Cryptanalysis of Bivium using a Boolean all solution solver

Virendra Sule
vrs@ee.iitb.ac.in
Dept. of Electrical Engg.

Anmol Yadav
anmol.y@iitb.ac.in
Dept. of Aerospace Engg.

Indian Institute of Technology Bombay
Mumbai 400076
India

March 2, 2018

Abstract

Cryptanalysis of Bivium is presented with the help of a new Boolean system solver algorithm. This algorithm uses a Boolean equation model of Bivium for a known keystream. The Boolean solver uses implicant based computation of satisfying assignments and is distinct from well known CNF-satisfiability solvers or algebraic cryptanalysis methods. The solver is also inherently parallel and returns all satisfying assignments of the system of equations in terms of implicants. Cryptanalysis of Bivium is classified in four categories of increasing strength and it is shown that the solver algorithm is able to complete the key recovery in category 2 in 48 hours by a Python code. (This benchmark is improved to 3 hours by a C++ code). Computational algorithms for formation of equations and symbolic operations are also developed afresh for handling Boolean functions and presented. Limitations of these implementations are determined with respect to Bivium model and its cryptanalysis which shall be useful for cryptanalysis of general stream ciphers.

1 Introduction

This paper reports 80 bit key recovery of the Bivium cipher from an output stream with a given IV using a Boolean system solver algorithm developed recently by one of the authors and announced in [4]. This solver is based on Boolean computations and is principally different from existing algebraic or

CNF-satisfiability (SAT) based solvers of Boolean systems. The algebraic model of Bivium chosen has Boolean functions in ANF hence the computational performance is not comparable to results on CNF model obtained by using known algebraic and SAT solvers. However the results establish the practical feasibility of cryptanalysis (of a specific category) of Bivium using sequential implementation of this algorithm.

In recent times algebraic cryptanalysis of block and stream ciphers has gained considerable importance as powerful methods and algorithms based on extended linearization and improved Grobner basis generation emerged [1, 7]. At the same time CNF satisfiability (SAT) based methods have also emerged as powerful approaches for cryptanalysis based on the CNF models of ciphers [2, 3, 8]. However we shall prefer to call SAT methods as *Boolean cryptanalysis* which distinguish themselves from algebraic, since these do not use algebraic operations such as arithmetic in finite fields and gcd computations on polynomials unlike in algebraic cryptanalysis but resort to Boolean operations and operations on Boolean functions. This paper is also about Boolean cryptanalysis and the purpose is to report an instance of solving a system of Boolean equations arising in the model of the Bivium cipher with an instance of an output stream. Bivium cipher has been one of the important case studies where performances of multiple cryptanalysis methods are available as in [3, 6]. Hence Bivium cryptanalysis should be useful for understanding limitations of computational algorithms and their implementations.

The problem of cryptanalysis, that of solving the key bits given the input output data of a cipher is a known hard computational problem in general. Cryptanalysis can help to estimate in practical time, the time in which instances of a specific cipher can be solved given sufficient resource. Solving practical cases of cryptanalysis for reduced scale ciphers in practical time is the most valuable aspect of cryptanalysis which helps in designing and engineering secure cryptographic primitives and infrastructure. However methods of algebraic as well as SAT based cryptanalysis suffer following two hurdles

1. Algebraic methods are based on mathematically complex operations and are not inherently designed for parallel computation. Hence these are difficult to parallelize and scale up for solving large systems of practical size over large number of parallel processors. Due to the complex nature of mathematical operations they also require extensive memory storage.

2. SAT based methods utilize relatively simple operations and most of

them are based on the DPLL algorithm and its later extensions. However these require representation of Boolean functions of the model in CNF form. This representation itself blows up in size (in number of clauses and variables) for practical systems. However, even if such a conversion to CNF model is achievable, the greatest disadvantage of SAT based methods is that they are only aimed for solving a *decision problems*, that of existence of a solution to return one solution and are not suitable for solving all solution assignments of a given CNF data set. Multiple solutions can exist for Boolean equations even when enough number of relations are available.

In realistic cryptanalysis problems typically arising from known plaintext attack, it is required to find all solutions of the key or continue to find solutions until match exists between available data of plaintext and ciphertext blocks for ensuring correctness of the solution. The concern here is not that of decision or existence of solution as it is already known that a solution exists for the relations but due to multiple number of solutions all solutions are necessary. This problem of finding all solutions of Boolean systems (termed in short as *AllSolve*) is theoretically of higher complexity and is not even comparable to the decision problem. For instance, it is well known that for the 2-SAT decision problem which is of class P, the All-2-SAT problem is of class $\#P$. Apart from this need to find all solutions, the true practical hurdle is the representation of all solutions of the Boolean system since the number of solutions of a Boolean system grows exponentially in number of free variables in the solution. Finally, in order to be able to solve realistic size problems the algorithm for cryptanalysis is required to have an inherent parallel structure and should also facilitate scaling up in number of parallel processes to utilize available parallel resources. Several known methods of algebraic cryptanalysis and those based on algorithms for SAT are not inherently parallel. Hence parallel solution of Boolean systems is still an unfinished agenda in computational science [13]. Cryptanalysis problems pose challenges in this agenda which have central importance not just in Cryptanalysis but also for the Science of discrete and parallel computation.

The Boolean solver algorithm used in this paper is an inherently parallel algorithm for the AllSolve problem of Boolean systems based on computation of local *implicants* of Boolean functions in each equation. An important feature of this approach is the representation of all solutions in terms of implicants of the system. While the number of all solutions increases exponentially in free variables, the number of implicants such as orthogonal implicants increases polynomially in the number of variables by suppressing

the free variables. Hence representation of solutions in terms of implicants is a natural way to compactly represent all solutions. We show in this paper that this algorithm succeeds in solving the Bivium equations even with a sequential implementation of this algorithm. Hence the performance of this algorithm can be greatly improved if such problems are solved by a parallel implementation.

1.1 Representation of equations by factors

Often the models of ciphers (either block or stream) involve mixed operations and hence constitute Boolean functions not inherently represented in standard forms such as CNF also called product of sums (POS) (or DNF called sum pf products (SOP)). Boolean functions in algebraic normal form (ANF) can be used for compact representation of equations involving mixed Boolean functions. Hence it is reasonable to assume that the model of the cipher be expressible in terms of equations with functions in ANF, although this too is not a universal recipe for a good representation of the problem. The cryptanalysis problem is usually stated as a Boolean system

$$C = F(P, K) \tag{1}$$

where C is the ciphertext (or known output) block of bits, P the known plaintext block (including the IV block) of input bits while K is the unknown bits of the secret key to be solved. We assume that these equations are brought to the algebraic normal form (ANF) and given as

$$g_i(X) = 0, i = 1, 2, \ldots, \tag{2}$$

where g_i are in ANF or alternatively the system is expressed in the product form

$$F(X) = \prod_{i=1}^{N} f_i(X) \tag{3}$$

where $f_i = g_i'$ the Boolean complements of g_i. The factors f_i are in general not elementary disjuctions as in the CNF representation of F. The cryptanalysis problem is stated as follows.

Problem 1. Construct a system of equations (2) from the known plaintext data, (in the Bivium case construct the equations for a given output key stream and IV) and compute the factors f_i of a Boolean formula (3) $F(X)$ where X are unknown key bits to be solved. Compute all assignments x of variables X such that the Boolean formula is satisfied, i.e. $F(x) = 1$.

2 Implicant based solver algorithm

The implicant based solver announced in [4] is used to solve all satisfying assignments of the formula $F(X)$. We shall call this algorithm *Implicantalgo* Any satisfying assignment x of the formula F must also satisfy each of the factors f_i. For a single non zero Boolean function f arising as a factor in a formula F, let $S(f)$ denote the set of all satisfying assignments. This set has a compact representation in terms of implicants of f which are terms t in the variables of f such that $t \leq f$ as Boolean functions (which is equivalent to stating that if for any assignment x, $t(x) = 1$ then $f(x) = 1$). The implicant based algorithm first computes the set $I(f)$ of a complete set of implicants of f which has the property that if $f(x) = 1$ for an assignment x then there exists an implicant t in $I(f)$ such that $t(x) = 1$. Any complete set of implicants is thus equivalent to all terms in a sum of product (SOP) form of f. If each factor f of a complex formula F has small number of variables (called a sparse formula) this computation is practically feasible. (In the actual implementation of this algorithm, we used the set $I(f)$ of *orthogonal* (OG) implicants. This set has the property that if $f(x) = 1$ for some assignment x then exactly one of the implicants t in $I(f)$ satisfies $t(x) = 1$). The Implicantalgo computes successively the OG implicants of the formula F whose factors f_i are given. If instead a Boolean system of functions $f_i = 0$ is to be solved, the factors chosen are complements $g_i = f_i'$ and the OG implicant set of the formula

$$G = \prod_i g_i$$

is computed. At each step the algorithm computes OG implicants of pivot functions f_i and hence also results in OG implicants of the formula F. Details of the algorithm with examples can be referred from [4]. In the present paper we show that this algorithm is able to solve the complex systems of algebraic equations arising in the cryptanalysis of Bivium. To the authors knowledge solutions to such large systems in large number of variables have not been announced by other methods such as Grobner basis or XL or CNF SAT based algorithms.

