Some Randomness Experiments on TRIVIUM

Subhabrata Samajder and Palash Sarkar
Applied Statistics Unit
Indian Statistical Institute
203, B.T.Road, Kolkata, India - 700108.
{subhabrata_r,palash}@isical.ac.in

March 22, 2014

Abstract

This paper develops two methods for exploring the structure of the stream cipher TRIVIUM. We consider whether it is possible to compute the algebraic normal form (ANF) of such functions. Since the key and the IV together make up 160 variables, doing this directly is not possible. Instead, one can choose a subset of the key and IV variables of size n and fix the other variables to constants.

As an application of this tool, we run some randomness experiments on the first output bit of TRIVIUM. Three types of tests were conducted on full (and reduced) round TRIVIUM. For the tests done, we fix a subset of n key variables and vary the remaining $160 - n$ key and IV bit positions. The first test tried to find polynomials which are non-random in some sense. This is along the line of work done by Aumasson et. al. on their work on cube testers. However, here we do not use any cube. We try to find polynomials corresponding to the first output bit of TRIVIUM which are non-random. Our experiments did reveal a number of polynomials which showed deviation from randomness.

The second test conducted checks the balancedness amongst the first l output bits of TRIVIUM. A proper statistical model for conducting such a test is proposed. Tests results shows that the first 8 output bits are unbalanced.

For the third test we consider N random choices of the constant values keeping the n key variables fixed. A simple test of hypothesis is applied to detect possible non-randomness in the distributions. Mostly, the results are negative. In a few cases, the results seem to indicate the presence of possible non-randomness, though, nothing conclusive can be inferred from this test.

The symbolic computation tool developed here can conceivably be used for exploring other features of TRIVIUM. Further, the idea behind the development of the tool can be used to build similar tools for other ciphers.

Keywords : Multivariate Polynomial Multiplication, Boolean Functions, Algebraic Normal Form (ANF), Stream Ciphers, TRIVIUM.

1 Introduction

TRIVIUM is a hardware oriented synchronous stream cipher that was submitted to the Profile II (hardware) of the eSTREAM competition by its authors Christophe De Cannière and Bart Preneel [DCP]. TRIVIUM maintains an internal state S of size 288 bits. The state S is further subdivided into 3 shorter shift registers S1, S2 and S3 of sizes 93, 84 and 111 bits each. It uses a simple quadratic state update function. At each round of the state update, 3 bits are computed using the state update function which are then fed into the three shift registers S1, S2 and S3, while the three most significant bits of S1, S2 and S3 are discarded. TRIVIUM uses 1152 rounds of key initialization. During key generation step, TRIVIUM uses the same state update function in addition to producing a bit, which is XOR of 6 state bits. Over the years it has received much attention from the research community due to its simple structure. However, there is still no known attack on full version of TRIVIUM which works better than exhaustive search.

To gain a better understanding of the full cipher, scaled-down variants, such as Bivium A and Bivium B [Rad06], have been suggested and studied. Apart from side channel attacks, the attacks on TRIVIUM can be broadly classified into two categories. The first one analyses these scaled-down variants like Bivium A and Bivium B, both of which uses two shift registers as their internal state instead of three and tries to extrapolate these results to the full TRIVIUM. The second approach has been to study the reduced-round variants of the cipher, i.e., TRIVIUM with 'r' rounds of key initialization where $r \leq 1152$. In this article we will be concentrating on the reduced-variant only.

In [O'N07], O'Neal claimed that TRIVIUM with 1152 rounds of key initialization may not be secure and thus proposed that the initialization rounds for TRIVIUM must be increased to $4 \times 1152 = 4608$ rounds in order to make it secure. After that the first attack on reduced-round variant of TRIVIUM was the cube attack [DS09]. In [DS09], Dinur and Shamir used the newly invented cube attack to successfully recover the key after 767 initialization rounds with 2^{45} bit operations and then showed that this can be further reduced to 2^{36} bit operations. In [ADMS09], Aumasson et. al. introduced a new class of attacks called the cube testers and developed distinguisher's that distinguished 790 rounds of TRIVIUM from random with 2^{30} complexity and were also able to detect non-randomness over 885 rounds in 2^{27} complexity, improving on the original 767-round cube attack.

Recently in [FV13], Fouque and Vannet increased the number of attacked initialization rounds by improving the time complexity of computing cube. They were able to find a key recovery attack requiring 2^{39} queries for 784 initialization rounds and were also able to provide another key recovering attack up to 799 rounds with a complexity of 2^{40} for queries and 2^{62} for the exhaustive search part. In their attack they used the Moebius Transform discussed in [Jou09], to improve on the time taken in the pre-processing stage of cube attack. The algorithm used in [FV13] first finds the truth table of the boolean function corresponding to the first output bit and then uses the algorithm given in [Jou09] to convert the truth table into its corresponding ANF. The second algorithm (Method - 2) given in this paper uses the same idea to compute the ANF of the boolean function corresponding to the first output bit of TRIVIUM. But instead of concentrating on cube attack this work concentrates on the randomness behavior of these boolean functions.

Our Results: The best attack so far in terms of the number of rounds attacked on reduced-round variants of TRIVIUM is the one given by Aumasson et. al. in [ADMS09]. In their work the input (key and IV) variables were divided into cube variables (CV) and superpoly variables (SV). Suppose $f(x_1, \ldots, x_c; y_1, \ldots, y_s)$ denote a boolean function in $c + s$ variables, where CV $= \{x_1, \ldots, x_c\}$ and SV $= \{y_1, \ldots, y_s\}$. Then superpoly s_{CV} of f corresponding to a cube of size c is defined as

$$s_{CV}(y_1, y_2, \ldots, y_s) = \oplus_{(x_1,x_2,\ldots,x_c) \in GF(2)^c} f(x_1, x_2, \ldots, x_c; y_1, y_2, \ldots, y_s),$$

which is an s-variable boolean function in the variables SV. For the cubes listed in the appendix of [DS09], the authors of the paper then checked for the following properties for every superpoly found:

1. Whether the truth table corresponding to the superpoly is balanced.
2. Whether the superpoly is a constant function.
3. Used the test by Alon et al. [AKK$^+$03] to check whether the superpoly is of degree d.
4. Checked whether a particular variable in SV occurs in the superpoly as a linear term.
5. Lastly to check whether a particular variable in SV is absent in the ANF of the superpoly.

Failing any one of these test, the authors termed the corresponding superpoly as non-random. With this the authors obtained the best result so far on reduced-round variant of TRIVIUM by detecting non-randomness over 885 rounds of TRIVIUM. The authors showed that for a cube of size 27 mentioned in [DS09], the key variables $1, 4, 5$ are absent in the corresponding superpoly, when the remaining other key bits are set to zero. It is argued that this is an evidence of non-randomness. This forms the motivation of our work.

In this paper instead using cubes we concentrated on the truth table representation and ANF of the first output bit after full (and reduced) round TRIVIUM. The paper uses two types of measurements for detecting non-randomness, which are different from those mentioned in [ADMS09]. The first one tries to find examples of polynomials which are non-random whereas the second one uses an aggregated measurement to look for any structural weakness. The first type is similar to the work of Aumasson et. al. [ADMS09], as both tries to find examples of polynomials which are non-random. However, this paper does not compute superpoly's to get such examples of non-random polynomials.

TRIVIUM uses very simple state update functions which are actually quadratic boolean functions. Since the state is updated over many initialization rounds, the boolean functions representing the output bits has a complex dependence on the key and IV variables. In our endeavor to finding non-randomness for full TRIVIUM we also addressed the problem of whether it is possible to explicitly write down the boolean functions for the state and the output bits in terms of the key and IV variables. If we consider all 160 key and IV variables, then this is practically impossible to do. Instead, we retain n of the key and IV variables and fix all the other variables to constants. For n up to 30, it is then possible to explicitly express each of the state bits and the first output bit (and also other output bits) as a boolean function of the n variables.

The paper gives two methods to symbolically compute the first output bit of TRIVIUM in n variables. The methods select n bits out of 160 and treats them as variables. The rest $160 - n$ are randomly fixed to constants. Method - 1 treats each state bit as a polynomial in n variables. The state bits are represented in their ANF. Method - 1 then computes the state after full (and reduced) round TRIVIUM by multiplying thrice two polynomials of n variables in their ANF using the Algorithm mentioned in [Sam13] at each step. Method - 1 then outputs the first output bit which is the XOR of six state bits. To get the truth table representation of the first output bit, Method - 1 uses the Algorithm mentioned in [Jou09] to convert the ANF into its corresponding truth table.

Method - 2, does the opposite of Method - 1. It first constructs the truth table of the first output bit. To do that it first computes the first output bit for all possible values of the n bits, which are to be treated as variables, keeping the remaining $160 - n$ bits fixed to a constant value. Thus we run TRIVIUM for 2^n possible key and IV inputs to get 2^n first output bits. This corresponds to the truth table of first output when viewed as a n variable polynomial. Method - 2 then converts this truth table into its corresponding ANF by using the Algorithm mentioned in [Jou09].

The simulation tools developed are then used to carry out some randomness experiments on TRIVIUM. A total of n key variables are chosen and the rest of the key variables and all the IV variables are randomly fixed to constants. The first output bit after full (or reduced) round is then an n-variable boolean function. Different random choices of the constants lead to different boolean functions. One can then consider the distribution of the monomials in these boolean functions. Similarly, one can also consider the distribution of number of zeroes and ones in its truth table representation, i.e., the balancedness of the first output bit. This notion was further extended to consider the balancedness amongst the first l output bits of TRIVIUM.

Simple tests are employed to detect possible non-randomness in this distribution. The tests done in this paper can be broadly classified into three types. The first type looks to identify a particular polynomial which in some sense is non-random. Some polynomials which failed the tests are reported. We detected Non-random polynomials even after full round of TRIVIUM. The second test was done to check for balancedness amongst the first l output bits. A proper statistical model for conducting such a test is given. Experiments were done by taking $n = 20, 30$ and $l = 8$. The results suggests that the first 8 output bits are unbalanced. Lastly, an aggregated test was done to detect non-randomness in the first output bit of TRIVIUM. For small values of r, the test routinely detected the presence of non-randomness. However once r grows, the test mostly reports the lack of non-randomness. In a few cases though the test seems to indicate the presence of possible non-randomness. Nothing conclusive, however, can be said from the aggregated test.

The idea behind the simulation tool that we have built for TRIVIUM can be easily adapted to build simulation tools for other ciphers. In fact, the state bits and the output bits of any cipher can be expressed as boolean functions of the key and IV bits. Further, any operation on the state bits can be expressed in terms of XOR and multiplication of the polynomials representing the state bits. So, using an efficient algorithm one can build a simulation tool for any cipher along the lines of what we have done for TRIVIUM.

2 A Brief Description of TRIVIUM

TRIVIUM maintains a 288-bit internal state "S" denoted by $S = (s_1, s_2, \ldots, s_{288})$ and uses two algorithms, namely a key initialization algorithm, which we call the key and IV setup, and a key stream generation algorithm. The state S is further divided into 3 shift registers, namely $S_1 = (s_1, s_2, \ldots, s_{93})$, $S_2 = (s_{94}, s_{95}, \ldots, s_{177})$ and $S_3 = (s_{178}, s_{179}, \ldots, s_{288})$.

