Solving Compressed Right Hand Side Equation Systems with Linear Absorption

Thorsten Ernst Schilling and Håvard Raddum

Selmer Center, University of Bergen
{thorsten.schilling,havard.raddum}@ii.uib.no

Abstract. In this paper we describe an approach for solving complex multivariate equation systems related to algebraic cryptanalysis. The work uses the newly introduced Compressed Right Hand Sides (CRHS) representation, where equations are represented using Binary Decision Diagrams (BDD). The paper introduces a new technique for manipulating a BDD, similar to swapping variables in the well-known sifting-method. Using this technique we develop a new solving method for CRHS equation systems. The new algorithm is successfully tested on systems representing reduced variants of Trivium.

Keywords: multivariate equation system, BDD, algebraic cryptanalysis, Trivium.

1 Introduction

Keystream generators produce pseudo-random sequences to be used in stream ciphers. A strong keystream generator must produce the sequence from a secret internal state such that it is very difficult to recover this initial state from the keystream. The security of a stream cipher corresponds to the complexity of finding the internal state that corresponds to some known keystream.

The relation between the keystream sequence and the internal state of the generator can be described as a system of algebraic equations. The variables in the system are the unknown bits of the internal state (at some time), and possibly some auxilliary variables. Solving the equation system will reveal the internal state of the generator, and hence break the associated stream cipher. Solving equation systems representing cryptographic primitives is known as algebraic cryptanalysis, and is an active research field.

This paper explores one approach for efficiently solving big equation systems, and is based on the work in [1], where the concept of Compressed Right Hand Side (CRHS) equations was introduced. A CRHS equation is a Binary Decision Diagram (BDD) together with a matrix with linear combinations of the variables in the system as rows. The problem of solving CRHS equation systems comes mainly from linear dependencies in the matrices associated with the BDD's. In this paper we introduce a new method for handling linear dependencies in CRHS equations, which we call *linear absorption*. The basis for linear absorption are two methods for manipulating BDD's. One of them is the technique of swapping

T. Helleseth and J. Jedwab (Eds.): SETA 2012, LNCS 7280, pp. 291–302, 2012.

variables in the well-known sifting method [2]. The other is similar, but, to the best of our knowledge, not described in literature earlier. We call it *variable XOR*.

We have tested the method of linear absorption on systems representing scaled versions of Trivium [3]. We are able to break small versions of Trivium using linear absorption, proving that the method works. From these tests we derive an early estimate for the complexity of breaking the full Trivium using linear absorption. Our results indicate that the complexity of solving systems representing scaled Triviums increases with a factor $2^{0.4}$ each time the size of the solution space doubles.

2 Preliminaries

2.1 Binary Decision Diagrams

A Binary Decision Diagram (BDD) [4, 5] is a directed acyclic graph. BDDs were initially mostly used in design and verification systems. Later implementations and refinement led to a broader interest in BDDs and they were successfully applied in the cryptanalysis of LFSRs [6] and the cipher Grain [7]. For our purposes, we think of a BDD in the following way, more thoroughly described in [1].

A BDD is drawn from top to bottom, with all edges going downwards. There is exactly one node on top, with no incoming edges. There are exactly two nodes at the bottom, labelled $\top$ and $\bot$, with no outgoing edges. Except for $\top$ and $\bot$ each node has exactly two outgoing edges, called the 0-edge and the 1-edge. Each node (except for $\top$ and $\bot$) is associated to a variable. There are no edges between nodes associated to the same variable, which are said to be at the same *level*. An order is imposed on the variables. The node associated to the first variable is drawn on top, and the nodes associated to the last variable are drawn right above $\top$ and $\bot$. Several examples of BDDs are found in the following pages.

A path from the top node to either $\top$ or $\bot$ defines a vector on the variables. If node F is part of the path and is associated to variable x, then x is assigned 0 if the 0-edge is chosen out from F, and x is assigned 1 if the 1-edge is part of the path. A path ending in $\top$ is called an *accepted* input to the BDD.