2.1 Special features of the implicant based algorithm

This algorithm has following central features which distinguish it from well known solvers based on algebraic and SAT based methods.

1. The Boolean factors f_i of the Boolean formula whose satisfying assignments are being computed, are not required to be in a special form such as CNF. These can be Boolean functions represented in mixed form as they naturally arise from a case study. For Bivium case we have chosen to represent them in ANF since in this form the factors can be computed easily from the Bivium algorithm.

2. All the factors f_i need not be available apriori i.e. the formula F need not be known completely at the start. New factors may be available as new bits of key stream get known. This is strikingly different from elimination based methods in which all factors containing a set of variables need to be known apriori or as in SAT based solvers.

3. The algorithm is inherently thread parallel. The starting set of implicants give rise to threads which fork further into new threads as further implicants are computed. Hence although the number of threads seem to increase exponentially, they also get discarded as contradictions arise such as $f_i/t = 0$ in further steps of Implicantalgo. The memory gets freed once a thread is discarded. In the present paper the results on performance of computation are limited due to the sequential implementation while this can be improved by several orders due to inherent parallel nature of the algorithm.

4. Each successful thread results into an implicant of the formula F. This way all implicants of the formula or a complete set $I(F)$ are discovered giving a compact representation of all satisfying solutions. Although such all satisfying assignments grow exponentially due to free variables, the implicant representation suppresses the free variables and gives a polynomial number of implicants representing all solutions. The SAT based solvers are not designed for computing all solutions and can be used to compute each solution in independent trial resulting in exponential number of trials.

Due to these features the Cryptanalysis of Bivium performed using Implicantalgo is unique as the algorithm is different from other methods. If the same algebraic model is used then absolute performance in time for same parameters of the key stream can be compared. Further, the scope for improvement in performance by parallel implementation of this algorithm is also a unique advantage of this algorithm.

3 Bivium: algorithm, model and cryptanalysis

Bivium is a reduced version of the Trivium cipher [10]. The reduction is obtained by just keeping two registers of trivium in the algorithm or alternatively loading the third register by zero initial loading. Here the two registers of Bivium are of length 93 and 84 respectively totally with 177 states. These are denoted as $(s_1, s_2, ..., s_{177})$. These states are initialized by 80 bits of key, 80 bits of IV and zeros. The exact location of key and IV bits in these registers is shown in the encryption and decryption algorithm below. But from any time instant k called clocking index, the algorithm for obtaining the keystream output $z(k)$ is as follows. Two latent states t_1 and t_2 are defined and updates of all states are carried out by following rules in algorithm 1.

Algorithm 1: Bivium algorithm

Input: states $s_1(k), \ldots, s_{177}(k)$
Output: $z(k)$ for $k = 1, 2, \ldots$

$t_1(k) = s_{66}(k) \oplus s_{93}(k)$
$t_2(k) = s_{162}(k) \oplus s_{177}(k)$
$z(k) = t_1(k) \oplus t_2(k)$
% Updates of t_1, t_2
$t_1(k+1) = t_1(k) \oplus s_{91}(k) * s_{92}(k) \oplus s_{171}(k)$
$t_2(k+1) = t_2(k) \oplus s_{175}(k) * s_{176}(k) \oplus s_{69}(k)$
% State update
$(s_1(k+1), s_2(k+1), ..., s_{93}(k+1)) = (t_2(k+1), s_1(k), ..., s_{92}(k))$
$(s_{94}(k+1), s_{95}(k+1), ..., s_{177}(k+1)) = (t_1(k+1), s_{94}(k), ..., s_{176}(k))$

The output sequence $z(k)$ is thus a quadratic non-linear function of the states at each instant k. The encryption of a plaintext bit stream $p(k)$ and decryption of the ciphertext bit stream are carried out by the following algorithms for encryption and decryption. These use the key stream $z(k)$ for XOR-ing with plaintext or ciphertext.

3.1 Algebraic model for cryptanalysis

The cryptanalysis of Bivium algorithm from the known plaintext attack thus amounts to recovering the key bits when the IV bits are given and if the sequence $z(k + 4 * 177)$ is available for consecutive values of k. From the Bivium algorithm 1, the sequence of outputs $z(k)$ and the IV bits allow

Algorithm 2: Encryption algorithm

Input: Key bits $K_1, K_2, \ldots, K_{80}$, IV bits $IV_1, IV_2, \ldots, IV_{80}$, Plaintext stream $p(k)$ for $k = 1, 2, \ldots$

Output: Ciphertext stream $c(k)$ for $k = 1, 2, \ldots$

1 % Loading of Key and IV
2 State at $k = 1$
3 $(s_1, s_2, \ldots, s_{93})(1) = (K_1, K_2, \ldots, K_{80}, 0, \ldots, 0)$
4 $(s_{94}, s_{95}, \ldots, s_{177})(1) = (IV_1, IV_2, \ldots, IV_{80}, 0, \ldots, 0)$
5 For $1, 2, \ldots$
6 $c(k) = p(k) \oplus z(k + 4 * 177)$

Algorithm 3: Decryption algorithm

Input: Key bits $K_1, K_2, \ldots, K_{80}$, IV bits $IV_1, IV_2, \ldots, IV_{80}$, Ciphertext stream $c(k)$ for $k = 1, 2, \ldots$

Output: Plaintext stream $p(k)$ for $k = 1, 2, \ldots$

1 % Loading of Key and IV
2 State at $k = 1$
3 $(s_1, s_2, \ldots, s_{93})(1) = (K_1, K_2, \ldots, K_{80}, 0, \ldots, 0)$
4 $(s_{94}, s_{95}, \ldots, s_{177})(1) = (IV_1, IV_2, \ldots, IV_{80}, 0, \ldots, 0)$
5 For $k = 1, 2, \ldots$
6 $p(k) = c(k) \oplus z(k + 4 * 177)$

construction of an algebraic model

$$f_k(K_1, K_2, \ldots, K_{80}) = z(k) \tag{4}$$

in terms of Boolean functions f_k for specified output indices k obtained online. If a sufficient number of these functions are computed, a unique solution to the key bits may result by solving (4) as a Boolean system using the algorithm Implicantalgo. We shall thus refer to (4) as the algebraic model of Bivium for given IV and the sequence of output indices k. The Bivium algorithm 1 also gives rise to another offline algebraic model with all 177 states as unkown bits. This is denoted as

$$g_k(s_1, s_2, \ldots, s_{177}) = z(k) \tag{5}$$

for specified output indices k. Such a model can be computed apriori. The cryptanalysis to recover the key based on this model from a given IV is discussed in the next section. The strength of the Bivium algorithm can be determined under various conditions of the knowledge of IV and the output index sequence which define categories of cryptanalysis.

3.2 Cryptanalysis categories for Bivium

Cryptanalysis is a methodology to determine conditions under which an attack can be mounted on the cipher with a known cryptanalytic solver with an objective to recovering the bits of the secret key. By Kirchhoff's principle in cryptography the only secret information about the encryption process is the secret key shared between parties used in an encryption algorithm. Hence the security of encryption process is not expected to be compromised by the side information which is not intentionally kept secret. A large part of side information may be available with efforts relatively inexpensive than the secret information and security may turn out vulnerable to such information. Hence the cryptanalysis is meant to simulate conditions under which such subversion of the encryption is feasible. The side information in stream ciphers can consist of several types of bits such as output sequence for indices before the key stream is actually used for encryption, bits of internal state during the encryption and rarely even the key bits. We consider following categories of cryptanalysis whose feasibility in (specified practical time, chosen as 24 hours, on a standrad desktop computer) indicates a category of weakness of the cipher. For Bivium we consider following categories of cryptanalysis.

1. Category 1. The secret key recovered from the algebraic model (4) of the first 80 bits of output $z(k)$ for $k = 1, 2, \ldots, 80$ in practically specified time.

2. Category 2. The secret key recovered from the algebraic model (4) from an intermediate sequence $z(k)$, $k = k_0 + 1, k_0 + 2, \ldots, k_0 + 80$ in practically specified time for $1 < k_0 < 4 * 177$.

3. Category 3. A minimum value of index $k_{\min}$ is always specified for any stream cipher for functional usage of the key stream $z(k)$ for encryption. For Bivium it is $k_{\min} = 4 * 177$. This category is defined by being able to solve the key from (4) in practically specified time from output stream $z(k)$, $k > k_{\min}$.

