New Distinguishers for Reduced Round Trivium and Trivia-SC using Cube Testers (Extended Abstract)

Anubhab Baksi[1], Subhamoy Maitra[1], Santanu Sarkar[2]

[1] Indian Statistical Institute, 203 B. T. Road, Kolkata 700 108, India
anubhab91@gmail.com, subho@isical.ac.in
[2] Indian Institute of Technology Madras, Sardar Patel Road, Chennai 600 036, India
sarkar.santanu.bir@gmail.com

Abstract. In this paper we experiment with cube testers on reduced round Trivium that can act as a distinguisher. Using heuristics, we obtain several distinguishers for Trivium running more than 800 rounds (maximum 829) with cube sizes not exceeding 27. In the process, we also exploit state biases that has not been explored before. Further, we apply our techniques to analyse Trivia-SC, a stream cipher proposed by modifying the parameters of Trivium and used as a building block for TriviA-ck (an AEAD scheme, which is submitted to the ongoing CAESAR competition). We obtain distinguishers till 900 rounds of Trivia-SC with a cube size of 21 only and our results refute certain claims made by the designers. These are the best results reported so far, though our work does not affect the security claims for the ciphers with full initialization rounds, namely 1152.

Keywords: Cryptanalysis, Cube Tester, Stream Cipher, Trivium, Trivia-SC.

1 Introduction

Cube attack, introduced by Dinur and Shamir [6,7], and cube tester, introduced by Aumasson et al. [8] have attracted serious attention recently. This is now being extensively used to cryptanalyse quite a few ciphers. One may also relate cube attacks with higher order differential cryptanalysis [14] proposed long back. For our explanations, let us consider the following model: Consider that the key, IV (Initialization Vector) and suitable padding bits are loaded in the initial state of a stream cipher. We denote this initial state by $S^{(0)}$ and obtain the state $S^{(r)}$ after the r-th step of evolution. In our model, we consider that the key bits are unknown; some of the IV bits (say, v many) are tried with all the $0, 1$ possibilities; rest of the IV bits are assigned to some specific value (we mostly experiment with the all-zero set-up); and the initial padding bits are fixed. Thus, for a single key, we try out 2^v IVs, and consider the GF(2) addition of all the distinct 2^v bits corresponding to each state location (denote it by $S_i^{(r)}$ for the bit at the state location i) or the output key-stream bit $f(S^{(r)})$ after t rounds of evolution during the Key Scheduling Algorithm (KSA). One may check that $S_i^{(r)}(k)$ and $f(S^{(r)}(k))$ are functions on the secret key bits $k = k_1, k_2, \ldots, k_\ell$, where the secret key is ℓ-bit long. Presuming that well designed stream ciphers have so clever update rules that, after certain number of evolutions t it becomes very hard (if not impossible with today's computational ability), to carry out the complete algebraic calculation of the functions $S_i^{(r)}(k)$ and $f(S^{(r)}(k))$, even with sophisticated tools that support symbolic computations (e.g., Sage [20]). However, over a random choice of keys (much less than 2^ℓ), if one can demonstrate that the behaviour of $f(S^{(r)}(k))$ can be distinguished from an ideal random number generator, then that is considered as a distinguisher (or, distinguishing attack) for that stream cipher. Since it may not be possible to obtain a distinguisher after the full rounds (say, τ) of evolution in KSA, one may try to study till t $(< \tau)$ rounds. The choice of proper v locations from the IV bits is an important task and naturally identifying a smaller set to obtain a distinguisher for larger t is the prime challenge here. We study this situation for the stream ciphers Trivium and Trivia-SC in this paper and provide new results in this direction. Similar to other existing cube attacks against Trivium, our work relies on heuristics to obtain interesting cubes and then we validate the results with experiments.

The Trivium [4] stream cipher, designed by De Cannière and Preneel, has a simple structure to understand and implement. It has been selected in the eStream [5] hardware portfolio and has also been standardized in ISO 29192 [15]. Even after more than a decade of effort by cryptanalysts, no attack could break the security conjectures made by the designers of Trivium. It has been studied extensively and there are several results with reduced initialization rounds of Trivium (see table 2). In this paper, we improve certain state of the art results by using suitable cubes of much smaller sizes and provide the distinguisher till 829 rounds which could not be reached so far.

TriviA-ck-v1 [1] is an AEAD (Authenticated Encryption with Associated Data) scheme, designed by Chakraborti & Nandi, and submitted to the ongoing "CAESAR - the Competition for Authenticated Encryption: Security, Applicability, and Robustness" project [3]. It uses the stream cipher Trivia-SC (sometimes refered to as SCTrivia or SC-Trivia, e.g., in [1, page 18], [2]) as a component. Trivia-SC is inspired from Trivium, but with a larger state size and also a larger key size. We succeed to find distinguishers for this cipher till 900 rounds that clearly refutes certain claims made by the designers.

1.1 Description of the Ciphers

Before proceeding further, let us describe the structures of Trivium and Trivia-SC. Table 1 briefly gives an overview.

Table 1: Overview of Trivia-SC and Trivium

Cipher	Key size	IV size	State size	Initialization rounds	# Bits involved in key-stream	# Operations in key-stream
Trivium	80	80	288	1152 (= 4 × 288)	6	5 $\oplus$
Trivia-SC	128	128	384	1152 (= 3 × 384)	8	6 $\oplus$, 1 $\wedge$

Throughout this document, and for both the ciphers, we use the convention of numbering both the key and IV bits as $1, 2, 3, \ldots$. Accordingly, we use $k_1, k_2, k_3, \ldots$ and $x_1, x_2, x_3, \ldots$ to represent the key and IV bits, respectively. We explain Trivium with a single array S and Trivia-SC with three arrays A, B, C, though each of these can be represented in any of the two manners.