There is a polynomial-time algorithm for reducing the number of nodes in a BDD, without changing the underlying function. It has been proven that a reduced BDD representing some function is unique up to variable ordering. In literature this is often referred to as a *reduced, ordered* BDD, but in this work we always assume BDDs are reduced, and that a call to the reduction algorithm is done whenever necessary.

2.2 Compressed Right Hand Side Equations

In [1] the concept of the Compressed Right Hand Side Equations was introduced. CRHS equations give a method for representing large non-linear constraints

along with algorithms for manipulating their solution spaces. In comparison to previous methods from the same family of algorithms [8–10] they offer an efficient way of joining equations with a very large number of solutions.

CRHS equations are a combination of the two different approaches *Multiple Right Hand Side Equations* [9] (MRHS equations) and BDDs. While MRHS equations were initially developed for cryptanalysis, BDDs were developed for other purposes. Combining the two provides us with a powerful tool for algebraic cryptanalysis. For instance, using CRHS equations it is possible to create a single large BDD representing the equation system given by the stream cipher TRIVIUM.

Definition 1 (CRHS Equation [1]). *A compressed right hand side equation is written as $Ax = \mathcal{D}$, where A is a binary $k \times n$-matrix with rows $l_0, \ldots, l_{k-1}$ and $\mathcal{D}$ is a BDD with variable ordering (from top to bottom) $l_0, \ldots, l_{k-1}$. Any assignment to x such that Ax is a vector corresponding to an accepted input in $\mathcal{D}$, is a satisfying assignment. If C is a CRHS equation then the number of vertices in the BDD of C, excluding terminal vertices, is denoted $\mathcal{B}(C)$.*

Example 1 (CRHS Equation). In order to write:

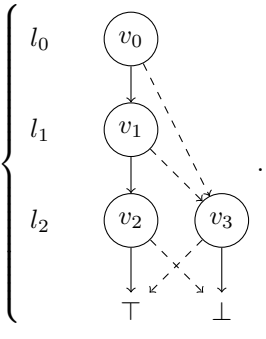
$$f(x_1, \ldots, x_6) = x_1x_2 + x_3 + x_4 + x_5 + x_6 = 0$$

as a CRHS equation one chooses a name for every linear component in $f(x_1, \ldots, x_6) = 0$. Here we decide to name the linear components $l_0 = x_1, l_1 = x_2, l_2 = x_3 + x_4 + x_5 + x_6$. Furthermore one needs to define an ordering on these linear components. For this example we we select the order l_0, l_1, l_2, from top to bottom.

The matrix A formed by the linear components is then our left hand side of the CRHS equation. The BDD formed by the possible values of l_0, l_1, l_2 in $f(x_1, \ldots, x_6) = 0$ together with the before defined order forms the right hand side of the CRHS equation.

The resulting CRHS equation is then:

$$\begin{bmatrix} x_1 & = l_0 \\ x_2 & = l_1 \\ x_3 + x_4 + x_5 + x_6 & = l_2 \end{bmatrix} = \left\{ \quad . \right. \tag{1}$$



The right hand side of the CRHS equation represents the possible values of l_0, l_1, l_2 in $f(x_1, \ldots, x_6) = 0$ in *compressed* form. The set of solutions of (1) is the union of all solutions of $Ax = L$, where L is a vector contained in the right hand side as an accepted input to the BDD. Naming equation (1) as E_0, we have $\mathcal{B}(E_0) = 4$.

2.3 Joining CRHS Equations

Given two CRHS equations A and B it is natural to ask: *What are the common solutions to A and B?*

In [1] an algorithm, called *CRHS Gluing* is introduced. The algorithm takes as input two CRHS equations and has as output a new CRHS equation which contains the solutions of the conjunction of the input. This algorithm is exponential in space and time consumption. Nevertheless, the constant of this exponential has been shown to be small enough for practical applications.