4. Category 4. The secret key recovered from any consecutive sequence of $z(k)$ from the offline model (5) when IV is known. In this case the model equations are solved for the initial state $x(k_m) = (s_i(k_m), i = 1, \ldots, 177)$ from a known sequence $z(k)$, $k > k_m = 4 * 177$. Then the Bivium algorithm is reversed and all solutions to $x(0)$ are determined. The key is determined when the IV in $x(0)$ match. Hence this requires solving for all solutions of the state update in reverse. Thus in this category the equations are known apriory and not built online from known output stream.

These categories can be suitably extended for other stream ciphers as well as new categories can be defined on the basis of possibility of side information. For most stream ciphers, a reconstruction in practically specified time, of the internal state of registers from the key stream data constitutes an almost complete break of the cipher. In this paper we report Cryptanalysis of Bivium in category 2.

4 Computation of the algebraic model

In this section we discuss generation of Boolean equations (4) for the Bivium model and then use the Boolean solver algorithm Implicantalgo to retrieve the Key. Few notations are used to represent Boolean functions in ANF and defining Boolean operations on them. First consider the notation called ANFstuc as defined below in terms of equations involving Boolean functions in ANF. This is explained in following example.

Here one or more variables define a term using a symbolic-AND. Similarly one or more terms form a function in ANF by symbolic-XOR. For

example let a function in ANF be $1 \oplus x_1 \oplus x_2x_3$. Here x_1, x_2 and x_3 are Boolean variables. For our code we represent this by $[1, [[1], [2, 3]]]$. This is a list of elements. The first is the constant of the function, second element is another list of lists. The first list is denotes first degree terms. The next list denotes higher degree terms etc. A term itself is a list of indices of elements. Hence the elements of the list are further lists. This is called ANFstuc notation for representing functions in ANF.

Example 1. In the ANFstuc notation the function

$$f(x_1, x_2, x_3) = 1 \oplus x_2 \oplus x_3 \oplus x_1x_3 \oplus x_2x_3 \oplus x_1x_2x_3$$

is represented by the list

$$[1, [[2], [3], [1, 3], [2, 3], [1, 2, 3]]]$$

4.1 Symbolic operations on ANFstuc

For obtaining the Boolean equations for Bivium, we need symbolic-AND and symbolic-XOR operations on Boolean functions. Symbolic-AND is carried out in algorithm 4.

Following example explains symbolic-AND and the result from algorithm 4 in ANFstuc notation

Example 2. Given two ANF functions $A = 1 \oplus x_1 \oplus x_2$ and $B = 1 \oplus x_2$, the function obtained by performing *symbolic AND* and then by algorithm 4 in ANFstuc form

$$\begin{aligned} A.B &= (1 \oplus x_1 \oplus x_2).(1 \oplus x_2) \\ &= (1 \oplus x_1 \oplus x_2) \oplus (x_2 \oplus x_1x_2 \oplus x_2x_2) \\ &= 1 \oplus x_1 \oplus x_2 \oplus x_2 \oplus x_1x_2 \oplus x_2 \\ &= 1 \oplus x_1 \oplus x_2 \oplus x_1x_2 \end{aligned}$$

In ANFstuc form A and B are represented as

$$\begin{aligned} A &= [1, [[1], [2]]] \\ B &= [1, [[2]]] \end{aligned}$$

Algorithm 4 returns $A.B$ in ANFstuc form as

$$SAND(A, B) = [1, [[1], [2], [1, 2]]]$$

Algorithm 4: Symbolic AND

```
1  function SAND (anfStuc1, anfStuc2)
   Input  : Two ANFstuc type data structures anfStuc1 and anfStuc2
   Output: Single ANFstuc data structure finalAnfStuc
2  begin
3      Initialize: finalAnfStuc ← [∅, [∅]]
4      for i ← 0 to length of anfStuc1[1] do
5          for j ← 0 to length of anfStuc2[1] do
6              tempTerm ← concatenate(anfStuc1[1][i], anfStuc2[1][j])
7              Remove all but one repeating elements of tempTerm
8              Append tempTerm to finalAnfStuc[1]
9      if anfStuc1[0] = 1 then
10         Append all elements of anfStuc2[1] to finalAnfStuc[1]
11     if anfStuc2[0] = 1 then
12         Append all elements of anfStuc1[1] to finalAnfStuc[1]
13     foreach element of finalAnfStuc[1] do
14         count ← number occurrences of element in finalAnfStuc[1]
15         if count  mod 2 = 0 then
16             Remove all occurrences of element from finalAnfStuc[1]
17         else
18             Remove all but one occurrence of element from
                finalAnfStuc[1]
19     finalAnfStuc[0] ← (anfStuc1[0] ∗ anfStuc2[0])  mod 2
20 return finalAnfStuc
```

Algorithm 5: Symbolic XOR

```
1  function SXOR (anfStuc1, anfStuc2)
   Input  : Two ANFstuc type data structures anfStuc1 and anfStuc2
   Output: Single ANFstuc data structure finalAnfStuc
2  begin
3      Initialize: finalAnfStuc ← [∅, [∅]]
4      for i ← 0 to length of anfStuc1[1] do
5          Append anfStuc1[1][i] to finalAnfStuc[1]
6      for i ← 0 to length of anfStuc2[1] do
7          Append anfStuc2[1][i] to finalAnfStuc[1]
8      foreach element of finalAnfStuc[1] do
9          count ← number occurrences of element in finalAnfStuc[1]
10         if count mod 2 = 0 then
11             Remove all occurrences of element from finalAnfStuc[1]
12         else
13             Remove all but one occurrence of element from
               finalAnfStuc[1]
14     finalAnfStuc[0] ← (anfStuc1[0] + anfStuc2[0]) mod 2
15 return finalAnfStuc
```

Next algorithm performs symbolic-XOR on given Boolean functions.

Example 3. Given two ANF functions $A = 1 \oplus x_1 \oplus x_2$ and $B = 1 \oplus x_2$, we perform symbolic-XOR and compare the result from algorithm 5

$$\begin{aligned} A \oplus B &= (1 \oplus x_1 \oplus x_2) \oplus (1 \oplus x_2) \\ &= 1 \oplus x_1 \oplus x_2 \oplus 1 \oplus x_2 \\ &= x_1 \end{aligned}$$

In ANFstuc form A and B are represented as

$$\begin{aligned} A &= [1, [[1], [2]]] \\ B &= [1, [[2]]] \end{aligned}$$

Using algorithm 5 we can get $A \oplus B$ in ANFstuc form as

$$SXOR(A, B) = [0, [[1]]]$$

4.2 Formation of Bivium ANF equations

Generation of the Boolean system model (4) or a pre-computed offline model (5) is an important step in the Cyptanalysis of Bivium in this paper. We generate ANF form of Boolean functions f_i in this model for a given IV in terms of unknown key bits as variables $K = (x_1, x_2, x_3 ... x_{80})$. Although several ways can be devised to generate equations from Boolean operations, this aspect of modeling is strongly dependent on available memory of the computer and has been seldom discussed in detail in Cryptanalysis literature. This step performs symbolic operations in terms of standard data structures of a computer language. For the present case of Bivium we have followed the following two step procedure.

1. Convert the unknown key bits of K as a set of 80 Boolean functions in ANF. Similarly convert the known IV as a set of 80 constant Boolean functions.

2. Use the symbolic Boolean operations of AND and XOR devised above to compute the output stream $z(k)$ as Boolean functions and denote these as f_k. The equations $f_k(K) = z(k)$ are converted to a formula (3) whose factors are $k_k(K) \oplus z(k) \oplus 1$ and are solved by Implicantalgo.

For instance an unknown bit x_i of the key is stored as the Boolean function in ANFstuc form as $[0, [[i]]]$ while a known bit IV_i is stored in ANFstuc

form as the constant function $[IV_i, [[]]]$. Once key and IV are converted to common data structure of ANFstuc the equation formation is carried out by algorithm 6 which is a symbolic representation of the Bivium algorithm 1.