Algorithm 1.1 Trivium: KLA

```
1: $(S_1, S_2, \ldots, S_{93}) \leftarrow (k_1, k_2, \ldots, k_{80}, 0, \ldots, 0)$
2: $(S_{94}, S_{95}, \ldots, S_{177}) \leftarrow (x_1, x_2, \ldots, x_{80}, 0, \ldots, 0)$
3: $(S_{178}, S_{179}, \ldots, S_{288}) \leftarrow (0, 0, \ldots, 0, 1, 1, 1)$
```

Algorithm 2.1 Trivia-SC: KLA

```
1: $(A_1, A_2, \ldots, A_{132}) \leftarrow (k_1, k_2, \ldots, k_{128}, 1, 1, 1, 1)$
2: $(C_1, C_2, \ldots, C_{147}) \leftarrow (x_1, x_2, \ldots, x_{128}, 1, \ldots, 1)$
3: $(B_1, B_2, \ldots, B_{105}) \leftarrow (1, 1, \ldots, 1)$
```

Algorithm 1.2 Trivium: KSA & PRGA

```
1: for $i = 1$ to $N$ do
2:     $t_1 \leftarrow S_{66} \oplus S_{93}; t_2 \leftarrow S_{162} \oplus S_{177}; t_3 \leftarrow S_{243} \oplus S_{288}$
3:     $z_i \leftarrow t_1 \oplus t_2 \oplus t_3$
4:     $t_1 \leftarrow t_1 \oplus S_{91} \wedge S_{92} \oplus S_{171}$
5:     $t_2 \leftarrow t_2 \oplus S_{175} \wedge S_{176} \oplus S_{264}$
6:     $t_3 \leftarrow t_3 \oplus S_{286} \wedge S_{287} \oplus S_{69}$
7:     $(S_1, S_2, \ldots, S_{93}) \leftarrow (t_3, S_1, \ldots, S_{92})$
8:     $(S_{94}, S_{95}, \ldots, S_{177}) \leftarrow (t_1, S_{94}, \ldots, S_{176})$
9:     $(S_{178}, S_{179}, \ldots, S_{288}) \leftarrow (t_2, S_{178}, \ldots, S_{287})$
10: end for
```