Here, we use a simpler and cheaper method of joining two CRHS equations. Given two BDDs $\mathcal{D}_1$ and $\mathcal{D}_2$, the notation $(\mathcal{D}_1 \to \mathcal{D}_2)$ is defined to simply mean that $\top$ in $\mathcal{D}_1$ is replaced with the top node in $\mathcal{D}_2$. The two $\bot$-nodes from $\mathcal{D}_1$ and $\mathcal{D}_2$ are merged into one $\bot$, and the resulting structure is a valid BDD.

Given the two CRHS equations $[L_1]x = \mathcal{D}_1$ and $[L_2]x = \mathcal{D}_2$ the result of joining them is

$$\begin{bmatrix} L_1 \\ L_2 \end{bmatrix} x = (\mathcal{D}_1 \to \mathcal{D}_2)$$

Any accepted path in $(\mathcal{D}_1 \to \mathcal{D}_2)$ gives accepted paths in both $\mathcal{D}_1$ and $\mathcal{D}_2$. In other words, any x such that $\begin{bmatrix} L_1 \\ L_2 \end{bmatrix} x$ yields an accepted path in $(\mathcal{D}_1 \to \mathcal{D}_2)$ gives solutions to the two initial CRHS equations.

When there are linear dependencies among the rows in $\begin{bmatrix} L_1 \\ L_2 \end{bmatrix}$ we get paths in $(\mathcal{D}_1 \to \mathcal{D}_2)$ that lead to false solutions. The problem of false solutions is the only problem preventing us from having an efficient solver for CRHS equation systems. This problem is addressed in Section 3.3.

Example 2 (Joining CRHS equations). The following two equations are similar to equations in a Trivium equation system. In fact, the right hand sides of the following are taken from a full scale Trivium equation system. The left hand matrices have been shortened.

$$\begin{bmatrix} x_1 & = l_0 \\ x_2 & = l_1 \\ x_3 + x_4 & = l_2 \end{bmatrix} = \left\{ \begin{array}{ll} l_0 & \\ l_1 & \\ l_2 & \end{array} \right. \quad \begin{array}{c} u_0 \\ u_1 \\ u_2 \quad u_3 \\ \top \qquad \bot \end{array} \quad , \quad \begin{bmatrix} x_4 & = l_3 \\ x_5 & = l_4 \\ x_6 + x_7 & = l_5 \end{bmatrix} = \left\{ \begin{array}{ll} l_3 & \\ l_4 & \\ l_5 & \end{array} \right. \quad \begin{array}{c} v_0 \\ v_1 \\ v_2 \quad v_3 \\ \top \qquad \bot \end{array} \tag{2}$$

The joining of the equations above is

$$\begin{bmatrix} x_1 = l_0 \\ x_2 = l_1 \\ x_3 + x_4 = l_2 \\ x_4 = l_3 \\ x_5 = l_4 \\ x_6 + x_7 = l_5 \end{bmatrix} = \left\{ \begin{array}{ll} l_0 & w_0 \\ l_1 & w_1 \\ l_2 & w_2 \quad w_3 \\ l_3 & w_4 \\ l_4 & w_5 \\ l_5 & w_6 \quad w_7 \\ & \top \end{array} \right. , \tag{3}$$

where $\bot$-paths in this last graph are omitted for better readability. The resulting equation has 8 nodes, where the corredsponding MRHS equation would have 16 right hand sides.

Joining two CRHS equations E_0 and E_1 is really nothing more than putting one on top of the other and connect them. If E_0 and E_1 are joined to form E, it is easy to see that $\mathcal{B}(E) = \mathcal{B}(E_0) + \mathcal{B}(E_1)$. The complexity of joining CRHS equations is linear, and we can easily build a single CRHS equation representing, for instance, the full Trivium. The CRHS equation representing the full Trivium will have less than 3000 nodes, but 2^{1336} paths in the long BDD, of which maybe only one is not a false solution.

