Some observations on ACORN v1 and Trivia-SC

Rebhu Johymalyo Josh[1] and Santanu Sarkar[2]

[1] Chennai Mathematical Institute,
SIPCOT IT Park, Siruseri,
Chennai- 603103, India
rebhu.webs@gmail.com
[2] Department of Mathematics,
Indian Institute of Technology Madras,
Chennai - 600036, India.
sarkar.santanu.bir@gmail.com

Abstract. In the first part of this paper, we study the security of Acorn v1, an authenticated encryption scheme submitted to the ongoing CAESAR competition. We perceive some interesting outcomes on the key stream bits of Acorn v1. In fact we observe that bit wise XOR of the first key stream bits for a fixed Key and IV but different associated data becomes 0.
In the second part of this paper, we provide slid pairs of modified Trivia-SC. For the original Trivia-SC, finding a slid pair is trivial as the padding is symmetric. Hence, it is in general assumed that finding slid pairs is difficult for asymmetric padding. Here we show that in this case also, getting a slid pair is possible.

Keywords: Acorn, Cube Attack, Cryptanalysis, SAT Solver, Stream Cipher, Trivia-SC.

1 Introduction

Acorn is a lightweight authenticated cipher which has been submitted to the ongoing CAESAR [9] competition. It uses a single State Register for encryption and authentication. It updates the state for 512 + *length of associated data* before encrypting the plain text. For encrypting each bit, the state update is run once each time for each character. This is used for encrypting the next bit of plain text. The current state is used for generating a bit called *Key stream Bit* which is XOR ed with the bit of plain text and output as one bit of cipher text.

Cube attacks, introduced by Dinur and Shamir [15], have been used extensively for cryptanalysis of Block ciphers and Stream ciphers. Cube attacks are very efficient on stream ciphers based on low degree NFSRs. Cube attacks can recover a secret key through queries to a black box polynomial using IV bits.

We experimented on Acorn using a modified cube attack by keeping the key and IV fixed, and changing the associated data. While analysing the values of the state which is later used for generation of the key stream at various point of the algorithm, we unearthed that for small length associated data, bitwise xor of the key stream bits remains 0 for almost every key and IV. We

further noticed that for any random key, IV, the length of *associated data* plays a compelling role. We ventured with all possible values of *assoiated data* and found that if *key stream Bit* is XORed for all possible *assoiated data*, then it yields 0 as the value; which signifies that we have even number of 1's as *key stream Bit*. Our motive in this paper is to recommend that *State Update* in the processing part should be run length of *associated data* + 1024 times instead of length of *associated data*+ 512 so that XORed value gives purely random values instead of homogeneous 0's.

Algebraic attack [10, 11] on a cipher relies on the solving of a system of multivariate polynomials. As an important result in this area, the block cipher Keeloq has been craked in real time using the algebraic attack [21, 3]. There are several methods to solve multivariate polynomials. There are several approaches using Gröbner basis such as Buchberger [8], F4 [18] etc. When the underlying field of the cipher is $GF(2)$, SAT solver is another elegant method in this direction that tries to solve the famous satisfiability (SAT) problem. The Boolean formula is given in the Conjunctive Normal Form (CNF). Massacci [24] introduced the idea of using SAT solvers towards cryptanalysis. The stream cipher Bivium was attacked using SAT solver [17] too. A slide-algebraic attack on Keeloq has been presented in [12] using the SAT solver.

The stream cipher Trivium [13] is in the hardware profile of the eStream portfolio [16] that has been designed by De Cannière and Preneel. It is a synchronous bit oriented cipher. The cipher is allowed to generate up to 2^{64} key stream bits from an 80 bit secret key and an 80 bit public IV. The design of Trivium is simple as well as sumptuous and it has been designed so as to require low hardware complexity. This cipher has the state size of 288 bits, consisting of three interconnected non-linear (degree 2) feedback shift registers of length 93, 84 and 111 bits respectively.