Algorithm 6: ANF Equation Formation

function FormEquations (IV, n)
Input : IV is 80 bit in *ANFstuc* form, n is the number of equations that are to be formed or length of key-stream
Output: n number of equations in *ANFstuc* form
begin
  Initialize Key: $K_i = [0, [[i]]]$ for $i = 1, 2, 3, ...80$
  Initialize States:
  $s_i = [K_1K_2, ..K_{80}, 0, 0, ..., 0, IV_1, IV_2, ..., IV_{80}, 0, 0, ..., 0]$
  Initialize List of Equations: $z \leftarrow [\varnothing]$
  Initialize temporary variables: $t_1 \leftarrow \varnothing; t_2 \leftarrow \varnothing$
  for $i \leftarrow 0$ **to** $i = n$ **do**
    $t_1 \leftarrow SXOR(s_{66}, s_{93})$ and $t_2 \leftarrow SXOR(s_{162}, s_{177})$
    Append $SXOR(t_1, t_2)$ **to** z
    $t_1 \leftarrow SXOR(t_1, SXOR(SAND(s_{91}, s_{92}), s_{171}))$
    $t_2 \leftarrow SXOR(t_2, SXOR(SAND(s_{175}, s_{176}), s_{69}))$
    $(s_1, s_2, ..., s_{93}) \leftarrow (t_2, s_1, ..., s_{92})$
    $(s_{94}, s_{95}, ..., s_{177}) \leftarrow (t_1, s_{94}, ..., s_{176})$
return z

4.3 Cryptanalysis category, formation of equations and solution

We shall report Cryptanalysis of Bivium in category 2 defined in the previous section. This amounts to solution of the system of equations in (4) for a chosen starting time index $k_0 < 4 * 177$ where the functions f_k are computed for a known 80 bits of IV and the unknown variables are key bits. The functions f_k are computed with the help of the algorithm 6 while the output stream $z(k)$ is computed using the standard implementation of the Bivium model as mentioned in [5]. The reason for using category 2 instead of the stronger category 3 is that the present implementation of the algorithm for equation formation in Python is still in its infancy and needs further efforts at overcoming challenges of memory management. At the present level the

implementation is too slow to fully compute the equations for the clocking time index of $4*177$. The practical time chosen for solution of the equations was chosen as 48 hours.

The system of equations formed for an output stream are solved by the algorithm Implicantalgo. In general, algebraic equations of this form need to be available in sufficient numbers to be able to find a unique or at most a small number of solutions. However in the theory of Boolean equations the elimination theory [12, 11] shows that all solutions can be obtained by elimination of variables even when a single equation is available. Hence it is not truly necessary to generate 80 equations to solve 80 Boolean unknowns by Boolean methods. However the number of solutions may be too large if the number of equations are too less. The scalability of the algorithm Implicantalgo is primarily a result of the sparsity of the individual functions in the system (or factors of the product form). This sparsity is more pronounced for the starting index k_0 much smaller than $4 * 177$ to allow the cryptanalysis to succeed in category 2. Hence to achieve the Cryptanalysis of Bivium in categories 3 and 4 further development in implementation of equation formation algorithm are necessary. Furhet such Cryptanalysis may not scale up to the required clocking index size without parallel implementation of the algorithm Implicantalgo. In the next section we present actual computation details with the help of examples.

5 Examples of equation generation and solution for category 2 cryptanalysis

In this section we compute examples to show that the Bivium model (4) can be actually generated and the equations can be solved in practical time using the sequential and unoptimized implementation of the algorithm Implicantalgo under the relaxed condition that the starting time for the output stream for the model is much smaller than the normal time of use which is $4 * 177$. This is classified as category 2 cryptanalysis.

5.1 Example with 161 bits of keystream, starting clock index 80

First the Key and IV are generated as random strings of 80 bits. The output stream $z(k)$ is then generated using standard implementation of Bivium cipher. The Key and IV are chosen as follows while the output stream is $z(80+k)$ for $k = 1, 2, \ldots, 81$. Hence totally 161 bits of output are generated.

The 81 outputs below are used for generating the functions in the model (4).

IV = 10101000001010011110110010101110001111110110010110110000
100110100110110001100100

Key = 01011110000110111000000010011010100000101010010101000
11110001001010000010001001

Output = 10010001110100100101100010001011011101100001000100 1
01000110111010110011110101011 0

The output stream above and the IV are then used as inputs to the equation generation algorithm 6. These equations represented in ANF are as follows.

Function	ANF form
f_1	$1 \oplus x_{68}$
f_2	$1 \oplus x_1x_2 \oplus x_{69}$
f_3	$1 \oplus x_1 \oplus x_2x_3 \oplus x_{70}$
f_4	$x_2 \oplus x_3x_4 \oplus x_{71}$
f_5	$1 \oplus x_3 \oplus x_4x_5 \oplus x_{15} \oplus x_{72}$
f_6	$1 \oplus x_4 \oplus x_5x_6 \oplus x_{16} \oplus x_{73}$
f_7	$x_5 \oplus x_6x_7 \oplus x_{17} \oplus x_{74}$
f_8	$x_6 \oplus x_7x_8 \oplus x_{18} \oplus x_{75}$
f_9	$x_7 \oplus x_8x_9 \oplus x_{19} \oplus x_{76}$
f_{10}	$x_8 \oplus x_9x_{10} \oplus x_{20} \oplus x_{77}$
f_{11}	$1 \oplus x_9 \oplus x_{10}x_{11} \oplus x_{21} \oplus x_{78}$
f_{12}	$1 \oplus x_{10} \oplus x_{11}x_{12} \oplus x_{22} \oplus x_{79}$
f_{13}	$1 \oplus x_{11} \oplus x_{12}x_{13} \oplus x_{23} \oplus x_{80}$
f_{14}	$1 \oplus x_{13}x_{14} \oplus x_{24}$
f_{15}	$x_{14}x_{15} \oplus x_{25}$
f_{16}	$x_{15}x_{16} \oplus x_{26}$
f_{17}	$x_1x_2 \oplus x_{16}x_{17} \oplus x_{27}$
f_{18}	$x_1 \oplus x_2x_3 \oplus x_{17}x_{18} \oplus x_{28}$
f_{19}	$1 \oplus x_2 \oplus x_3x_4 \oplus x_{18}x_{19} \oplus x_{29}$
f_{20}	$x_3 \oplus x_4x_5 \oplus x_{19}x_{20} \oplus x_{30}$
f_{21}	$x_4 \oplus x_5x_6 \oplus x_{20}x_{21} \oplus x_{31}$
f_{22}	$1 \oplus x_5 \oplus x_6x_7 \oplus x_{21}x_{22} \oplus x_{32}$
f_{23}	$1 \oplus x_6 \oplus x_7x_8 \oplus x_{22}x_{23} \oplus x_{33}$
f_{24}	$x_7 \oplus x_8x_9 \oplus x_{23}x_{24} \oplus x_{34}$
f_{25}	$1 \oplus x_8 \oplus x_9x_{10} \oplus x_{24}x_{25} \oplus x_{35}$