3 Solving Large CRHS Equation Systems

After joining several CRHS equations together the left hand side of the resulting equation may contain linear dependencies which are not reflected in the right hand side BDD. The matrix of the CRHS equation contains rows which sum to 0. The BDD on the other hand is oblivious to this fact and contains paths which sum to 1 on the affected variables.

Since the set of solutions of the CRHS equation is the union of solutions to the individual linear systems formed by each vector of the right hand side, we need to filter out those vectors which yield an inconsistent linear system. Let for example the left hand side of a CRHS equation contain the linear combinations l_i, l_j and l_k and assume we found that $l_i + l_j + l_k = 0$. The BDD might nevertheless contain a path which assigns l_i, l_k and l_k to values that make their sum equal to 1.

Since we know that this path in the BDD does not correspond to a solution we would like to eliminate it from the BDD.

In practical examples from the cryptanalysis of Trivium we end up with the situation that almost all paths on the right hand side are of this kind, i.e., not corresponding to the left hand side. The major problem is that we cannot easily *delete a path* by some simple operation, e.g., deleting a node. This is because there are many paths passing through a single node.

In order to delete all invalid solutions from a CRHS equation, we introduce the techniques *Variable XOR* and *Linear Absorption* in the following. They are new methods for the manipulation of BDDs and can be used to take care of removing paths which correspond to false solutions.

3.1 Variable Swap

A usual operation on a BDD is to swap variable levels [2] while preserving the function the BDD represents. This means to change the permutation of variables in a BDD by exchanging adjacent positions of two variables. This is done for example to change the size of a specific BDD. We will use this technique in the following and give a short introduction to it.

The origins of the BDD data structure lie within the *Shannon Expansion* [11]. In the following let be $F = f(x_0, \ldots, x_{n-1})$, $F_{x_r} = f(x_0, \ldots, x_{r-1}, 1, x_{r+1}, \ldots, x_{n-1})$ and $F_{\overline{x_r}} = f(x_0, \ldots, x_{r-1}, 0, x_{r+1}, \ldots, x_{n-1})$. Then by the Shannon expansion every Boolean function can be represented in the form

$$F = x \cdot F_x + \overline{x} \cdot F_{\overline{x}}. \tag{4}$$

We write the function as a BDD with the root node denoted $F = (x, F_x, F_{\overline{x}})$. Here x is the variable defining the level of the node, F_x is the node connected through the 1-edge and $F_{\overline{x}}$ is the node connected to the 0-edge. F_x and $F_{\overline{x}}$ are called the co-factors of the node F.

Let the variable coming after x in the variable order be y. To expand (4) by the variable y, we have to expand the subfunctions F_x and $F_{\overline{x}}$ accordingly:

$$F = x \cdot (y \cdot F_{xy} + \overline{y} \cdot F_{x\overline{y}}) + \overline{x} \cdot (y \cdot F_{\overline{x}y} + \overline{y} \cdot F_{\overline{x}\overline{y}}). \tag{5}$$

Again, as a root node of a BDD we have $F = (x, (y, F_{xy}, F_{x\overline{y}}), (y, F_{\overline{x}y}, F_{\overline{x}\overline{y}}))$ but this time with explicitly written co-factors. Assume we would like to swap the order of x and y. Then we can equivalently write (5) as

$$F' = y \cdot (x \cdot F_{xy} + \overline{x} \cdot F_{\overline{x}y}) + \overline{y} \cdot (x \cdot F_{x\overline{y}} + \overline{x} \cdot F_{\overline{x}\overline{y}}) \tag{6}$$

which leads us to the new node representation of $F' = (y, (x, F_{xy}, F_{\overline{x}y}), (x, F_{x\overline{y}}, F_{\overline{x}\overline{y}}))$. Now the order of the variables x and y is swapped. Since (5) and (6) are equivalent so are our BDD nodes before and after the swap.