Algorithm 2.2 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:     $z_i \leftarrow A_{66} \oplus A_{132} \oplus B_{69} \oplus B_{105} \oplus C_{66} \oplus C_{147} \oplus (A_{102} \wedge B_{66})$
9: end for
```

Let us now describe the stream cipher Trivium [4]. As stated, this cipher consists of a 288-bit state, that can be seen as three Non-linear Feedback Shift Registers (NFSRs). One can consider these three NFSRs together and thus we denote this as a single 288-bit register S, and its bits by $S_1, S_2, \ldots, S_{288}$ respectively. The KLA (Key Loading Algorithm), by which the key and IV are initially loaded to the register, is given in algorithm 1.1. After the completion of KLA, the Key Scheduling Algorithm (KSA) routine is carried out. The cipher is clocked 1152 times without producing any key-stream bit. After KSA, required number of key-stream bits are generated by the Pseudo-Random Generation Algorithm (PRGA) routine. The algorithm of KSA and PRGA are same, so we put it in 1.2. The only difference between these two is that the step 3 of algorithm 1.2 remains 'off' in case of KSA (first 1152 rounds), and remains 'on' afterwards in PRGA.

In a similar fashion, 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 $C \equiv (S_{238}, S_{239}, \ldots, S_{384})$. Likewise Trivium, the KLA routine is described in algorithm 2.1. The KSA follows it (the evolution is done for 1152 rounds without producing any key-stream bit), which is followed by PRGA, which are given in algorithm 2.2. Likewise Trivium, the algorithm for both KSA and PRGA are the same, except for step 8 is 'off' during KSA, and 'on' afterwards in PRGA.

By carefully observing the description of Trivia-SC in [1, step 10 and 11 of algorithm 3], one may find out a subtle difference: unlike Trivium [4, section 2.1], which first outputs a key-stream bit and then updates the state, Trivia-SC first runs the evolution then outputs the key-stream bit.

1.2 Existing Works & Our Contributions

Study of cube attacks on Trivium has received serious attention in literature. Table 2 summarizes the relevant state of the art results, including our own work. Cube attacks are key recovery attacks, which attempt to guess at least one bit (with probability significantly > .5) of a secret key with complexity less than the brute-force attack. On the other hand, cube testers are distinguishing attacks. As stated in section 1, they try to distinguish a given cipher from a truly random scenario. Informally, such type of attacks may or may not manipulate the secret key bits, depending on which we define the following two terms:

Table 2: Overview of cube attacks and cube testers on Trivium

Author(s): Type of the attack	Cube size : Round(s)
Dinur, Shamir [6] (Eurocrypt'09) [7] (Eprint'08): Cube attack	$12 : 672 - 685$; $23 : 735 - 747$; $29 : 767 - 774$
Aumasson, Dinur, Meier, Shamir [8] (FSE'09): Cube tester	$24 : 772$; $30 : 790$ (Distinguisher) $24 : 842$; $27 : 885$ (Non-randomness)
Stankovski [9] (Indocrypt'10): Cube tester	$44 : 806$ (Distinguisher) $45 : 1026$; $54 : 1078$ (Non-randomness)
Knellwolf, Meier, Naya-Plasencia [12] (SAC'11): Cube tester	$25 : 772, 782, 789, 798$ (Distinguisher) $25 : 868$ (2^{31} key space); $25 : 953, 961$ (2^{26} key space) (Non-randomness)
Fouque, Vannet [13] (FSE'13): Cube attack	$30 : 784$; $37 : 799$
Our work: Cube tester (Distinguisher)	$13 : 710$; $20 : 766, 792$; $21 : 777, 801$; $22 : 804, 810$; $27 : 822, 823, 826, 829$

- **Distinguisher** A detector of some state/key-stream bits, if those are deviated from truely random scenario, where the attacker controls the (all or some) public variables only.
- **Non-randomness** A detector of the same, when the attacker has control over some private variables too.

It may be noted from table 2 that one may obtain non-randomness for more rounds than obtaining the distinguisher. This is not surprising, since, the key bits may also be manipulated that may give rise to certain class of weak keys in case of non-randomness. In this paper, we look at the distinguishing attack, where we have control over the IVs only. In this respect, we obtain the improved results with cube sizes in the range of twenties. One may note that results with significantly larger cubes have been reported in [9]. However, those results are hard to emulate without high end facilities. Thus, further efforts have been made later in [12,13] to obtain relatively smaller cubes that can be checked quickly on commonly available laptop or desktop computers. In that direction, our efforts provide currently the best known results as evident from table 2. Our heuristics are built over the idea of [9]. We first make some experiments with a few runs to obtain a potentially good cube and then run that for larger iterations to see how far the cube can be used to identify a distinguisher. That is, with our heuristics, we are able to quickly discard the cubes that may not provide good distinguishing results in further rounds. We also optimize our software implementation to certain level to obtain faster execution.

Trivia-SC, being a recently proposed cipher, have not attracted much cryptanalysis yet. Very recently, Xu, Zhang and Feng [11] points out a distinguisher against Trivia-SC with complexity 2^{152}, which is larger than the brute force attack of complexity 2^{128}. Now, let us explain the implication of our work to Trivia-SC. The designers of this cipher claimed in [1, chapter 3.1, page 15]:

> "It has been observed from the results that the output bit polynomial for Trivia-SC with 896 or more rounds of initialization behaves as random polynomial ...."

However, with our heuristics, we could discover cubes of size 13 and 21 respectively to obtain distinguishers till 897 and 900 rounds. This clearly refutes the claim of the designers.

Informally speaking, the security margin of Trivium, by itself is very tight. This is one of the excellent stream ciphers that is tempting to the cryptanalysts due to its very small and simple structure. In fact, in [10], it has been commented that it may not be judicious to increase the key size of Trivium with some increase in state as weaknesses can be found on Trivium with complexity little beyond the exhaustive key search. We show in this paper that Trivia-SC, which increases the key size as well as state size of Trivium, can be distinguished for significantly more rounds with small cubes. Thus, one may conclude that Trivia-SC is weaker compared to Trivium with respect to cube testers.

2 Finding the Cubes: Results on Trivium

The prime question about the efficiency of cube attack and cube tester kind of cryptanalysis is as asked in [16, slide 69] that how to discover the best cubes. We study this issue in detail, propose some heuristics, and present some *good* cubes for Trivium and Trivia-SC. As evident from the literature, there is not much concrete theory available to obtain a good cube, and most of the efforts are based on clever heuristics. The backbone of a cube tester is to obtain a (as minimal as possible) set of public variables, and experimentally check for the presence of bias for this set, if any, at some rounds rounds over certain iterations.

At this point, let us refer to the notations that we have described in the introduction. Since we are considering reduced round model of the ciphers, we consider as if the key-stream bits are available from just after the completion of KLA. In this regard, we have already noted that $S_i^{(r)}(k)$ are the state bits and $f(S^{(r)}(k))$ are functions on the key bits only after the t-th round. For Trivium, by $S_i^{(0)}(k)$ we mean the state bits just after the KLA and $f(S^{(0)}(k))$ is the output key-stream bit at that point. Consider that U be the set of all IVs and we consider a subset V of U, with $v = |V|$. The IVs in V will be taken as the cube.

2.1 Maximum all zero [9]

Let us first describe the basic idea of GreedyBitSet heuristic proposed in [9]. We explain that in algorithm 3. Here the idea is to obtain the set of IV bits in the cube V so that $\Pr[f(S^{(r)}(k)) = 1] = 0$. As we will keep on updating the cubles, let us refer this as indexed by V (as and when required), i.e., we require $P[f_V(S^{(r)}(k)) = 1] = 0$ for all the rounds till r. Since it is being checked whether $P[f_V(S^{(r)}(k)) = 1] = 0$, i.e., whether the function $f_V(S^{(r)}(k))$ is identically zero, only a very few run with random keys will suffice with high confidence for this test. In [9], it is also described how in each step more than one bits can be added; this was proposed so that the local optima can be better avoided. From now on, we refer to GreedyBitSet as ‘maximum all zero’ approach.

Algorithm 3 GreedyBitSet

```
Input: Key $k$, set of IV bits $U$, desired cube size $t$
Output: Cube $V$ of size $t$
$V \leftarrow \emptyset$
for $i \leftarrow 0$ to $t-1$ do
    $V \leftarrow$ GreedyAdd1Bit$(k, U, V)$
end for
return $V$
```