$$
\begin{aligned}
f_{26} &\quad 1 \oplus x_9 \oplus x_{10}x_{11} \oplus x_{25}x_{26} \oplus x_{36} \\
f_{27} &\quad 1 \oplus x_{10} \oplus x_{11}x_{12} \oplus x_{26}x_{27} \oplus x_{37} \\
f_{28} &\quad x_{11} \oplus x_{12}x_{13} \oplus x_{27}x_{28} \oplus x_{38} \\
f_{29} &\quad x_{12} \oplus x_{13}x_{14} \oplus x_{28}x_{29} \oplus x_{39} \\
f_{30} &\quad x_{13} \oplus x_{14}x_{15} \oplus x_{29}x_{30} \oplus x_{40} \\
f_{31} &\quad 1 \oplus x_{14} \oplus x_{15}x_{16} \oplus x_{30}x_{31} \oplus x_{41} \\
f_{32} &\quad 1 \oplus x_{15} \oplus x_{16}x_{17} \oplus x_{31}x_{32} \oplus x_{42} \\
f_{33} &\quad 1 \oplus x_{16} \oplus x_{17}x_{18} \oplus x_{32}x_{33} \oplus x_{43} \\
f_{34} &\quad x_{17} \oplus x_{18}x_{19} \oplus x_{33}x_{34} \oplus x_{44} \\
f_{35} &\quad 1 \oplus x_{18} \oplus x_{19}x_{20} \oplus x_{34}x_{35} \oplus x_{45} \\
f_{36} &\quad x_{19} \oplus x_{20}x_{21} \oplus x_{35}x_{36} \oplus x_{46} \\
f_{37} &\quad 1 \oplus x_{20} \oplus x_{21}x_{22} \oplus x_{36}x_{37} \oplus x_{47} \\
f_{38} &\quad x_{21} \oplus x_{22}x_{23} \oplus x_{37}x_{38} \oplus x_{48} \\
f_{39} &\quad 1 \oplus x_{22} \oplus x_{23}x_{24} \oplus x_{38}x_{39} \oplus x_{49} \\
f_{40} &\quad x_{23} \oplus x_{24}x_{25} \oplus x_{39}x_{40} \oplus x_{50} \\
f_{41} &\quad 1 \oplus x_{24} \oplus x_{25}x_{26} \oplus x_{40}x_{41} \oplus x_{51} \\
f_{42} &\quad x_{25} \oplus x_{26}x_{27} \oplus x_{41}x_{42} \oplus x_{52} \\
f_{43} &\quad 1 \oplus x_{26} \oplus x_{27}x_{28} \oplus x_{42}x_{43} \oplus x_{53} \\
f_{44} &\quad 1 \oplus x_{27} \oplus x_{28}x_{29} \oplus x_{43}x_{44} \oplus x_{54} \\
f_{45} &\quad 1 \oplus x_{28} \oplus x_{29}x_{30} \oplus x_{44}x_{45} \oplus x_{55} \\
f_{46} &\quad 1 \oplus x_{29} \oplus x_{30}x_{31} \oplus x_{45}x_{46} \oplus x_{56} \\
f_{47} &\quad x_{30} \oplus x_{31}x_{32} \oplus x_{46}x_{47} \oplus x_{57} \\
f_{48} &\quad x_{31} \oplus x_{32}x_{33} \oplus x_{47}x_{48} \oplus x_{58} \\
f_{49} &\quad 1 \oplus x_{32} \oplus x_{33}x_{34} \oplus x_{48}x_{49} \oplus x_{59} \\
f_{50} &\quad x_{33} \oplus x_{34}x_{35} \oplus x_{49}x_{50} \oplus x_{60} \\
f_{51} &\quad x_{34} \oplus x_{35}x_{36} \oplus x_{50}x_{51} \oplus x_{61} \\
f_{52} &\quad 1 \oplus x_{35} \oplus x_{36}x_{37} \oplus x_{51}x_{52} \oplus x_{62} \\
f_{53} &\quad 1 \oplus x_{36} \oplus x_{37}x_{38} \oplus x_{52}x_{53} \oplus x_{63} \\
f_{54} &\quad x_{37} \oplus x_{38}x_{39} \oplus x_{53}x_{54} \oplus x_{64} \\
f_{55} &\quad x_{38} \oplus x_{39}x_{40} \oplus x_{54}x_{55} \oplus x_{65} \\
f_{56} &\quad 1 \oplus x_{15} \oplus x_{39} \oplus x_{40}x_{41} \oplus x_{55}x_{56} \oplus x_{66} \\
f_{57} &\quad 1 \oplus x_{16} \oplus x_{40} \oplus x_{41}x_{42} \oplus x_{56}x_{57} \oplus x_{67} \\
f_{58} &\quad 1 \oplus x_{17} \oplus x_{41} \oplus x_{42}x_{43} \oplus x_{57}x_{58} \oplus x_{68} \\
f_{59} &\quad 1 \oplus x_{18} \oplus x_{42} \oplus x_{43}x_{44} \oplus x_{58}x_{59} \oplus x_{69} \\
f_{60} &\quad x_{19} \oplus x_{43} \oplus x_{44}x_{45} \oplus x_{59}x_{60} \oplus x_{70} \\
f_{61} &\quad 1 \oplus x_{20} \oplus x_{44} \oplus x_{45}x_{46} \oplus x_{60}x_{61} \oplus x_{71} \\
f_{62} &\quad 1 \oplus x_{21} \oplus x_{45} \oplus x_{46}x_{47} \oplus x_{61}x_{62} \oplus x_{72} \\
f_{63} &\quad 1 \oplus x_{22} \oplus x_{46} \oplus x_{47}x_{48} \oplus x_{62}x_{63} \oplus x_{73} \\
f_{64} &\quad 1 \oplus x_{23} \oplus x_{47} \oplus x_{48}x_{49} \oplus x_{63}x_{64} \oplus x_{74} \\
f_{65} &\quad x_{24} \oplus x_{48} \oplus x_{49}x_{50} \oplus x_{64}x_{65} \oplus x_{75}
\end{aligned}
$$

f_{66} $x_{25} \oplus x_{49} \oplus x_{50}x_{51} \oplus x_{65}x_{66} \oplus x_{76}$

f_{67} $1 \oplus x_{26} \oplus x_{50} \oplus x_{51}x_{52} \oplus x_{66}x_{67} \oplus x_{77}$

f_{68} $x_1x_2 \oplus x_{15} \oplus x_{27} \oplus x_{51} \oplus x_{52}x_{53} \oplus x_{67}x_{68} \oplus x_{78}$

f_{69} $1 \oplus x_1 \oplus x_2x_3 \oplus x_{16} \oplus x_{28} \oplus x_{52} \oplus x_{53}x_{54} \oplus x_{68}x_{69} \oplus x_{79}$

f_{70} $x_2 \oplus x_3x_4 \oplus x_{15}x_{16} \oplus x_{16} \oplus x_{17} \oplus x_{29} \oplus x_{53} \oplus x_{54}x_{55} \oplus x_{69}x_{70} \oplus x_{80}$

f_{71} $x_3 \oplus x_4x_5 \oplus x_{12} \oplus x_{15} \oplus x_{16} \oplus x_{16}x_{17} \oplus x_{18} \oplus x_{30} \oplus x_{54} \oplus x_{55}x_{56} \oplus x_{70}x_{71}$

f_{72} $x_4 \oplus x_5x_6 \oplus x_{13} \oplus x_{16} \oplus x_{17}x_{18} \oplus x_{18} \oplus x_{19} \oplus x_{31} \oplus x_{55} \oplus x_{56}x_{57} \oplus x_{71}x_{72}$

f_{73} $1 \oplus x_5 \oplus x_6x_7 \oplus x_{14} \oplus x_{17} \oplus x_{18}x_{19} \oplus x_{20} \oplus x_{32} \oplus x_{56} \oplus x_{57}x_{58} \oplus x_{72}x_{73}$

f_{74} $1 \oplus x_6 \oplus x_7x_8 \oplus x_{15} \oplus x_{18} \oplus x_{19}x_{20} \oplus x_{21} \oplus x_{33} \oplus x_{57} \oplus x_{58}x_{59} \oplus x_{73}x_{74}$

f_{75} $1 \oplus x_7 \oplus x_8x_9 \oplus x_{16} \oplus x_{19} \oplus x_{20}x_{21} \oplus x_{22} \oplus x_{34} \oplus x_{58} \oplus x_{59}x_{60} \oplus x_{74}x_{75}$

f_{76} $x_8 \oplus x_9x_{10} \oplus x_{17} \oplus x_{20} \oplus x_{21}x_{22} \oplus x_{23} \oplus x_{35} \oplus x_{59} \oplus x_{60}x_{61} \oplus x_{75}x_{76}$

f_{77} $1 \oplus x_9 \oplus x_{10}x_{11} \oplus x_{18} \oplus x_{21} \oplus x_{22} \oplus x_{22}x_{23} \oplus x_{24} \oplus x_{36} \oplus x_{60} \oplus x_{61}x_{62} \oplus x_{76}x_{77}$

f_{78} $1 \oplus x_{10} \oplus x_{11}x_{12} \oplus x_{19} \oplus x_{22} \oplus x_{23}x_{24} \oplus x_{24} \oplus x_{25} \oplus x_{37} \oplus x_{61} \oplus x_{62}x_{63} \oplus x_{77}x_{78}$

f_{79} $x_{11} \oplus x_{12}x_{13} \oplus x_{20} \oplus x_{23} \oplus x_{24} \oplus x_{24}x_{25} \oplus x_{26} \oplus x_{38} \oplus x_{62} \oplus x_{63}x_{64} \oplus x_{78}x_{79}$

f_{80} $1 \oplus x_1x_2 \oplus x_{12} \oplus x_{13}x_{14} \oplus x_{21} \oplus x_{24} \oplus x_{25}x_{26} \oplus x_{26} \oplus x_{27} \oplus x_{39} \oplus$
$x_{63} \oplus x_{64}x_{65} \oplus x_{79}x_{80}$

f_{81} $x_1 \oplus x_1x_2x_{26} \oplus x_2x_3 \oplus x_{12}x_{80} \oplus x_{13} \oplus x_{14}x_{15} \oplus x_{22} \oplus x_{25} \oplus x_{26}x_{27}$
$\oplus x_{28} \oplus x_{40} \oplus x_{64} \oplus x_{65}x_{66}$

These equations are then solved by algorithm Implicantalgo which resulted in two implicant solutions. The solutions covert to the following bit strings

$$\begin{aligned} K_1 &= 01011110000110111000000010011010100000101010010101000 \\ &\quad 11110001001010000010001001 \\ K_2 &= 00111001100000100011101011111011001010010001111100011 \\ &\quad 00001011100011100000110111001 \end{aligned}$$

We can see that the K_1 matches with the Key that we had chosen initially. The time taken for formation of equations was approximately 0.15 sec. while the time required to solve the equations to get two solutions was close to 63 min.