Moreover, it becomes clear that swapping two variables is a local operation, in the sense that only nodes at levels x and y are affected. If one would like to swap the levels x and y (where as above x is before y in the BDD permutation) one has to apply the operation above to every node at level x and change it accordingly.

Example 3 (Variable Swap).

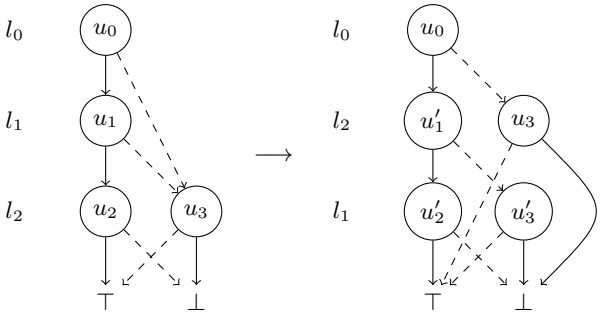


Fig. 1. Swapping l_1 and l_2

On the left side in Fig. 1 a BDD along with its permutation (l_0, l_1, l_2) is depicted. In order to swap levels l_1 and l_2, i.e., change the permutation to (l_0, l_2, l_1), one has to apply the swapping routine described above to all nodes at level l_1. In this case $u_1 = (l_1, u_2, u_3)$ is the only node affected. With explicitly written co-factors we get $u_1 = (l_1, (l_2, \top, \bot), (l_2, \bot, \top))$. From the swapping procedure above we know that the resulting new node is $u_1' = (l_2, (l_1, \top, \bot), (l_1, \bot, \top)) = (l_2, u_2', u_3')$. Node u_3 stays unchanged.

3.2 Variable XOR

In this section we introduce a new method for manipulating BDDs, the *variable XOR* operation. As the name suggests, we change a variable by XORing a different variable onto it. To preserve the original function we have to change the BDD accordingly. Below we explain how this is done. In fact, the procedure is quite similar to Variable Swap, and is only a local operation.

Let x and y be two consecutive BDD variables (x before y) and $\sigma = x + y$. We want to transform (5) into:

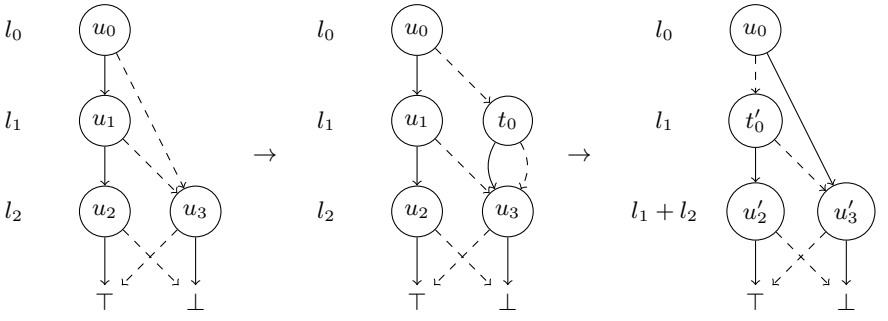
$$F' = x \cdot (\sigma \cdot F_{x\sigma} + \overline{\sigma} \cdot F_{x\overline{\sigma}}) + \overline{x} \cdot (\sigma \cdot F_{\overline{x}\sigma} + \overline{\sigma} \cdot F_{\overline{x}\overline{\sigma}}). \tag{7}$$

We can see that if $x = 1$ then $F_{x\sigma} = F_{x\overline{y}}$ and $F_{x\overline{\sigma}} = F_{xy}$. Similarly if $x = 0$ then $F_{\overline{x}\sigma} = F_{\overline{x}y}$ and $F_{\overline{x}\overline{\sigma}} = F_{\overline{x}\overline{y}}$. With that in mind (7) can be written as

$$F' = x \cdot (\sigma \cdot F_{x\overline{y}} + \overline{\sigma} \cdot F_{xy}) + \overline{x} \cdot (\sigma \cdot F_{\overline{x}y} + \overline{\sigma} \cdot F_{\overline{x}\overline{y}}) \tag{8}$$

which leads immediately to the new node representation
$F' = (x, (\sigma, F_{x\overline{y}}, F_{xy}), (\sigma, F_{\overline{x}y}, F_{\overline{x}\overline{y}}))$. With this manipulation extra care has to be taken of edges incoming to nodes at the y-level that *jumps* over the x-level. Here temporary nodes have to be introduced since y goes over into σ and cannot longer be *addressed* directly.