2.1 Key and IV Setup

The algorithm is initialized by loading an 80-bit key into the first 80-bits of the state S, i.e., $s_1, s_2, \ldots, s_{80}$ and an 80-bit IV into the state bits $s_{94}, s_{95}, \ldots, s_{173}$ and setting all remaining bits to 0, except for s_{286}, s_{287}, and s_{288}, which are set to 1. Each round of the iterative process extracts the values of 15 specific state bits and uses them to update 3 bits of the state. This is repeated for $4 \times 288 = 1152$ times. This can be summarized by the following pseudo-code (Algorithm 1):

Algorithm 1: TRIVIUM - Key and IV Setup.

```
$(s_1, s_2, \ldots, s_{93}) \leftarrow (K_1, K_2, \ldots, K_{80}, 0, \ldots, 0)$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_1, IV_2, \ldots, IV_{80}, 0, 0, 0, 0)$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, \ldots 0, 1, 1, 1)$
for $i = 1$ to $4 \cdot 288$ do
    $t_1 \leftarrow s_{66} \oplus s_{91} \cdot s_{92} \oplus s_{93} \oplus s_{171}$
    $t_2 \leftarrow s_{162} \oplus s_{175} \cdot s_{176} \oplus s_{177} \oplus s_{264}$
    $t_3 \leftarrow s_{243} \oplus s_{286} \cdot s_{287} \oplus s_{288} \oplus s_{69}$
    $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
    $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
    $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
end
```

2.2 Key Stream Generation

The key stream generation algorithm is similar to that of the key initialization algorithm except that at each round, a single bit which is a linear function of six state bits, is output before the state update. This process repeats itself until the requested $N \leq 2^{64}$ bits of key stream is generated. The complete description is given by the following pseudo-code (Algorithm 2):

Algorithm 2: TRIVIUM - Key Stream Generation.

```
for $i = 1$ to $N$ do
    $t_1 \leftarrow s_{66} \oplus s_{93}$
    $t_2 \leftarrow s_{162} \oplus s_{177}$
    $t_3 \leftarrow s_{243} \oplus s_{288}$
    $z_i \leftarrow t_1 \oplus t_2 \oplus t_3$
    $t_1 \leftarrow t_1 \oplus s_{91} \cdot s_{92} \oplus s_{171}$
    $t_2 \leftarrow t_2 \oplus s_{175} \cdot s_{176} \oplus s_{264}$
    $t_3 \leftarrow t_3 \oplus s_{286} \cdot s_{287} \oplus s_{69}$
    $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
    $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
    $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
end
```

3 Multiplication of Algebraic Normal Forms

In this section, we give a brief description of the Algorithm used to multiply two boolean functions in their ANF's. The crux of the algorithm is the observation that for all variables x involved, the fact that $x^2 = x$ can be utilized in reducing the number of multiplications involved at the cost of extra additions. This idea immediately gives a recursive algorithm to multiply two boolean functions given by their ANFs. In [Sam13], the authors then converted this recursive algorithm into a tree based algorithm and then used some known programming techniques to arrive at an efficient implementation.

3.1 Basic Idea

Let, $p(x_1, \ldots, x_n), q(x_1, \ldots, x_n) \in \mathbb{R}$, where $\mathbb{R} = GF(2)[x_1, x_2, \ldots, x_n] / \langle x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle$. Write,

$$\begin{aligned} p(x_1, \ldots, x_n) &= x_n \cdot p_1(x_1, \ldots, x_{n-1}) \oplus p_0(x_1, \ldots, x_{n-1}) \\ q(x_1, \ldots, x_n) &= x_n \cdot q_1(x_1, \ldots, x_{n-1}) \oplus q_0(x_1, \ldots, x_{n-1}). \end{aligned}$$

Then,

$$\begin{aligned} pq &= (p_1 q_1) x_n^2 \oplus (p_1 q_0 \oplus p_0 q_1) x_n \oplus p_0 q_0 \\ &= (p_1 q_1 \oplus p_1 q_0 \oplus p_0 q_1) x_n \oplus p_0 q_0; \qquad [\text{Since, } x_n^2 = x_n \text{ in } \mathbb{R}.] \\ &= \{(p_1 \oplus p_0)(q_1 \oplus q_0) \oplus p_0 q_0\} x_n \oplus p_0 q_0. \end{aligned}$$

Thus, the number of $(n-1)$-variate multiplications required is 2 instead of 4 at the cost of one extra addition. This immediately gives a recursive algorithm to multiply two boolean functions in their ANF's.

Directly applied, this idea leads to a recursive algorithm. The recursive algorithm can be modified to obtain an iterative algorithm. It turns out that the iterative algorithm can be seen to work as follows. The ANF of the two polynomials are converted to the truth table formats; the truth table formats are multiplied; and then the truth table format of the product is converted to the ANF format. The conversions between ANF and truth table formats are the same as the algorithm described at [Jou09, SRD]. For the sake of completeness the algorithm details and the implementation used in this paper are given in Appendix C.

4 Symbolically Computing the ANF of the First Output Bit of TRIVIUM

In this section, we give details of how to symbolically compute TRIVIUM. Two such methods, namely Method - 1 and Method - 2, are given. Parallels can be drawn for other ciphers.

4.1 Method - 1

Let us denote the key K by $(k_1, k_2, \ldots, k_{80})$ and the IV by $(iv_1, iv_2, \ldots, iv_{80})$. We consider each state bit of TRIVIUM as an n-variable boolean function. From Algorithm 1, it can be seen that TRIVIUM uses an 80-bit key and an 80-bit IV. If instead of bits, we consider the key and the IV as variables then the state is initialized as follows:

$$\begin{aligned}(s_1, s_2, \ldots, s_{93}) &\leftarrow (k_1, k_2, \ldots, k_{80}, 0, \ldots, 0),\\ (s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (iv_1, iv_2, \ldots, iv_{80}, 0, 0, 0, 0),\\ (s_{178}, s_{179}, \ldots, s_{288}) &\leftarrow (0, \ldots 0, 1, 1, 1).\end{aligned}$$

Then during each state update (or iteration), these state bits gets multiplied and added in the boolean function ring defined over the variables K and IV. Thus, considering each state bit as a boolean function in $n = 80 + 80 = 160$ variables, one can view each state update as performing 3 multiplications (1 for each $t_i, i = 1, 2, 3.$) and 9 additions (3 for each $t_i, i = 1, 2, 3.$) in the boolean function ring defined over the variables K and IV. Addition in the boolean function ring defined over the variables K and IV, is just bitwise XOR, whereas for multiplication corresponds to multiplication of two boolean functions in their ANF's. We illustrate this with an example. After state initialization, notice that $s_{66} = k_{66}, s_{91} = 0, s_{92} = 0, s_{93} = 0$ and $s_{171} = iv_{78}$. Therefore during the first state update,

$$t_1 = k_{66} \oplus iv_{78}.$$

Also, during state update,

$$\begin{aligned}(s_1, s_2, \ldots, s_{93}) &\leftarrow (t_3, s_1, \ldots, s_{92})\\ (s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (t_1, s_{94}, \ldots, s_{176})\\ (s_{178}, s_{179}, \ldots, s_{288}) &\leftarrow (t_2, s_{178}, \ldots, s_{287}).\end{aligned}$$

Doing this naively means copying 2^n bits 288 times, which is very costly. Hence, to avoid copying we use another array, S_I say, of integer of size 288, initialized with the identity permutation, i.e., $S_I[i] = i$, for all $i = \{1, 2, \ldots, 288\}$. Also the computation of t_1, t_2 and t_3 are done on the state bits s_{93}, s_{177} and s_{288} itself, since these bits are discarded. Then after each round, instead of copying the state bits, we just do the following simple operations to simulate the state update.

$$\begin{gathered}S_I[i+1] = S_I[i], \text{ for all } i = 1, \ldots, 288 \text{ such that } i \neq 93, 177, 288;\\ S_I[1] = t_3; \qquad S_I[94] = t_1; \qquad S_I[177] = t_2.\end{gathered}$$

Notice that $n = 160$ is infeasible on current computers. Also because of memory restrictions, we cannot go beyond 30. Hence, we randomly select the key and IV bit positions which we want to treat as variables. These selected bit positions are then renamed as variables $k_1, k_2, \ldots, k_{n_k}$ and $iv_1, iv_2, \ldots, iv_{n_{iv}}$, such that $n_k + n_{iv} = n(\leq 30)$. The rest of the key and IV bit positions are then set randomly to either 0 or 1. We then symbolically compute the full (or reduced) round TRIVIUM and get the state bits as polynomials in $k_1, k_2, \ldots, k_{n_k}$ and $iv_1, iv_2, \ldots, iv_{n_{iv}}$ after r rounds of key initialization. We then consider the first output bit which is the bitwise XOR of six state bits, namely $s_{66}, s_{93}, s_{162}, s_{177}, s_{243}$ and s_{288} and hence a polynomial in $k_1, k_2, \ldots, k_{n_k}$ and $iv_1, iv_2, \ldots, iv_{n_{iv}}$.

Initially we implemented this algorithm in the software 'SAGE'. But we found that 'SAGE' uses a quadratic time algorithm to multiply two boolean functions. Hence, our implementation was quite slow. In fact, the implementation could not handle dense polynomials, i.e., polynomials with number of monomials of the order of 2^{n-1} (where n = number of variables), for $n \geq 18$. Thus, the implementation ran out of memory when the number of initialization rounds were increased to 600 or so. So, we needed a fast implementation of multiplication of two boolean functions in their ANFs. We used the implementation MultANF$_{64}$ of multiplication described in Appendix C.4. As mentioned earlier, using this algorithm, two 30-variable boolean functions can be multiplied in less than 2 seconds on a 3 GHz processor. Carrying out the simulation of full 1152 rounds of TRIVIUM with $n = 30$ requires 3456 multiplications and the entire simulation requires about one-and-half hours.

4.2 Method - 2

The previous section, gave a method to symbolically compute TRIVIUM up to r rounds of key initialization using the MultANF$_{64}$. This section, gives a different method for symbolically computing TRIVIUM. Algorithm 3 (in Appendix A) converts the ANF of an n-variable boolean function to its corresponding truth table representation and Algorithm 4 (in Appendix A) converts the truth table representation to its corresponding ANF. Method - 2 first constructs the truth table of the first output bit z_1. It then uses Algorithm 4 to transform the truth table representation of z_1 into its corresponding ANF.

Method - 2 randomly chooses n bit positions and fixes the remaining $160 - n$ bits to a fixed constant value as before. Then for each of the possible 2^n values of the selected n bit positions, the first output bit of full round TRIVIUM is obtained. This output bit is then inserted into the appropriate position of the 2^n bit size truth table corresponding to the first output bit z_1. After repeating the above procedure 2^n times, the truth table corresponding to the first output bit z_1 is obtained. To improve on time, a fast implementation of TRIVIUM is used, where 64 rounds of TRIVIUM are computed in parallel. The truth table obtained is then converted to the ANF using Algorithm 9 (given in Appendix C.2). Thus, computing the ANF of z_1.

We explain this new procedure with the help of an example, where $n = 10$. All the n bits are selected from the key bits. Let the selected 10 key bit positions be $1, 4, 22, 38, 42, 44, 53, 56, 61, 78$, i.e., the key bits $k_1, k_4, k_{22}, k_{38}, k_{42}, k_{44}, k_{53}, k_{56}$, k_{61}, k_{78} are treated as variables. Assume that the remaining 70 bits of key are fixed to the constant value OX01C87E277448253F4 and the 80 bit IV is fixed to the constant value OX49DA7EF6B4EC27037844, where the numbers are given in hexadecimal. Now the 10 key bit positions can take all possible $2^{10} = 1024$ values from 0 to 1023. Iterate over all possible values of the key bits $k_1, k_4, k_{22}, k_{38}, k_{42}, k_{44}, k_{53}, k_{56}, k_{61}, k_{78}$ and keep the remaining 70 bits of key and the 80 bits of IV fixed to the constant values. This produces 1024 different 80-bit key and 1 80-bit IV common for all these 80-bit keys. For each of these 1024 80 bit keys and the 80 bit IV, the first output bit (or any other bit) of full (or reduced) round of TRIVIUM is computed using a fast 64-bit implementation of TRIVIUM.

The truth table corresponding to 10 variables is of size 1024 bits. We therefore take a 1024 bit array. The truth table is then constructed by inserting in the appropriate position the value of the first output bit computed using a 64-bit implementation of TRIVIUM. Let b denote the first output bit when the variables $k_1, k_4, k_{22}, k_{38}, k_{42}, k_{44}, k_{53}, k_{56}, k_{61}, k_{78}$ take the integer value i. Then the truth table is constructed by inserting b in the i^{th} bit position of the truth table. The truth table once constructed is then converted to its corresponding ANF using Algorithm 9 of Appendix C. Thus, computing the ANF of z_1 when it is viewed as a function of the variables $k_1, k_4, k_{22}, k_{38}, k_{42}, k_{44}, k_{53}, k_{56}, k_{61}, k_{78}$ and the remaining key and IV bits fixed to the above mentioned constant.

5 Randomness Experiments

In this section, we utilize the symbolic computation tools developed in the previous section, to conduct some basic randomness tests on full (or reduced) round TRIVIUM. The focus is the first output bit of TRIVIUM, which according to the previous discussion, is an n-variable boolean function in n_k key variables and n_{iv} IV variables. The goal is to look for any structural defect in the first output bit of TRIVIUM. To be specific, we studied whether the first output polynomial behaves like a random polynomial or not. There were two types of experiments we looked into. In addition to the this, another test was also conducted to check for balancedness amongst the first l output bits of TRIVIUM. To be specific experiments were conducted by taking $l = 8$.