```
▷ GreedyAdd1Bit$(k, U, V)$
Input: $k, U$, cube $V$ of size $n$
Output: Cube $V$ of size $n+1$
$BestBit \leftarrow$ None; $max \leftarrow -1$
for all $x \in U \setminus V$ do
    $V' \leftarrow V \cup \{x\}$
    $i$ is the maximum round up to & including which all $f_{V'}(S^{(r)}(k))$'s are zero
    if $i > max$ then
        $max \leftarrow i$; $BestBit \leftarrow x$
    end if
end for
return $V \cup \{BestBit\}$
```

2.2 Our Strategies: Maximum last zero & Maximum frequency of zero

Instead of the 'maximum all zero' approach [9], we consider the 'maximum last zero' & 'maximum frequrncy of zero' approaches. That is, instead of considering the IV bit(s) for which we get all zero up to & including a particular round, we now consider the IV bit(s) for which we get the maximum last zero and maximum frequency of zero, respectively. A bit formally, line 4 in the above algorithm should be replaced by

1. "i is the last round for which the corresponding $f_{V'}(S^{(r)}(k)) = 0$", and
2. "i is the round for which the corresponding $f_{V'}(S^{(r)}(k)) = 0$ gives maximum zero count (counted up to some fixed round, say, R)",

respectively for maximum last zero and maximum frequrncy of zero. We explain these 3 heuristics with an example below.

Consider the function for the first 10 rounds. By 0 we mean that the function is identically zero, and we denote it by ? otherwise. Let $V = \emptyset$ and for the cube $\{x_1\}$, sequence of initial $f_{\{1\}}(S^{(r)}(k))$ is $\overbrace{0 \cdots 0}^{6}?0??$; whereas for $\{x_2\}$ (respectively, $\{x_3\}$), say the sequence of initial $f_{\{2\}}(S^{(r)}(k))$ is $\overbrace{0 \cdots 0}^{5}????0$ (respectively, $f_{\{3\}}(S^{(r)}(k))$ is $\overbrace{0 \cdots 0}^{5}?00?0$). While the strategy of [9] will choose the first one, we will choose the second one ('maximum last zero') and the third one ('maximum frequency of zero'). Similar to [9], in our strategy too, one may consider more than one IV bits can be included at a time for getting rid of local optima. For R (up to & including which round we will be counting the frequency), in 'maximum frequency of zero', we take 1152, the full KSA round for the ciphers.

We also consider a few other issues for obtaining *good* cubes with the following ideas:

1. Once we get a cube $V = \{x_1, x_2, \ldots, x_v\}$, we also consider the cubes of size $v + 1$ as $\{x_1, x_2, x_3, \ldots, x_v, x_{v+1}\}$, where x_{v+1} is chosen randomly from $\in U \setminus V$ and check its usefulness as a *good* cube.
2. Following [9,12], we sometimes consider the IV bits in V with a gap of three. This performs well for Trivium, though for Trivia-SC this is not that effective.
3. Once we get a cube $V = \{x_1, x_2, \ldots, x_v\}$, we also consider the cubes of size $v - 1$ as $\{x_2, x_3, \ldots, x_v\}$, $\{x_1, x_3, \ldots, x_v\}$, …, $\{x_1, x_2, \ldots, x_{v-1}\}$ and check whether they are also *good* cubes.
4. Once we get a cube $V = \{x_1, x_2, \ldots, x_v\}$, we also consider the cubes $\{x_1 - 1, x_2, \ldots, x_v\}$, $\{x_1, x_2 - 1, \ldots, x_v\}$, …, $\{x_1, x_2, \ldots, x_v - 1\}$ and $\{x_1 - 1, x_2 - 1, \ldots, x_v - 1\}$ (assuming $x_i > 1$, wherever necessary).
5. We also study GreedyBitSet heuristic along with our own heuristics. In all our experiments, we note that our heuristics outperform GreedyBitSet. Also, it does not prove that GreedyBitSet is indeed weaker to our heuristics, since some randomness is always associated and initially we test with very small runs.
6. While nothing is said about tie cases in [9], we notice that one particular tie case candidate performs much better than others. Hence, we propose another minor modification: At line 5 of algorithm 3 '>' should be replaced by '≥', and the next line will look for the best IV bit(s) which outperform(s) others (if any). Althogh, it is not possible for us to check all tie case candidates with our computational power, we try to cover as many as possible. We notice that the 'maximum last zero' heuristic gives maximum number of tie case in comparison to other two heuristics.
7. While finding the cubes, we mostly study $3+3+3+3+2+2+2+2+1+1+\ldots$ strategy, i.e., starting from an empty cube, we first set 3 IV bits at a time; then with that 3-cube, we insert another 3 IV bits; etc.
8. While we mostly consider IV bits of $U \setminus V$ to be assigned to zero, we also make experiments with random choice of those IV's in certain cases

As for the convention, we denote by $\star$, $*$, †, ‡ the cubes we obtain by using maximum last zero approach, maximum frequency of zero approach, the idea of random bit insertion, and the idea of cubes of a gap of 3, respectively.

Based on these ideas we try out many cubes with small number of iterations. Once we obtain a cube that seems promising, we go for longer runs for exact estimation of probabilities to obtain distinguishers after significant number of rounds. It is needless to mention that our method do not use any automated tool and we need to inspect quite a large amount of data by 'trial and error'.

2.3 Experimental Results

Before proceeding to the results in detail, we state our ideas of how we implement in computers and how we detect the biases.

