Solving Compressed Right Hand Side Equation Systems with Linear Absorption

Thorsten Ernst Schilling, Håvard Raddum
thorsten.schilling@ii.uib.no,havard.raddum@ii.uib.no

Selmer Center, University of Bergen

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.

Key words: 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 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:

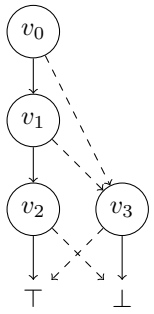
$$f(x_1, \ldots, x_6) = x_1x_2 + x_3 + x_4 + x_5 + x_6 = 0$$

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

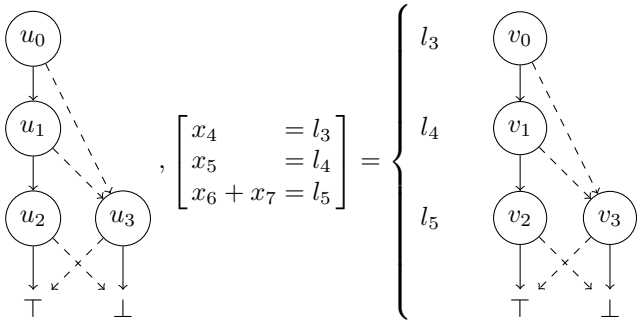
$$\begin{bmatrix} L_1 \\ L_2 \end{bmatrix} x = (\mathcal{D}_1 \to \mathcal{D}_2)$$

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.

$$\begin{bmatrix} x_1 & = l_0 \\ x_2 & = l_1 \\ x_3 + x_4 & = l_2 \end{bmatrix} = \left\{ \begin{array}{ll} l_0 & \\ l_1 & \\ l_2 & \end{array} \right. \quad , \quad \begin{bmatrix} x_4 & = l_3 \\ x_5 & = l_4 \\ x_6 + x_7 & = l_5 \end{bmatrix} = \left\{ \begin{array}{ll} l_3 & \\ l_4 & \\ l_5 & \end{array} \right.$$



(2)

The joining of the equations above is

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

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).

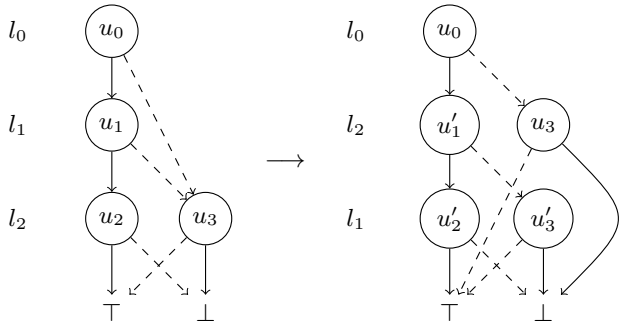


Fig. 1. Swapping l_1 and l_2.

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:

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

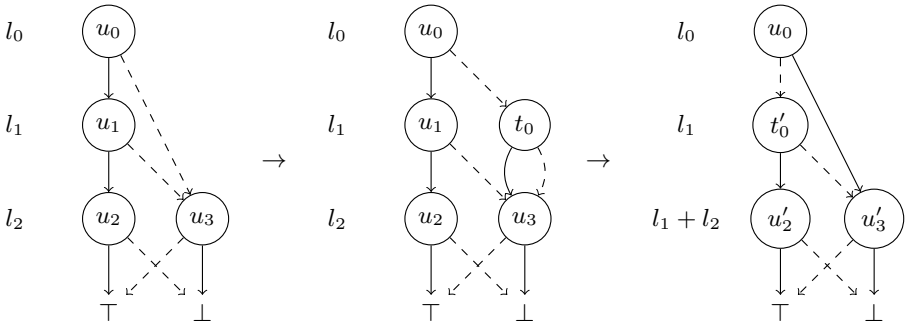
We can see that if $x = 1$ then $F_{x\sigma} = F_{x\overline{y}}$ and $F_{x\overline{\sigma}} = F_{xy}$. Similarly if $x = 0$ then $F_{\overline{x}\sigma} = F_{\overline{x}y}$ and $F_{\overline{x}\overline{\sigma}} = F_{\overline{x}\overline{y}}$. With that in mind (7) can be written as

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

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 & 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 non-linear 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.