A Differential Fault Attack on Trivium has been presented in [25] that exploits SAT solvers. Biryukov and Wagner first introduced slide attack [6]. One alluring property of the slide attack is that its complexity is independent of the number of rounds of the cipher. Slide attacks on the block cipher GOST were presented in [7, 5]. The slide attack on stream cipher is to study how given a Key-IV, one can efficiently obtain another Key-IV such that the generated output key streams in Pseudo Random Generator Algorithm (PRGA) are exact shifts of each other throughout the key stream generation. These Key-IV pairs are referred as "slid pairs" following [23, Section 3.2]. That is, in this model, the attacker needs to provide two different key-IV pairs (K_1, IV_1) and (K_2, IV_2), such that they can generate same key streams with c many bit shifts. These pairs are denoted as $[(K_1, IV_1), (K_2, IV_2), c]$ in [23] while analysing Trivium. The work of [23] could achieve slid pairs up to 115 shifts. Later it was improved up to 212 shifts in [1]. Slide attacks on the stream ciphers Grain v1 and Grain 128 were proposed in [14, 22]. Recently slide attack is mounted on Grain 128a in [4]. For Trivia-SC, mounting slid attack is very easy as proposed in [20]. This is because padding in Trivia-SC is symmetric. Hence attacker proposed to use

asymmetric padding in Trivia-SC. In this paper, we have presented slid pair for this modified version of Trivia-SC up to 280 shifts.

2 Acorn

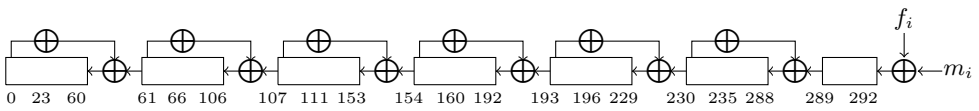

Fig. 1. The concatenation of 6 LFSRs in ACORN. f_i indicates the overall feedback bit for the ith step; m_i indicates the message bit for the i-th step.

index / array	**0 to 127**	**128 to 255**	**256**	**257 to 1535**
m	k	IV	1	0
a	1	1	1	1
b	1	1	1	1

Table 1. Initialization

index / array	**0 to *length(ad)-1***	***length(ad)***	***length(ad)+1* to *length(ad)+255***	***length(ad)+256* to *length(ad)+511***
m	ad	1	0	0
a	1	1	1	0
b	1	1	1	1

Table 2. Processing

2.1 Description of the Cipher

Notations:

- d : data of 1536 bits
- k : the key used of 128 bits
- IV : IV used of 128 bits
- S_i : Current State of 293 bits
- $S_{(i+1)}$: Next State of 293 bits
- a : control bit of 1536 bits
- b : control bit of 1536 bits
- p : plain text
- ad : associated data

Algorithm 1 Pseudo Code

function UPDATE(S_i,m_i,a_i,b_i)
$S_{i,289} \leftarrow S_{i,289} \oplus S_{i,235} \oplus S_{i,230}$
$S_{i,230} \leftarrow S_{i,230} \oplus S_{i,196} \oplus S_{i,193}$
$S_{i,193} \leftarrow S_{i,193} \oplus S_{i,160} \oplus S_{i,154}$
$S_{i,154} \leftarrow S_{i,154} \oplus S_{i,111} \oplus S_{i,107}$
$S_{i,107} \leftarrow S_{i,107} \oplus S_{i,66} \oplus S_{i,61}$
$S_{i,61} \leftarrow S_{i,61} \oplus S_{i,23} \oplus S_{i,0}$
$f_i \leftarrow$ FEEDBACK(S_i, a_i, b_i)
for j:=0 to 291 **do**
$S_{i+1,j} \leftarrow S_{i,j+1}$
$S_{i+1,292} \leftarrow f_i \oplus m_i$
function FEEDBACK(S_i,a_i,b_i)
$ks_i \leftarrow$ KEY STREAM(S_i)
return $f_i \leftarrow S_{i,0} \oplus (\sim S_{i,107}) \oplus (S_{i,244}\&S_{i,23}) \oplus (S_{i,244}\&S_{i,160}) \oplus (S_{i,23}\&S_{i,160}) \oplus (S_{i,230}\&S_{i,111}) \oplus ((\sim S_{i,230})\&S_{i,66}) \oplus (a_i\&S_{i,196}) \oplus (b_i\&ks_i)$
function KEY STREAM(S_i)
return $ks_i \leftarrow S_{i,12} \oplus S_{i,154} \oplus (S_{i,235}\&S_{i,61}) \oplus (S_{i,235}\&S_{i,193}) \oplus (S_{i,61}\&S_{i,193})$
function INITIALIZATION
$S_0 \leftarrow 0$
for i:=0 to 127 **do**
$m_i \leftarrow k_i$
for i:=0 to 127 **do**
$m_{i+128} \leftarrow IV_i$
$m_{256} \leftarrow 1$
for i:=1 to 1279 **do**
$m_{256+i} \leftarrow 0$
for i:=1 to 1535 **do**
$a \leftarrow 1$
$b \leftarrow 1$
for i:=0 to 1535 **do**
$S_{i+1} \leftarrow$ UPDATE(S_i, m_i, a_i, b_i)
function PROCESSING
for i:=0 to length(ad)-1 **do**
$m_i \leftarrow ad_i$
$m_{length(ad)} \leftarrow 1$
for i:=1 to 511 **do**
$m_{i+length(ad)} \leftarrow 0$
for i:=0 to 511+length(ad) **do**
$b_i \leftarrow 1$
for i:=0 to 255+length(ad) **do**