For the tests performed in this paper random samples were generated by randomly selecting $n = n_k, (n_{iv} = 0)$ key bit positions. Keeping these n bit positions fixed, the rest $160 - n$ bit positions were randomly set to either 0 or 1. Let the key variables be $k_{i_1}, \ldots, k_{i_n}$ and denote the rest of the $160 - n$ key and IV variables by $\mathbf{v}$. If S_r denotes the state after r rounds, then z_1 is the XOR of the polynomials corresponding to the state bits $S_r[66], S_r[93], S_r[162], S_r[177], S_r[243]$ and $S_r[288]$. Thus,

$$z_1 = S_r[66] \oplus S_r[93] \oplus S_r[162] \oplus S_r[177] \oplus S_r[243] \oplus S_r[288].$$

Then z_1 can be written in the following form.

$$z_1 \quad = \quad f(k_{i_1}, \ldots, k_{i_n}, \mathbf{v}). \tag{1}$$

Choose independent and uniform random values for the variables in $\mathbf{v}$. Then z_1 can be considered to be a random polynomial in the variables $k_{i_1}, \ldots, k_{i_n}$. Each such value of the variables in $\mathbf{v}$ corresponds to a different polynomial in the variables $k_{i_1}, \ldots, k_{i_n}$.

5.1 Test for Finding Non-Random First Bit Polynomials

Keep the n bit positions fixed and select independent and uniform random values for the variables in $\mathbf{v}$. For each such value of $\mathbf{v}$, we have selected two test criteria.

1. The first one checks whether the number of zeroes and ones in the truth table of z_1 are close to expected value 2^{n-1} or not. This is because a random function is expected to contain as many zeroes as ones in its truth table, i.e., the number of zeroes and ones are expected to be close to 2^{n-1}.

2. The second test checks whether for a particular degree d $(0 \le d \le n)$ and a particular polynomial, the number of monomials of degree d deviates significantly away from the expected value, which for a random polynomial is $\frac{1}{2}\binom{n}{d}$. The test also does the same for the total number of monomials present in a particular polynomial. In this case the expected number of monomials is 2^{n-1}.

We have reported those polynomials that have failed at least one of these criteria.

Denote by u^* a uniform random n-variable polynomial. For a random polynomial the probability of occurrences of each monomial m is $\frac{1}{2}$. Then the occurrences of a particular monomial in the random polynomial u^* defines a Bernoulli process with $p = \frac{1}{2}$. Therefore the number of monomials for a given degree d $(0 \le d \le n)$ follows Binomial $\mathcal{B}\left(\binom{n}{d}, \frac{1}{2}\right)$ distribution. The total number of monomials follows Binomial $\mathcal{B}\left(2^n, \frac{1}{2}\right)$ distribution. Again for a random polynomial u^*, the probability of the t^{th} $(0 \le i \le 2^{n-1})$ entry of its truth table being zero is $\frac{1}{2}$. Therefore the total number of zeroes and ones in the truth table both follows Binomial $\mathcal{B}\left(2^n, \frac{1}{2}\right)$ distribution.

Consider the test on the weight of the function. Given a probability α there is an interval I_α, such that, the weight of a random function u^* lies in the interval I_α with probability α. So, if the observed value of the weight of the first output bit function f (see equation 1) lies outside I_α, then this may be taken as an indication of non-randomness of f. Similarly, one can test for the total number of monomials and the number of monomials of a fixed degree.

For values of 2^n and $\binom{n}{d}$ greater than or equal to 30, binomial distributions for the number of monomials of a particular degree, weight and the total number of zeroes in the truth table are all well approximated by the normal distributions $\mathcal{N}\left(\frac{1}{2}\binom{n}{d}, \frac{1}{2}\sqrt{\binom{n}{d}}\right)$ and $\mathcal{N}\left(2^{n-1}, 2^{\frac{n-2}{2}}\right)$. Hence, for all values of d except for $d = 0, 1, n-1, n$, the distribution for the number of monomials of a particular degree is well approximated by the normal distribution. Let N_d be the random variable corresponding to the observed values of the number of monomials of degree d. And let N_T and N_{T0} denote the random variables corresponding to the observed values for total number of monomials and number of zeroes in the truth table, respectively. Then all the random variables except N_0, N_1, N_{n-1}, N_n approximately follows the normal distribution.

The test statistic for the random variable N_d is $\frac{N_d - \frac{1}{2}\binom{n}{d}}{\sqrt{\frac{1}{4}\binom{n}{d}}}$, whereas the test statistic corresponding to the random variables N_T and N_{T0} are given by $\frac{R-2^{n-1}}{\sqrt{2^{n-2}}}$, where R is either N_T or N_{T0}. For a given α, the test then checks whether the observed values of these random variables lie in the interval corresponding to α or not. Thus, if $[-u_\alpha, u_\alpha]$ denotes the interval corresponding to α, then the test corresponds to checking whether the observed absolute value of the test statistic n_d $(2 \leq d \leq n-2)$ or n_T or n_{T0} is greater than u_α or not.

5.1.1 Experimental Results

The experiments where conducted by taking values of $n = 10, 20$ and 30. Also, for all the experiments six values of alpha were taken, namely,

$$\alpha_1 = \left(1 - \frac{1}{2^2}\right), \alpha_2 = \left(1 - \frac{1}{2^3}\right), \alpha_3 = \left(1 - \frac{1}{2^4}\right),$$
$$\alpha_4 = \left(1 - \frac{1}{2^5}\right), \alpha_5 = \left(1 - \frac{1}{2^6}\right), \alpha_6 = \left(1 - \frac{1}{2^7}\right). \tag{2}$$

The values in (2) roughly corresponds to 75%, 87.5%, 93.75%, 96.88%, 98.44% and 99.22%, respectively. For a standard normal distribution $I_{\alpha_1} = [-1.15, 1.15]$, $I_{\alpha_5} = [-2.15, 2.15]$ and $I_{\alpha_6} = [-2.66, 2.66]$. The size of I_α increases with the increase in the value of α. So for the test corresponding to the weight of the function f at level α_6, the test fails if the absolute value of the observed N_T is greater than 2.66. In other words the test fails if $\mid n_T \mid > 2.66$. Also if a observed value lies outside the interval I_α then the observed value is further away from the expected value when the value of α (say α_6) is high rather than when its value is low (say α_1). This immediately implies that the tests corresponding to the six values of alpha are not independent. Since $\alpha_2 > \alpha_1$, rejection in case of α_2 automatically implies rejection for α_1, as the interval corresponding to α_2 is a superset of the interval corresponding to α_1. Thus, if the test fails for α_6 then the test also fails for all other values of α. Also failing at α_6 implies that the observed value is significantly away form the expected value and hence can be considered as an evidence of non-randomness.

Although we did not find any polynomial for which most of these test fails, we were able to get some polynomials which fails the test for certain parameters. The following tables list some of these polynomials.

Table 1 gives some polynomials for $n = 10, 20, 30$, whose truth table representations has number of zeroes lying outside the interval I_{α_6}. In the Table the column "Key Variables" indicate the key bit positions that were treated as variables. The columns "Key Constant" and "IV Constant" gives the values of $80 - n$ and 80 bits of the key and IV bits which were set to constant values.

Table 1: Table showing list of some polynomials with number of zeroes and one's lying outside the interval for $\alpha = \alpha_6$. The values given in the table are for $n = 10, 20, 30$ and 1152 key initialization rounds of TRIVIUM.

n	**Key Variables**	**Key Constant**	**IV Constant**	**Number of 0's**
10	1, 4, 22, 38, 42, 44, 53, 56, 61, 78	OX452D5AA716418A9CC	OXBC925DE125682B159CB4	465
		OX1476803AD7850AD36	OXA1D62667224E6CF221CF	465
		OX31D5EC5914E3D922F	OXE24571405777B5521A	555
		OX54CD8D3B53FC0A114	OXD4702BB150946D98D944	556
		OX238009F2E69728CB8	OX68131089DB607D1981F1	556

Continued on next page

Table 1 – *Continued from previous page*

n	**Key Variables**	**Key Constant**	**IV Constant**	**Number of 0's**
10	1, 4, 22, 38, 42, 44, 53, 56, 61, 78	OX53DB1C63D36BB4FD2	OXCF5050997F8601AB88EF	558
		OX42F216A6B2AFCEC17	OX30E66D573F151F784B58	560
		OX17485DC470A73061E	OXD54A1D5A59055062EFB6	571
	15, 16, 20, 27, 31, 37, 41, 45, 58, 73	OX27F50AF693342B6F9	OX706CCD7801037A0A49	437
20	0, 1, 9, 10, 14, 19, 27, 29, 41, 42, 52, 55, 62, 64, 68, 69, 71, 75, 78, 79	OX3625E972822DB6A	OXB2D91DF4E87047E9B8C6	522657
		OX80F5C4876AADE17	OXA380363693475CFCCEB	522768
		OXB7521EE35C15C4B	OX309D70CFFD406A96299A	522860
		OXBCEFBB60D3A6BAF	OXB0EC6893275307067F03	522862
		OXCD8AC4B29BEE0B1	OX1DFF5B9FFE4363C2F1A3	522902
30	1, 4, 7, 9, 10, 12, 13, 14, 15, 21, 25, 27, 30, 31, 32, 33, 34, 44, 52, 54, 55, 56, 58, 59, 62, 66, 69, 70, 74, 79	OX290C10B0294D2	OX586A33527C2928DDE2C6	536920658
	7, 15, 20, 21, 22, 26, 29, 30, 32, 33, 34, 41, 42, 49, 52, 54, 55, 56, 57, 59, 60, 63, 64, 65, 66, 72, 75, 76, 78, 79	OX1FD41217D312F	OXC8C051B0D49C69D1A7DD	536822130
		OX12C5E491E4B6F	OX99E4748853D60D6617EC	536920867

Table 2, gives some polynomials for $n = 10, 20, 30$, whose total number of monomials in its ANF lies outside interval I_{α_6}. In the table the column "Monomial Degrees" refers to the degree of monomials for which in addition to the total number of monomials, the number of monomials for that particular degree also fails the tests at level α_6.

Table 2: Table showing list of some polynomials with its total number of monomials lying outside the interval for $\alpha = \alpha_6$. The values given in the table are for $n = 10, 20, 30$ and 1152 key initialization rounds of TRIVIUM.

n	**Key Variables**	**Key Constant**	**IV Constant**	**Monomial Degrees**
10	1, 4, 22, 38, 42, 44, 53, 56, 61, 78	OX37FE4B0255D1D295C	OXD70079FAE0F0308EC206	6, 8
		OX457B6B0466DE7552E	OXD167CC3093E7E699466	None
		OX0484EB9A3E80085D	OX9B10785F6BF67CA8D5CB	None
		OX243E3DFA82D00EE44	OXB4526FDF61F96D7FCAE3	None
	15, 16, 20, 27, 31, 37, 41, 45, 58, 73	OX5EE252240CE406D5	OX3F0E2249DE7C031CF797	None
20	0, 1, 9, 10, 14, 19, 27, 29, 41, 42, 52, 55, 62, 64, 68, 69, 71, 75, 78, 79	OX56B4A18579E0D3E	OXAC576EF0BDDE67E72619	None
		OXFC12A46241151AD	OX10E6744E590F46973ADD	13
		OXADC520A5DA98587	OX77EC7B17675B6489CAD8	None
		OXC6AFA4B133A47F7	OX61207A01BCC272B683F9	None
		OX43ED55256B3CFF5	OX822E158DE22B7390747F	None
		OXAA1BE875BC0B948	OXE49D3F5E9DF3726567A	10
		OX44B684623514BE0	OX9CB0767A4B911C07655B	13
		OXF9BB1A903D2B55A	OXBEFF617BF05E74ED8172	11
30	7, 15, 20, 21, 22, 26, 29, 30, 32, 33, 34, 41, 42, 49, 52, 54, 55, 56, 57, 59, 60, 63, 64, 65, 66, 72, 75, 76, 78, 79	OXE65F1294C96A	OXEB482AFBDFE04F8DAD56	14
		OX128D80C2688E3	OX3CF5643BE9AD30EAC0C8	None
		OX199D831A8D833	OX9F7651D0129823F00C61	None
		OX1DBD945A6AD33	OXDB855A93A2834AC2FE5C	15

Apart form this, the experiment found examples of many polynomials which failed the six tests for monomials of certain degrees at level α_6. However, we could not find a single polynomial which fails all the six tests for both balancedness, total number of monomials and monomials of certain degree. However, more and more examples of polynomials failing the tests for balancedness, total number of monomials and monomials of certain degree can be found, as the value of α is decreased form α_6 to α_1

For example let us consider the case when $\alpha = \alpha_4$. Table 6 (Appendix D) gives a list of few polynomials which not only fails the test for balancedness but also the test for the total number of monomials at level α_4. In addition, the table also gives the monomial degrees for which the test fails. The column "Monomial Degrees" is similar to the one in Table 2. In our experiments we found 3 polynomials for $n = 10$ and 2 polynomials for $n = 30$ which had failed the test. However we could not find any example for $n = 20$.

