Algebraic analysis of Trivium-like ciphers

Sui-Guan Teo[1,2], Kenneth Koon-Ho Wong[1,2], Harry Bartlett[1,2], Leonie Simpson[1,2], and Ed Dawson[1]

[1] Institute for Future Environments,
Queensland University of Technology
teosuiguan@gmail.com, {kk.wong, h.bartlett, lr.simpson, e.dawson}@qut.edu.au
[2] Science and Engineering Faculty,
Queensland University of Technology
2 George Street, Brisbane Qld 4000, Australia

Keywords: Stream ciphers, Trivium, Trivium-N, Bivium-A, Bivium-B, algebraic attacks

Abstract Trivium is a bit-based stream cipher in the final portfolio of the eSTREAM project. In this paper, we apply the approach of Berbain et al. to Trivium-like ciphers and perform new algebraic analyses on them, namely Trivium and its reduced versions: Trivium-N, Bivium-A and Bivium-B. In doing so, we answer an open question in the literature. We demonstrate a new algebraic attack on Bivium-A. This attack requires less time and memory than previous techniques which use the F4 algorithm to recover Bivium-A's initial state. Though our attacks on Bivium-B, Trivium and Trivium-N are worse than exhaustive keysearch, the systems of equations which are constructed are smaller and less complex compared to previous algebraic analysis. Factors which can affect the complexity of our attack on Trivium-like ciphers are discussed in detail.

1 Introduction

Trivium [7] is a bit-based stream cipher designed by De Canniére and Preneel in 2005 and selected in the final portfolio of the eSTREAM project [12]. Trivium has a 288-bit nonlinear feedback shift register (NLFSR) and uses an 80-bit key and an 80-bit IV to generate keystream. However, it is more commonly represented in the literature as based on three non-autonomous binary NLFSRs: A, B, and C, of sizes 93, 84 and 111 bits respectively [4]. Each key-IV pair can be used to generate up to 2^{64} bits of keystream. In each iteration, Trivium uses nonlinear state-update functions to update three bits of the internal state. The keystream bit is expressed as a linear combination of the contents of six stages of its internal state. Trivium's structural simplicity makes it a popular cipher to cryptanalyse, but to date, there are no attacks in the public literature which are faster than exhaustive keysearch. To aid the analysis of Trivium, Bivium-A and Bivium-B were proposed by Raddum [16] in 2006 and Trivium-N was proposed by Schilling and Raddum [17] in 2011.

Both Bivium-A and Bivium-B have a 177-bit NLFSR and use an 80-bit key and an 80-bit IV to generate keystream. Bivium-A/B are also more commonly represented in the literature as based on two non-autonomous binary NLFSRs: A and B, of sizes 93 and 84 bits respectively. In each iteration, nonlinear state-update functions update two bits of the internal state. For Bivium-A, each keystream bit is a linear combination of the contents of two stages of its internal state, while for Bivium-B, each keystream bit is a linear combination of the contents of four stages of its internal state. Trivium-N is a scaled down version of Trivium, where N is the total internal state size. Trivium-N preserves the full structure of the original Trivium cipher, but scales down the sizes of the three registers proportionally.

Algebraic attacks [8] are commonly applied when analysing stream ciphers. They have been particularly effective in attacking keystream generators based on linear feedback shift registers (LFSRs). However, solving systems of equations produced by stream ciphers which use nonlinear state-update functions is more complex as the degree of equations increases as more iterations of the state-update function are applied. This may make the system of equations computationally infeasible to solve. Two different approaches to reducing the degree of such equations are proposed by Raddum [16] and Berbain et al. [3].

Raddum proposed using a algebraic relabelling technique, where the state-update bits of Trivium and Bivium-A/B are represented using new variables, instead of nonlinear combinations of initial state bits. Using this relabelling technique, the degree of equations added into the system never grows higher

Parameter	Bivium-A	Bivium-B	Trivium	Trivium-N
q	2	2	3	3
q'	1	1	1	1
j	1	2	3	3

Table 1. Values of q, q', and j for Trivium-like stream ciphers

than the degree of the nonlinear equation used in a keystream generator's state-update function. The tradeoff however, is that the system of equations now contains more variables.

For keystream generators which use a linear output function (which Trivium-like ciphers are examples of), Berbain et al. [3] presented an approach of expressing new feedback bits of an NLFSR as linear combinations of keystream bits and internal state bits. By expressing the new feedback bits of an NLFSR as linear combinations of keystream bits and the previous internal state bits, the equations representing the feedback bits of an NLFSR will always be linear. Berbain et al. claim that their technique can be extended to ciphers in which $q > 1$ bits of internal state are non-linearly updated at each step and q or more linear combinations of the state are output as keystream. However, the extension of these techniques to ciphers in which $q > 1$ bits of internal state are non-linearly updated, while only $q' < q$ linear combinations of the state-bits are output has not been demonstrated and is posed as an open question.