$a_i \leftarrow 1$

for i:=255 to 511 **do**

$a_{i+length(ad)} \leftarrow 0$

for i:=0 to 511 + length(ad) **do**

$S_{i+1} \leftarrow$ UPDATE(S_i, m_i, a_i, b_i)

function ENCRYPTION

for i:=0 to length(p)-1 **do**

$m_{i+length(ad)+512} \leftarrow p_i$

$m_{length(p)+length(ad)+512} \leftarrow 1$

for i:=0 to 511 **do**

$m_{i+length(p)+length(ad)+512} \leftarrow 0$

for i:=length(ad)+512 to length(ad)+length(p)+767 **do**

$a_i \leftarrow 1$

for i:=length(p)+length(ad)+768 to length(ad)+length(p)+1023 **do**

$a_i \leftarrow 0$

for i:=length(ad)+512 to length(ad)+length(p)+1023 **do**

$b_i \leftarrow 0$

for i:=length(ad)+512 to length(ad)+length(p)+1023 **do**

$S_{i+1} \leftarrow$ UPDATE(S_i, m_i, a_i, b_i)

$c_i = p_i \oplus$ KEY STREAM(S_i)

return c

function MAIN(p,ad,key,IV)

INITIALIZATION

PROCESSING

ENCRYPTION

$Print(c)$

Here, we are ommitting the part where authentication tag is generated as our analysis is regarding the cipher text only.

2.2 Our Analysis on Acorn

The cube attack is a method of cryptanalysis applicable to a wide variety of symmetric-key algorithms, published by Itai Dinur and Adi Shamir. It have been used extensively for cryptanalysis of Block ciphers and Stream ciphers. Cube attacks are very efficient on stream ciphers based on low degree NFSRs. Cube attacks can recover a secret key through queries to a black box polynomial using IV bits. In our analysis, we have tweaked the cube attack by keeping the key and IV persistent and went ahead with altering the associated data. While examining the values of the *State* at different points of the program before the encryption, we noticed that we have equal number of 0's and 1's returned by *Key stream* function during *Encryption*. We XOR ed each value obtained and found that at the end of 64000 random keys and plain text and all possible

associated data of given length, the value obtained is 0. Say the length of *associated data* is n, then with all 2^n possible *associated data*, the XOR ed value is 0 till some specific bit corresponding to n. We bunched together 64 keys together for single place in an unsigned long long integer in C by left shifting 64 randomly generated bits and then operated on it normally as it is described in the algorithm. We also bunched 64 IVs together for single place in an unsigned long long integer in C. This will help in finding the experimental results at least 32 times faster and won't hamper the normal result as we will experiment with randomly generated keys and IVs.

2.2.1 Experimental Results We have executed the code on Ubuntu 14.04 64-bit with hardware specifications of Intel Core i5-3210M dual-core CPU @ 2.50GHz × 4 threads, 8 GB RAM. We used multi-threading to find out the bit index till which the XOR ed value is 0 for different length of *associated data*.