5.2 Example with 165 bits of keystream, starting index 80

This example was chosen of comparable size to the above with a different Key and IV. The 85 bits of output stream $z(80 + k)$ for $k = 1, 2, \ldots, 85$ was used to build the model. The randomly generated Key, IV and the

keystream output are as follows

$$
\begin{array}{lcl}
IV & = & 11011000101100000011010001000000001010000110010011100 \\
 & & 11011011011110001000010110 \\
\text{Key} & = & 10101011000110001101101000011111001010111100101010001 \\
 & & 01000110100111001010001111 0 \\
\text{Output} & = & 00100001011100010101001100100110010100100011010111 11 \\
 & & 01101010100111011100101101000001 0
\end{array}
$$

Following 85 functions of the model (4) were generated by the algorithm 6 which are represented in ANF.

$$
\begin{array}{ll}
f_1 & 1 \oplus x_{68} \\
f_2 & 1 \oplus x_1x_2 \oplus x_{69} \\
f_3 & 1 \oplus x_1 \oplus x_2x_3 \oplus x_{70} \\
f_4 & 1 \oplus x_2 \oplus x_3x_4 \oplus x_{71} \\
f_5 & 1 \oplus x_3 \oplus x_4x_5 \oplus x_{15} \oplus x_{72} \\
f_6 & 1 \oplus x_4 \oplus x_5x_6 \oplus x_{16} \oplus x_{73} \\
f_7 & 1 \oplus x_5 \oplus x_6x_7 \oplus x_{17} \oplus x_{74} \\
f_8 & 1 \oplus x_6 \oplus x_7x_8 \oplus x_{18} \oplus x_{75} \\
f_9 & 1 \oplus x_7 \oplus x_8x_9 \oplus x_{19} \oplus x_{76} \\
f_{10} & x_8 \oplus x_9x_{10} \oplus x_{20} \oplus x_{77} \\
f_{11} & 1 \oplus x_9 \oplus x_{10}x_{11} \oplus x_{21} \oplus x_{78} \\
f_{12} & x_{10} \oplus x_{11}x_{12} \oplus x_{22} \oplus x_{79} \\
f_{13} & 1 \oplus x_{11} \oplus x_{12}x_{13} \oplus x_{23} \oplus x_{80} \\
f_{14} & 1 \oplus x_{13}x_{14} \oplus x_{24} \\
f_{15} & 1 \oplus x_{14}x_{15} \oplus x_{25} \\
f_{16} & 1 \oplus x_{15}x_{16} \oplus x_{26} \\
f_{17} & 1 \oplus x_1x_2 \oplus x_{16}x_{17} \oplus x_{27} \\
f_{18} & x_1 \oplus x_2x_3 \oplus x_{17}x_{18} \oplus x_{28} \\
f_{19} & x_2 \oplus x_3x_4 \oplus x_{18}x_{19} \oplus x_{29} \\
f_{20} & 1 \oplus x_3 \oplus x_4x_5 \oplus x_{19}x_{20} \oplus x_{30} \\
f_{21} & 1 \oplus x_4 \oplus x_5x_6 \oplus x_{20}x_{21} \oplus x_{31} \\
f_{22} & 1 \oplus x_5 \oplus x_6x_7 \oplus x_{21}x_{22} \oplus x_{32} \\
f_{23} & x_6 \oplus x_7x_8 \oplus x_{22}x_{23} \oplus x_{33} \\
f_{24} & x_7 \oplus x_8x_9 \oplus x_{23}x_{24} \oplus x_{34} \\
f_{25} & 1 \oplus x_8 \oplus x_9x_{10} \oplus x_{24}x_{25} \oplus x_{35}
\end{array}
$$

$$
\begin{aligned}
f_{26} &\quad 1 \oplus x_9 \oplus x_{10}x_{11} \oplus x_{25}x_{26} \oplus x_{36} \\
f_{27} &\quad x_{10} \oplus x_{11}x_{12} \oplus x_{26}x_{27} \oplus x_{37} \\
f_{28} &\quad x_{11} \oplus x_{12}x_{13} \oplus x_{27}x_{28} \oplus x_{38} \\
f_{29} &\quad x_{12} \oplus x_{13}x_{14} \oplus x_{28}x_{29} \oplus x_{39} \\
f_{30} &\quad x_{13} \oplus x_{14}x_{15} \oplus x_{29}x_{30} \oplus x_{40} \\
f_{31} &\quad 1 \oplus x_{14} \oplus x_{15}x_{16} \oplus x_{30}x_{31} \oplus x_{41} \\
f_{32} &\quad 1 \oplus x_{15} \oplus x_{16}x_{17} \oplus x_{31}x_{32} \oplus x_{42} \\
f_{33} &\quad x_{16} \oplus x_{17}x_{18} \oplus x_{32}x_{33} \oplus x_{43} \\
f_{34} &\quad x_{17} \oplus x_{18}x_{19} \oplus x_{33}x_{34} \oplus x_{44} \\
f_{35} &\quad 1 \oplus x_{18} \oplus x_{19}x_{20} \oplus x_{34}x_{35} \oplus x_{45} \\
f_{36} &\quad x_{19} \oplus x_{20}x_{21} \oplus x_{35}x_{36} \oplus x_{46} \\
f_{37} &\quad 1 \oplus x_{20} \oplus x_{21}x_{22} \oplus x_{36}x_{37} \oplus x_{47} \\
f_{38} &\quad x_{21} \oplus x_{22}x_{23} \oplus x_{37}x_{38} \oplus x_{48} \\
f_{39} &\quad x_{22} \oplus x_{23}x_{24} \oplus x_{38}x_{39} \oplus x_{49} \\
f_{40} &\quad 1 \oplus x_{23} \oplus x_{24}x_{25} \oplus x_{39}x_{40} \oplus x_{50} \\
f_{41} &\quad x_{24} \oplus x_{25}x_{26} \oplus x_{40}x_{41} \oplus x_{51} \\
f_{42} &\quad x_{25} \oplus x_{26}x_{27} \oplus x_{41}x_{42} \oplus x_{52} \\
f_{43} &\quad x_{26} \oplus x_{27}x_{28} \oplus x_{42}x_{43} \oplus x_{53} \\
f_{44} &\quad x_{27} \oplus x_{28}x_{29} \oplus x_{43}x_{44} \oplus x_{54} \\
f_{45} &\quad x_{28} \oplus x_{29}x_{30} \oplus x_{44}x_{45} \oplus x_{55} \\
f_{46} &\quad 1 \oplus x_{29} \oplus x_{30}x_{31} \oplus x_{45}x_{46} \oplus x_{56} \\
f_{47} &\quad 1 \oplus x_{30} \oplus x_{31}x_{32} \oplus x_{46}x_{47} \oplus x_{57} \\
f_{48} &\quad x_{31} \oplus x_{32}x_{33} \oplus x_{47}x_{48} \oplus x_{58} \\
f_{49} &\quad 1 \oplus x_{32} \oplus x_{33}x_{34} \oplus x_{48}x_{49} \oplus x_{59} \\
f_{50} &\quad x_{33} \oplus x_{34}x_{35} \oplus x_{49}x_{50} \oplus x_{60} \\
f_{51} &\quad 1 \oplus x_{34} \oplus x_{35}x_{36} \oplus x_{50}x_{51} \oplus x_{61} \\
f_{52} &\quad 1 \oplus x_{35} \oplus x_{36}x_{37} \oplus x_{51}x_{52} \oplus x_{62} \\
f_{53} &\quad 1 \oplus x_{36} \oplus x_{37}x_{38} \oplus x_{52}x_{53} \oplus x_{63} \\
f_{54} &\quad x_{37} \oplus x_{38}x_{39} \oplus x_{53}x_{54} \oplus x_{64} \\
f_{55} &\quad 1 \oplus x_{38} \oplus x_{39}x_{40} \oplus x_{54}x_{55} \oplus x_{65} \\
f_{56} &\quad x_{15} \oplus x_{39} \oplus x_{40}x_{41} \oplus x_{55}x_{56} \oplus x_{66} \\
f_{57} &\quad x_{16} \oplus x_{40} \oplus x_{41}x_{42} \oplus x_{56}x_{57} \oplus x_{67} \\
f_{58} &\quad 1 \oplus x_{17} \oplus x_{41} \oplus x_{42}x_{43} \oplus x_{57}x_{58} \oplus x_{68} \\
f_{59} &\quad 1 \oplus x_{18} \oplus x_{42} \oplus x_{43}x_{44} \oplus x_{58}x_{59} \oplus x_{69}
\end{aligned}
$$