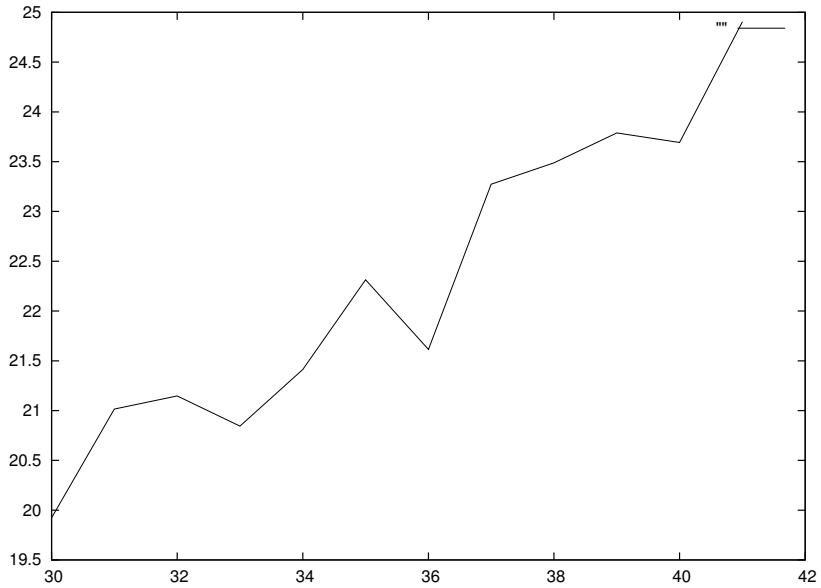


Fig. 2. Trend of complexities for Trivium-N

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 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. 14th International Conference on Information Security and Cryptology, Seoul, Korea, November 30 - December 2, 2011, to appear in Lecture Notes in Computer Science (2011)
2. Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design **12** (1993) 42–47
3. Cannière, C.D., Preneel, B.: Trivium specifications. ECRYPT Stream Cipher Project (2005)
4. Akers, S.: Binary decision diagrams. IEEE Transactions on Computers **27**(6) (1978) 509–516
5. Somenzi, F.: Binary decision diagrams. In: Calculational System Design, volume 173 of NATO Science Series F: Computer and Systems Sciences, IOS Press (1999) 303–366
6. Krause, M.: Bdd-based cryptanalysis of keystream generators. CRYPTO '98, Lecture Notes in Computer Science **1462** (1998) 222–237
7. Stegemann, D.: Extended BDD-Based Cryptanalysis of Keystream Generators. Selected Areas in Cryptography 2007, Lecture Notes in Computer Science **4876** (2007) 17–35
8. Raddum, H.: MRHS Equation Systems. Selected Areas in Cryptography 2007, Lecture Notes in Computer Science **4876** (2007) 232–245
9. Raddum, H., Semaev, I.: Solving multiple right hand sides linear equations. Designs, Codes and Cryptography **49**(1) (2008) 147–160
10. Schilling, T.E., Raddum, H.: Solving equation systems by agreeing and learning. Proceedings of Arithmetic of Finite Fields, WAIFI 2010, Lecture Notes in Computer Science **6087** (2010) 151–165
11. Shannon, C.E.: The synthesis of two-terminal switching circuits. Bell Systems Technical Journal **28** (1949) 59–98
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`.

Chapter 6

Other Results

This chapter contains two other results which did not fit in any other publication. They represent new approaches either on how to solve the main problem studied in this thesis or to reduce the technical complexity of the task. As opposed to the other scientific results in this thesis the ones presented in this chapter are *works in progress*.

The first result might make the main approach used in this thesis easier to understand. Primarily for those working more in the fields of algorithms and graph theory, the presented method might transfer the theme of cryptanalysis into a more familiar realm. Due to the relatively intuitive structure of the presented reduction of the problem, one can say that it bridges the two areas.

The second result represents more of a technical improvement. One algorithm is incorporated into another by preprocessing the input of the former. By doing this, additional information can be obtained which would otherwise require a whole separate processing from a second algorithm. This reduces the implementation complexity of a potential solving algorithm.

6.1 Independent Set Reduction

One technique to show the *NP*-hardness of a problem A is to give a *deterministic polynomial time reduction* of another problem B, known to be *NP*-hard, to A. By such a reduction we mean a function f, which takes as input a problem instance $b \in B$ and has as output a problem instance of A. That is a function $f : B \to A$ which can be computed in polynomial time and $f(b) \in A \Leftrightarrow b \in B$.

Such a reduction demonstrates that problem A is at least as hard to solve as B since all instances of B can be solved if we can solve A. Reductions which satisfy these requirements are also called Karp- or Cook-reductions [10, 28, 41] and are in the following denoted by $B \leq_p A$.

Since those reductions can be done in deterministic polynomial time, the asymptotical running time of B cannot be larger than that of A, and therefore the complexity of A is dominant.

The problem of solving a set of equations is almost intuitively *NP*-complete since there exists an obvious relation to *SAT* solving. Other decision problems – such as the INDEPENDENT SET-problem – belong to one of the first problems shown to be *NP*-complete [46], but might need a proper formal reduction for clarification.

In the following we will show an alternative reduction of the decision version of the Gluing/Agreeing algorithm to the INDEPENDENT SET-problem (IS-problem). Not for the purpose of demonstrating (again) the *NP*-completeness of the IS-problem, but rather to provide an alternative formulation of the Gluing/Agreeing in the context of graph theory. Along with it, we will present some remarks on the structure of the resulting graphs. A similar reduction of the SAT-problem $\leq_p$ IS-problem along with more details on polynomial time reductions and complexity theory in general can be found in [46].

Then we will present an algorithm by Fomin et. al. [22] to solve the IS-problem on instances yielded by random equation systems and/or by a cipher. Finally, we will present some experimental results and conclusions.

To begin, we define the problem of finding a solution to a set of equations as a decision problem of the Gluing/Agreeing algorithm.

Definition 6 (GA-problem)**.** *Let*

$$\mathcal{S} = \{S_0, S_1, \ldots, S_{m-1}\} \tag{6.1}$$

be an input to the Gluing/Agreeing algorithm where $S_i = (X_i, V_i)$ is a symbol consisting of a set of variables X_i and a set of satisfying vectors $V_i = \{v_0, v_1, \ldots, v_{r-1}\}$ defined in X_i.

The decision problem is then to ask if there exists a set of vectors $\mathcal{V} \subseteq V_0 \cup V_1 \cup \ldots \cup V_{m-1}$ such that

1. *for each V_i it holds that $|V_i \cap \mathcal{V}| = 1$, and*
2. *for each $a, b \in \mathcal{V}$ with $a \in V_i$ and $b \in V_j$ it is true for $X_{i,j} = X_i \cap X_j$ that $a[X_{i,j}] = b[X_{i,j}]$.*

It is easy to see that every such set $\mathcal{V}$ satisfying both conditions in definition 6 constitutes a solution to the input of the Gluing/Agreeing algorithm.

We want to reduce the GA-problem to the well known *k*-IS-problem. Again, we will give a definition of the problem as the decision version.

Definition 7 (*k*-IS-problem)**.** *Given a graph $\mathcal{G} = (V, E)$ with vertices V, edges E and a non-negative integer k, does there exist a subset $\mathcal{I} \subseteq V$ of size k such that $\mathcal{I}$ is an independent set, i.e., $\mathcal{I}$ induces an edgeless subgraph on $\mathcal{G}$?*

6.1.1 Reduction

Now we will introduce our new reduction GA-problem $\leq_p$ IS-problem. That is a method to transform an input to the GA-problem to an input of the *k*-IS-problem. Then we will show that this reduction is correct, i.e., the instance of the GA-problem with *m* symbols has a solution if and only if the resulting *k*-IS-problem has a solution with $k = m$ and that the reduction can be done in polynomial time.