Contributions of paper. In this paper, we answer Berbain et al.'s above-mentioned open question. Using Trivium-like ciphers as a case-study, we present a new method of representing the feedback bits of Trivium, Trivium-N and Bivium-A/B as linear combinations of internal state bits and keystream bits. Trivium-like ciphers make excellent case-studies, as $q' < q$ in all three cases. To assist us in our analysis, we introduce a new variable j, which describes the number of registers the keystream generation function takes inputs from. We show that the value of j has a significant impact on the success of our algebraic attack on these ciphers. The values of q', q and j for Bivium-A, Bivium-B, Trivium and Trivium-N are given in Table 1.

2 Algorithm specifications and current cryptanalysis

In this section, we review the specifications of Trivium and Trivium-N in Section 2.1 and Section 2.2 respectively and Bivium-A/B in Section 2.3. We omit the description of the initialisation process for the four ciphers, as it has no impact on our analysis. The reader is referred to the specifications of Trivium [7] and Bivium-A/B [16] for a description of the initialisation process.

2.1 Trivium

Let A_i denote the stages for register A and $A_i(t)$ represent the contents of A_i at time t, for $0 \le i \le 92$. Similar notations are used for registers B and C. The state-update functions of Trivium are as follows:

$$A_i(t+1) = \begin{cases} A_{24}(t) \oplus C_{45}(t) \oplus C_0(t) \oplus C_1(t)C_2(t) & i = 92, \\ A_{i+1}(t) & 0 \le i \le 91. \end{cases}$$

$$B_i(t+1) = \begin{cases} B_6(t) \oplus A_{27}(t) \oplus A_0(t) \oplus A_1(t)A_2(t) & i = 83, \\ B_{i+1}(t) & 0 \le i \le 82. \end{cases}$$

$$C_i(t+1) = \begin{cases} C_{24}(t) \oplus B_{15}(t) \oplus B_0(t) \oplus B_1(t)B_2(t) & i = 110, \\ C_{i+1}(t) & 0 \le i \le 109. \end{cases}$$

At time t, Trivium's output function generates a keystream bit as follows:

$$z(t) = A_{27}(t) \oplus A_0(t) \oplus B_{15}(t) \oplus B_0(t) \oplus C_{45}(t) \oplus C_0(t)\ ,\ t \ge 0$$

We extend this representation of Trivium and consider the keystream as a sequence related to the three underlying register sequences. The initial bits in sequence A produced by register A are A_i, for $0 \leq i \leq 92$. Similarly, the initial bits in sequence B produced by register B are B_i, for $0 \leq i \leq 83$, and the initial bits in sequence C produced by register C are C_i, for $0 \leq i \leq 110$. The values of these initial elements for all three sequences are determined by the initialisation process.
The new sequence bits $A_{\alpha+92}$, $B_{\alpha+83}$ and $C_{\alpha+110}$ produced after α iterations of Trivium's state-update function can be calculated as follows:

$$A_{\alpha+92} = A_{\alpha+23} \oplus C_{\alpha+44} \oplus C_{\alpha-1} \oplus C_{\alpha}C_{\alpha+1} \ , \alpha \geq 1 \tag{1}$$
$$B_{\alpha+83} = B_{\alpha+5} \oplus A_{\alpha+26} \oplus A_{\alpha-1} \oplus A_{\alpha}A_{\alpha+1} \ , \alpha \geq 1 \tag{2}$$
$$C_{\alpha+110} = C_{\alpha+23} \oplus B_{\alpha+14} \oplus B_{\alpha-1} \oplus B_{\alpha}B_{\alpha+1} \ , \alpha \geq 1 \tag{3}$$

The keystream bit can be written as a linear combination of two bits each from sequences A, B and C as follows:

$$z_{\alpha-1} = A_{\alpha+26} \oplus A_{\alpha-1} \oplus B_{\alpha+14} \oplus B_{\alpha-1} \oplus C_{\alpha+44} \oplus C_{\alpha-1} \ , \alpha \geq 1 \tag{4}$$

Note that the sequence-based approach is analogous to the relabelling approach of Raddum [16]. In the sequence-based approach, the equations representing sequence bits $A_{\alpha+92}$, $B_{\alpha+83}$ and $C_{\alpha+110}$ for $\alpha \geq 1$ are the same equations which represent the feedback bits of Trivium in Raddum's approach. In this paper, we use these sequence equations in our algebraic analysis.