Example 4 (Variable XOR).



The first diagram shows the initial BDD in which the variable levels l_1 and l_2 are to be XORed. The second diagram represents how the auxilliary node t_0 needs to be introduced since the edge (u_0, u_3) ignores the l_1 level. Then the variable XOR procedure is applied to both u_1 and t_0, and the resulting BDD is reduced. After the application of the modification of equation (5) to (7) the result of the variable XOR method to variables l_1 and l_2 of the initial diagram is depicted.

3.3 Linear Absorption

We are now ready to explain the method of linear absorption.

Assume we have a BDD with $(l_0, \ldots, l_{k-1})$ as the ordered set of linear combinations associated with the levels. We can easily find all linear dependencies among the l_i's. Assume that we have found the dependency $l_{i_1} + l_{i_2} + \ldots + l_{i_r} = 0$, where $i_1 < i_2 < \ldots < i_r$.

By using variable swap repeatedly, we can move the linear combination l_{i_1} down to the level just above l_{i_2}. Then we use variable XOR to replace l_{i_2} with $l_{i_1} + l_{i_2}$. Next, we use varlable swap again to move $l_{i_1} + l_{i_2}$ down to the level just above l_{i_3}, and variable XOR to replace l_{i_3} with $l_{i_1} + l_{i_2} + l_{i_3}$. We continue in this way, picking up each l_{i_j} that is part of the linear dependency, until we replace l_{i_r} with $l_{i_1} + l_{i_2} + \ldots + l_{i_r}$. Let us call the level of nodes associated with $l_{i_1} + l_{i_2} + \ldots + l_{i_r}$ for the zero-level.

We know now that the zero-level has the 0-vector associated with it. This implies that any path in the BDD consistent with the linear constraint we started with has to select a 0-edge out of a node on the zero-level. In other words, all 1-edges going out from this level lead to paths that are inconsistent with the linear constraint $l_{i_1} + l_{i_2} + \ldots + l_{i_r} = 0$, and can be deleted.

After deleting all outgoing 1-edges, there is no longer any choice to be made for any path going out from a node at the zero-level. If F is a node at the zero-level, any incoming edge to F can go directly to F_0, jumping the zero-level altogether. After all incoming edges have been diverted to jump the zero-level,

all nodes there can be deleted, and the number of levels in the BDD decreases by one. We are now certain that any path in the remaining BDD will never be in conflict with the constraint $l_{i_1} + l_{i_2} + \ldots + l_{i_r} = 0$; we say that the linear constraint has been absorbed.

We can repeat the whole process, and absorb one linear constraint at the time, until all remaining l_i are linearly independent. At that point, any remaining path in the BDD will yield a valid solution to the initial equation system.

4 Experimental Results

We have tested Linear Absorption on equation systems representing scaled versions of Trivium.

4.1 Trivium and Trivium-*N*

Trivium is a synchronous stream cipher and part of the ECRYPT Stream Cipher Project portfolio for hardware stream ciphers. It consists of three connected nonlinear feedback shift registers (NLFSR) of lengths 93, 84 and 111. These are all clocked once for each keystream bit produced.

Trivium has an inner state of 288 bits, which are initialized with 80 key bits, 80 bits of IV, and 128 constant bits. The cipher is clocked 1152 times before actual keystream generation starts. The generation of keystream bits and updating the registers is very simple. For algebraic cryptanalysis purposes one can create four equations for every clock; three defining the inner state change of the registers and one relating the inner state to the keystream bit. Solving this equation system in time less than trying all 2^{80} keys is considered a valid attack on the cipher.