$$
\begin{aligned}
f_{60} \quad & 1 \oplus x_{19} \oplus x_{43} \oplus x_{44}x_{45} \oplus x_{59}x_{60} \oplus x_{70} \\
f_{61} \quad & x_{20} \oplus x_{44} \oplus x_{45}x_{46} \oplus x_{60}x_{61} \oplus x_{71} \\
f_{62} \quad & x_{21} \oplus x_{45} \oplus x_{46}x_{47} \oplus x_{61}x_{62} \oplus x_{72} \\
f_{63} \quad & 1 \oplus x_{22} \oplus x_{46} \oplus x_{47}x_{48} \oplus x_{62}x_{63} \oplus x_{73} \\
f_{64} \quad & 1 \oplus x_{23} \oplus x_{47} \oplus x_{48}x_{49} \oplus x_{63}x_{64} \oplus x_{74} \\
f_{65} \quad & 1 \oplus x_{24} \oplus x_{48} \oplus x_{49}x_{50} \oplus x_{64}x_{65} \oplus x_{75} \\
f_{66} \quad & x_{25} \oplus x_{49} \oplus x_{50}x_{51} \oplus x_{65}x_{66} \oplus x_{76} \\
f_{67} \quad & 1 \oplus x_{26} \oplus x_{50} \oplus x_{51}x_{52} \oplus x_{66}x_{67} \oplus x_{77} \\
f_{68} \quad & x_1x_2 \oplus x_{15} \oplus x_{27} \oplus x_{51} \oplus x_{52}x_{53} \oplus x_{67}x_{68} \oplus x_{78} \\
f_{69} \quad & x_1 \oplus x_2x_3 \oplus x_{16} \oplus x_{28} \oplus x_{52} \\
& \oplus x_{53}x_{54} \oplus x_{68}x_{69} \oplus x_{79} \\
f_{70} \quad & x_2 \oplus x_3x_4 \oplus x_{15} \oplus x_{15}x_{16} \oplus x_{17} \oplus x_{29} \\
& \oplus x_{53} \oplus x_{54}x_{55} \oplus x_{69}x_{70} \oplus x_{80} \\
f_{71} \quad & x_3 \oplus x_4x_5 \oplus x_{12} \oplus x_{15} \oplus x_{16} \\
& \oplus x_{16}x_{17} \oplus x_{17} \oplus x_{18} \oplus x_{30} \\
& \oplus x_{54} \oplus x_{55}x_{56} \oplus x_{70}x_{71} \\
f_{72} \quad & x_4 \oplus x_5x_6 \oplus x_{13} \oplus x_{16} \oplus x_{17}x_{18} \\
& \oplus x_{18} \oplus x_{19} \oplus x_{31} \oplus x_{55} \oplus x_{56}x_{57} \oplus x_{71}x_{72} \\
f_{73} \quad & 1 \oplus x_5 \oplus x_6x_7 \oplus x_{14} \oplus x_{17} \oplus x_{18}x_{19} \\
& \oplus x_{20} \oplus x_{32} \oplus x_{56} \oplus x_{57}x_{58} \oplus x_{72}x_{73} \\
f_{74} \quad & x_6 \oplus x_7x_8 \oplus x_{15} \oplus x_{18} \oplus x_{19}x_{20} \\
& \oplus x_{21} \oplus x_{33} \oplus x_{57} \oplus x_{58}x_{59} \oplus x_{73}x_{74} \\
f_{75} \quad & 1 \oplus x_7 \oplus x_8x_9 \oplus x_{16} \oplus x_{19} \\
& \oplus x_{20} \oplus x_{20}x_{21} \oplus x_{22} \oplus x_{34} \oplus x_{58} \\
& \oplus x_{59}x_{60} \oplus x_{74}x_{75} \\
f_{76} \quad & 1 \oplus x_8 \oplus x_9x_{10} \oplus x_{17} \oplus x_{20} \\
& \oplus x_{21}x_{22} \oplus x_{22} \oplus x_{23} \oplus x_{35} \\
& \oplus x_{59} \oplus x_{60}x_{61} \oplus x_{75}x_{76} \\
f_{77} \quad & 1 \oplus x_9 \oplus x_{10}x_{11} \oplus x_{18} \oplus x_{21} \\
& \oplus x_{22} \oplus x_{22}x_{23} \oplus x_{24} \oplus x_{36} \oplus x_{60} \\
& \oplus x_{61}x_{62} \oplus x_{76}x_{77} \\
f_{78} \quad & x_{10} \oplus x_{11}x_{12} \oplus x_{19} \oplus x_{22} \oplus x_{23} \\
& \oplus x_{23}x_{24} \oplus x_{24} \oplus x_{25} \oplus x_{37} \\
& \oplus x_{61} \oplus x_{62}x_{63} \oplus x_{77}x_{78} \\
f_{79} \quad & x_{11} \oplus x_{12}x_{13} \oplus x_{20} \oplus x_{23} \oplus x_{24}x_{25} \\
& \oplus x_{25} \oplus x_{26} \oplus x_{38} \oplus x_{62} \oplus x_{63}x_{64} \oplus x_{78}x_{79} \\
f_{80} \quad & x_1x_2 \oplus x_{12} \oplus x_{13}x_{14} \oplus x_{21} \oplus x_{24} \\
& \oplus x_{25}x_{26} \oplus x_{27} \oplus x_{39} \oplus x_{63} \oplus x_{64}x_{65} \oplus x_{79}x_{80}
\end{aligned}
$$

$$
\begin{aligned}
f_{81} \quad & x_1 \oplus x_1x_2x_{26} \oplus x_2x_3 \oplus x_{12}x_{80} \oplus x_{13} \oplus \\
& x_{14}x_{15} \oplus x_{22} \oplus x_{25} \oplus x_{26}x_{27} \\
& \oplus x_{28} \oplus x_{40} \oplus x_{64} \oplus x_{65}x_{66} \oplus x_{80} \\
f_{82} \quad & x_1x_2 \oplus x_1x_2x_3 \oplus x_1x_2x_{28} \oplus x_1x_{27} \\
& \oplus x_2 \oplus x_2x_3x_{27} \oplus x_3x_4 \oplus x_{12}x_{13} \\
& \oplus x_{13} \oplus x_{14} \oplus x_{15}x_{16} \\
& \oplus x_{23} \oplus x_{26} \oplus x_{27}x_{28} \oplus x_{29} \oplus x_{41} \\
& \oplus x_{65} \oplus x_{66}x_{67} \\
f_{83} \quad & 1 \oplus x_1x_3x_4 \oplus x_1x_{29} \oplus x_2x_3 \oplus x_2x_3x_4 \\
& \oplus x_2x_3x_{29} \oplus x_2x_{28} \oplus x_3 \oplus x_3x_4x_{28} \\
& \oplus x_4x_5 \oplus x_{13}x_{14} \oplus x_{15} \oplus x_{16}x_{17} \\
& \oplus x_{24} \oplus x_{27} \oplus x_{28}x_{29} \oplus x_{30} \\
& \oplus x_{42} \oplus x_{66} \oplus x_{67}x_{68} \\
f_{84} \quad & x_1 \oplus x_2x_4x_5 \oplus x_2x_{30} \oplus x_3x_4 \oplus x_3x_4x_5 \\
& \oplus x_3x_4x_{30} \oplus x_3x_{29} \oplus x_4 \oplus x_4x_5x_{29} \\
& \oplus x_5x_6 \oplus x_{14} \oplus x_{14}x_{15} \oplus x_{16} \oplus x_{17}x_{18} \\
& \oplus x_{25} \oplus x_{28} \oplus x_{29}x_{30} \oplus x_{31} \\
& \oplus x_{43} \oplus x_{67} \oplus x_{68}x_{69} \\
f_{85} \quad & 1 \oplus x_2 \oplus x_3 \oplus x_3x_5x_6 \oplus x_3x_{31} \\
& \oplus x_4x_5x_6 \oplus x_4x_5x_{31} \oplus x_4x_{30} \oplus x_5 \\
& \oplus x_5x_6x_{30} \oplus x_6x_7 \oplus x_{15} \oplus x_{15}x_{16} \\
& \oplus x_{16} \oplus x_{17} \oplus x_{18}x_{19} \oplus x_{26} \oplus x_{29} \\
& \oplus x_{30} \oplus x_{30}x_{31} \oplus x_{32} \oplus x_{44} \oplus x_{68} \oplus x_{69}x_{70}
\end{aligned}
$$