Length x of *Associated Data*	**Bit y till which XOR ed value is 0**
10	341
12	349
14	358
16	371
18	379

Table 3. Experimental results for different *Associated Data* length on Acorn.

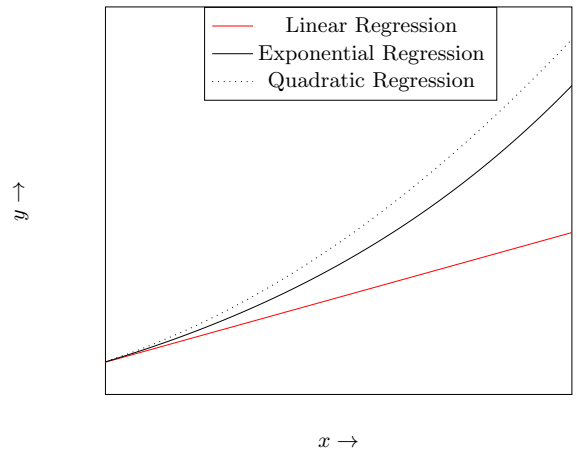

Fig. 2. Extrapolation with different methods.

2.2.2 Extrapolation We used different extrapolation to find out the length of *Associated Data* for which the first 512 bits will be 0. Here y is the number

of bits and x is the length of *Associated Data*. Different regression formulae have been obtained from [29].

Linear Regression. Our formula is in this case $y = 4.9 \times x + 291$. Thus when $x = \mathbf{45}$, y becomes 512.

Exponential Regression. Formula $y = 296.9076103 \times e^{1.363091448 \times 10^{-2} x}$. So $y = 512 \Leftrightarrow x \approx \mathbf{40}$

Quadratic Regression. Formula $y = 7.142857143 \times 10^{-2} x^2 + 2.9x + 304.4285714$ gives $y = 512 \Leftrightarrow x \approx \mathbf{37}$

2.3 Inference

From the above extrapolation, we say that if the length of *Associated Data* is roughly about 45 bits or more, then updating for 512 rounds and XORing the obtained *key stream* bit over all possible data will yield 0. Hence it is conspicuous that a mere 512 rounds of updating the *State* in the processing part may not be sufficient for proper mixing. Hence if we have the *State Update* for 1024 rounds instead of 512, then for Linear Regression, $y = 1024 \Leftrightarrow x \approx \mathbf{150}$; for Exponential Regression, $y = 1024 \Leftrightarrow x \approx \mathbf{91}$; and for Quadratic Regression, $y = 1024 \Leftrightarrow x \approx \mathbf{82}$. According to the model of Acorn, $0 \leq$ *length of Associated Data* $\leq 2^{64}$. So 1024 rounds of *State Updation* will be a fortress to prevent an attack from the cipher text.

3 TriviA-ck

TriviA-ck-v1 [2] is an AEAD (Authenticated Encryption with Associated Data) scheme, designed by Chakraborti and Nandi, and it is submitted to the ongoing "CAESAR - the Competition for Authenticated Encryption: Security, Applicability, and Robustness." It uses the stream cipher Trivia-SC (sometimes referred to as SCTrivia or SC-Trivia, e.g., in [2, page 18]) and an efficient universal hash function VPV-Hash. Trivia-SC is inspired from Trivium, but with a larger state and key size.

3.1 Description of the Cipher

Before proceeding further, let us describe the structures of Trivia-SC. Table 4 gives an brief overview.

Table 4. Overview of Trivia-SC

Cipher	Key size	IV size	State size	Initialization rounds	# Bits involved in key stream	# Operations in key stream
Trivia-SC	128	128	384	1152 ($= 3 \times 384$)	8	6 $\oplus$, 1 $\wedge$

We use the convention of numbering for both key and IV bits as $1, 2, 3, \ldots$. Accordingly, we use $k_1, k_2, k_3, \ldots$ and $v_1, v_2, v_3, \ldots$ to represent the key and IV bits, respectively. We explain Trivia-SC with three arrays A, B, C.