2.2 Trivium-N

Trivium-N's NLFSRs A and B have a state size $93 \times \beta$ bits and $84 \times \beta$ bits respectively, rounded to the nearest integers, where $\beta = \frac{N}{288}$. The size of C will be the integer which, when summed with the size of A and B, gives N. For example, for $N = 40$, $\beta = \frac{40}{288} \approx 0.1389$ and the sizes of registers A, B and C are 13, 12 and 15 bits respectively. The stages used in the state-update function and keystream generation functions in Trivium-N are also scaled down by a factor of β, rounded to the nearest integer. For example, if 24 is used as a tap position in Trivium, the corresponding tap position for Trivium-40 will be $24 \times 0.1389 = 3.3336 \approx 3$. The tap positions 1 and 2 are unchanged to ensure that the state-update functions remain nonlinear. The state-update functions for Trivium-40 are as follows:

$$A_i(t+1) = \begin{cases} A_3(t) \oplus C_6(t) \oplus C_0(t) \oplus C_1(t)C_2(t) & i = 12, \\ A_{i+1}(t) & 0 \leq i \leq 11. \end{cases}$$
$$B_i(t+1) = \begin{cases} B_1(t) \oplus A_4(t) \oplus A_0(t) \oplus A_1(t)A_2(t) & i = 11, \\ B_{i+1}(t) & 0 \leq i \leq 10. \end{cases}$$
$$C_i(t+1) = \begin{cases} C_3(t) \oplus B_2(t) \oplus B_0(t) \oplus B_1(t)B_2(t) & i = 14, \\ C_{i+1}(t) & 0 \leq i \leq 13. \end{cases}$$

The keystream equation for Trivium-40 is:

$$z(t) = A_4(t) \oplus A_0(t) \oplus B_2(t) \oplus B_0(t) \oplus C_6(t) \oplus C_0(t) \ , t \geq 0$$

Table 2 shows the size of registers A, B and C, along with the keysize for various Trivium-N ciphers analysed in this paper.

Current algebraic cryptanalysis of Trivium and Trivium-N. After observing 288 bits of keystream, Raddum's [16] system of equations for Trivium consisted of 954 equations in 954 variables. Applying techniques from graph theory to this system of equations, Raddum estimated that the initial state of Trivium can be recovered in about 2^{164} operations. Simonetti et al. [19] attempted to solve Raddum's system of equations for Trivium using the F4 algorithm [13] implemented in Magma [6]

N	A	B	C	Key size	N	A	B	C	Key size
40	13	12	15	11	60	19	17	24	17
50	16	15	19	14	70	23	20	27	19

Table 2. Size of registers and keysizes for various Trivium-N

and reported that the computation did not finish. Borghoff et al. [5] investigated the use of heuristic optimization methods such as hill climbing algorithms as a technique for solving Raddum's system of equations, but again, the estimated complexity is worse than exhaustive keysearch. Schilling and Raddum [17,18] analysed Trivium and Trivium-N using an algebraic technique called the Compressed Right Hand Side representation combined with Binary Decision Diagrams (BDD) [1]. The number of paths in the BDD ranged from $2^{93.77}$ for $N = 40$ to $2^{160.49}$ for $N = 70$. As these numbers do not take into account the complexity of finding the correct path in the BDD which solves the equations, their approach is clearly worse than exhaustive keysearch. Other forms of algebraic attacks include attacks using SAT-solvers [15] and cube attacks [2,9,14]. However, none of the attacks are better than exhaustive keysearch for the original Trivium proposal.

2.3 Bivium-A/B specifications

Let A_i denote the stages for register A and $A_i(t)$ represent the contents of A_i at time t, for $0 \le i \le 92$. A similar notation is used for register B. The state-update functions of both Bivium-A and Bivium-B are as follows:

$$A_i(t+1) = \begin{cases} A_{24}(t) \oplus B_{15}(t) \oplus B_0(t) \oplus (B_1(t)B_2(t)), & i = 92, \\ A_{i+1}(t), & 0 \le i \le 91. \end{cases}$$

$$B_i(t+1) = \begin{cases} B_6(t) \oplus A_{27}(t) \oplus A_0(t) \oplus (A_1(t)A_2(t)), & i = 83, \\ B_{i+1}(t), & 0 \le i \le 82. \end{cases}$$