The reduction consists of two steps:

Step 1 First we create an empty undirected graph $\mathcal{G} = (V, E)$. For every V_i in (6.1) we construct a clique consisting of vertices labeled by vectors in V_i and insert them into $\mathcal{G}$. That is for every vector $v \in V_i$ we insert a vertex labeled v into V and connect the vertex labeled by v to all other vertices with labels in V_i.

Step 2 After *Step 1* there exists for every vector v in $V_0 \cup V_1 \cup \ldots \cup V_{m-1}$ a corresponding vertex in $\mathcal{G}$ labeled v. For all pairs of symbols S_i, S_j with $X_{i,j} = X_i \cap X_j \neq \emptyset$ we compare every pair of vectors $u \in V_i, w \in V_j$. If $u[X_{i,j}] \neq w[X_{i,j}]$ we insert the edge uw into $\mathcal{G}$.

Lemma 1. *$\mathcal{G}$ has an independent set of size m, if and only if $\mathcal{S}$ has a solution. Furthermore, the reduction above can be done in polynomial time.*

Proof. We will show that the vertices of $\mathcal{G}$ which form an independent set of size m are, as their corresponding vectors, a solution set for $\mathcal{S}$.

($\Rightarrow$) Assume $\mathcal{G}$ has an independent set $\mathcal{I}$ of size m. By *Step 1* $\mathcal{G}$ consists of m cliques, so $\mathcal{I}$ can contain at most 1 vertex of every clique. Since all vertices labeled by vectors in V_i form a clique, the first requirement of the GA-problem is met, namely that at most one vector per V_i is selected.

Furthermore, since by *Step 2* all vertices whose corresponding vectors are unequal in their projection on common variables are connected, $\mathcal{I}$ can only contain those vertices whose corresponding vectors are equal in their projection. That satisfies the second requirement of the GA-problem.

($\Leftarrow$) Assume $\mathcal{S}$ has a solution $\mathcal{V}$. Then we can form an independent set $\mathcal{I}$ in $\mathcal{G}$ by selecting all vertices with labels in $\mathcal{V}$. This is an independent set of size m in $\mathcal{G}$ since only one vertex per clique is selected (by construction in *Step 1*). Furthermore none of these vertices share an edge, since by *Step 2* only those vertices are connected whose corresponding vectors are unequal in their projection of common variables. Thus, if $\mathcal{I}$ would not be an independent set of size m, $\mathcal{V}$ would not be a solution to $\mathcal{S}$, which contradicts the assumption.

Step 1 of the reduction can be done in $\mathcal{O}(m \cdot |V|_{max}{}^2)$ where $|V|_{max}$ is the maximum number of vectors in any V_i. *Step 2* is comparing each pair of vectors for each pair of symbols and therefore runs in $\mathcal{O}(m^2 \cdot |V|_{max}{}^2)$. The whole reduction can therefore be done in $\mathcal{O}(m^2 \cdot |V|_{max}{}^2)$. □

Example 6 (Reduction example). Let

S_0	x_0	x_1	x_2	x_3
a_0	0	0	0	1
a_1	0	1	0	0
a_2	1	0	1	0

,

S_1	x_2	x_3	x_4	x_5
b_0	0	0	0	1
b_1	0	1	0	0
b_2	1	0	1	0

,

S_2	x_0	x_1	x_4	x_5
c_0	0	0	0	1
c_1	0	1	0	0
c_2	1	0	1	0

(6.2)

be a given GA-problem instance. We will apply the two-step reduction explained above in order to transform S_0, S_1, S_2 into an IS-problem instance. In *Step 1* we create a clique for every symbol.

The only possible independent set of size $k = 3$ in Step 2 of Figure 6.1 is indicated in orange. The only solution to equations (6.2) is therefore the combination of vectors a_2, c_2, b_2 which is written $(1,0,1,0,1,0)$ in the variables $(x_0, x_1, x_2, x_3, x_4, x_5)$.

6.1.2 Graph Structure

Graphs resulting from this reduction are different from random graphs, even when the input was a random equation system. Let us assume that we know the following about the input instance:

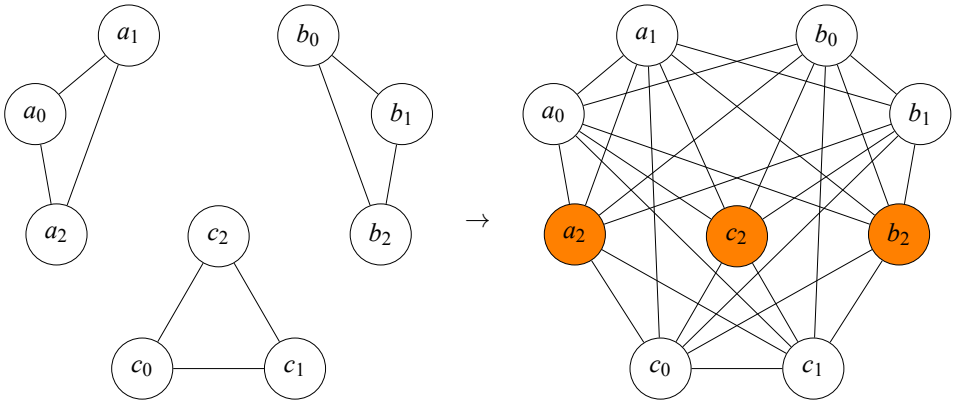


Figure 6.1: 2-Step Reduction to IS-problem

1. The equation system is over $\mathbb{F}_q$,
2. in m equations,
3. with a sparsity of $\leq l$,
4. all equations are pair-wise agreeing,
5. and the system contains no trivial partial solutions, i.e., fixed variables.