Since we need to run a lot of independent runs to detect biases in our computers, we need to have a quite fast implementation. We use C programming language with GCC [19] compiler under Linux environments. We make our codes run faster by exploiting two levels of parallelism. In the first place, we use each variable of type `unsigned long long`, which has 64 bits of precision, thus we can run 64 independent stream ciphers parallelly. Roughly speaking, this gives us nearly 64-fold speed up. Secondly, we update 64 KSA rounds at a time, then generate 64 key-stream bits (instead of updating 1 KSA round, then generating 1 key-stream bit); then again update 64 KSA rounds; etc. This strategy works for both the ciphers because of the update patterns of its registers. Empirical results show that the speed-up obtained is almost two-fold in our software implementation. With such an efficient software implementation, we could experiment and observe the biases from a few seconds to at most in a day till 22 size cubes in laptops. However, for the experiments with the 27 size cube and for the biases for more than 820 rounds in Trivium, we required much higher computational power and used a machine with 120 processors of 2.80 GHz clock each for almost 2 weeks in a multiuser environment.

Now we describe the statistical criteria needed to detect a bias (i.e., to be sure that it is not a false alarm). Consider that we have two key-stream bits, one is generated from an ideally random source (say I) and the other one is the key-stream bits of a stream cipher (say J). Consider that the probability of ocurrance of some event in I is p and the same from J is $p(1+q)$, where q is significantly smaller than p. Then one can distinguish I, J in $\mathcal{O}\left(\frac{1}{pq^2}\right)$ trials, i.e., one needs to have those many key-stream bits available in J. This underlines one component of the complexity. Thus, if $p = \frac{1}{2}$ and $q = \pm\frac{1}{2^u}$, then the complexity is $\mathcal{O}\left(2^{2u+1}\right)$. The other component here is 2^v, where v is the cube size. That is the total complexity to obtain a cube distinguisher is 2^{v+2u+1}.

Now, the main issue is what constant is to be taken for $\mathcal{O}\left(\frac{1}{pq^2}\right)$ trials when actually running the experiments. For $\frac{1}{pq^2}$ trials the success probability of distinguishing J from I is approximately 69%, whereas it increases to more than 99.9% with $\frac{39}{pq^2}$ trials. Thus, for all the experiments when we try to be confident about a bias, we run at least $\frac{39}{pq^2}$ trials to obtain proper estimate of the deviation q. However, for complexity figures, we consider $\frac{1}{pq^2}$ trials as it provides success probability significantly more than 50%.

With this background, let us present some interesting experimental results in table 3 (Trivium) and table 6 (Trivia-SC). For a short-hand notation, we write $\mathcal{P}^{(r)}$ in place of $\Pr[f(S^{(r)}(k)) = 1]$. In the first colums of the tables 3 & 6, we present two rounds, r and r_0; where r gives the maximum round up to which a bias (given in column 2) can be detected with complexity given in last column; and r_0 gives the maximum round up to which an exact bias of zero (i.e., $\mathcal{P}^{(r_0)} = 0$) can be detected in 2^{v+1} complexity. One may note that the starting index for the IV bits is 1 and the key-stream couting for Trivia-SC starts from 0. We mark by those IV indices by bold fonts, which are inserted to a previous cube to yield a new cube.

Table 3: Cubes obtained for Trivium giving bias in key-stream

r (r_0)	$\mathcal{P}^{(r)}$	Cube indices	(#)	Complexity
710 (686)	0.462977	1, 3, 10, 12, 14, **23**, 38, 45, 48, 50, 69, 75, 79	(13)†	$2^{21.511}$
766 (747)	0.496747	2, 4, 6, 8, 11, 13, 15, 17, 19, 26, 28, 30, 32, 37, 55, 58, 67, 73, 76, 79	(20)∗	$2^{35.528}$
777 (747)	0.491104	2, 4, 6, 8, 11, 13, 15, 17, 19, 26, 28, 30, 32, **34**, 37, 55, 58, 67, 73, 76, 79	(21)∗	$2^{33.625}$
792 (736)	0.495842	2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 47, 50, 68, 74, 77, 80	(20)‡	$2^{34.820}$
801 (741)	0.497301	2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, **44**, 47, 50, 68, 74, 77, 80	(21)‡	$2^{37.067}$
804 (766)	0.497907	2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47, 50, **56**, 68, 74, 77, 80	(22)‡	$2^{38.800}$
822 823 (778)	0.495747 0.496973	2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47, 50, **53**, 56, **59**, **62**, **65**, 68, **71**, 74, 77, 80	(27)‡	$2^{41.755}$ $2^{42.736}$

One very special distinguisher in table 3 is in the first row, this cube is originally obtained from a 12 cube given in [6, table 1, row 15 - excluding the header], which gives a key-recovery attack on Trivium at round 685. We extended the given cube by inserting 23, which gives a clear distinguisher till round 710.

In table 3 (last 4 rows), the IV bits are differing by 3 or multiples of 3. The 27-cube (last row) gives a bias for maximal round, this is the full cube of gap 3. So far, we obtain clear bias up to 823 round, and we are trying to see (with the same cube) if the bias is present in some later rounds also. Note that in Section 2.4, we show how one can obtain a distinguisher for Trivium at 810-th round using the 22-cube (second row from bottom) and at 826-th & 829-th round using the 27-cube mentioned here. One may also note that the detection complexity (with the same cube) rises when we try to detect biases at later rounds; e.g., the detection complexity (with the 27-cube) for round 822 is $2^{41.755}$, which is nearly doubled ($2^{42.736}$) in case of round 823.

2.4 Exploiting Biases Observed at the State Bits of Trivium

One may note that the key-stream bit of Trivium is the XOR of six state bits. Thus, if one considers:

i. each of those six state bits are biased (this can be experimentally checked in this scenario), and
ii. each of the state bits are independent (this is actually not true as each of the state bits is a function of the same secret key bits; but in a stream cipher design it is expected that after quite a few rounds of initialization, the individual state bits should not have much correlation and one may assume the independence),