Algorithm 2 Trivia-SC: KLA

1: $(A_1, A_2, \ldots, A_{132}) = (k_1, k_2, \ldots, k_{128}, 1, 1, 1, 1)$
2: $(C_1, C_2, \ldots, C_{147}) = (v_1, v_2, \ldots, v_{128}, 1, \ldots, 1)$
3: $(B_1, B_2, \ldots, B_{105}) = (1, 1, \ldots, 1)$

Algorithm 3 Trivia-SC: KSA & PRGA

1: **for** $i = 1$ to N **do**
2: $t_1 \leftarrow A_{66} \oplus A_{132} \oplus (A_{130} \wedge A_{131}) \oplus B_{96}$
3: $t_2 \leftarrow B_{69} \oplus B_{105} \oplus (B_{103} \wedge B_{104}) \oplus C_{120}$
4: $t_3 \leftarrow C_{66} \oplus C_{147} \oplus (C_{145} \wedge C_{146}) \oplus A_{75}$
5: $(A_1, A_2, \ldots, A_{132}) \leftarrow (t_3, A_1, \ldots, A_{131})$
6: $(B_1, B_2, \ldots, B_{105}) \leftarrow (t_1, B_1, \ldots, B_{104})$
7: $(C_1, C_2, \ldots, C_{147}) \leftarrow (t_2, C_1, \ldots, C_{146})$
8: $z_i \leftarrow A_{66} \oplus A_{132} \oplus B_{69} \oplus B_{105} \oplus C_{66} \oplus C_{147} \oplus (A_{102} \wedge B_{66})$

Let us now describe the stream cipher Trivia-SC. Trivia-SC consists of 3 NFSRs: A, B and C of size 132, 105 and 147 bits; whose bits are represented as $A_1, A_2, \ldots, A_{132}$, $B_1, B_2, \ldots, B_{105}$ and $C_1, C_2, \ldots, C_{147}$ respectively. We represent the internal state register (384 bits) by S and denote its bits by $S_1, S_2, \ldots, S_{384}$, where $A \equiv (S_1, S_2, \ldots, S_{132})$, $B \equiv (S_{133}, S_{134}, \ldots, S_{237})$ and $C \equiv (S_{238}, S_{239}, \ldots, S_{384})$. The KLA routine is described in algorithm 2 which is followed by the KSA. The evolution is done for 1152 rounds without producing any key stream bit, which is followed by PRGA, which are given in algorithm 3.

3.2 Attack using SAT Solver

3.2.1 Description of the attack using SAT solver In Trivia-SC, we have 128-bit secret key and 128-bit IV. We consider a total of 128+128=256 variables which corresponds to respective Key and IV. After c rounds of KSA, the internal state $S^{(c)}$ will be a function of key and IV for any non-negative integer c. Now if $S^{(c)}$ satisfies $384 - 256 = 128$ fixed locations of the starting state $S^{(0)}$, then $S^{(c)}$ would also be a starting state of a different key and IV. The situation is presented pictorially in Figure 3.

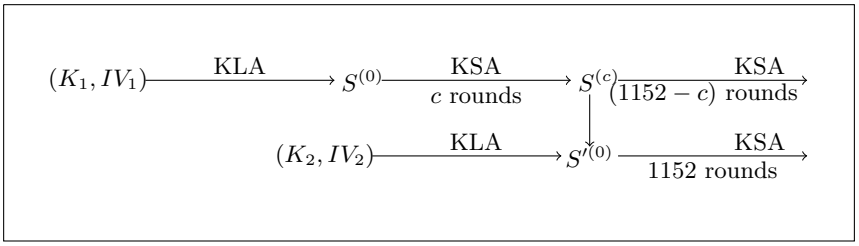

Fig. 3. Slid attack on Trivia-SC

In [20], it has been shown that slid pairs in Trivia-SC is not difficult.

The idea is as follows: In Trivia-SC, the initial state $S = (A, B, C)$ is loaded as

$A = (k_1, \ldots, k_{128}, 1, 1, 1, 1)$

$B = (\overbrace{1, \ldots, 1}^{105})$

$C = (v_1, \ldots, v_{128}, \overbrace{1, \ldots, 1}^{19})$.