Then we can make the following observations about the resulting graph:

Lemma 2. *The graph will contain m complete subgraphs. I.e., for each (X_i, V_i) with $r = |V_i|$ there exists the complete subgraph K_r in $\mathcal{G}$.*

Proof. By construction. □

Lemma 3. *For any pair of complete subgraphs K_r^i, K_s^j containing r and s nodes, respectively in $\mathcal{G} = (V, E)$ let η_{ij} be the size of the cut-set between K_r^i and K_s^j, i.e., $\eta_{ij} = |\{uv \in E \mid u \in K_r^i \text{ and } v \in K_s^j\}|$. Then either $\eta_{ij} = 0$ or $r \leq \eta_{ij} \leq r(s-1)$, $r \leq s$.*

Proof. If two pair-wise agreeing equations have no common variables, their vectors cannot disagree – hence their cliques have no common edges, i.e., $\eta_{ij} = 0$. If they on the other hand have common variables, we know that they must have vectors that differ in these variables since by assumption there are no trivial partial solutions. Each vector in K_r^i therefore has to be unequal to at least one vector in K_s^j, i.e., $r \leq \eta_{ij}$. Since the equations are agreeing pair-wise we also know that no vector can be unequal to all vectors of K_s^j and therefore $\eta_{ij} \leq r(s-1)$. □

This casts some constraints on the structure of the resulting graph after the transformation, even if the input equation system was purely random (and then pair-wise agreed). Unfortunately, it is unclear at this point if this observations can give any advantage when implementing a solving algorithm and the question remains an open problem.

6.1.3 IS-Solving Algorithm

We have now established a method to reduce the problem of solving a set of equations to the problem of finding a k-independent set in a graph. The next step is obviously to try to solve the transformed instances, and to do so we will present the algorithm `mis` introduced by Fomin et. al. [22]. The algorithm will return the size of the biggest independent set in the input instance. If we know that our input instance of equations before the transformation has a solution, we can actually answer that question immediately, i.e., the size is m. On the other hand, we cannot say which vertices are contained in the set. By tracing `mis`' calculation we can reconstruct the independent set and ultimately provide the solution to our input equation system.

The reason for `mis`' significance in the context of equation solving is that one of its operations, namely *folding*, transforms the *equations* (which correspond to the complete graphs K_r^i) in a non-trivial way. That means that the modification to the input equations is new and to our knowledge not duplicated by any other algorithm.

In the following we call $N(v) = \{u \in V | uv \in E\}$ the *open neighborhood* of v. $N[v] = N(v) \cup \{v\}$ is the *closed neighborhood* of v. Further, we say that $\alpha(\mathcal{G})$ denotes the size of a maximum independent set in a graph $\mathcal{G}$.

We can then find that for a graph $\mathcal{G}$ the following two observations are true:

- For every connected component C of $\mathcal{G}$,

$$\alpha(\mathcal{G}) = \alpha(C) + \alpha(\mathcal{G} \setminus C).$$

- For any two vertices v, w s.t. $N[w] \subseteq N[v]$ it is true that

$$\alpha(\mathcal{G}) = \alpha(\mathcal{G} \setminus \{v\}).$$

The following two Lemmas along with their proofs can be found in [22]. They will play a central role in the algorithm `mis`.

Folding A vertex v is *foldable* if $N(v) = \{u_1, u_2, \ldots, u_{d(v)}\}$ contains no anti-triangle. *Folding* a vertex v is then applying the following steps to $\mathcal{G}$ in order to obtain the modified graph $\widetilde{\mathcal{G}}(v)$:

1. Add vertex u_{ij} for each anti-edge $u_i u_j$ in $N(v)$
2. Add edges between each u_{ij} and vertices in $N(u_i) \cup N(u_j)$
3. Add one edge between each pair of new vertices
4. Remove $N[v]$

Lemma 4. *Consider a graph $\mathcal{G}$, and let $\widetilde{\mathcal{G}}(v)$ be the graph obtained by folding a foldable vertex v. Then*

$$\alpha(\mathcal{G}) = 1 + \alpha(\widetilde{\mathcal{G}}(v)).$$

Mirroring A *mirror* of a vertex v is any $u \in \bigcup_{w \in N(v)} N(w)$ s.t. $N(v) \setminus N(u)$ forms a clique or $N(v) \setminus N(u) = \emptyset$. $M(v)$ denotes the set of all mirrors of v.

Lemma 5. *For any graph $\mathcal{G}$ and for any vertex v of $\mathcal{G}$,*

$$\alpha(\mathcal{G}) = max\{\alpha(\mathcal{G} \setminus (\{v\} \cup M(v))), 1 + \alpha(\mathcal{G} \setminus N[v])\}.$$

With the observations and lemmas above we are ready to state the algorithm `mis`. It takes as an argument a graph $\mathcal{G}$ and returns the size of the biggest independent set in $\mathcal{G}$. As stated before, the set of vertices forming the maximum independent set in $\mathcal{G}$ can be derived by backtracking `mis`' steps.

Algorithm 3 mis

```
1: procedure MIS(G)
2:     if(|V(G)| ≤ 1) return |V(G)|
3:     if(∃ connected component C ⊂ G) return mis(C) + mis(G \ C)
4:     if(∃ vertices v,w such that N[w] ⊆ N[v]) return mis(G \ {v})
5:     if(∃ vertex v with deg(v) = 2) return 1+mis(G̃(v))
6:     select a vertex v of maximum degree which minimizes |E(N(v))|
7:     return max{α(G \ ({v} ∪ M(v))), 1 + α(G \ N[v])}
8: end procedure
```

Algorithm 3 represents just one method utilizing the previously presented lemmas on the properties of independent sets to solve the problem. Other algorithms, e.g., more specialized in solving instances resulting from the reduction, can be imagined and might improve the running time profoundly. For further details on the method used to find a maximum independent set, along with examples of the techniques, one can consult [22].

6.1.4 Experimental Results

We applied this algorithm to a small set of instances coming from random equation systems and examples from algebraic cryptanalysis. Unfortunately our simple implementation of the presented algorithm did not yield any improvement over the other techniques presented. Furthermore, the case of a foldable vertex occurred rather rarely.