Bivium-A's output function generates a keystream bit as follows:

$$z(t) = B_{15}(t) \oplus B_0(t) \ , \ t \ge 0$$

Similarly, the output function of Bivium-B is as follows:

$$z(t) = A_{27}(t) \oplus A_0(t) \oplus B_{15}(t) \oplus B_0(t) \ , \ t \ge 0$$

We extend this representation of Bivium-A/B in a similar way to our treatment of Trivium by considering the keystream as a sequence related to the two underlying register sequences. Using this approach, the new sequence bits $A_{\alpha+92}$, $B_{\alpha+83}$ produced after α iterations of Bivium-A/B's state-update function can be calculated as follows:

$$A_{\alpha+92} = A_{\alpha+23} \oplus B_{\alpha+14} \oplus B_{\alpha-1} \oplus B_\alpha B_{\alpha+1} \ , \ \alpha \ge 1 \tag{5}$$

$$B_{\alpha+83} = B_{\alpha+5} \oplus A_{\alpha+26} \oplus A_{\alpha-1} \oplus A_\alpha A_{\alpha+1} \ , \ \alpha \ge 1 \tag{6}$$

The keystream bit for Bivium-A can be written as a linear combination of two bits from sequence A as follows:

$$z_{\alpha-1} = B_{\alpha+14} \oplus B_{\alpha-1} \ , \ \alpha \ge 1 \tag{7}$$

The keystream bit for Bivium-B can be written as a linear combination of two bits each from sequences A and B as follows: $z_{\alpha-1} = A_{\alpha+26} \oplus A_{\alpha-1} \oplus B_{\alpha+14} \oplus B_{\alpha-1}$, for $\alpha \ge 1$.

Current algebraic cryptanalysis. There are several attacks on Bivium-A and Bivium-B which are faster than exhaustive keysearch. Raddum [16] used his relabelling approach and formed a system of equations for Bivium-A/B which consists of 399 equations in 399 variables. Combining techniques from graph theory with these system of equations, he was able to recover the initial state of Bivium-A in about a day. He estimated that the complexity of recovering the initial state of Bivium-B will take about 2^{56} seconds. Eibach et al. [11] also identified factors which can have an impact on the computation of a Gröbner basis on Bivium-B, and proposed an optimised Gröbner Basis attack. They estimate that by guessing 42 internal state bits, an optimised Gröbner Basis attack on Bivium-B can recover the initial state in about $2^{39.12}$ seconds. Other forms of algebraic analysis on Bivium-A/B use Boolean Satisfiability (SAT) solvers [10, 15] to recover the initial state. By guessing certain bits of initial state, these attacks using SAT solvers are faster than exhaustive keysearch.

3 New analysis of Trivium-like ciphers

In this section, we apply Berbain et al.'s approach to the analysis of the three Trivium-like ciphers: Bivium-A in Section 3.1, Trivium in Section 3.2 and Trivium-N in Section 3.3. As the analysis of Trivium can easily be applied to Bivium-B, the reader is referred to Appendix A for our analysis of Bivium-B. We note however, that although the system of equations for Bivium-B using our approach is less complex than Raddum's approach, it is still worse than exhaustive keysearch.

3.1 New analysis of Bivium-A

Bivium-A's keystream bit $z_{\alpha-1}$ depends on two sequence bits produced by register B, $B_{\alpha+14}$ and $B_{\alpha-1}$, for $\alpha \geq 1$. The keystream bit z_{69} can be calculated as: $z_{69} = B_{84} \oplus B_{69}$. Applying Berbain at al.'s technique to the equation for z_{69}, we can determine the equation for calculating the sequence bit B_{84}:

$$B_{84} = z_{69} \oplus B_{69} \tag{8}$$

The equations representing the sequence bits $B_{\alpha+83}$ for $\alpha \geq 1$ can then be written as follows: $B_{\alpha+83} = z_{\alpha+68} \oplus B_{\alpha+68}$, for $\alpha \geq 1$. In this section, we analyse two different approaches for recovering the initial state of Bivium-A. Both these approaches use a divide-and-conquer approach to form two systems of equations.

First approach. We use a divide-and-conquer approach to recover the initial state of Bivium-A. This involves forming two systems of equations and solving them sequentially. The first system of equation recovers the sequence produced by register B, while the second system of equation recovers the sequence produced by register A.