After clocking the registers once, the state becomes $S' = (A', B', C')$, where $A' = (t_3, k_1, \ldots, k_{128}, 1, 1, 1)$, $B' = (t_1, 1, \ldots, 1)$ and $C' = (t_2, v_1, \ldots, v_{128}, 1, \ldots, 1)$.

Thus if $k_{128} = v_{128} = t_1 = 1$, state S' is also a valid initial state, generated by the key $(t_3, k_1, \ldots, k_{127})$ and IV $(t_2, v_1, \ldots, v_{127})$. So S and S' generate 1-bit shifted key streams. To overcome this attack, it has been suggested in [20] to take first bit of B register as 0.

Thus to find slid pairs in this modified version, we have 128 equations over 256 variables. We used SAT solver to obtain the solutions. Note that some locations of the state $S^{(c)}$ would be complicated if we write the complete expressions in the CNF form as c increases. To overcome this bottleneck, at each round of KSA, we introduce three new variables x_i, y_i and z_i and replace t_1, t_2 and t_3 by x_i, y_i and z_i respectively. Thus, at each round, we introduce three new variables and three equations. So after c rounds, we have $128 + 3c$ equations (128 many fixed values in a starting state) over $256 + 3c$ variables. Since we have more variables than constraints, it is expected to find a solution to such a system of equations. To solve these, we use the SAT solver Cryptominisat-2.9.5 [27] installed in SAGE [28].

3.2.2 Example We have implemented the code in SAGE 5.12 on a Linux Mint 17.1 Cinnamon 64-bit. The hardware platform is a laptop with a Intel Core i5-4200U @ 1.6 Ghz × 2 and 4 GB RAM. Below, we present one slid pair in hexadecimal form.

Shift c	Solution time in seconds
270	1.23
272	2.24
274	6.12
276	15.82
278	29.74
280	407.11

Table 5. Experimental results for different shifts on modified Trivia-SC.

In example 1, we obtain (K_1, IV_1) and (K_2, IV_2) in 407.11 seconds, and these Key-IV pairs produce 280 bit shifted key streams.

Example 1. K_1 : `09c2e824283614f6034c2fee86e2e9b6`

IV_1 : `e6cf0aac80f27ea07436c1c05137582b`.

key streams: `b3425795e81caa3a6d5e934a464427c7251748080a7e50`
`bdb3a0de00196662eff03370` $\overline{\texttt{484d089cebca}}$ $\overline{\texttt{7e28e90cfe0}}$

K_2 : `2c41a48c695055f80c6b23d4c5c9db96`

IV_2 : `cf7040af7c63795f0d254746d25c778b`

key streams: `484d089cebca7e28e90cfe085926d614a476dca7c1424`

4 Conclusion

From the experiments we conducted on Acorn v1, our analysis concludes that it is might not be totally secure with length of *associated data* + 512 rounds of *State Update* in the processing of associated data. 1024 + length of *associated data* rounds of *State Update* will be safer than the proposed approach.

In the second part of the paper, we study a modified version of TriviA-ck, where padding is asymmetric. For symmetric padding, getting slid pair is very easy. We have shown that it is possible to obtain slid pairs even in the case of asymmetric padding.

References

1. A. Baksi, S. Maitra and S. Sarkar. An Improved Slide Attack on Trivium. IPSI Transaction on Internet Research. Published in January, 2015
2. A. Abortion and M. Nani. TriviA-ck-v1. Available at `http://competitions.cr.yp.to/round1/triviackv1.pdf`.