Another problem during the short phase of experimentation, was the insufficiently optimized approach in handling the graphs generated from input equation systems.

6.1.5 Conclusion & Further Work

We have presented an alternative approach for solving a system of sparse equations over some finite field. A reduction to the well-known IS-problem was shown, along with a very short analysis of the resulting graph structure. Furthermore, we presented an algorithm by Fomin et. al. to solve the general IS-problem.

The presented algorithm by Fomin et. al. is directed toward the general IS-problem, i.e., not specialized on input from equation system. It is possible that additional heuristics in the algorithm can speed up the solving process.

It is furthermore unclear how much the graph structure, which is rather specific even for random equation systems, influences the hardness of the IS-problem. A further analysis of the graph structure of equation systems might lead to a greater insight and to more efficient algorithms.

Another problem during the investigation of the possibilities of this approach were the rather intricate implementational details. It might be possible that an efficient implementation of the presented approach can already compete against other existing techniques and we therefore think that a further investigation of the IS-problem in relation to the solving of equation systems might be worthwhile.

6.2 Algorithm Unification

When experimenting with versions of the Gluing/Agreeing algorithm and with the Syllogism method of solving equation systems [58], it becomes clear that running either one of them on one and the same instance can lead to different new information. That is, one algorithm can produce a different output from another. This fact is not very surprising in itself since they are two different algorithms. Nevertheless, the aim of both algorithms is the same and their guess and verify strategy is similar.

We denote by $\mathfrak{A}(r)$ the output of the Agreeing algorithm, by $\mathfrak{S}(r)$ the output of the Syllogism algorithm on an instance r. We can say that in general $\mathfrak{A}(r) \neq \mathfrak{S}(r)$ (there might exists r where the output is equal). We can illustrate this by the following example.

Example 7 (Agreeing vs. Syllogism). The equation system $r = \{S_0, S_1, S_2\}$ is pairwise agreeing

S_0	x_0	x_1	x_3
a_0	0	0	0
a_1	0	0	1
a_2	1	0	1
a_3	1	1	1

,

S_1	x_1	x_2	x_4
b_0	0	0	1
b_1	1	0	0
b_2	1	0	1
b_3	1	1	1

,

S_2	x_0	x_2	x_5
c_0	0	0	0
c_1	0	1	1
c_2	1	0	0
c_3	1	1	0

(6.3)

and $\mathfrak{A}(r) = r$ would not yield any modification. On the other hand $\mathfrak{S}(r)$ would result in a modification to S_2, i.e., the vector c_1 would not be contained in $\mathfrak{S}(r)$.

The case for the Syllogism algorithm is similar. Let us assume that our instance r is now

S_0	x_0	x_1	x_2
a_0	0	0	0
a_1	0	0	1
a_2	0	1	0
a_3	1	0	0
a_4	1	1	1

,

S_1	x_0	x_1	x_2
b_0	0	0	0
b_1	0	1	1
b_2	1	0	1
b_3	1	1	0
b_4	1	1	1

.

Then $\mathfrak{S}(r) = r$ despite the fact that we can *learn* from $\mathfrak{A}(r)$ that a_1, a_2, a_3, b_1, b_2 and b_3 cannot be part of any common solution to S_0 and S_2.

The straightforward way to make use of both algorithms simultaneously would be to run them consecutively on the same instance. While this is a a very easy solution which guarantees that the output contains no information which either algorithm would not be able to filter out, it would make the code of an implementation more complicated,

therefore more error-prone and probably less efficient. Less efficient since it is not guaranteed that Syllogism might yield additional information over Agreeing, and vice versa.

Another problem is that the analysis of the run-time of this compound algorithm might be more complicated, and it seems to be the least favorable solution. A *unified* algorithm, which would behave exactly as the composition $\mathfrak{A} \circ \mathfrak{S}$, could circumvent this problem. The following section describes how such an algorithm can be constructed by adding extra information to an Agreeing instance during a separate stage of preprocessing, and how this information is used during Agreeing to yield the same result as running first Agreeing, and then the Syllogism algorithm on the same instance.

This preprocessing step can then be applied to any Agreeing instance and changes complexity estimates of the algorithm only relative to the size of the input instance.

6.2.1 Syllog Preprocessing

First we have to recall that the Agreeing algorithm can work on *tuples* [44] or *pockets* [47]. Both approaches exploit the fact that it is not necessary to repeatedly calculate projections of vectors in common variables to other symbols. Instead, tuples or pockets are used to keep track of pairs of sets of vectors which are equal in their projection across different symbols. While the tuple-approach is limited to equivalences between sets[1], we make use of pockets which can express one-way implications and give us a higher degree of freedom to form the necessary constraints.

We will explain the preprocessing technique with the help of the following three symbols:

$$\begin{array}{c|cc} S_a & x_i & x_j \\ \hline a_0 & 0 & 0 \\ a_1 & 0 & 1 \\ a_2 & 1 & 0 \\ a_3 & 1 & 1 \end{array}, \quad \begin{array}{c|cc} S_b & x_j & x_k \\ \hline b_0 & 0 & 0 \\ b_1 & 0 & 1 \\ b_2 & 1 & 0 \\ b_3 & 1 & 1 \end{array}, \quad \begin{array}{c|cc} S_c & x_i & x_k \\ \hline c_0 & 0 & 0 \\ c_1 & 0 & 1 \\ c_2 & 1 & 0 \\ c_3 & 1 & 1 \end{array}. \tag{6.4}$$

These symbols are obviously pair-wise agreeing and do not contain any information. They contain each possible binary vector in two variables for all pairs (x_i, x_j), (x_j, x_k) and (x_i, x_k). In order that the Agreeing algorithm becomes active (as in *propagating knowledge*) one has to delete at least two vectors from either symbol. However, we know that if we delete the two correct vectors from two different symbols, the Syllogism algorithm can propagate some knowledge.