Small Scale Trivium. In [1] a reduced version of Trivium, called Trivium-N was introduced. N is an integer value which defines the size of the inner state of that particular version of Trivium. Trivium-288 is by our construction equivalent to the originally proposed Trivium.

All versions of Trivium-N with $N < 288$ try to preserve the structure of the original Trivium as well as possible. This yields equation systems which are comparable to the full cipher. Other small scale version of Trivium e.g., Bivium [12], in which an entire NLFSR was removed, seems to be too easy to solve.

4.2 Results

We have constructed CRHS equation systems representing Trivium-N for several values of N, and run the algorithm for absorbing linear constraints described in Section 3.3. For $N \leq 41$ we were able to absorb all linear constraints, which means that any remaining path in the BDD is a valid solution to the system (we have also verified this).

The number of nodes in the BDD grows very slowly when absorbing the first linear constraints, but increases more rapidly when the linear constraints of length two have been absorbed. We know, however, that the number of paths will be very small once all linear constraints have been absorbed since we expect a unique, or very few, solution(s). Thus the number of nodes must also decrease quickly after the number of absorbed constraints is past some tipping point. For each instance we have recorded the maximum number of nodes the BDD contained during execution, and used this number as our measure of complexity. The memory consumtion is dominated by the number of nodes, and in our implementation each node took 60 bytes. The memory requirement in bytes can then be found approxiamtely by multiplying the number of nodes with 60.

The results for testing the algorithm on Trivium-N for $30 \leq N \leq 41$ is written below.

N	max. # of nodes
30	$2^{19.92}$
31	$2^{21.02}$
32	$2^{21.15}$
33	$2^{20.84}$
34	$2^{21.41}$
35	$2^{22.32}$
36	$2^{21.61}$
37	$2^{23.27}$
38	$2^{23.49}$
39	$2^{23.79}$
40	$2^{23.69}$
41	$2^{24.91}$

The number of solutions (paths) in each instance was found to be between 1 and 3. The number of levels in the final BDD was 73 for $N = 30$, and 97 for $N = 41$.

The numbers above have been produced using only a single test for each N. We can expect some variation in the maximum number of nodes when re-doing tests using different initial states for some particular Trivium-N. The numbers are plotted in Fig. 2 to show the general trend in the increase of complexity.

4.3 Extrapolating

We can use the least-square method to fit a linear function to the data points we have. Letting 2^M be the maximum number of nodes needed, the linear function that best approximates our data is $M = 0.4N + 7.95$.

When N increases by 1, the size of the solution space for the variables in the initial state doubles. However, the total number of variables in the system increases by three when N increases by 1. This is because we need to clock the cipher one step further to have enough known keystream for a unique solution, and each clock introduces three new variables. Hence we can say that the size