If the value of α was further relaxed then we get more examples of polynomials where all the three conditions fail. Tables 5, 6, 7, 8, 9 (Appendix D) gives examples of such polynomials which fails all the three tests at level $\alpha_5, \alpha_4, \alpha_3, \alpha_2$ and α_1, respectively. In case of $\alpha = \alpha_5$, we can see from Table 5 that corresponding to $n = 10$ we have only two polynomials which failed the tests, whereas we could not find any such examples for $n = 20, 30$. However, when the value of α was relaxed to α_1, we found 45, 61 and 28 polynomials for $n = 10, 20$ and 30, respectively. The tables also show a steady increase in the number of monomials of a particular degree failing the test as α decreases.

5.2 Balancedness Test for the First l Output Bits

This section generalizes the first test of Section 5.1, where tests were done on the weight of the truth table corresponding to the first output bit z_1. In this section, instead of taking just one bit, test were done on the first l output bits of TRIVIUM. Let $X_i \in \{0,1\}, i \in \{1,2,3,\ldots,l\}$ denote the random variable corresponding to the i^{th} output bit of TRIVIUM and x_i its corresponding observer value at a particular run of TRIVIUM. Then for random polynomials, $X_i \overset{i.i.d.}{\sim} \mathcal{B}er\left(\frac{1}{2}\right), i \in \{1,2,3,\ldots,l\}$, where *i.i.d.* means independently and identically distributed and $\mathcal{B}er(p)$ stands for Bernoulli distribution with probability of success p. Therefore, $\Pr\left[X_1 = x_1, X_2 = x_2, \ldots, X_l = x_l\right] = \frac{1}{2^l}$, for all $(x_1, x_2, \ldots, x_l) \in \mathbb{F}_2^l$. Thus the joint distribution of $(X_1, X_2, \ldots, X_l)$ is an uniform distribution over $\mathbb{F}_2^l$. Let, $(a_1, a_2, \ldots, a_l)$ denote a particular value of the l-tuple. Define,

$$Y_{(a_1,a_2,\ldots,a_l)} = \begin{cases} 1; & \text{if}\,(x_1, x_2, \ldots, x_l) = (a_1, a_2, \ldots, a_l)\,; \\ 0; & \text{if}\,(x_1, x_2, \ldots, x_l) \neq (a_1, a_2, \ldots, a_l)\,; \end{cases}$$

Then $Y_{(a_1,a_2,\ldots,a_l)} \sim \mathcal{B}er\left(\frac{1}{2^l}\right)$. For a particular instance of TRIVIUM, n bits are treated as variables and the remaining $160 - n$ are fixed to constants. The truth table of the all the i^{th} output function $z_i, i \in \{1,2,3,\ldots,l\}$, is generated by 2^n runs of TRIVIUM. Let $N = 2^n$. Define,

$$Y_{(a_1,a_2,\ldots,a_l)} = \sum_{i=0}^{N-1} Y_{i,(a_1,a_2,\ldots,a_l)},$$

where $Y_{i,(a_1,a_2,\ldots,a_l)}$ denote the random variable $Y_{(a_1,a_2,\ldots,a_l)}$ corresponding to the i^{th} run of TRIVIUM. Therefore, expectation $E\left[Y_{(a_1,a_2,\ldots,a_l)}\right] = \frac{N}{2^l}$ and variance $Var\left(Y_{(a_1,a_2,\ldots,a_l)}\right) = \frac{N\left(2^l-1\right)}{2^{2l}}$, for all $(a_1, a_2, \ldots, a_l) \in \mathbb{F}_2^l$. Hence for large N, $Y_{(a_1,a_2,\ldots,a_l)} \sim \mathcal{N}\left(\frac{N}{2^l}, \sqrt{\frac{N\left(2^l-1\right)}{2^{2l}}}\right)$, for all $(a_1, a_2, \ldots, a_l) \in \mathbb{F}_2^l$. This gives, for a particular confidence level $(1-\alpha)$, an interval $I_\alpha = [-z_\alpha, z_\alpha]$, such that $\Pr\left[\left|\frac{Y_{(a_1,a_2,\ldots,a_l)} - \frac{N}{2^l}}{\sqrt{\frac{N\left(2^l-1\right)}{2^{2l}}}}\right| \le z_\alpha\right] = 1 - \alpha$. Thus, balancedness check for a particular choice of $(a_1, a_2, \ldots, a_l)$ boils down to testing the following null hypothesis

$$H_0 : p = \frac{1}{2^l} = p_0 \text{ (say)}$$

against the alternate hypothesis

$$H_A : p \neq \frac{1}{2^l}.$$

5.2.1 Experimental Results

Experiments were done by taking values of $n = 20, 30$ and $l = 8$. For $n = 20$, $Y_{(a_1,a_2,\ldots,a_8)} \sim \mathcal{N}\left(2^{12}, 2^2\sqrt{255}\right)$ and for $n = 30$, $Y_{(a_1,a_2,\ldots,a_8)} \sim \mathcal{N}\left(2^{14}, 2^7\sqrt{255}\right)$ for all $(a_1, a_2, \ldots, a_8) \in \mathbb{F}_2^8$. Hypothesis tests were done to check for balancedness for all 256 possible choices of $(a_1, a_2, \ldots, a_8)$. The confidence level was set at 0.95, i. e., α was set at 0.05. For each $n = 20$ and 30, the bits to be treated as variables were fixed. The remaining $160 - n$ bits positions were varied to generate 1000 different polynomials z_i (for all $i \in \{1, 2, \ldots, 8\}$) after full round key initialization of TRIVIUM. The experiments revealed for each of $n = 20$ and 30 for almost all the choices of $160 - n$ bits, the balancedness test fails for some choices of $(a_1, a_2, \ldots, a_8)$. Table 3, gives 10 such examples for $n = 20, 30$. In the table, the column corresponding to "Strings failing the hypothesis test" gives the integer value of the string $(a_1, a_2, \ldots, a_8)$ for which the hypothesis test had failed. Thus, a the value of "61" in the column corresponds to the binary string whose hexadecimal representation if given by "OX3D".

Table 3: Table showing a list of 10 polynomials each for $n = 20, 30$ and $l = 8$, all of which had failed the Balancedness test for some of the strings $(a_1, a_2, \ldots, a_8) \in \mathbb{F}_2^8$.

n	**Key Variables**	**Key Constant**	**IV Constant**	**Strings failing the hypothesis test**
20	8, 18, 19, 20, 23, 28, 29, 30, 39, 40, 41, 47, 48, 53, 56, 59, 64, 70, 72, 73	OXA21768F225AE936	OX74B078DB62E31BA54359	61, 110, 149, 182, 209, 253
		OX3BE24042DB0C421	OXAA567C4955D946047304	28, 35, 43, 49, 50, 53, 65, 94, 114, 118, 162, 168, 178, 183, 1194, 213, 249
		OX88F3FEC0318538E	OX2B87779A19863BB18905	107, 112, 142, 143, 168, 239, 252
		OX36D00CE2606D134	OX8EB134639711212E2CAA	4, 9, 33, 38, 43, 78, 101, 110, 122, 130, 135, 146, 164, 191, 204, 216
		OX424625207D56F6	OX738C279B7471D987F4C	0, 16, 37, 38, 54, 123, 141, 175, 248
		OXA7BFA8E42BC4D73	OX58673EABBD7642461C9A	15, 74, 77, 90, 120, 127, 132, 174, 185, 188, 207, 210, 221, 237, 239
		OXF2556362F79482A	OX17BF3C9AF7E3312DA8FF	20, 21, 41, 43, 59, 60, 66, 99, 118, 135, 141, 143, 153, 155, 158, 161, 176, 180, 185, 216, 234, 236
		OXDFB9A8909C9A446	OXADB914D021983A6F88E8	32, 76, 96, 125, 140, 156, 180, 245
		OXF6865827A5FEA40	OX9DCA24B3F9315AB9097	45, 94, 97, 133, 152, 155, 167, 180, 196, 225, 231
		OX1D633C22B81B580	OX1CA17053918671838005	10, 12, 40, 82, 94, 107, 114, 116, 122, 126, 132, 162, 192, 207, 219, 220, 240
30	1, 2, 4, 6, 10, 11, 13, 18, 20, 21, 22, 27, 29, 30, 35, 38, 39, 42, 45, 48, 50, 51, 53, 54, 56, 63, 68, 69, 74, 78	OX106BA503BA685	OX1A778F3E4684B2757A8	23, 29, 51, 59, 105, 110, 114, 133, 146, 147, 173, 198, 237
		OX2C04D2321BEDC	OXBF571F23FA7F36149C19	14, 60, 61, 68, 88, 117, 118, 138, 143, 144, 151, 186, 194, 216, 240
		OX132E52C33FBF2	OXDC1353073BD8705A0D7C	5, 41, 48, 77, 88, 98, 104, 157, 162, 167, 177
		OX301DE482002B9	OX97BF60A02F1A42824416	16, 29, 74, 83, 101, 113, 154, 196, 249
		OX381CB23A685B3	OX80456D9C2C25542DEC70	18, 56, 71, 80, 165, 243
		OX2F5B96D7CBC82	OXA59A226B1EF14F604E03	4, 7, 8, 23, 27, 40, 50, 188, 214, 239, 241
		OX261570967C24B	OXA81B2FF2CB216187FFDA	8, 55, 66, 90, 93, 108, 117, 139, 163, 199, 209
		OX26AD5652E935E	OX9C7D56FDF873804C19	15, 21, 57, 68, 83, 99, 117, 129, 134, 152, 219, 246
		OX1E6FC6E452F65	OX5BDB3100E0B26DCA4981	7, 22, 33, 52, 225
		OXFD990AD8A539	OX770A7F52CAE040BC7DAD	1, 3, 5, 7, 17, 20, 80, 101, 114, 136, 146, 171, 177, 180, 196, 223, 230, 239, 247, 255

5.3 Aggregate Tests

Consider any monomial m. Then the probability that m occurs in u^* defines a Bernoulli trial with probability $p = \frac{1}{2}$. The total number of monomials of degree d follows Binomial $\mathcal{B}\left(\binom{n}{d}, \frac{1}{2}\right)$ distribution and the total number of monomials occurring in u^* follows $\mathcal{B}\left(2^n, \frac{1}{2}\right)$ distribution. We conducted a simple test of hypothesis to determine whether z_1 behaves like a uniform random polynomial in the following manner. A total of $N = 1000$ (except for $n = 30$, for which $N = 544$ was taken) independent and uniform random choices of constant values were made for the variables in $\mathbf{v}$. Each such value gives rise to a different polynomial z_1: denote the polynomials so formed by $z_{1,1}, \ldots, z_{1,N}$. For each $z_{1,i}$, we recorded the distribution of monomials together with the total number of monomials, i.e., we recorded the number of occurrences n_d, of degree d monomials, for each $d = 0, 1, \ldots, n$ together with the total number of monomials $\sum_{i=0}^{n} n_d$. Let, $\tilde{X} = \left(\tilde{X}_1, \tilde{X}_2, \ldots \tilde{X}_N\right)$ where each $\tilde{X}_i = \left(X_{i,0}, X_{i,1}, \ldots, X_{i,n}, X_{i,2^n}\right); i = 1, 2, \ldots, N$ and $X_{i,j}$ = number of monomials of degree j in $z_{1,i}$ for $j = 0, 1, 2, \ldots, N$; and $X_{i,2^n}$ = total number of monomials in $z_{1,i}$. Consider the random variables $T_d = \sum_{i=1}^{N} X_{i,d}$ $(0 \leq d \leq n)$ and $T_{\sum_{i=1}^{N}} X_{i,2^n}$. Recall, that independent and uniform random choices were made for the variables in $\mathbf{v}$. Hence, $X_{i,d} \overset{i.i.d}{\sim} \mathcal{B}\left(\binom{n}{d}, \frac{1}{2}\right)$ for all $0 \leq d \leq n$ and $X_{i,2^n} \overset{i.i.d}{\sim} \mathcal{B}\left(2^n, \frac{1}{2}\right)$. Therefore, $T_d \sim \mathcal{B}\left(N \cdot \binom{n}{d}, \frac{1}{2}\right)$ for all $0 \leq d \leq n$ and $T_{2^n} \sim \mathcal{B}\left(N \cdot 2^n, \frac{1}{2}\right)$.