For example the deletion of a_0 (denoted as $\overline{a_0}$) yields the implication $\overline{x_i} \Rightarrow x_j$. The deletion of either b_2 or b_3 can give us new information which can be used to reduce S_c, i.e.:

- $\overline{b_2}$ would yield $x_j \Rightarrow x_k$ and therefore $\overline{x_i} \Rightarrow x_k$ which deletes c_0, or
- $\overline{b_3}$ which would yield $x_j \Rightarrow \overline{x_k}$ and therefore $\overline{x_i} \Rightarrow \overline{x_k}$ which deletes c_1.

[1] The tuple {A,B} represents the fact that: If the set of vectors A is excluded from a common solution then B must be excluded, and vice-versa.

We can express this fact with the following implications in the terms of marking vectors:

$$\begin{aligned} \{a_0, b_2\} &\Rightarrow \{c_0\} \\ \{a_0, b_3\} &\Rightarrow \{c_1\}, \end{aligned}$$

where $A \Rightarrow B$ is to be understood as that the marking (*deletion*) of all vectors in A entails the marking of all vectors in B, but not the other way around.

Proceeding in the same way for all possible implications

$$\begin{aligned} x_i^{(\alpha)} \Rightarrow x_j^{(\beta)} \Rightarrow x_k^{(\gamma)},\ x_i^{(\alpha)} \Rightarrow x_k^{(\beta)} \Rightarrow x_j^{(\gamma)} \\ x_j^{(\alpha)} \Rightarrow x_i^{(\beta)} \Rightarrow x_k^{(\gamma)},\ x_j^{(\alpha)} \Rightarrow x_k^{(\beta)} \Rightarrow x_i^{(\gamma)} \\ x_k^{(\alpha)} \Rightarrow x_i^{(\beta)} \Rightarrow x_j^{(\gamma)},\ x_k^{(\alpha)} \Rightarrow x_j^{(\beta)} \Rightarrow x_i^{(\gamma)} \end{aligned}$$

where $\alpha, \beta, \gamma \in \{0,1\}$, $x_r^{(0)} = \overline{x_r}$ and $x_r^{(1)} = x_r$ can give us all pockets we need to express the behavior of the Syllogism algorithm.

All pockets which can be derived from all possible implications $x_i^{(\alpha)} \Rightarrow x_j^{(\beta)} \Rightarrow x_k^{(\gamma)}$ on (6.4) is shown in Figure 6.2.

$$\begin{aligned} p_0 &= (\{a_0, b_2\}, 8) & p_6 &= (\{a_3, b_0\}, 10) \\ p_1 &= (\{a_0, b_3\}, 9) & p_7 &= (\{a_3, b_1\}, 11) \\ p_2 &= (\{a_1, b_0\}, 8) & p_8 &= (\{c_0\}, \emptyset) \\ p_3 &= (\{a_1, b_1\}, 9) & p_9 &= (\{c_1\}, \emptyset) \\ p_4 &= (\{a_2, b_2\}, 10) & p_{10} &= (\{c_2\}, \emptyset) \\ p_5 &= (\{a_2, b_3\}, 11) & p_{11} &= (\{c_3\}, \emptyset) \end{aligned}$$

Figure 6.2: Syllogism pockets for $x_i^{(\alpha)} \Rightarrow x_j^{(\beta)} \Rightarrow x_k^{(\gamma)}$.

Again, just like in the Syllogism algorithm, the transitive closure of the implications is maintained, but this time with the help of pockets and the Agreeing algorithm. Instead of finding new implications we have expressed the technique of Syllogism as the deletion of vectors in an equation system.

The following algorithm `spre` implements such a preprocessing. It takes as input an equation system $\mathcal{S} = \{S_0, S_1, \ldots, S_{m-1}\}$ and returns all newly derived pockets according to the rules above from $\mathcal{S}$.

The crucial operations begin at line 5 in the algorithm, i.e., the generation of the pockets. Assume without loss of generality that $(\alpha, \beta, \gamma) = (0,0,0)$. Then set A will after the execution of line 5 contain all vectors from symbol S_a such that $v[x_i, x_j] = (0,0)$. The deletion of all these vectors suggests that if $x_i = 0$ then $x_j = 1$ which is the implication $\overline{x_i} \Rightarrow x_j$. In line 6 all vectors of symbol S_b are collected in B for which it is true that $v[x_j, x_k] = (1,0)$. A deletion of all these vectors would immediately yield that if $x_j = 1$ then $x_k = 1$, i.e., $x_j \Rightarrow x_k$. If both implications become simultaneously true due to the deletion (marking) of all vectors in A and B, we can derive by $\overline{x_i} \Rightarrow x_j \Rightarrow x_k$ the fact that $\overline{x_i} \Rightarrow x_k$. We therefore know that in this case from symbol S_c all vectors with $v[x_i, x_k] = (0,0)$ need to be deleted, exactly the vectors which are collected in line 7.

Algorithm 4 Syllogism Preprocessing

procedure SPRE($\mathcal{S}$)
  $N \leftarrow \emptyset$
  for all triples (x_i,x_j,x_k), s.t., $x_i,x_j \in X_a,\ x_j,x_k \in X_b$ and $x_i,x_k \in X_c$ **do**
    for all binary vectors (α,β,γ) **do**
      $A \leftarrow \{v \in V_a \mid v[x_i,x_j] = (\alpha,\beta)\}$
      $B \leftarrow \{v \in V_b \mid v[x_j,x_k] = (\overline{\beta},\gamma)\}$
      $C \leftarrow \{v \in V_c \mid v[x_i,x_k] = (\alpha,\gamma)\}$
      $p_r \leftarrow (A \cup B, s)$
      $p_s \leftarrow (C,\emptyset)$
      Insert p_r and p_s into N
    end for
  end for
  return N
end procedure

Example 8 (spre algorithm). Assume we apply spre on equation system (6.3) and we say that $x_i = x_0,\ x_j = x_1$ and $x_k = x_2$. At some point the algorithm would be in the state that $(\alpha,\beta,\gamma) = (0,1,1)$. At that point A would be empty since there is no vector $v[x_0,x_1] = (0,1)$ in V_0. Likewise there does not exist any vector $v[x_1,x_2] = (0,1)$ in V_1 and therefore $B = \emptyset$. This situation is equivalently expressed by the two implications

$$\overline{x_0} \Rightarrow \overline{x_1}$$
$$\overline{x_1} \Rightarrow \overline{x_2}$$

which yield $\overline{x_0} \Rightarrow \overline{x_2}$. The set C contains all vectors $v[x_0,x_2] = (0,1)$. Since $A \cup B$ is empty we get an empty p_r pocket and all vectors in p_s, namely c_1 have to be deleted immediately.