We start to form the first system of equations with 84 variables. For the first 69 iterations, we add two equations and one variable into the system of equations: one equation representing the keystream, and one equation and variable representing the sequence bit for register B. A set of these equations are:

$$z_{\alpha-1} \oplus B_{\alpha-1} \oplus B_{\alpha+14} = 0 \ , \ \alpha \geq 1 \tag{9}$$

$$B_{\alpha+83} \oplus z_{\alpha+68} \oplus B_{\alpha+68} = 0 \ , \ \alpha \geq 1 \tag{10}$$

where Equation 9 is a keystream equation for Bivium-A and Equation 10 the new equation representing Bivium-A's sequence bit $B_{\alpha+83}$ derived from our new analysis. After the first 69 iterations, we do not need to add subsequent keystream equations, as these would have already been added into the system of equations and are redundant. To illustrate this, consider the equation describing the sequence bit B_{84}: $B_{84} = z_{69} \oplus B_{69}$. When we try to add the equation describing z_{69} at the 70'th iteration into the system of equations, we have

$$\begin{aligned} z_{69} \oplus B_{69} \oplus B_{84} &= 0 \\ z_{69} \oplus B_{69} \oplus z_{69} \oplus B_{69} &= 0 \end{aligned} \tag{11}$$

Equation 11 cancels out and does not allow us to represent a sequence bit in terms of a linear combination of keystream bits and other sequence bits. For the last 39 iterations, we only add the equations representing the sequence bits for B and would have generated enough sequence bits for register B to have generated 177 bits of keystream, the same amount of keystream which needed to be observed to construct Raddum's fully-determined system of equations. This gives us a final system of equations after 108 iterations consisting of 192 variables in 177 equations. Solving this system of equations using the F4 algorithm gives us 2^{15} possible solutions.

For each of these 2^{15} possible solutions, we form the second system of equations to recover the initial state of A, substituting the sequence bits of B recovered in the first system of equations into the second system of equations. We start to form the second system of equations with 93 variables. At each iteration, we add two equations and one variable: one equation and variable relating the sequence bits of register A with the sequence bits of A and B, and one equation relating the sequence bit of register B with the sequence of A and B. It is not necessary to add the keystream equations into this system of equations, as these would have already been solved in the first system of equations. A set of equations added in this system of equations consists of:

$$A_{\alpha+92} \oplus B_{\alpha+14} \oplus B_{\alpha-1} \oplus B_{\alpha}B_{\alpha+1} \oplus A_{\alpha+23} = 0 \ , \ \alpha \geq 1$$
$$B_{\alpha+83} \oplus A_{\alpha+26} \oplus A_{\alpha-1} \oplus A_{\alpha}A_{\alpha+1} \oplus B_{\alpha+5} = 0 \ , \ \alpha \geq 1$$

After 108 iterations, we have a system of equations consisting of 201 variables in 216 equations. Solving this system of equations can recover the sequence of A. An attacker can then use this initial state to generate some keystream, and check if the keystream generated matches that which was captured. If it is, the attacker can be confident they have recovered the correct initial state.

During the construction of the first system of equations, it is necessary to generate a system of equations beyond 69 sets of equations, even though adding any further variables and equations does not reduce the number of solutions obtained when the first system of equations are solved. To illustrate this, let us assume we had only generated 69 new equations representing the new sequence bits B_{i+83}, for $0 \leq i \leq 68$ and obtain a system of equations consisting of 153 variables in 138 equations. During the construction of the second system of equations, we also add 69 sets of equations. This will result in a second system of equations consisting of 162 variables in 138 equations. Solving the second system of equations will give us 2^{24} possible solutions. Combined with the number of solutions in the first system of equations, this gives us a total number of $2^{15} \times 2^{24} = 2^{39}$ possible solutions. While this is still better than exhaustive keysearch, it is clearly more than the 2^{15} possible solutions we have to originally try in our proposed first approach.

Second approach. In our first approach, the second system of equations consists of 201 variables in 216 equations. As there are more equations than variables, some equations can be removed from the system and may not increase the number of expected solutions. In this section, we investigate the effect that removing these equations and variables has on the system of equations obtained.

The construction of the second system of equations requires adding two new variables and one equation at each iteration. The set of equations due to be added on the 93rd iteration is:

$$A_{185} \oplus B_{107} \oplus B_{92} \oplus B_{93}B_{94} \oplus A_{116} = 0$$
$$B_{176} \oplus A_{119} \oplus A_{92} \oplus A_{93}A_{94} \oplus B_{98} = 0$$

At this point we obtain a system of equations consisting of 186 variables in 186 equations. To build this second system of equations, we need $84 + 93 = 177$ variables in register B to form the second system of equations. This means adding $177 - 84 = 93$ sets of equations into our first system of equations. We use the same construction method used in forming the first system of equations for our first approach to form the system of equations in our second approach. This first system of equations consist of 177 variables in 162 equations. As each equation contains one keystream variable, only 162 keystream bits are needed to solve the first system of equation. This is less than the 177 keystreams bits needed to uniquely define the initial state. Therefore the second approach will not be effective at recovering the initial state and we do not consider this approach in our analysis of Trivium-like ciphers any further. In the remainder of this paper, our approach on Bivium-A refers to the first approach presented earlier.