Separate tests were carried out to determine whether monomials of degree d occur with probability $p = 1/2$ and also whether the distribution of all monomials (without degree restriction) is with probability $p = 1/2$. In each case, we tested the null hypothesis

$$H_0 : p = \frac{1}{2} = p_0 \text{ (say)}$$

against the alternate hypothesis

$$H_A : p \neq \frac{1}{2}.$$

For sufficiently large N, the binomial distributions $\mathcal{B}\left(N \cdot \binom{n}{d}, \frac{1}{2}\right)$ and $\mathcal{B}\left(N \cdot 2^n, \frac{1}{2}\right)$ are well approximated by normal $\mathcal{N}\left(N \cdot \binom{n}{d} \cdot p_0, \sqrt{N \cdot \binom{n}{d} \cdot p_0 \cdot (1 - p_0)}\right)$ and $\mathcal{N}\left(N \cdot 2^n \cdot p_0, \sqrt{N \cdot 2^n \cdot p_0 \cdot (1 - p_0)}\right)$, respectively. Therefore, for sufficiently large N,

$$T_d \sim \mathcal{N}\left(N \cdot \binom{n}{d} \cdot p_0, \sqrt{N \cdot \binom{n}{d} \cdot p_0 \cdot (1 - p_0)}\right)$$

and

$$T_{2^n} \sim \mathcal{N}\left(N \cdot 2^n \cdot p_0, \sqrt{N \cdot 2^n \cdot p_0 \cdot (1 - p_0)}\right).$$

The test statistic T'_d corresponding to the degree d monomials is then given by

$$T'_d = \frac{T_d - N \cdot \binom{n}{d} \cdot p_0}{\sqrt{N \cdot \binom{n}{d} \cdot p_0 \cdot (1 - p_0)}}$$

and the test statistic corresponding to the total number of monomials is given by

$$T'_{2^n} = \frac{T_{2^n} - N \cdot 2^n \cdot p_0}{\sqrt{N \cdot 2^n \cdot p_0 \cdot (1 - p_0)}},$$

where both T'_d $(0 \leq d \leq n)$ and T'_{2^n} follows $\mathcal{N}(0, 1)$.

By the argument presented above all of these $T'_i, i = 0, 1, \ldots, n, 2^n$ follow normal $\mathcal{N}(0, 1)$. The null hypothesis was tested at 95% interval, which boils down to checking whether the absolute value of the observed test statistic

t'_d is less than 1.96 (since, for a random variable X, following normal $\mathcal{N}(0,1)$, $\Pr[-1.96 \le X \le 1.96] = 0.95$) or not. Thus for a particular sample $\tilde{x}$, we check if $| t'_d | \le 1.96$ or not. The above experiment was repeated for each $n = 10, 15, 16, 20$ and $r =$ 576, 767, 864, 885, 1152 (full round) rounds of key initialization. For $n = 30$, we have considered only the full round TRIVIUM.

5.3.1 Experimental Results

The results of the randomness test described above are given in Table 4. In the table, the column corresponding to "Key Variables" gives the bit positions which were taken as variables in the experiment. The column corresponding to "Rejected Monomial Degrees" denote the degrees of monomials for which, in the corresponding experiment, the hypothesis test failed. Also, the last column "Total Reject" answers the question whether the hypothesis test for the total number of monomials failed or not. In the table "None" means that none of the hypothesis tests corresponding to different monomial degrees failed. In the table, by rejection of degree zero, we imply that there was an imbalance in the constant term of the polynomials occurring in those particular samples.

Let, us explain what we mean with the help of an example. Consider the case when number of initialization rounds $r = 1152, n = 10$ and the position of the key bits which are treated as variables are $6, 9, 12, 15, 23, 26, 33, 35, 57, 61$. In Table 4, there are 2 rows corresponding to this entry. In the first row, it is observed that the randomness test failed for degree 5 only whereas in the later case it was found that none of the tests had failed. For degree 5, the test suggest that we accept the null hypothesis if the value of the test statistic based on the current data lies in the interval [125508.04, 126491.96], otherwise we reject the null hypothesis. In the first case, the value of the test statistics was 125481, while in the second case it was 125819. We can see that these values are close to the lower bound of the acceptance region. This shows that the results of the tests are dependent on the current random sample.

Table 4: Table Showing the Results of the Randomness Test.

r	$n(=n_k)$	Key Variables	Rejected Monomial Degrees	Total Reject?
576	10	3, 5, 11, 20, 35, 36, 41, 46, 58, 69	7, 8, 9, 10	Yes
	15	3, 5, 8, 11, 19, 20, 28, 30, 35, 36, 41, 46, 58, 69, 75	3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15	Yes
	16	3, 5, 8, 11, 19, 20, 28, 30, 35, 36, 41, 46, 58, 63, 69, 75	3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16	Yes
	20	5, 10, 11, 12, 16, 19, 20, 23, 28, 29, 30, 35, 36, 41, 52, 53, 58, 59, 69, 71	1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20	Yes
767	10	3, 5, 11, 20, 35, 36, 41, 46, 58, 69	6	No
	15	3, 5, 8, 11, 19, 20, 28, 30, 35, 36, 41, 46, 58, 69, 75	2	No
	16	3, 5, 8, 11, 19, 20, 28, 30, 35, 36, 41, 46, 58, 63, 69, 75	None	No
	20	5, 10, 11, 12, 16, 19, 20, 23, 28, 29, 30, 35, 36, 41, 52, 53, 58, 59, 69, 71	13	No
864	10	3, 5, 11, 20, 35, 36, 41, 46, 58, 69	6	No
	15	3, 5, 8, 11, 19, 20, 28, 30, 35, 36, 41, 46, 58, 69, 75	5	No
	16	3, 5, 8, 11, 19, 20, 28, 30, 35, 36, 41, 46, 58, 63, 69, 75	None	No
	20	5, 10, 11, 12, 16, 19, 20, 23, 28, 29, 30, 35, 36, 41, 52, 53, 58, 59, 69, 71	5, 17	No
885	10	3, 5, 11, 20, 35, 36, 41, 46, 58, 69	None	No
	15	0, 2, 4, 10, 13, 20, 22, 28, 29, 44, 46, 50, 62, 64, 79	8, 14	No
		3, 5, 8, 11, 19, 20, 28, 30, 35, 36, 41, 46, 58, 69, 75	4	No
		6, 9, 23, 24, 26, 27, 33, 49, 50, 55, 56, 66, 72, 76, 78	None	No
		7, 12, 13, 22, 24, 29, 30, 48, 49, 50, 53, 55, 56, 63, 77	None	No
		10, 18, 22, 25, 27, 29, 35, 47, 53, 65, 69, 73, 74, 76, 77	14	No

Continued on next page

Table 4 – *Continued from previous page*

r	$n(=n_k)$	Key Variables	Rejected Monomial Degrees	Total Reject?
885	16	3, 5, 8, 11, 19, 20, 28, 30, 35, 36, 41, 46, 58, 63, 69, 75	2	No
		3, 6, 9, 16, 37, 40, 46, 47, 52, 53, 58, 70, 71, 72, 75, 77	3	No
		3, 9, 18, 19, 21, 32, 35, 39, 50, 52, 57, 58, 60, 70, 71, 72	None	No
		6, 13, 15, 21, 30, 31, 41, 42, 47, 50, 51, 54, 64, 69, 70, 79	9, 15	No
		7, 12, 25, 38, 49, 50, 54, 55, 59, 60, 61, 63, 70, 71, 74, 78	12, 13, 14	No
	20	5, 10, 11, 12, 16, 19, 20, 23, 28, 29, 30, 35, 36, 41, 52, 53, 58, 59, 69, 71	11, 13	No
		8, 11, 12, 17, 19, 21, 26, 27, 38, 39, 44, 45, 47, 59, 63, 69, 70, 73, 74, 75	0, 14, 15	No
		4, 6, 8, 14, 15, 29, 37, 38, 40, 41, 43, 47, 49, 50, 52, 57, 59, 61, 63, 71	None	No
		1, 8, 9, 14, 18, 22, 24, 29, 32, 33, 36, 38, 47, 49, 57, 59, 62, 68, 78, 79	None	No
		4, 7, 13, 14, 18, 21, 32, 37, 38, 49, 52, 54, 59, 61, 68, 69, 73, 74, 76, 78	None	No
1152	10	2, 4, 14, 16, 18, 29, 36, 53, 55, 73	10	No
		3, 5, 11, 20, 35, 36, 41, 46, 58, 69	8	No
		4, 15, 26, 38, 39, 53, 58, 62, 65, 69	5	No
		4, 13, 22, 53, 67, 68, 70, 76, 78, 79	None	No
		6, 9, 12, 15, 23, 26, 33, 35, 57, 61	5	No
			None	No
		6, 13, 32, 34, 36, 41, 43, 57, 67, 79	None	No
		8, 15, 21, 25, 37, 38, 44, 45, 48, 62	6	No
	15	0, 2, 14, 38, 39, 45, 50, 52, 55, 62, 64, 68, 74, 78, 79	13, 15	No
		1, 3, 20, 26, 36, 40, 43, 50, 55, 57, 58, 62, 72, 73, 79	8	No
		2, 3, 8, 16, 19, 24, 25, 29, 30, 31, 45, 48, 51, 68, 71	None	No
		2, 5, 8, 12, 21, 35, 37, 41, 53, 56, 57, 64, 69, 73, 74	8	No
		3, 5, 8, 11, 19, 20, 28, 30, 35, 36, 41, 46, 58, 69, 75	None	No
	16	1, 6, 7, 10, 12, 20, 26, 29, 34, 36, 39, 51, 52, 70, 72, 79	5, 9	No
		3, 5, 8, 11, 19, 20, 28, 30, 35, 36, 41, 46, 58, 63, 69, 75	None	No
		5, 6, 10, 16, 23, 39, 44, 49, 52, 62, 65, 67, 73, 75, 76, 78	10	Yes
		7, 9, 18, 19, 27, 29, 38, 39, 40, 43, 55, 66, 67, 74, 76, 77	0, 2	No
		23, 24, 26, 32, 36, 37, 41, 48, 49, 50, 52, 55, 57, 58, 71, 74	None	No
	20	0, 5, 8, 9, 12, 13, 17, 21, 24, 30, 32, 33, 36, 41, 49, 56, 59, 65, 69, 77	None	No
		0, 7, 19, 22, 28, 29, 31, 32, 33, 34, 36, 42, 56, 60, 62, 63, 64, 65, 66, 77	5	No
		3, 5, 20, 31, 32, 34, 35, 36, 37, 41, 49, 50, 59, 61, 63, 68, 70, 71, 72, 79	16	No
		4, 8, 12, 17, 18, 21, 28, 34, 35, 39, 43, 44, 45, 47, 52, 64, 65, 66, 69, 77	0	No
		5, 10, 11, 12, 16, 19, 20, 23, 28, 29, 30, 35, 36, 41, 52, 53, 58, 59, 69, 71	None	No
	30	7, 15, 20, 21, 22, 26, 29, 30, 32, 33, 34, 41, 42, 49, 52, 54, 55, 56, 57, 59, 60, 63, 64, 65, 66, 72, 75, 76, 78, 79	23, 25	No

6 Conclusion

In this we paper proposed two methods to symbolically compute TRIVIUM. We have conducted some randomness tests on the polynomial representing the first output bit. The first tests gives examples of non-random polynomials. This study is along the line of cube testers [ADMS09], where the authors found examples of such polynomials using cube attack techniques. The second study checks for balancedness across the first l output bits of TRIVIUM. Experimental results are reported for $l = 8$. It was noticed that the first 8 bits of TRIVIUM are in general not balanced. In the third study we have conducted an aggregated randomness test using Hypothesis testing techniques. The results of the experiments mostly indicate the absence of any overall non-randomness. By overall non-randomness we mean that there is no clinching evidence which may suggest that probability of occurrence of monomials of a particular degree d is not equal to half.