6.2.2 Conclusion & Further Work

In this section we presented a method to *emulate* the Syllogism algorithm in the Agreeing algorithm without any modification of the Agreeing. With the preprocessing routine spre, we make full use of the transitive closure on implications throughout the equation system by running the Agreeing algorithm on the augmented set of pockets.

Open questions are for example if and how this augmentation of the pocket database influences the learning described in [47]. The overall complexity of Agreeing is only influenced by the number of the pockets, but a new estimate on the complexity in terms of the *number of equations*, *number of variables* and *sparsity* for the augmented algorithm would be interesting.

Bibliography

[1] 2012. http://www.ecrypt.eu.org/stream/. 2.3

[2] 2012. http://www.satcompetition.org/. 3.3

[3] ALBRECHT, M., CID, C., FAUGÈRE, J. C., AND PERRET, L. On the relation between the MXL family of algorithms and Gröbner basis algorithms. Cryptology ePrint Archive, Report 2011/164, 2011. http://eprint.iacr.org/. 3.2.2

[4] ARS, G., FAUGÈRE, J. C., IMAI, H., KAWAZOE, M., AND SUGITA, M. Comparison between XL and Gröbner Basis Algorithms. In *Advances in Cryptology — ASIACRYPT 2004, Springer-Verlag* (2004), pp. 338–353. 3.2.2

[5] BARD, G. V. *Algebraic Cryptanalysis*. Springer-Verlag, 2009. 2.4

[6] BARD, G. V., COURTOIS, N. T., AND JEFFERSON., C. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers. Cryptology ePrint Archive, Report 2007/024, 2007. http://eprint.iacr.org/. 3.3.1

[7] BOGDANOV, A., KHOVRATOVICH, D., AND RECHBERGER, C. Biclique Cryptanalysis of the Full AES. *Cryptology ePrint Archive, Report 2011/449* (2011). http://eprint.iacr.org/. 2.3

[8] CID, C., MURPHY, S., AND ROBSHAW, M. J. B. Small Scale Variants of the AES. In *Proceedings of the 12th international conference on Fast Software Encryption* (Berlin, Heidelberg, 2005), FSE'05, Springer-Verlag, pp. 145–162. 2.4

[9] COOK, D., AND KEROMYTIS, A. *CryptoGraphics: Exploiting Graphics Cards For Security (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. 3.1

[10] COOK, S. A. An overview of computational complexity. *Communications of the ACM 26* (1983), 401–408. 6.1

[11] COURTOIS, N., ALEXANDER, K., PATARIN, J., AND SHAMIR, A. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. *Lecture Notes in Computer Science 1807* (2000), 392–407. 3.2, 3.2.1, 3.2.2

[12] COURTOIS, N., AND PATARIN, J. About the XL Algorithm over GF(2). *Lecture Notes in Computer Science 2612* (2003), 141–157. 3.2.1

[13] COURTOIS, N., AND PIEPRZYK, J. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. *Lecture Notes in Computer Science 2501* (2002), 267–287. 2.4

[14] COURTOIS, N. T., BARD, G. V., AND WAGNER, D. Algebraic and Slide Attacks on KeeLoq. In *Fast Software Encryption*, K. Nyberg, Ed. Springer-Verlag, Berlin, Heidelberg, 2008, pp. 97–115. 2.4

[15] DAVIS, M., LOGEMANN, G., AND LOVELAND, D. A Machine Program for Theorem-Proving. *Commun. ACM 5*, 7 (1962), 394–397. 3.3.1

[16] DAVIS, M., AND PUTNAM, H. A Computing Procedure for Quantification Theory. *J. ACM 7*, 3 (1960), 201–215, doi: 10.1145/321033.321034. 3.3.1

[17] DIFFIE, W. The First Ten Years of Public-Key Cryptography. In *Innovations in Internetworking*. Artech House, Inc., 1988, pp. 510–527. 2.3

[18] DINUR, I., AND SHAMIR, A. Cube Attacks on Tweakable Black Box Polynomials. Cryptology ePrint Archive, Report 2008/385, 2008. `http://eprint.iacr.org/`. 3.2

[19] EULER, L. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae 8* (1741), 128–140. 2.2

[20] FAUGÈRE, J. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra 139*, 1-3 (1999), 61–88. 3.2.2

[21] FAUGÈRE, J. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). *Proceedings of the 2002 international symposium on Symbolic and algebraic computation* (2002), 75–83, doi: 10.1145/780506.780516. 3.2.2

[22] FOMIN, F. V., GRANDONI, F., AND KRATSCH, D. A Measure & Conquer Approach for the Analysis of Exact Algorithms. *J. ACM 56* (2009), 25:1–25:32. 6.1, 6.1.3, 6.1.3

[23] GOLDBERG, E., AND NOVIKOV, Y. BerkMin: A fast and robust SAT-solver. *Discrete Applied Mathematics 155*, 12 (2007), 1549–1561. 3.3.1

[24] GREENLAW, R., AND HOOVER, H. *Fundamentals of the Theory of Computation: Principles and Practice*. Morgan Kaufmann, 1998. 2.1.1

[25] GÜNEYSU, T., KASPER, T., NOVOTNÝ, M., PAAR, C., AND RUPP, A. Cryptanalysis with COPACOBANA. *IEEE Trans. Comput. 57*, 11 (Nov. 2008), 1498–1513, doi: 10.1109/TC.2008.80. 3.1