then one may use the Piling-up lemma [17, chapter 3.3.1]. Consider that $X_1, X_2, \ldots, X_q$ are independent binary random variables. Let, $\Pr(X_i = 1) = \frac{1}{2}(1 - \epsilon_i)$, where $\epsilon_i \geq 0\ \forall i = 1, 2, \ldots, q$. Here, according to the notations of Trivium, $q = 6$ and

$$X_1 \leftarrow S_{66}, X_2 \leftarrow S_{93}, X_3 \leftarrow S_{162}, X_4 \leftarrow S_{177}, X_5 \leftarrow S_{243}, X_6 \leftarrow S_{288}.$$

Let $X = X_1 \oplus X_2 \oplus \cdots \oplus X_6$. This provides the key-stream bits in reduced round Trivium. Then $\Pr(X = 1) = \frac{1}{2}(1 - \prod_{i=1}^{6} \epsilon_i)$. One may easily note that $\epsilon = \prod_{i=1}^{6} \epsilon_i$ is much smaller than each of the ϵ_i's and thus it requires much less complexity to experimentally detect higher individual bias ϵ_i rather than the composite bias ϵ.

First, we explain the scenario for the 22 size cube (second row from the bottom in table 3). In this case, with experiments, we do obtain biases at rounds 801 and 804 with high confidence, but it is not possible for the round 810. However, the biases corresponding to the six state bits could be identified properly and from that we can identify the bias of the key-stream bit at 810 rounds. Table 4 explains the scenario clearly.

Table 4: Obtaining the bias of the key-stream bit from the biases of the state bits (22 cube)

Round	ϵ_1	ϵ_2	ϵ_3	ϵ_4	ϵ_5	ϵ_6	Observed (ϵ)	Estimated ($\hat{\epsilon}$)
801	0.220354	1.000000	0.81480	1.000000	0.0319700	1.000000	0.014384	0.005740
804	0.096706	1.000000	0.912068	0.980624	0.012794	0.970144	0.004522	0.001073
810	0.019522	1.000000	0.447160	0.931532	0.003122	0.992302	0.000308	0.000025

The bias for the key-stream bit at 810 round is so small that we need to run the experiment for far more times than we have done for the state bits. However, for the biases in the state bits, we have run enough experiments and thus from those biases, using Piling-up lemma, one may estimate the bias of the key-stream bit. One may note that we accumulate the experimental data for the relevant state bits ($\epsilon_i, i = 1, 2, \ldots, 6$) and the output key-stream bit (ϵ) as well. At the same time, we use the experimental data of the state bits and the Piling-up Lemma to obtain the estimated bias ($\hat{\epsilon}$) for the output key-stream bit. With the given data, the observed & estimated probabilities for this key-stream are found 0.499846 and 0.499987, respectively. Note that, to be confident with the observed probability, we need to run the experiment for around $2^{24.330}$ random keys for each cube (i.e., total detection complexity is around $2^{24.330} \times 2^{22} = 2^{46.330}$), whereas with the estimated probability, we need to run the experiment for around $2^{31.462}$ random keys for each cube (i.e., total detection complexity is around $2^{31.462} \times 2^{22} = 2^{53.462}$). Both of these cases are quite challenging to execute with our computational facilities. However, the state biases ($\epsilon_i, i = 1, 2, \ldots, 6$) are such that it is enough for us to experiment with $2^{17.647}$ (in case of ϵ_5, where the probability is 0.498439) random keys for each cube (i.e., total detection complexity is around $2^{17.647} \times 2^{22} = 2^{39.647}$) to discover the biases with very high confidence.

We also like to point out that with such cubes, there are certain state bits where we can obtain bias till further rounds. As for an example, at the 854-th round, the input corresponding to $X_6 \leftarrow S_{288}$ remains biased to a significant extent. This shows that there exists at least one input to the key-stream generation process for which significant biases can be observed. However, the output may not demonstrate significant bias as for the other five inputs the biases could not be observed with the amount of experiment we have done.

We also record similar results with our 27-cube (last row of table 3). So far, we obtain that this scenario at round 829, and further attempts are ongoing to extend this round with the same cube. Table 5 shows necessary calculations with the 27 cube, the cases of round 820, 822, 826 & 829 are also given here. At this point, we would like to point out that the bias at rounds 826 and 829 are not observed directly from the key-stream. The observed and estimated probabilities at round 826 are respectively 0.499661 & 0.499859; and that of round 829 are 0.499900 & 0.499969, respectively. The complexity of detection for these 2 cases are given by $2^{25.575}$ & $2^{28.955}$ respectively for each cube (that means, the total detection complexties are respectively $2^{52.575}$ & $2^{55.955}$). Likewise the previous cases (in table 4), one may observe that it is too difficult to detect the key-stream bias directly with our present computational power. However, we can detect even the most difficult state bias (in case of ϵ_5, where the actual probabilities are 0.495017 and 0.497464 for round 826 and 829 respectively) with complexity for each cube $2^{14.298}$ and $2^{16.246}$ for round 826 and for round 829 respectively (i.e., total detection complexities for this state are given by $2^{14.298} \times 2^{27} = 2^{41.298}$ and $2^{16.246} \times 2^{27} = 2^{43.246}$ respectively, which are within our scope). Here also, we can find one biased state bit ($X_6 \leftarrow S_{288}$), which participates in key-stream at round 881.

Table 5: Obtaining the bias of the key-stream bit from the biases of the state bits (27 cube)

Round	ϵ_1	ϵ_2	ϵ_3	ϵ_4	ϵ_5	ϵ_6	Observed (ϵ)	Estimated ($\hat{\epsilon}$)
820	0.128770	1.000000	0.962410	1.000000	0.021830	1.000000	0.010078	0.002705