Acknowledgement We would like to thank Sanjay Bhattacherjee and Raju Maiti for those insightful discussions and the anonymous referees for comments that helped improve the presentation of this paper.

References

[ADMS09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In *Fast Software Encryption*. Springer-Varlag, 2009.

[AKK$^+$03] Noga Alon, Tali Kaufman, Michael Krivelevich, Simon Litsyn, and Dana Ron. Testing low-degree polynomials over GF (2). In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 188–199. Springer, 2003.

[CMP07] Chris Charnes, Cameron McDonald, and Josef Pieprzyk. Attacking Bivium with MiniSat. Technical Report 2007/413, Cryptology ePrint Archive, Report, http://eprint.iacr.org/2007/040, 2007.

[DCP] Christophe De Cannière and Bart Preneel. Trivium-specifications. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/030 (2005).

[DS09] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. *Advances in Cryptology-EUROCRYPT 2009*, pages 278–299, 2009.

[EJT07] Håkan Englung, Thomas Johansson, and Mettem Sönmez Turan. A Framework for Chosen IV Statistical Analysis of Stream Ciphers. In *INDOCRYPT*, pages 268–281, 2007.

[FKM08] Simon Fischer, Shahram Khazaei, and Willi Meier. Chosen IV Statistical Analysis for Key Recovery Attacks on Stream Ciphers. In *AFRICACRYPT*, pages 236–245, 2008.

[FV13] Pierre-Alain Fouque and Thomas Vannet. Improving Key Recovery to 784 and 799 rounds of Trivium using Optimized Cube Attacks. In *Fast Software Encryption*. Springer, 2013.

[Jou09] Antoine Joux. *Algorithmic cryptanalysis*. CRC Press, 2009.

[MB07] Alexander Maximov and Alex Biryukov. Two Trivial Attacks on Trivium. In *Selected Areas In Cryptography*, pages 36–55, 2007.

[O'N07] Sean O'Neil. Algebraic Structure Defectoscopy. In *Special ECRYPT Workshop–Tools for Cryptanalysis*, 2007.

[Rad06] Håvard Raddum. Cryptanalytic Results on Trivium. Technical Report 2006/039, eSTREAM, ECRYPT Stream Cipher Project, Report, http://www.ecrypt.eu.org/stream/papersdir/2006/039.ps, 2006.

[Sam13] Samajder, Subhabrata and Sarkar, Palash. Fast Multiplication of the Algebraic Normal Forms of Two Boolean Functions. In Lilya Budaghyan, Tor Helleseth and Matthew G. Parker, editor, *WCC 2013*, pages 373–385, 2013. (pdf available at `http://www.selmer.uib.no/WCC2013/pdfs/Samajder.pdf`).

[SPF10] Ilaria Simonetti, Ludovic Perret, and Jean Charles Faugère. Algebraic Attack Against Trivium. In *First International Conference on Symbolic Computation and Cryptography, SCC*, volume 8, pages 95–102, 2010.

[SRD] Mohamed Abdelraheem Sondre Ronjom and Lars Eirik Danielsen. (`http://www.ii.uib.no/~mohamedaa/odbf/help/ttanf.pdf`).

[TK07] Mettem Sönmez Turan and O. Kara. Linear approximations for 2-Round Trivium. In *Proc. First International Conference On Security of Information and Networks (SIN 2007)*, pages 96–105. Trafford Publishing, 2007.

[Vie07] M. Vielhaber. Breaking One.Fivium By AIDA : An Algebraic IV Differential Attack. Technical Report 2007/413, Cryptology ePrint Archive, Report, http://eprint.iacr.org/2007/413, 2007.

A Iterative Algorithms

Algorithm 3: PRE_PROCESS (A, B, n, i)

```
Input: $A, B, n, i$
for $i = 0, 1, 2, \ldots, n-1$ do
    for $j = 0, 1, 2, \ldots, 2^i - 1$ do
        for $k = 0, 1, \ldots, 2^{n-i-1} - 1$ do
            $A[2^{n-i-1} + j \cdot 2^{n-i} + k] = A[2^{n-i-1} + j \cdot 2^{n-i} + k] \oplus A[j \cdot 2^{n-i} + k]$
            $B[2^{n-i-1} + j \cdot 2^{n-i} + k] = B[2^{n-i-1} + j \cdot 2^{n-i} + k] \oplus B[j \cdot 2^{n-i} + k]$
        end
    end
end
```

Algorithm 4: POST_PROCESS (C, n, i)

```
Input: $C$, $n$, $i$
for $i = n-1, n-2, n-3, \ldots, 0$ do
    for $j = 0, 1, 2, \ldots, 2^i - 1$ do
        for $k = 0, 1, \ldots, 2^{n-i-1} - 1$ do
            $C[2^{n-i-1} + j \cdot 2^{n-i} + k] = C[2^{n-i-1} + j \cdot 2^{n-i} + k] \oplus C[j \cdot 2^{n-i} + k]$
        end
    end
end
```

B MultANF$_w$

C Multiplication of Algebraic Normal Forms and its Implementation

C.1 Conversion from ANF to Truth Table

Polynomials in $\mathbb{R}$ are represented using a sequence of bits. In this sequence, the presence of every monomial is denoted by a single bit. Since the number of such possible monomials in $\mathbb{R}$ is 2^n, 2^n bits are used to represent

Algorithm 5: PRE_PROCESS$_w$ (A, B, n, i)

```
Input: $A, B, n, i$
for $j = 0, 1, 2, \ldots, 2^i - 1$ do
    for $k = 0, 1, \ldots, 2^{n-i-1} - 1$ do
        $A[2^{n-i-1} + j \cdot 2^{n-i} + k] = A[2^{n-i-1} + j \cdot 2^{n-i} + k] \oplus A[j \cdot 2^{n-i} + k]$
        $B[2^{n-i-1} + j \cdot 2^{n-i} + k] = B[2^{n-i-1} + j \cdot 2^{n-i} + k] \oplus B[j \cdot 2^{n-i} + k]$
    end
end
```

Algorithm 6: UNPACK (X, n) : Unpacks a w-bit word to a byte array.

```
Input: a $w$-bit word $X$; $n := \log_2 w - 3$.
for $i = 0, 1, 2, \ldots, n - 1$ do
    temp = Bit-wise AND of $X$ and $M_{i+1}$
    temp = SHIFT right temp by $2^{n+3-i-1}$ (according to our assumption, the left-most bit is the LSB)
    $X$ = temp XOR $X$
end
```

any polynomial in $\mathbb{R}$.

It is clear that one can compute the values of p_0, $(p_0 \oplus p_1)$, q_0 and $(q_0 \oplus q_1)$ independently and then multiply them to get the required $p_0 q_0$ and $(p_0 \oplus p_1) \cdot (q_0 \oplus q_1)$. Thus one needs to compute p_0 and $p_0 \oplus p_1$ (respectively, q_0 and $q_0 \oplus q_1$) from p (respectively, q). This is explained by the pseudo-code presented as Algorithm 3 in Appendix A, where the input polynomials p and q are given as 2^n bit arrays A and B, respectively. It is repeated for both the polynomials p and q.

C.2 Multiplication of Truth Table

After the PRE_PROCESS step, notice that the arrays A and B, contains the truth table representation of the corresponding polynomials p and q. This is because, at each iteration of the algorithm the polynomial p (respectively q) is divided into p_0 (respectively q_0) and $p_0 \oplus p_1$ (respectively $q_0 \oplus q_1$). Now, p_0 (respectively q_0) corresponds to the situation when $x_n = 0$, and $p_0 \oplus p_1$ (respectively $q_0 \oplus q_1$) corresponds to the situation when $x_n = 1$. Thus, after n such steps the bit value $A[i]$ (respectively, $B[i]$) corresponds to the values of p (respectively, q) evaluated at $x_1 = i_1, x_2 = i_2, \ldots, x_n = i_n$, where $i_1 i_2 \ldots i_n$ is the binary representation of i and $i \in \{0, 1, 2, \ldots, 2^n - 1\}$. The Algorithm then does a bit-wise AND of the arrays A and B and stores the corresponding results in another array C. This corresponds to multiplication of the truth tables of q with that of q.

As extracting a bit from a byte is costly table lookups are used. Here instead of going all the way down to the n^{th} level, the algorithm stops at level $n - \beta$ and use table lookups to perform multiplication of two β-variable polynomials. The value of β is taken to be 3, because the table corresponding to $\beta = 4$ becomes very large. Thus the polynomials p and q are packed into byte arrays and byte level XOR's are used to multiply them.

C.3 Conversion of Truth Table to Algebraic Normal Form

This step is similar to the one described in Section C.1. The only difference is that in this step as opposed to Section C.1 one traverses from the leaf to the root of the computation tree. In other words the outer loop runs from $n - 1$ to 0 instead of 0 to $n - 1$ in Section C.1. Algorithm 4 of Appendix A, gives the corresponding pseudo code.

Algorithm 7: EXTRACT_AND_LOOKUP (X, Y, Z, n) : Extracts bytes from w-bit words X and Y, does a table look-up and stores the result in the corresponding byte of Z.

```
Input: w-bit words $X$, $Y$, $Z$; table $T$; $n := \log_2 w - 3$
for $i = 0, 1, \dots, 2^n - 1$ do
    if $i = 0$ then
        $Z := T[X$ AND $B_1][Y$ AND $B_1]$
    end
    else
        temp := $T[(X$ AND $B_{i+1})$ SHIFT left by $i \cdot 2^3$ bits.][$(Y$ AND $B_{i+1})$ SHIFT left by $i \cdot 2^3$ bits.]
        (According to our assumption the left-most bit is the LSB).
        $Z$ := temp XOR $Z$

    end
end
```

Algorithm 8: PACK (Z, n) : Packs a w-bit word into a byte array.

```
Input: a w-bit word $Z$; $n := \log_2 w - 3$.
for $i = n-1, n-2, n-3, \dots, 0$ do
    temp = Bit-wise AND of $Z$ and $M_{i+1}$
    temp = SHIFT right temp by $2^{n+3-i-1}$ (according to our assumption, the left-most bit is the LSB)
    $Z$ = temp XOR $Z$
end
```

C.4 The Implementation

One may use w-bit XOR instead of 8-bit, assuming the architecture allows w-bit word arithmetic, where $w = 2^k, k \geq 3$. The motivation is to save on the number of 8-bit XOR's. Thus, using one w-bit XOR, one can save $2^{\log_2 w - 3}$ many XOR's. However, doing it this way one can only go up to $n - \log_2 w$ level, since, as mentioned in the previous section, maintaining a table of size greater than 3-variables is not feasible. Hence using w-bit words, involves, an additional task of UNPACKING and PACKING the w-bit word into bytes so that one can use the 8-bit table lookup.

Instead of directly copying the w-bit words to and back from byte arrays, a constant amount of extra space is used to get an algorithm which not only saves the cost of copying but also saves on the number of XOR's. The idea is to use $2^{\log_2 w - 3}$ many w-bit word masks, say $M_1, \dots M_{2^{\log_2 w - 3}}$ plus an additional temporary variable "temp", where M_i contains 1 in the bit positions $j \cdot 2^{\log_2 w - i} + k, j \in \{0, 2, 4, \dots 2^i - 2\}$ and $k \in \{0, 1, 2, \dots, 2^{\log_2 w - i} - 1\}$ and 0 elsewhere. The M_i's actually simulate each level of the PRE_PROCESS and POST_PROCESS described above for $n = \log_2 w$ and corresponding to each w-bit word.

Algorithm 9: POST_PROCESS$_w$ (C, n, i)

```
Input: $C$, $n$, $i$
for $j = 0, 1, 2, \dots, 2^i - 1$ do
    for $k = 0, 1, \dots, 2^{n-i-1} - 1$ do
        $C[2^{n-i-1} + j \cdot 2^{n-i} + k] = C[2^{n-i-1} + j \cdot 2^{n-i} + k] \oplus C[j \cdot 2^{n-i} + k]$
    end
end
```

Algorithm 10: MultANF$_w$ (T, A, B, C, n, w) : A non recursive algorithm to multiply two boolean functions in their ANF's.

```
Input: 8-bit Look-up Table $T$; Two polynomials $A$ and $B$; $C$ for Result; number of varibles $n$; word
       size $w$
Output: C := Product of $A$ and $B$
for $i = 0, 1, 2, \dots, n - \log_2 w - 1$ do
    PRE_PROCESS$_w(A, B, n - \log_2 w, i)$
end
for $i = 0, 1, 2, \dots, 2^{n-\log_2 w} - 1$ do
    UNPACK ($A[i]$, $\log_2 w - 3$)
    UNPACK ($B[i]$, $\log_2 w - 3$)
    EXTRACT_AND_LOOKUP ($A[i]$, $B[i]$, $C[i]$, $\log_2 w - 3$)
    PACKING ($C[i]$, $\log_2 w - 3$)
end
for $i = n - \log_2 w - 1, n - \log_2 w - 2, n - \log_2 w - 3, \dots, 0$ do
    POST_PROCESS$_w(C, n - \log_2 w, i)$
end
```

We summarize our discussion by giving the w-bit non-recursive algorithm, MultANF$_w$ (see Algorithm 10 of Appendix B) as presented in [Sam13]. The routine MultANF$_w$ takes as input T, A, B, n, w, where A and B are the corresponding w-bit word representation of two n-variate polynomials ($n > \log_2 w \geq 3$) and $T_{256\times256}$ is a 8-bit table look-up. MultANF$_w$ multiplies the polynomials A and B with the help of table T and stores the result in C.