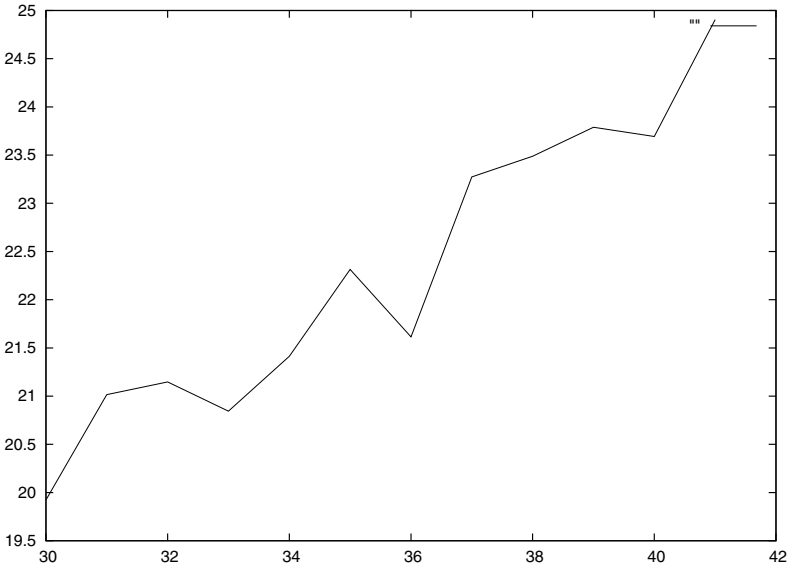


Fig. 2. Trend of complexities for Trivium-N

of the problem instance increases by a factor 2^3 for each increase in N. The complexity of our solving method only increases with a factor of approximately $2^{0.4}$ on the tested instances, which we think is quite promising.

Admittedly, we have too little data to draw any clear conclusions, but it is still interesting to see what value of M we get for $N = 288$. Based on the data we have, we find that currently we need to be able to handle around 2^{123} nodes in a BDD for successfully attacking the full Trivium.

5 Conclusions and Future Work

We have introduced how to alter a BDD to preserve the underlying function when two variables are XORed. Together with variable swap, we have introduced a new solving method in algebraic cryptanalysis, which we call linear absorption. The solving technique works on equations represented in CRHS form.

The work in this paper gives more insight into how to solve some of the open questions in [1], and provides a complete solving method. We have shown how the method works on systems representing scaled versions of Trivium. The structure of the equations is exactly the same in the down-scaled and the full versions of Trivium, it is only the number of equations and variables that differ. Our tests thus gives some information on the complexity of a successful algebraic attack on the full Trivium.

Unfortunately, we have not had the time to test linear absorption on other ciphers, or test more extensively on Trivium-N. This is obviously a topic for further research. We also hope to further investigate the problem of how to find a path in a BDD that satisfies a set of linear constraints. There may be tweaks to the algorithm of linear absorption, or there may be a completely different and better method. In any case, we hope to see more results on solving methods for CRHS equation systems.

References

1. Schilling, T.E., Raddum, H.: Analysis of trivium using compressed right hand side equations. In: 14th International Conference on Information Security and Cryptology, Seoul, Korea, November 30-December 2. LNCS (2011)
2. Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. In: Proceedings of the 1993 IEEE/ACM International Conference on Computer-aided Design, vol. 12, pp. 42–47 (1993)
3. Cannière, C.D., Preneel, B.: Trivium specifications. ECRYPT Stream Cipher Project (2005)
4. Akers, S.B.: Binary decision diagrams. IEEE Transactions on Computers 27(6), 509–516 (1978)
5. Somenzi, F.: Binary decision diagrams. In: Calculational System Design. NATO Science Series F: Computer and Systems Sciences, vol. 173, pp. 303–366. IOS Press (1999)
6. Krause, M.: BDD-Based Cryptanalysis of Keystream Generators. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 222–237. Springer, Heidelberg (2002)
7. Stegemann, D.: Extended BDD-Based Cryptanalysis of Keystream Generators. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 17–35. Springer, Heidelberg (2007)
8. Raddum, H.: MRHS Equation Systems. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 232–245. Springer, Heidelberg (2007)
9. Raddum, H., Semaev, I.: Solving multiple right hand sides linear equations. Designs, Codes and Cryptography 49(1), 147–160 (2008)
10. Schilling, T.E., Raddum, H.: Solving Equation Systems by Agreeing and Learning. In: Hasan, M.A., Helleseth, T. (eds.) WAIFI 2010. LNCS, vol. 6087, pp. 151–165. Springer, Heidelberg (2010)
11. Shannon, C.E.: The synthesis of two-terminal switching circuits. Bell Systems Technical Journal 28, 59–98 (1949)
12. McDonald, C., Charnes, C., Pieprzyk, J.: Attacking Bivium with MiniSat. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/040 (2007), `http://www.ecrypt.eu.org/stream`