It can be noticed that as the clocking progresses, the terms in the functions start differing in these two examples but often the variable terms are same and the functions only differ by constants. On solving the equations with algorithm Implicantalgo we end up with a single implicant. Converting the implicant to bit string we obtain K_1 which completely matches the Key that was chosen initially. These are obtained as follows

$$
\begin{aligned}
\text{Implicant} \quad = \quad & x_1x_2'x_3x_4'x_5x_6'x_7x_8x_9'x_{10}'x_{11}'x_{12}x_{13}x_{14}'x_{15}' \\
& x_{16}'x_{17}x_{18}x_{19}'x_{20}x_{21}x_{22}'x_{23}x_{24}'x_{25}'x_{26}'x_{27}' \\
& x_{28}x_{29}x_{30}x_{31}x_{32}x_{33}'x_{34}'x_{35}x_{36}'x_{37}x_{38}'x_{39} \\
& x_{40}x_{41}x_{42}x_{43}'x_{44}'x_{45}x_{46}'x_{47}x_{48}'x_{49}x_{50}'x_{51}' \\
& x_{52}'x_{53}x_{54}'x_{55}x_{56}'x_{57}'x_{58}'x_{59}x_{60}x_{61}'x_{62}x_{63}' \\
& x_{64}'x_{65}x_{66}x_{67}x_{68}'x_{69}'x_{70}x_{71}'x_{72}x_{73}'x_{74}'x_{75}' \\
& x_{76}x_{77}x_{78}x_{79}x_{80}' \\
K_1 \quad = \quad & 10101011000110001101101000011111001010111100 10 \\
& 10100010100011010011100101000111 10
\end{aligned}
$$

In the algorithm for solution of these equations we chose the first and then

the second functions as pivots to compute the starting implicants. These functions are f_1 and f_2 and their implicants are

$$\begin{aligned} I(f_1) &= \{x'_{68}\} \\ I(f_2) &= \{x_1x_2x'_{68}x_{69},\quad x_1x'_2x'_{68}x'_{69},\quad x'_1x_2x'_{68}x'_{69},\quad x'_1x'_2x'_{68}x'_{69}\} \end{aligned}$$

Substituting x_{68} in the second set of implicants the threads are defined by the four implicants of $f_2/(x'_{68})$ i.e. with $x_{68} = 0$. The time taken for the formation of the equations was 0.25 sec. Time taken to solve the equations and obtain the results was 179 min (which is about 3 hrs). The second implicant $x_1x'_2x'_{69}$ started the thread that converged to the solution. The number of threads that were spawned for the above implicants were recorded as 1744628, 1609617, 1665674, 1391531 respectively. However all except one thread finally gave a solution. This computation was run using a single core of Intel(R) Xeon(R) CPU E5-1620 v3 @ 3.50GHz processor with 8 GB of RAM.

5.3 Example with 200 bits of keystream, starting index 110

In this example a key stream after 110 bits of clocking was considered. Equations corresponding to last 90 bits of keystream were considered for solving for the key. The key, IV and output stream are as follows

$$\begin{aligned} \text{Key} &= 11101110111001111000101001100111010001101101101010001101111001101100100111110010 \\ IV &= 00011000011101001010100110010111000101010110111001101001010100111011100011001001 \\ \text{Output} &= 10010010010011100100000011000011110100111001101011100101101111100100010100100010010000100001111 \end{aligned}$$

The 90 functions generated for output stream indices $110+k, k = 1, 2, \ldots, 90$ for this IV are far too complex to show here hence we omit the functions. The time taken to solve these the equations to get the key was 2883 min which is close to 48 hrs.

5.4 Updated solving time

Implementing the solver in C++ brings down the time to 3 hrs for solving the equations in Section 5.3

5.5 Time performances on formation of equations

Following time performances for formation for different number of ANF equations for Key and IV chosen as in the example of 165 bit ketstream above were obtained. (The table bellow summarizes the average time taken by a pure python implementation of Algorithm 6 using PyPy 5.1.2, with GCC 5.3.1 compiler. (The average time is computed after taking multiple trials with same parameters).

Number of equations	Avg. time (sec)
165	0.24
200	0.52
300	4.24
350	136.69
355	216.93
360	309.17
362	403.42

For formation of 365 equations however, the code ran for more then 24 hrs. and still did not end up with the equations. This is due to the fact that, formation of equation is highly RAM consuming process. After forming 362 equations the total RAM on the machine (8 GB) is full. The system starts filling up the swap memory. This inherently makes the process run much slower. But after a point the swap is also full and the system becomes non responsive, crashing the system. For this simulation above machine with 14 GB swap memory was used. Finally it was observed in all these Cryptanalysis experiments that while memory usage of the algorithm 6 was very heavy, the CPU usage was very light. On the other hand the memory usage of the solver algorithm Implicantalgo was very light but its CPU usage was very high. Since the equation solver is an inherently parallel algorithm the overall Boolean cryptanalysis can be scaled up well by parallelization.

The solver implementation is done in Python. For optimization we used *PyPy* (https://pypy.org/), a compiler for the python language. This provides minimum of 2x speed up than python interpreter for the solving the equations. The solver is found to be light on RAM usage and heavy on CPU usage.

6 Conclusions

Cryptanalysis of Bivium is shown to be approachable by forming an algebraic model of the algorithm and solving the model equations by a Boolean

system solver which does not utilize a CNF representation as in SAT solvers nor algebraic methods of algebraic cryptanalysis. Assuming a known key stream for clock indices upto 362 instants and known IV, the equation formation is found to be feasible by the sequential implementation of the algorithms proposed in 403 sec. The equations were shown to be solvable by sequential implementation of the Boolean solver algorithm in approximately 48 hrs for last 90 equations of a known key stream of 200 instants. This forms category 2 class cryptanalysis of Bivium. The success of the Boolean solver can also be attributed to sparsity of the equations. The full scale cryptanalysis of category 3 for key stream length greater than $4 * 177$ was not feasible by the current implementation since this implementation results in a memory crash at length 365 key stream for equation formation. Hence cryptanalysis in category 3 is not likely to be feasible without parallel implementation and optimization of implementation for equation formation. It was observed by load measurement that the equation formation algorithm is heavy on memory storage and very light on CPU utilization while the solver algorithm is very light on memory usage and heavy on CPU usage. The solver algorithm has a potential to scale up for solving more complex systems for category 3 cryptanalysis by parallel implementation. Similarly the equation formation algorithm can be improved by distributing the memory and parallel computation. The algebraic model based cryptanalysis requires symbolic computation involving formal algebraic and Boolean operations on unknowns. In literature, need for symbolic computation has not been considered a serious hurdle for cryptanalysis but is certainly a practical issue which needs to be resolved. Hence symbolic model building of ciphers and solutions of Boolean equations are serious topics of further research in cryptanalysis. Results of this paper are important at least from two points of view. One, that the Boolean systems shown to be solvable successfully by the solver are sufficiently more complex as compared to those reported by other algebraic solvers. Second, these systems are not in the CNF form as in other SAT solvers. Moreover the solver is able to compute all solutions in contrast to SAT solvers.

First author acknowledges support for this work from the project
15DITIR009 of the center NCETIS at IIT Bombay, India

References

[1] Gregory Bard. Algebraic cryptanalysis. Springer 2009.

[2] Dudek P., Kurkowski M., Srebrny M. (2012) Towards Parallel Direct SAT-Based Cryptanalysis. In: Wyrzykowski R., Dongarra J., Karczewski K., Waśniewski J. (eds) Parallel Processing and Applied Mathematics. PPAM 2011. Lecture Notes in Computer Science, vol 7203. Springer, Berlin, Heidelberg

[3] Tobias Eibach , Enrico Pilz , Gunnar Völkel, Attacking Bivium using SAT solvers, Proceedings of the 11th international conference on Theory and applications of satisfiability testing, p.63-76, May 12-15, 2008, Guangzhou, China

[4] Virendra Sule. Implicant based parallel all solution solver for Boolean satisfiability. arXiv.org/cs.DS/1611.09590v3, 6 Feb 2017.

[5] Yun Tian, Gongliang Chen, Jianhua Li: On the Design of Trivium. IACR Cryptology ePrint Archive 2009.

[6] McDonald, C., Charnes, C., Pieprzyk, J.: Attacking Bivium with Minisat. Technical Report 2007/040, ECRYPT Stream Cipher Project 2007

[7] Kenneth Koon-Ho Wong, Gregory V. Bard. Improved Algebraic Cryptanalysis of QUAD, Bivium and Trivium via Graph Partitioning on Equation Systems, Information Security and Privacy: 15th Australasian Conference, ACISP 2010

[8] McDonald, C., Charnes, C., Pieprzyk, J.: An Algebraic Analysis of Trivium Ciphers based on the Boolean Satisfiability Problem, https://eprint.iacr.org/2007/129.pdf

[9] H. Raddum: Cryptanalytic results on TRIVIUM. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/039, 2006. http://www.ecrypt.eu.org/stream

[10] C. De Cannière, B. Preneel: TRIVIUM – a stream cipher construction inspired by block cipher design principles. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/030, 2005. http://www.ecrypt.eu.org/stream/trivium.html

[11] F. M. Brown. Boolean reasoning. The logic of Boolean equations. Dover, 2006.

[12] Sergiu Rudeanu. Boolean functions and equations. North Holland, Amsterdam, 1974.

[13] Youssef Hammadi and C. M. Wintersteiger. Seven challenges in parallel SAT solving. Challenge paper AAAI 2012 Sub-Area spotlights track. Association of Advancement of Artificial Intelligence.