[26] JEROSLOW, R. G., AND WANG, J. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence 1* (1990), 167–187. 10.1007/BF01531077. 3.3.1

[27] KAHN, D. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*, rev sub ed. Scribner, 1996. 2.3

[28] KARP, R. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, R. Miller and J. Thatcher, Eds. Plenum Press, 1972, pp. 85–103. 2.1.3, 2.1.4, 6.1

[29] KIPNIS, A., AND SHAMIR, A. Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. *Lecture Notes in Computer Science 1666* (1999), 788–788. 2.4, 3.2.1

[30] KNUTH, D. *The Art of Computer Programming Vol. 1*. Addison-Wesley, 1973. 2.1.2

[31] LANDAU, E. *Handbuch der Lehre von der Verteilung der Primzahlen*. B. G. Teubner, 1909. 2.1.2

[32] LANDAUER, R. Irreversibility and Heat Generation in the Computing Process. *IBM Journal of Research and Development 5* (1961), 183–191. 3.1

[33] LAURITZEN, N. *Concrete Abstract Algebra: From Numbers to Gröbner Bases*. Cambridge University Press, 2003. 3.2.2, 3.2.2

[34] MARQUES-SILVA, J. The Impact of Branching Heuristics in Propositional Satisfiability Algorithms. In *Proceedings of the 9th Portuguese Conference on Artificial Intelligence: Progress in Artificial Intelligence* (London, UK, 1999), EPIA '99, Springer-Verlag, pp. 62–74. 3.3.1

[35] MARQUES-SILVA, J. P., AND SAKALLAH, K. A. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers 48* (1999), 506–521. 3.3.1

[36] MCDONALD, C., CHARNES, C., AND PIEPRZYK, J. Attacking Bivium with MiniSat. *ECRYPT Stream Cipher Project* (2007). 3.3

[37] MOSKEWICZ, M., MADIGAN, C., ZHAO, Y., ZHANG, L., AND MALIK, S. Chaff: Engineering an Efficient SAT solver. In *Proceedings of the 38th conference on Design automation* (2001), ACM New York, NY, USA, pp. 530–535. 3.3.1, 3.3.1

[38] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. Data Encryption Standard. In *Federal Information Processing Standards Publication 46* (1977). 2.3

[39] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. Triple DES. In *Federal Information Processing Standards Publication 46-3* (1999). 2.3

[40] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. Advanced Encryption Standard. In *Federal Information Processing Standards Publication 197* (2001). 2.3

[41] PAPADIMITRIOU, CH. H. *Computational Complexity*. Addison-Wesley, 1994. 6.1

[42] RADDUM, H. MRHS Equation Systems. *Lecture Notes in Computer Science 4876* (2007), 232–245. 3.4, 3.4.3, 3.4.3

[43] RADDUM, H., AND SEMAEV, I. New Technique for Solving Sparse Equation Systems. *IACR Cryptology ePrint Archive* (2006). 3.4

[44] RADDUM, H., AND SEMAEV, I. Solving Multiple Right Hand Sides linear equations. *Designs, Codes and Cryptography 49*, 1 (2008), 147–160. 3.4.3, 3.4.3, 6.2.1

[45] RØNJOM, S., AND RADDUM, H. On the Number of Linearly Independent Equations Generated by XL. *Lecture Notes in Computer Science 5203* (2008), 239–251. 3.2.1

[46] ROTHE, J. *Complexity Theory and Cryptology. An Introduction to Cryptocomplexity*. EATCS Texts in Theoretical Computer Science. Springer, 2005. 2.1.3, 2.2.1, 6.1

[47] SCHILLING, T. E., AND RADDUM, H. Solving Equation Systems by Agreeing and Learning. *Lecture Notes in Computer Science* (2010), 151–165. 3.4.2, 6.2.1, 6.2.2

[48] SCHÖNING, U. *Theoretische Informatik — kurzgefaßt (3. Aufl.)*. Hochschultaschenbuch. Spektrum Akademischer Verlag, 1997. 2.1.1, 2.1.2

[49] SELMAN, B., KAUTZ, H. A., AND COHEN, B. Local Search Strategies for Satisfiability Testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* (1996), pp. 521–532. 3.3

[50] SEMAEV, I. Sparse Algebraic Equations over Finite Fields. *SIAM Journal on Computing 39*, 2 (2009), 388–409, doi: 10.1137/070700371. 3.4, 3.4.2

[51] SHANNON, C. Communication Theory of Secrecy Systems. *Bell System Technical Journal 28* (1949), 656–715. 2.3

[52] SHANNON, C. E. A Mathematical Theory of Communication. *Bell system technical journal 27* (1948). 2.3

[53] SOOS, M., NOHL, K., AND CASTELLUCCIA, C. Extending SAT Solvers to Cryptographic Problems. In *SAT* (2009), pp. 244–257. 3.3

[54] U.S. DEPARTMENT OF COMMERCE, NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. Announcing Request for Candidate Algorithm nominations for a New Cryptograhpic Hash Algorithm (SHA–3) Family. *Federal Register 212* (2007). 2.3

[55] VAN HENTENRYCK, P., AND MICHEL, L. *Constraint-Based Local Search*. The MIT Press, 2005. 3.5

[56] WARSHALL, S. A theorem on Boolean matrices. *J. Assoc. Comput. Mach. 9* (1962). 3.5

[57] YANG, J., AND GOODMAN, J. Symmetric Key Cryptography on Modern Graphics Hardware. In *Proceedings of the Advances in Crypotology 13th international conference on Theory and application of cryptology and information security* (Berlin, Heidelberg, 2007), ASIACRYPT'07, Springer-Verlag, pp. 249–264. 3.1

[58] ZAKREVSKIJ, A., AND VASILKOVA, I. Reducing Large Systems of Boolean Equations. In *4th International Workshop on Boolean Problems* (2000), pp. 21–22. 3.5, 6.2