822	0.137094	0.999174	0.944876	1.000000	0.018380	1.000000	0.009132	0.002379
826	0.036620	0.982358	0.791138	1.000000	0.009966	0.995456	0.000678	0.000282
829	0.019288	0.890656	0.741232	1.000000	0.005072	0.946106	0.000200	0.000061

One may also note (from tables 4, 5) that, ϵ is much higher than $\hat{\epsilon}$. So, the detection complexity for the biases will be indeed less than which is governed by last columns of tables 4, 5. The piling-up lemma is not obeyed exactly, and this is not unusual; since the state bits are not actually independent. However, our analysis shows that one can assume independence with a very low margin of error.

3 Results on Trivia-SC

For Trivia-SC, the cubes and related results are presented in table 6. One challenging task here is to find a distinguisher for more than 896 rounds in the initialization as the designers commented that they could not discover any non-randomness beyond that. We show here that non-randomness can indeed be discovered till round 900 with cube size as small as 21.

Table 6: Cubes for Trivia-SC giving bias in key-stream

r (r_0)	$\mathcal{P}^{(r)}$	Cube indices	(#)	Complexity
802 (801)	0.433241	1, 2, 3, 5, 9, 24, 54, 69, 84	(9)†	$2^{15.810}$
897 (882)	0.246883	1, 2, 3, 7, 16, 22, 32, 37, 46, 52, 67, 82, 97	(13)†	$2^{15.964}$
890 (889)	0.475783	9, 10, 11, 15, 24, 25, 26, 30, 45, 60, 75, 90, 105, 106, 121, 126	(16)⋆	$2^{25.736}$
900 (884)	0.418750	4, 10, 13, 16, 19, 25, 34, 40, 46, 49, 55, 70, 79, 85, 91, 94, 100, 115, 121, 125, 127	(21)∗	$2^{27.243}$
900 (885)	0.415210	4, 10, 13, 16, 19, 25, 34, 40, **44**, 46, 49, 55, 70, 79, 85, 91, 94, 100, 115, 121, 125, 127	(22)∗	$2^{28.120}$

We obtain a cube of size 13 (second row of table 6) giving bias at round 897 for Trivia-SC. In fact, the same cube can be extended with additional IV's to obtain improved probabilities (towards closer to zero) at the same round as shown in table 7.

In table 8, we show for Trivia-SC that starting from a fixed cube of size 12: {4, 10, 19, 25, 40, 55, 70, 85, 100, 121, 125, 127} (which we obtain using maximum frequency of zero approach, under $3+3+3+3$ strategy), we insert 4 new IV bits to get a 16 cube by the same approach (maximum frequency of zero), but by two different strategies. In one strategy, we insert one bit at a time $(12+1+1+1+1)$, and in the other we insert 2 bits at a time $(12+2+2)$. We show that the latter way is better than the former in

Table 7: Cubes obtained for Trivia-SC by inserting new IVs

Cube size	14	16	17	18	19	20
Additional IV's	18	8, 68	112	23	33	127
$\mathcal{P}^{(897)}$ (r_0)	0.149 (882) †	0.113 (882) †	0.056 (882) †	0.0326 (883) †	0.012 (883) †	0.006 (883) †

table 8. Particularly, the maximum round (880) which shows bias at key-stream has not been improved from the 12-cube in the former strategy. One may note that the cube at last row of table 8 is a subset of last two rows of table 6.

Table 8: Effects of inserting variable number of IV bits at a time in Trivia-SC

Strategy	r (r_0)	$\mathcal{P}^{(r)}$	Cube indices (#)	Complexity
$3+3+3+3$	880 (849)	0.428235	4, 10, 19, 25, 40, 55, 70, 85, 100, 121, 125, 127 (12)∗	$2^{18.601}$
$+1+1+1+1$	880 (865)	0.182461	4, 10, **16**, 19, 25, **34**, 40, 55, 70, **81**, 85, **91**, 100, 121, 125, 127 (16)∗	$2^{18.310}$
$+2+2$	895 (879)	0.493476	4, 10, **13**, 19, 25, 40, **46**, 55, 70, 85, **91**, 100, **115**, 121, 125, 127 (16)∗	$2^{29.520}$

Further, we show that exploiting very small cubes of size 9 only, one can obtain distinguishers for more than 800 rounds in Trivia-SC. This indicates that under this kind of analysis, Trivia-SC is indeed weaker than Trivium.

One may note, there are some interesting observations regarding the well-applicability of the heuristics. As an instance, consider the case of Trivia-SC (table 6). We get a 16-cube which gives a 0 probability at 889-th round, but the last detectable bias is located at 890-th round (row 3). But, with a 21 cube, the last zero probability is at 884th round, while the bias is clearly detectable at 900th round (row 4).

We would like to point out that the state bias related analysis, as in Section 2.4, cannot directly be applied on Trivia-SC because of the AND operation. However, one may indeed use the state biases towards estimating the non-randomness in keystream. This we will explore in the full version of this paper.

3.1 One Suggested Key-stream Expression for Trivia-SC

Here we show that, even with retaining everything as it is (in algorithms 2.1, 2.2), the security of Trivia-SC can be greatly improved just by changing the key-steam expression (at step 8 of algorithm 2.2). The idea here is to use one such expression for generating key-stream which has better cryptographic properties than that one chosen by the designers. We use a rotation symmetric Boolean function on 5 variables as given in [18, section 3.2.1]. The algebraic normal form of the function is: $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_1 \wedge x_2 \oplus x_2 \wedge x_3 \oplus x_3 \wedge x_4 \oplus x_4 \wedge x_5 \oplus x_5 \wedge x_1 \oplus x_1 \wedge x_2 \wedge x_3 \oplus x_2 \wedge x_3 \wedge x_4 \oplus x_3 \wedge x_4 \wedge x_5 \oplus x_4 \wedge x_5 \wedge x_1 \oplus x_5 \wedge x_1 \wedge x_2$; we substitute x_1, x_2, …, x_5 by B_{66}, A_{102}, C_{147}, B_{105}, A_{132} respectively; and XOR the resulting expression with $A_{66} \oplus B_{69} \oplus C_{66}$ (to make use of all the 8 state bits, as in the original design). Thus, we get our suggested key-stream expression as:

$(A_{66} \oplus B_{69} \oplus C_{66}) \oplus [B_{66} \oplus A_{102} \oplus C_{147} \oplus B_{105} \oplus A_{132} \oplus B_{66} \wedge A_{102} \oplus A_{102} \wedge C_{147} \oplus C_{147} \wedge B_{105} \oplus B_{105} \wedge A_{132} \oplus A_{132} \wedge B_{66} \oplus B_{66} \wedge A_{102} \wedge C_{147} \oplus A_{102} \wedge C_{147} \wedge B_{105} \oplus C_{147} \wedge B_{105} \wedge A_{132} \oplus B_{105} \wedge A_{132} \wedge B_{66} \oplus A_{132} \wedge B_{66} \wedge A_{102}]$.

The part inside $[\cdot]$ has the best possible cryptographic properties (in terms of nonlinearity, resiliency, algebraic degree and autocorrelation values) among all 5 variable Boolean functions. We run our heuristics on this proposed design too. The best cubes we obtain this way are given in table 9. One may note that for the same size of the cubes, the distinguishers are for much smaller rounds; e.g., we can not go beyond round 819 here, whereas in the original Trivia-SC, we could go up to 900-th round. Hence, one can conclude that the resulting stream cipher seems more secure than than Trivia-SC considering cube attack.

Future AEAD designers, who will be relying on a stream cipher, may use this idea to make their design even better. One may argue that incorporating a different Boolean function may increase the gate count in hardware implementation. This argument has merit for a stream cipher design in constrained environment. However, for an AEAD scheme, there are additional components involved and thus, the increase in gate count with a 5-variable function will be minimal, if at all. Particularly, Trivium, being a hardware profile finalist in eStream, typically uses much lesser gates; but this is not the case for the the TriviA-ck AEAD scheme, as a whole.

Table 9: Cubes for our suggested variant of Trivia-SC giving bias in key-stream

r (r_0)	$\mathcal{P}^{(r)}$	Cube indices	(#)	Complexity
746 (745)	0.474401	5, 14, 44, 59, 80, 89, 119, 121, 125	(9)$\star$	$2^{18.576}$
749 (703)	0.489759	2, 5, 11, 20, 26, 41, 80, 86, 122	(9)$*$	$2^{21.219}$
781 (780)	0.142615	5, **8**, 14, **29**, **35**, 44, 59, **68**, **74**, 80, 89, **103**, **107**, 119, 121, 125	(16)$\star$	$2^{17.969}$
813 (768)	0.495023	2, 5, **7**, 11, 20, **22**, 26, **35**, 41, **56**, **71**, 80, 86, **92**, **101**, 122	(16)$*$	$2^{30.301}$
819 (804)	0.075385	2, 5, 7, 11, **14**, 20, 22, 26, 35, 41, **50**, 56, **69**, 71, 80, **83**, 86, 92, **95**, 101, 122	(21)$*$	$2^{22.472}$

4 Conclusion

In this paper, we discuss some new heuristics, and also their usefulness to find bias after more than 800 (namely, 823) rounds of Trivium with small sized cubes. To the best of our knowledge, biases till such higher rounds could not be reached earlier with such small cubes, where size of the cubes do not exceed 27. In another direction, we also provide a novel idea for exploiting the state biases to estimate bias in keystream till 829 rounds of Trivium. In fact, such state biases could be observed till 881 rounds. It is expected that further extensions can be made over our heuristics to cryptanalyse more rounds and to use it in key-recovery attacks. We also study Trivia-SC that is a part of the AEAD scheme TriviA-ck. We provide empirical evidences to show that Trivia-SC is weaker than Trivium and thus propose certain modifications in keystream generation.

References

1. A. Chakraborti, M. Nandi. TriviA-ck-v1. Available at `http://competitions.cr.yp.to/round1/triviackv1.pdf`.
2. A. Chakraborti, M. Nandi. Important Features and Flexibilities of TriviA. Presentation at DIAC 2014, available at `http://2014.diac.cr.yp.to/slides/nandi-trivia.pdf`.
3. CAESAR: Competition for authenticated encryption: Security, applicability, and robustness. Available at `http://competitions.cr.yp.to/caesar.html`.
4. C. De Cannière, B. Preneel. Trivium. Available at `http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf`.
5. eSTREAM: the ECRYPT Stream Cipher Project. Available at `http://www.ecrypt.eu.org/stream/`.
6. I. Dinur, A. Shamir. Cube Attacks on Tweakable Black Box Polynomials. In Eurocrypt 2009, LNCS, Vol. 5479, pp. 278-299, 2009.
7. I. Dinur, A. Shamir. Cube Attacks on Tweakable Black Box Polynomials. In Eprint, 2008. Available at `http://eprint.iacr.org/2008/385.pdf`.
8. J-P. Aumasson, I. Dinur, W. Meier, A. Shamir. Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In FSE 2009, LNCS, Vol. 5665, pp. 1–22, 2009.
9. P. Stankovski. Greedy Distinguishers and Nonrandomness Detectors. In Indocrypt 2010, LNCS, Vol. 6498, pp. 210–226, 2010.
10. A. Maximov, A. Biryukov. Two trivial attacks on Trivium. In SAC 2007, LNCS, Vol. 4876, pp. 36–55, 2007.
11. C. Xu, B. Zhang, D. Feng. Linear Cryptanalysis of FASER128/256 and TriviA-ck. In Indocrypt 2014, LNCS, Vol. 8885, pp. 237–254, 2014.
12. S. Knellwolf, W. Meier, M. Naya-Plasencia. Conditional Differential Cryptanalysis of Trivium and KATAN. In SAC 2011, LNCS, Vol. 7118, pp. 200–212, 2011.
13. P.-A. Fouque, T. Vannet. Improving Key Recovery to 784 and 799 Rounds of Trivium Using Optimized Cube Attacks. In FSE 2013, LNCS, Vol. 8424, pp. 502–517, 2013.
14. L. R. Knudsen. Truncated and higher order differentials. In FSE 1994, LNCS, Vol. 1008, pp. 196–2111, 1994.
15. ISO/IEC 29192-2:2012. Available at `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=56552`.
16. W. Meier. Cube Testers and Key Recovery in Symmetric Cryptography. Presentation at Indocrypt 2009, available at `http://indocrypt09.inria.fr/slides_cube_ind09.pdf`
17. D. R. Stinson. Cryptography Theory and Practice. Chapman & Hall/CRC. Third Edition, 2006.
18. P. Stănică, S. Maitra. Rotation symmetric Boolean functions—Count and cryptographic properties. In Discrete Applied Mathematics 156(10): 1567-1580, 2008.
19. GCC, the GNU Compiler Collection. Available at `https://gcc.gnu.org/`.
20. Sage: Open Source Mathematics Software. Available at `http://www.sagemath.org/`.