To do this, the MultANF$_w$ routine calls the subroutines "PRE_PROCESS$_w$" (Algorithm 5 of Appendix B), "UNPACK" (Algorithm 6 of Appendix B), "EXTRACT_AND_LOOKUP" (Algorithm 7 of Appendix B), "PACK" (Algorithm 8 of Appendix B) and "POST_PROCESS$_w$" (Algorithm 9 of Appendix B). The subroutine PRE_PROCESS$_w$ corresponds to the operations while descending down the recursion tree, whereas the subroutine "POST_PROCESS_w" corresponds to the operations while ascending up the recursion tree. Notice that the subroutine 'UNPACK" is called twice once each for the w-bit words $A[i]$ and $C[i]$. The subroutine EXTRACT_AND_LOOKUP extracts each byte from the w-bit words A and B; does the corresponding table lookup and then stores the value returned by the table in the exact byte position of C.

One can choose different values of w. For $w = 64$, the implementation MultANF$_{64}$ is particularly fast. It is reported in Table 1 of [Sam13] that two 30-variable boolean functions can be multiplied in less than 2 seconds on a 3GHz processor. We later use MultANF$_{64}$ to build our simulation tool for TRIVIUM.

D Tables

Table 5: Table showing list of some polynomials with its total number of monomials and the number of zeroes in its truth table lie outside the interval for $\alpha = \alpha_5$. The values given in the table are for $n = 10, 20, 30$ and 1152 key initialization rounds of TRIVIUM.

n	Key Variables	Key Constant	IV Constant	Monomial Degrees
10	1, 4, 22, 38, 42, 44, 53, 56, 61, 78	OX37BDD3EAD0BAFABC0	OXDB565D9DB98F4E3389C5	None
		OX47E214EB5727E04C9	OXD34F684B1055DAECE93	7

Table 6: Table showing list of some polynomials with its total number of monomials and the number of zeroes in its truth table lie outside the interval for $\alpha = \alpha_4$. The values given in the table are for $n = 10, 20, 30$ and 1152 key initialization rounds of TRIVIUM.

n	**Key Variables**	**Key Constant**	**IV Constant**	**Monomial Degrees**
10	1, 4, 22, 38, 42, 44, 53, 56, 61, 78	OX37BDD3EAD0BAFABC0	OXDB565D9DB98F4E3389C5	None
		OX47E214EB5727E04C9	OXD34F684B1055DAECE93	7
		OX4547D85442C8D68CF	OXD08829A188F6241E7C2D	5, 8
30	7, 15, 20, 21, 22, 26, 29, 30, 32, 33, 34, 41, 42, 49, 52, 54, 55, 56, 57, 59, 60, 63, 64, 65, 66, 72, 75, 76, 78, 79	OX11D3963CFE658	OXE9F618EC66862A0DEB4E	12
		OX191766116C74F	OX1E8B7D71045E0F56A4EA	4, 24

Table 7: Table showing list of some polynomials with its total number of monomials and the number of zeroes in its truth table lie outside the interval for $\alpha = \alpha_3$. The values given in the table are for $n = 10, 20, 30$ and 1152 key initialization rounds of TRIVIUM.

n	**Key Variables**	**Key Constant**	**IV Constant**	**Monomial Degrees**
10	1, 4, 22, 38, 42, 44, 53, 56, 61, 78	OX37BDD3EAD0BAFABC0	OXDB565D9DB98F4E3389C5	None
		OX47E214EB5727E04C9	OXD34F684B1055DAECE93	2, 7
		OX4547D85442C8D68CF	OXD08829A188F6241E7C2D	5, 8
20	0, 1, 9, 10, 14, 19, 27, 29, 41, 42, 52, 55, 62, 64, 68, 69, 71, 75, 78, 79	OX6725535534737CA	OXDDAE21B901422A1643A	None
		OXDDAE21B901422A1643A	OXF51C70E932C18D17D41	None
30	7, 15, 20, 21, 22, 26, 29, 30, 32, 33, 34, 41, 42, 49, 52, 54, 55, 56, 57, 59, 60, 63, 64, 65, 66, 72, 75, 76, 78, 79	OX11D3963CFE658	OXE9F618EC66862A0DEB4E	12
		OX3C25D092EFEF9	OXEE0C5AA49BA676F04E05	14, 15
		OX613242AFA99E	OX74996C308B57426EC1FF	4, 13
		OX191766116C74F	OX1E8B7D71045E0F56A4EA	4, 24

Table 8: Table showing list of some polynomials with its total number of monomials and the number of zeroes in its truth table lie outside the interval for $\alpha = \alpha_2$. The values given in the table are for $n = 10, 20, 30$ and 1152 key initialization rounds of TRIVIUM.

n	**Key Variables**	**Key Constant**	**IV Constant**	**Monomial Degrees**
10	1, 4, 22, 38, 42, 44, 53, 56, 61, 78	OX37BDD3EAD0BAFABC0	OXDB565D9DB98F4E3389C5	3, 4
		OX47E214EB5727E04C9	OXD34F684B1055DAECE93	2, 7
		OX4547D85442C8D68CF	OXD08829A188F6241E7C2D	4, 5, 8
		OX22C82FA5C4FF1FFA7	OX8E6C7CCC6DA42DE02582	6
		OX46F73734324A8C3CF	OXED6F602BFE6161C4B002	5, 7, 8
		OX548E1A39B23F3483B	OX2C2B1447188F2DF15053	6, 8
		OX52A20EB6B6861FD2B	OX69EF224FC6FB72AC6C37	2, 3
		OX243E3DFA82D00EE44	OXB4526FDF61F96D7FCAE3	5, 6
		OX21FEF73EB0DC5739A	OX7598278A31B96B6E06F5	3

Continued on next page

Table 8 – *Continued from previous page*

n	**Key Variables**	**Key Constant**	**IV Constant**	**Monomial Degrees**
10	1, 4, 22, 38, 42, 44, 53, 56, 61, 78	OX36B1281D43A9240B3	OXE58A191A1E6C333C8EFD	3, 5
		OX1185DD59742FE8169	OX89B62A60C21C42A0E6B2	4,
		OX37FE4B0255D1D295C	OXD70079FAE0F0308EC206	6, 7, 8
20	0, 1, 9, 10, 14, 19, 27, 29, 41, 42, 52, 55, 62, 64, 68, 69, 71, 75, 78, 79	OX6725535534737CA	OXDDAE21B901422A1643A	8, 12, 15, 17
		OXC481FD7BC1F523	OXC37057969DFB005C79DC	5, 6, 8, 9
		OXDF7305B0CFDB228	OXC9C17DF5198908297669	8
		OX3480FFC0AD084D6	OXF51C70E932C18D17D41	9, 15
		OX80A26F93FFE786E	OX49025F652E977970AAA3	3, 5, 6, 9
		OX99997304FBA97AB	OX91A0123D835369D66539	3, 10, 11
		OX1438B4C6E410610	OXEC881E225AE17BE12D06	6, 16
		OX2B8619E6B23FD69	OXF24C75F66F5957352674	None
		OX2E93E577A837AAC	OXF50C2B06C5B100F1D712	2, 14
		OXFDAFFE872B1ECA6	OX63F0791A5BD92EA49167	2, 7, 10
		OX1E15EFE0723A1A0	OXAFF45320480C32FE05AD	11, 12, 13, 14, 17
		OX94252897FEBA	OX40B53BED60BA2A4EF7BD	7
		OX5A4644E0DCF37F1	OXAFE71BE0360E0C918B9C	3, 9, 13
		OXAA07D7C6F262C91	OX4F821468B1891D2AD371	2, 3, 4, 12
		OX748AA0B4C4431F6	OX6BB3415153E252D74428	9
		OX82641E96DDFE210	OXA045545ADF754FE49440	4, 16, 17
30	7, 15, 20, 21, 22, 26, 29, 30, 32, 33, 34, 41, 42, 49, 52, 54, 55, 56, 57, 59, 60, 63, 64, 65, 66, 72, 75, 76, 78, 79	OX11D3963CFE658	OXE9F618EC66862A0DEB4E	12, 15, 18
		OX3C25D092EFEF9	OXEE0C5AA49BA676F04E05	14, 15, 23
		OXCDCA70B4903D	OX28094F93A84519B6030	2, 12, 21, 26
		OX613242AFA99E	OX74996C308B57426EC1FF	4, 13, 15, 19, 24, 26, 27
		OX191766116C74F	OX1E8B7D71045E0F56A4EA	4, 18, 24, 25

Table 9: Table showing list of some polynomials with its total number of monomials and the number of zeroes in its truth table lie outside the interval for $\alpha = \alpha_1$. The values given in the table are for $n = 10, 20, 30$ and 1152 key initialization rounds of TRIVIUM.

n	**Key Variables**	**Key Constant**	**IV Constant**	**Monomial Degrees**
10	1, 4, 22, 38, 42, 44, 53, 56, 61, 78	OX37BDD3EAD0BAFABC0	OXDB565D9DB98F4E3389C5	2, 3, 4, 5
		OX22C82FA5C4FF1FFA7	OX8E6C7CCC6DA42DE02582	6
		OX47E214EB5727E04C9	OXD34F684B1055DAECE93	2, 4, 7
		OX4547D85442C8D68CF	OXD08829A188F6241E7C2D	4, 5, 8
		OX46F73734324A8C3CF	OXED6F602BFE6161C4B002	5, 7, 8
		OX548E1A39B23F3483B	OX2C2B1447188F2DF15053	6, 7, 8
		OX25999FA70096CE14A	OX534E7B0E0099371CEC2B	7
		OX32C3B8564711127E2	OXC3B65FA580064682A886	2, 5, 6, 7, 8
		OX36700C6F525F4A15E	OX76C0CA72E4037279E52	2, 5, 7
		OX1104F75E2114D99C6	OX99545A9EDB9B664CF25E	5, 6
		OX52A20EB6B6861FD2B	OX69EF224FC6FB72AC6C37	2, 3, 6