Comparison of attacks. Details of the systems of equations formed for using Raddum's approach and our approach are shown in Table 3. The linear equation column lists the number of linear equations in the respective system of equations, while the quadratic column entry lists the number of quadratic equations for the respective system of equations. The total equation column lists the total number of equations for the respective system equations, while the K.S. column lists the length of keystream needed to solve the relevant system of equations. The variable column lists the number of variables in the respective system of equations. Finally, the expected solution (Expected solns) column lists the expected number of solutions obtained by solving the respective system of equations.

For Raddum's technique, we need to solve a single system of equations consisting of 399 variables in 399 equations and require 177 bits of keystream to solve. Of these 399 equations, 177 are linear and 222 are quadratic. Our approach requires solving two systems of equations sequentially. The first system of equations has 192 variables in 177 linear equations and requires 177 bits of keystream to solve, while the second system has 108 linear and 108 quadratic equations and does not require any further keystream bits to solve. As we are solving two systems of equations separately, the complexity of our attacks is more likely less compared to Raddum's technique. We evaluate these findings in our experimental results, as is shown in Table 4. For the purposes of comparison, we attempted, via computer experiments, to

Technique	Linear eqn.	Quadratic eqn.	Total eqn.	K.S. (bits)	Variables	Expected Solns
Raddum	177	222	399	177	399	1
Our approach (Step 1)	177	0	177	177	192	2^{15}
Our approach (Step 2)	108	108	216	0	201	1

Table 3. Details for the system of equations in Bivium-A for both approaches

recover the initial state of Bivium-A using two different approaches: Raddum's system of equations and our approach. The time and memory complexities of the two approaches were compared. We solve the system of equations in these techniques using the F4 algorithm implemented in Magma. In both cases, the keystream were generated using the same initial state. We then attempted to solve the system of equations in two ways: (1) without guessing any bits, solve the system of equations for Bivium-A and (2) load 15 correct initial state bits of Register B, and solve the system of equations for Bivium-A.

The maximum amount of memory set in our experiments is 87 040 MB (85 GB). Table 4 lists the time and memory requirements needed for Magma to solve the system of equations. A Did Not Finish (DNF) entry signifies that Magma was not able to solve the system of equations using the allocated amount of memory. Magma was not able to solve Raddum's system of equations despite the allocation of 87 040 MB of RAM for the experiments when we did not guess any bits in the system of equations, whereas the attack using our approach was able to obtain a solution in 13.9 hours using 135.56 MB of RAM. If 15 correct bits of B were loaded into register B, Raddum's and our approach were able to recover the initial state within nine seconds using at most 13.34 MB of RAM. However, our approach required only 3.29 seconds to obtain a solution, whereas needed for Raddum's approach needed 8.95 seconds. Note that it is difficult to directly compare the effectiveness of our approach against the attacks by others [15,16] as those attacks were implemented on different software and hardware platforms, which may have an effect on the time and memory required to recover the initial state of Bivium-A.

	No Guessing			Load correct 15 bits into register B		
	Time	Memory (MB)	K.S (bits)	Time	Memory (MB)	K.S (bits)
Raddum [16]	DNF	87 040	177	8.95 s	13.34	177
Our approach	13.9 hrs	135.56	177	3.29 s	13.31	177

Table 4. Time, memory, and data complexities for recovering initial state of Bivium-A

3.2 New algebraic analysis on Trivium

The keystream generation function of Trivium takes as input one linear combination of six sequence bits: two sequence bits from A, B and C respectively. The 67'th and 70'th keystream bits are:

$$z_{66} = A_{93} \oplus A_{66} \oplus B_{81} \oplus B_{66} \oplus C_{111} \oplus C_{66} \tag{12}$$

$$z_{69} = A_{96} \oplus A_{69} \oplus B_{84} \oplus B_{69} \oplus C_{113} \oplus C_{68} \tag{13}$$