3. W. Aerts, E. Biham, D. De Moitie, E. De Mulder, O. Dunkelman, S. Indesteege, N. Keller, B. Preneel,G. A. E. Vandenbosch and I. Verbauwhede. A Practical Attack on KeeLoq. J. Cryptology, 25(1):136–157, 2012.
4. S. Banik, S. Maitra, S. Sarkar and M. S. Turan. A Chosen IV Related Key Attack on Grain-128a. In ACISP 2013, LNCS, Vol. 7959, pp. 13–26, 2008.
5. E. Biham, O. Dunkelman and N. Keller. Improved Slide Attacks. In FSE 2007, LNCS, Vol. 4593, pp. 153–166, 2007.
6. A. Biryukov and D. Wagner. Slide Attacks. In FSE 1999, LNCS, Vol. 1636, pp. 245–259, 1999.
7. A. Biryukov and D. Wagner. Advanced Slide Attacks. In EUROCRYPT 2000, LNCS, Vol. 1807, pp. 589–606, 2000.
8. B. Buchberger. An Algorithm for Finding the Basis Elements of the Residue Class Ring of a Zero Dimensional Polynomial Ideal. PhD thesis, Johannes Kepler University of Linz (JKU), 1965.
9. Caesar: Competition for authenticated encryption: Security, applicability, and robustness. `http://competitions.cr.yp.to/caesar.html`.
10. N. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Over-defined Systems of Equations. In ASIACRYPT 2002, LNCS, Vol. 2501, pp. 267–287, 2002.
11. N. Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In CRYPTO 2003, LNCS, Vol. 2729, pp. 176–194, 2003.
12. N. Courtois, G. V. Bard and D. Wagner. Algebraic and Slide Attacks on KeeLoq. In FSE 2008, LNCS, Vol. 5086, pp. 97–115, 2008.
13. C. De Cannière and B. Preneel. TRIVIUM - a stream cipher construction inspired by block cipher design principles. eSTREAM, ECRYPT Stream Cipher Project.
14. C. De Cannière, Ö. Küçük and B. Preneel. Analysis of Grain's Initialization Algorithm. In AFRICACRYPT 2008, LNCS, Vol. 5023, pp. 276–289, 2008.
15. I. Dinur and A. Shamir. Cube Attacks on Tweakable Black Box Polynomials. In EUROCRYPT 2009, LNCS, Vol. 5479, pp. 278-299, 2009.
16. The ECRYPT Stream Cipher Project. eSTREAM Portfolio of Stream Ciphers. Revised on September 8, 2008.
17. T. Eibach, E. Pilz and G. Völkel. Attacking Bivium Using SAT Solvers. In SAT 2008, LNCS, Vol. 4996, pp. 63–76, 2008.
18. J. C. Faugère. A new efficient algorithm for computing Gröbner bases (F4). Journal of Pure and Applied Algebra, 139(1):6188, 1999.
19. P. A. Fouque and T. Vannet. Improving Key Recovery to 784 and 799 rounds of Trivium using Optimized Cube Attacks. To appear in FSE 2013.
20. Available at `https://groups.google.com/forum/#!searchin/crypto-competitions/trivia/crypto-competitions/Uzgt-2t3knM/kjv5kWKJ3nAJ`
21. S. Indesteege, N. Keller, O. Dunkelman, E. Biham and B. Preneel. A Practical Attack on KeeLoq. In EUROCRYPT 2008, LNCS, Vol. 4965, pp. 1–18, 2008.
22. Y. Lee, K. Jeong, J. Sung and S. Hong. Related-Key Chosen IV Attacks on Grain-v1 and Grain-128. In ACISP 2008, LNCS, Vol. 5107, pp. 321–335, 2008.
23. D. Priemuth-Schmid and A. Biryukov. Slid Pairs in Salsa20 and Trivium. In INDOCRYPT 2008, LNCS, Vol. 5365, pp. 1–14, 2008.
24. F. Massacci. Using Walk-SAT and Rel-Sat for Cryptographic Key Search. In IJCAI 1999, pp. 290–295 1999.
25. M. S. E. Mohamed, S. Bulygin and J. Buchmann. Improved Differential Fault Analysis of Trivium. In COSADE 2011, Darmstadt, Germany, February 24–25, 2011.
26. H. Raddum. Cryptanalytic results on trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/039.
27. M. Soos. CryptoMiniSat-2.9.5. `http://www.msoos.org/cryptominisat2/`.
28. W. Stein. Sage Mathematics Software. Free Software Foundation, Inc., 2009. Available at `http://www.sagemath.org`. (Open source project initiated by W. Stein and contributed by many).
29. `www.xuru.org`.