Continued on next page

Table 9 – *Continued from previous page*

n	**Key Variables**	**Key Constant**	**IV Constant**	**Monomial Degrees**
10	1, 4, 22, 38, 42, 44, 53, 56, 61, 78	OX243E3DFA82D00EE44	OXB4526FDF61F96D7FCAE3	5, 6, 7
		OX21FEF73EB0DC5739A	OX7598278A31B96B6E06F5	2, 3
		OX36B1281D43A9240B3	OXE58A191A1E6C333C8EFD	2, 3, 5
		OX373444186660E5FF4	OXEB195ADDBE903C2D1056	2
		OX1185DD59742FE8169	OX89B62A60C21C42A0E6B2	4
		OX13C98DD112B9E4345	OXC02E2FE44559226743EA	2, 6
		OX25964FF8044895C95	OXCFC776D1C4E100F35C85	2, 4, 5
		OX2628BA81850F8F769	OX1FCF571CE4612534B608	4, 8
		OX043DACA1A2026DBDA	OXBDC5DCD77F921AABDF6	4, 7
		OX2126745C279E5A10C	OX1248772E03E133CE0B7B	2, 7
		OX586B4E14496FAC82	OX9C2138672ABB3DDDC1F2	5
		OX368252990744C0C7	OX74DF32D819F351C27B0E	2, 3, 4
		OX532D3BE97067D5129	OX7BED67B3ABB91C35FA5D	5
		OX1382D64B413855656	OX95972A312ED14A3BF60	6
		OX52FFFC0FF6FD88EBD	OXF66572321FFA19728935	3, 7
		OX439C148480CEF8257	OX74FB1106EE59338B8704	3
		OX14B0825030BA0A96B	OX207C5A11622E7FE89689	2, 3, 7
		OX568F9EDC3FA5CFC5	OXCD8D6086AF815B848C24	3, 6, 8
		OX37A8A2D3F4AD45193	OXD49B28D3ABC66F27C37E	3, 6
		OX457B6B0466DE7552E	OXD167CC3093E7E699466	5, 6
		OX2CE451DE26F01574	OX1C1342B3181C132E7CCF	8
		OX2188425AC63CCD33F	OX90C929DD67D3678472EE	2, 4, 5, 6, 8
		OX2454FEF2819CFDFE8	OX9E71576A5F36051743D5	6, 7
		OX04D00A4F9785AC8C4	OX8DCA7A16BF435CE5940B	5
		OX37FE4B0255D1D295C	OXD70079FAE0F0308EC206	4, 6, 7, 8
		OX53CD508F74BBC7DBE	OXC37E2A7F2F8164D022BB	5, 7, 8
		OX048DAFF69510A4B0E	OX3A1A19897D5D77691BDB	5
		OX4708A09334FCAFE4F	OX4A4C60F2FECB3B1FFA4F	2, 3, 5
		OX4559C324D7A96E402	OX76093CA6A6820F188476	6
		OX11CCD131147B71B01	OX82C85493CE4525CC267A	4, 7
		OX43C5AE65F459310FF	OXE29F6C5FF9B122017D55	6
		OX074D96A9360193375	OXFA274E2AFE2E68F3ECE6	6
		OX073A6C0377AF88B83	OX6351165578DB3B77F014	3, 5
		OX11D7C2096469B8D59	OX85164E66AB350C5BCD85	2
20	0, 1, 9, 10, 14, 19, 27, 29, 41, 42, 52, 55, 62, 64, 68, 69, 71, 75, 78, 79	OXCD8AC4B29BEE0B1	OX1DFF5B9FFE4363C2F1A3	2, 5, 8, 13, 17
		OX6725535534737CA	OXDDAE21B901422A1643A	8, 12, 15, 16, 17
		OXC481FD7BC1F523	OXC37057969DFB005C79DC	5, 6, 8, 9, 14
		OXDF7305B0CFDB228	OXC9C17DF5198908297669	4, 6, 8, 17, 18
		OXE8EF47A657A1A10	OX47B07462C84600BBD4C3	8, 12, 17
		OX3480FFC0AD084D6	OXF51C70E932C18D17D41	4, 6, 9, 14, 15
		OXF79DC891384AFF0	OXDDAF2C635F1725E5722C	2, 3, 5, 11, 15, 17
		OX63CBCE13DFA0AFF	OXC314A7271C748FEB77B	3, 11, 15
		OX80A26F93FFE786E	OX49025F652E977970AAA3	3, 4, 5, 6, 9, 14

Continued on next page

Table 9 – *Continued from previous page*

n	**Key Variables**	**Key Constant**	**IV Constant**	**Monomial Degrees**
20	0, 1, 9, 10, 14, 19, 27, 29, 41, 42, 52, 55, 62, 64, 68, 69, 71, 75, 78, 79	OX99997304FBA97AB	OX91A0123D835369D66539	3, 10, 11
		OX1438B4C6E410610	OXEC881E225AE17BE12D06	2, 6, 7, 9, 16
		OX2B8619E6B23FD69	OXF24C75F66F5957352674	10, 13
		OX3907E66406C1230	OXCC4F643389D174E09308	9, 11, 12, 17
		OX8DC9AD07E1DD5B7	OX6F925A43AABE3589016F	2, 9, 18
		OX1C2904C43FF6577	OXE73A4739E4C117D70E8E	7, 11, 12, 13, 16, 17
		OX2E93E577A837AAC	OXF50C2B06C5B100F1D712	2, 3, 5, 13, 14
		OXFDAFFE872B1ECA6	OX63F0791A5BD92EA49167	2, 5, 7, 10, 15
		OX1B3537B58870F55	OXA33A411FC4173976088F	2, 3, 5, 8, 9, 10, 14, 15, 18
		OX1E15EFE0723A1A0	OXAFF45320480C32FE05AD	2, 6, 11, 12, 13, 14, 15, 17
		OX137AD462B48819D	OXECFE45F642EA682545D6	7, 10, 17
		OX94252897FEBA	OX40B53BED60BA2A4EF7BD	7, 10, 11
		OX2EEDD3E63910450	OX33953B8FBB3E5DB89240	6, 7, 17
		OX7E51F8E40591536	OXB9C6CEF795453064BE6	9, 10, 14, 15, 17
		OXD241FF6460C6208	OX5C725E6A96F9353ECCE8	2, 5, 12, 14, 18
		OX895517354CC7691	OX9099444B625C731D4A9F	4, 13
		OX18093FF2EA21B89	OX3627A654C272E098ABA	8, 15, 17
		OX69219DE1A75C6B5	OX59FB6A44546208BBA473	5, 8, 9, 12, 13, 15
		OX8CA5215750A80AE	OXBBFE302CAEC030A95C66	2, 7, 11, 18
		OX48F6A050FA29FD4	OXF13127F0DE9243B0AA33	10, 12
		OXA869C2C7AE9EEDD	OXFC4D4C6DE13E62F8530E	2, 6, 7
		OX54D805626D001BA	OXCC3F38A3216216E4DD16	10, 15
		OXFF859A14E90D2D3	OX2AC86B44BBEF20B3A8EB	3
		OX92E4EE44F4AA9B0	OXB2E35370E80D3FA438FE	5, 7, 8, 9, 10, 11, 12, 14
		OXC906517717370E8	OX407C62FEAAD57DE22435	6, 8, 9, 15
		OX81ABA2F10E414F1	OXD8DC3E5436B30D553DFF	13, 15
		OX5E2587E52504105	OX492770685C260EB94076	3, 4, 5, 7, 9, 11, 14
		OX66BE1B647D74852	OXE2B64E025081086192F3	2, 4, 5, 7, 8, 11, 13
		OXC987FF2679818EA	OXECBC1922F4E82FE0E298	3, 9, 14, 15, 16
		OX399995668AA5766	OXADDE1B907E417C75C073	9, 18
		OXA01F4042A39D2AF	OXB42E343741AE2885A4B8	8, 10, 11, 15, 16
		OXEB000173340180D	OX9D4811B44D956C1C4122	9, 10
		OX6FEDB601C5D0F7	OXACF51C7A59AC427CBA18	2, 4, 13, 14, 16, 17, 18
		OX349068A2D3BE11B	OXF51A7B67A45C5173EDB0	2, 6, 9, 10, 11, 12
		OX9904566610C1359	OX5C921B727602478B4F1F	2, 11, 13, 18
		OX631BFA24A283F98	OX8BEE5BDF986177DEFCB7	2, 4, 7, 8, 14, 15, 17
		OX41B6D6E060B45	OXF4301269A0A373516F83	4, 6, 7, 9, 15
		OX6AE3E77490E6D0B	OXED4B6FCC7E5B1FFAA681	2, 6, 12, 15, 17
		OXF9BB1A903D2B55A	OXBEFF617BF05E74ED8172	6, 9, 11, 18
		OXADB103911781696	OX78001622345E7535AF89	2, 3, 10, 12, 13, 14
		OXD4BAF7074479E09	OX3CB7239F46CD1A18C135	5, 6, 13, 16, 18
		OX31C341E7D2823C0	OX44E6375DA9C323B5EFCF	15, 16, 17
		OX5A4644E0DCF37F1	OXAFE71BE0360E0C918B9C	3, 9, 13

Continued on next page

Table 9 – *Continued from previous page*

n	**Key Variables**	**Key Constant**	**IV Constant**	**Monomial Degrees**
20	0, 1, 9, 10, 14, 19, 27, 29, 41, 42, 52, 55, 62, 64, 68, 69, 71, 75, 78, 79	OXBCF4D6769BCEFB	OX6D344C7B4AC745BC07FC	10, 11, 12, 18
		OXD989E1257E60721	OX85F9933760D63491E6D	2, 8, 13, 18
		OXFE4EC2B5DF70C87	OX7E0D7707ABF24E5811D8	4, 7, 8, 10, 11,14, 17
		OX6BB3225B97D099	OX5BD374B78F0F10E1F552	6
		OX3666DCC1529A055	OX998A7DC1F1F94C9CAFB0	3, 9
		OXAA07D7C6F262C91	OX4F821468B1891D2AD371	2, 3, 4, 5, 7, 12
		OXCDB760C050E0196	OX2FDE16E5A5517466E581	10, 16, 18
		OX748AA0B4C4431F6	OX6BB3415153E252D74428	9, 17
		OX82641E96DDFE210	OXA045545ADF754FE49440	4, 12, 13, 16, 17
30	7, 15, 20, 21, 22, 26, 29, 30, 32, 33, 34, 41, 42, 49, 52, 54, 55, 56, 57, 59, 60, 63, 64, 65, 66, 72, 75, 76, 78, 79	OX2A3B12E5B3AAA	OX9107625D556D1E48A3B5	3, 7, 13, 16, 17, 18, 20, 23, 25
		OX11D3963CFE658	OXE9F618EC66862A0DEB4E	9, 12, 13, 14, 15, 18
		OXF8534CF8A0C4	OXCEDD5CCE04F12DF6FA42	7, 9, 10, 14, 15, 18
		OXE0D05859F75D	OXDE0D17F6F4A032F4345A	2, 4, 5, 8, 13, 15, 16, 18
		OX1767659F97A78	OXDB0E189FAA7523B7F38C	3, 5, 11, 12, 13, 16, 18, 19, 24
		OX3C25D092EFEF9	OXEE0C5AA49BA676F04E05	4, 8, 10, 14, 15, 23, 24
		OX358F63BC9862E	OX2B8A12AF7C7513BFB545	4, 5, 8, 10, 14, 15, 16, 23, 25
		OXCDCA70B4903D	OX28094F93A84519B6030	2, 6, 12, 16, 21, 24, 26
		OX186E1140CAE7A	OXE5893222F3CF2AD91C84	3, 10, 17, 21, 24
		OXF5633C0E0766	OX3A2161ED1A9A6C545C99	6, 8, 14, 16, 17, 18, 23, 25
		OX3EF1C76CC3786	OX91441019D7A5F99C0E2	5, 8, 15, 16, 19, 22, 26, 27
		OX1E3305EE66BF7	OX84052206580263DB7246	3, 9, 12, 14, 23
		OX2BFEF0DB6F4F7	OX21D64C13071A1E0AA4DF	10, 14, 24, 28
		OX22BE07DCB8255	OX14B48826D4E3EE8AA4A	10, 11, 13, 14, 17, 22, 24, 25
		OX2C53E5CA904F8	OX4CF8318FB91A7BD1C2D0	4, 5, 8, 14, 17, 18, 19, 22, 24, 27
		OX93726691E2D0	OX2C4C389C765606937AF4	6, 10, 15, 18, 19, 13, 16
		OX3ED1D244BD2B1	OXBB5B758E8FB029E57666	2, 6, 7, 8, 15, 23, 25
		OX2DF6C79AE5433	OX920D16223BEE4EB5822E	3, 4, 8, 9, 12, 13, 20
		OX378C02C3FDF2B	OX77681A2286592408308D	3, 10, 11, 13, 15, 22, 23, 26, 28
		OX3B97E24D24147	OX694F280DCB2B108F1385	2, 9, 11, 12, 24, 28
		OXC7294829B50A	OXF303799BF930108F4B0F	4, 8, 11, 15, 16
		OX378763FE6C96	OX2FBC5FB87C8125734B6E	2, 3, 6, 7, 8, 12, 16, 18, 23, 25, 28
		OX3BECF5CC75818	OXC3B33304C9FD300B28F3	5, 12, 14, 17, 19, 21, 22, 23, 24
		OX3049A0E2FB512	OX50986952B94273E8F099	9, 13, 16, 24, 26
		OX313C07B28B127	OX30005763E511714B24C0	10, 11, 17, 18, 25, 26
		OX613242AFA99E	OX74996C308B57426EC1FF	4, 5, 8, 11, 12, 13, 15, 17, 19, 21, 22, 24, 25, 26, 27
		OX5AE80FC1F4DB	OXD5776A122F7F7B0049B1	3, 10, 15, 21, 23, 24
		OX191766116C74F	OX1E8B7D71045E0F56A4EA	4, 13, 17, 18, 24, 25