Using Berbain et al.'s approach to reorder Equations 12 and 13, we can determine the equation describing the sequence bits for registers A, B and C:

$$A_{\alpha+92} = z_{\alpha+65} \oplus A_{\alpha+65} \oplus B_{\alpha+80} \oplus B_{\alpha+65} \oplus C_{\alpha+110} \oplus C_{\alpha+65}\ ,\ \alpha \geq 1$$
$$B_{\alpha+83} = z_{\alpha+68} \oplus A_{\alpha+95} \oplus A_{\alpha+68} \oplus B_{\alpha+68} \oplus C_{\alpha+112} \oplus C_{\alpha+67}\ ,\ \alpha \geq 1$$
$$C_{\alpha+110} = z_{\alpha+65} \oplus A_{\alpha+92} \oplus A_{\alpha+65} \oplus B_{\alpha+80} \oplus B_{\alpha+65} \oplus C_{\alpha+65}\ ,\ \alpha \geq 1$$

However, we can not use all three sets of equations simultaneously since they are actually equivalent. For example, calculating B_{84} requires knowledge of the sequence bit A_{96}. The equation representing A_{96} is:

$$A_{96} = z_{69} \oplus A_{69} \oplus B_{84} \oplus B_{69} \oplus C_{114} \oplus C_{69} \tag{14}$$

If we substitute Equation 14 into the equation representing the sequence bit B_{84}, we get

$$B_{84} = z_{69} \oplus z_{69} \oplus A_{69} \oplus B_{84} \oplus B_{69} \oplus C_{114} \oplus C_{69} \oplus A_{69} \oplus B_{69} \oplus C_{114} \oplus C_{69} = B_{84}$$

This does not allow us to express B_{84} in terms of the sequence bits and keystream. Therefore, our technique only allows us to express the sequence bits for a single register in terms of internal state and keystream bits. Therefore, the divide-and-conquer approach used in our analysis of Bivium-A cannot be applied here.

In the following analysis, we express the sequence bits A_i, for $i \geq 93$ using our approach, and the sequence bits of B_i, for $i \geq 84$ and C_i, for $i \geq 111$ using the sequence update function described in Section 2.1. This new system of equations starts off with 288 variables. For each of the first 66 iterations, we add three new variables and four new equations to the system of equations. A set of these equations consists of:

$$A_{\alpha+92} \oplus z_{\alpha+65} \oplus A_{\alpha+65} \oplus B_{\alpha+80} \oplus B_{\alpha+65} \oplus C_{\alpha+110} \oplus C_{\alpha+65} = 0,\ \ ,\ \alpha \geq 1$$
$$B_{\alpha+83} \oplus B_{\alpha+5} \oplus A_{\alpha+26} \oplus A_{\alpha-1} \oplus A_{\alpha}A_{\alpha+1} = 0\ ,\ \alpha \geq 1$$
$$C_{\alpha+110} \oplus C_{\alpha+23} \oplus B_{\alpha+14} \oplus B_{\alpha-1} \oplus B_{\alpha}B_{\alpha+1} = 0\ ,\ \alpha \geq 1$$
$$z_{\alpha-1} \oplus A_{\alpha+26} \oplus A_{\alpha-1} \oplus B_{\alpha+14} \oplus B_{\alpha-1} \oplus C_{\alpha+44} \oplus C_{\alpha-1} = 0\ ,\ \alpha \geq 1$$

After 66 iterations, similar to Bivium-A and Bivium-B, adding the keystream equations into the system of equations are redundant, and we can stop adding variables and equations into the system. This gives us a final system of equations consisting of 486 variables in 264 equations. The comparison of both systems of equations is shown in Table 5. Column headings are the same as Table 3. Our system of equations has less quadratic equations compared to Raddum's technique. Raddum's technique has 666 quadratic equations, compared to 132 quadratic equations in ours. The drawback of our system of equations however, is that our system of equations have a greater excess of variables over equations to Raddum's technique. In Raddum's technique, solving a system consisting of 954 variables in 954 equations should yield a unique solution. In contrast, solving our system of equations consisting of 486 variables in 264 equations using the F4 algorithm will yield 2^{222} possible solutions, which is worse than exhaustive keysearch. Reasons for this will be discussed in Section 4.4.

Technique	Linear eqn.	Quadratic eqn.	Total eqn.	K.S. (bits)	Variables	Expected solns
Raddum	288	666	954	288	954	1
Our approach	132	132	264	132	486	2^{222}

Table 5. Details for the system of equations in Trivium

3.3 New analysis on Trivium-N

We constructed the system of equations for certain Trivium-N ciphers, where N ranged from 40–70 and calculated the complexity of an attack using our approach. The details of the system of equations constructed using our approach, along with the number of paths in the BDD constructed by Schilling and Raddum [17] is shown in Table 6. Comparing the number of solutions for various Trivium-N ciphers

