical Analysis for Key Recovery Attack on Stream ciphers, AFRICACRYPT, (2008).
[8] E. Pasalic, Transforming chosen IV attack into a key differential attack : how to break Trivium and similar designs, *eprint.iacr.org/2008/443*, (2008).
[9] E. Pasalic, Key differentiation attacks on stream cipher, *eprint.iacr.org/2008/443*, (2008).
[10] A. Maximov, A. Biryukov, Two Trivial Attacks on Trivium, *ecrypt.eu.org/stream/ triviump3.html*, (2007).
[11] C. McDonald, C. Charnes, J. Pieprzyk, Attacking Bivium with MiniSAT, *ecrypt.eu. org/stream//triviump3.html*, (2007).
[12] M. S. Turan, O. Kara, Linear Approximations for 2-round Trivium, *ecrypt.eu.org/ stream/triviump3.html*, (2007).
[13] M. Vielhaber, Breaking ONE.FIVIUM by AIDA, an Algebraic IV Differential Attack, *eprint.iacr.org/2007/413*, (2007).
[14] H. Yupu, G. Juntao, L. Qing, Floating Fault Analysis of Trivium under Weaker Assumptions, *eprint.iacr.org/2009/169*, (2009).
[15] Y. Hu, F. Zhang, Y. Zhang, Hard Fault Analysis of Trivium, *eprint.iacr.org/2009/ 333*, (2009).
[16] D. Priemuth-Schmid, A. Biryukov, Slide Pairs in Salsa20 and Trivium, *eprint.iacr. org/2008/405*, (2008).
[17] M. Hojsik, B. Rudolf, Differential Fault Analysis of Trivium, LNCS 5086, *FSE 2008*, pp. 158-172, Springer/Heidelberg (2008).
[18] M. Hojsik, B. Rudolf, Floating Fault Analysis of Trivium, LNCS 5365, *IndoCrypt 2008*, pp. 229-250, Springer/Heidelberg (2008).
[19] E. Biham, A. Shamir, Differential Fault Analysis of Secret Key Cryptosystems, LNCS 1294, *Advance in Cryptology-Crypto 97*, pp. 513-525, Springer/Heidelberg (1997).
[20] M. Afzal, A. Masood, Modification in the Design of Trivium to Increase its Security Level, *eprint.iacr.org/2009/250*, (2009).
[21] Y. Tian, G. Chen, J. Li, On the Design of Trivium, *eprint.iacr.org/2009/431*, (2009).
[22] I. Dinur, A. Shamir, Cube Attacks on Tweakable Black Box Polynomials, *eprint.iacr. org/2008/385*, (2008).
[23] S.S. Bedi, N.R. Pillai, Cube Attacks on Trivium, *eprint.iacr.org/2009/015*, (2009).
[24] J.P. Aumasson, I. Dinur, W. Meier, A. Shamir, Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium, LNCS 5665, *FSE 2009*, pp. 1-22, Springer/Heidelberg (2009).
[25] I. Dinur, A. Shamir, Slide Channel Cube Attacks on Block Ciphers, *eprint.iacr.org/20 09/127*, (2009).
[26] A. Zhang, C.W. Lim, K. Khoo, Extensions of the Cube Attack, *eprint.iacr.org/20 09/049*, (2009).
[27] A. Zhang, C.W. Lim, K. Khoo, W. Lei, J. Pieprzyk, Extensions of the Cube Attack based on Low Annihilators, *eprint.iacr.org/2009/049*, (2009).
[28] eSCARGO project : Sheffield University, *http://www.sheffield.ac.uk/eee/escargot/*.