Cipher	Linear eqn.	Quadratic eqn.	Total eqn.	K.S. (bits)	Variables	Expected solns	Paths in BDD [17]
Trivium-40	18	18	36	18	67	2^{31}	$2^{93.77}$
Trivium-50	22	22	44	22	83	2^{39}	$2^{115.60}$
Trivium-60	26	26	52	26	99	2^{47}	$2^{140.35}$
Trivium-70	32	32	64	32	118	2^{54}	$2^{160.49}$

Table 6. Details for the system of equations in Trivium-N

in Table 6 against the number of paths in the BDD for the same Trivium-N ciphers in Schilling and Raddum's paper, we see that our approach's system of equation is less complex. For example, for $N = 70$, we have a system of equation consisting of 64 equations in 118 variables, which when solved, can give 2^{54} possible solutions, whereas Trivium-70's system of equations using Schilling and Raddum's approach has $2^{160.49}$ paths in its BDD. However, our approach is still worse than exhaustive keysearch for all Trivium-N ciphers analysed. In a subsequent paper, Schilling and Raddum [18] improved on their earlier results. For $N = 40$, they note that the maximum number of nodes in the BDD for $N = 40$ is $2^{23.69}$. However, there is insufficient detail in their description to determine the time complexity of computing the solution using the BDD.

4 Discussion

In Section 3, we investigated the possibility of recovering the initial state of Trivium-like ciphers using the approach of Berbain et al. They left the application of their technique to keystream generators which updated $q > 1$ bits of internal state at each iteration and only output $q' < q$ linear combinations of the state-bits as an open question. In this paper, we answer this open question. In this section, we will show that it may be possible to recover the initial states of some Trivium-like ciphers faster than exhaustive keysearch using our analysis. This depends on three factors:

- The relationship between j (the number of registers the keystream generation function takes as input to generate keystream) and q (the number of registers whose internal state is updated at each iteration).
- The largest index among the stages in a register used as input to the output function and whose sequence bits are expressed using Berbain et al.'s approach.
- The size of each register in Trivium-like ciphers.

4.1 Case when $j < q$

In the case of Bivium-A, where $j < q$, it was shown that it is possible to recover the initial state of the cipher using the F4 algorithm with a complexity that is less than recovering the same initial state using

Raddum's relabelling technique in conjunction with the F4 algorithm. It is shown through computer simulations that the recovery of Bivium-A's initial state using our approach can be significantly faster than Raddum's technique if the same amount of keystream is used in both attacks. The complexity of our new approach can be calculated as follows: Let T_f denote the total time taken needed for the F4 algorithm to solve both systems of equations; with T_1 and T_2 denoting the time needed for the F4 algorithm to solve the first and second system of equations respectively, and N_A denoting the number of solutions obtained in solving the first system of equations. The complexity of recovering the initial state of Trivium-like ciphers where $j < q$ is:

$$T_f = T_1 + (N_A \times T_2) \tag{15}$$

4.2 Case when $j = q$

For Bivium-B, Trivium and Trivium-N, $j = q$. In this case it is not possible to use a divide-and-conquer approach to recover the initial state of the keystream generators. To recover the initial state of these ciphers using our approach requires us to solve a single system of equations. However when $j = q$, using our approach to build this system of equations is problematic. Since a keystream equation is essentially being used to represent a sequence bit, our approach does not allow us to add an additional (keystream) equation after a certain number of iterations. After this point, the numbers of variables and equations added into the system of equations at each iteration are the same, as was demonstrated in our analysis of Bivium-B, Trivium and Trivium-N. If the numbers of variables are greater than the number of equations prior to the point at which the keystream equation becomes redundant, this system will always have more variables than equations. Adding further equations does not allow us to reduce the number of solutions obtained when the system of equations is solved and also adds to the complexity of solving these equations. In Bivium-B, Trivium and Trivium-N, trying all possible solutions obtained to determine which is the correct initial state when the system of equations is solved is worse than exhaustive keysearch. The complexity of recovering the initial state of Trivium-like ciphers where $j = q$ is $T_f = T_s + N_A$, where T_s is the time taken to solve the system of equations and N_A is the number of solutions obtained when the system of equations is solved.

4.3 Determining N_A

In our analyses of Trivium-like ciphers, the number of solutions obtained when solving a particular system of equations is found by counting the number of equations and variables added at each iteration. This set of solutions can be generalised. To do this, we need to examine the state-update function and output function used in Trivium-like ciphers. Let S_R be the total size of the registers whose sequence bit(s) is not written in terms of keystream and internal state bits *and* whose stages are used during the construction for a particular system of equations. For Trivium-like ciphers where $j < q$, this system of equations is the first system of equations. For Trivium-like ciphers where $j = q$, this system of equation is the sole system of equations obtained when we perform an algebraic analysis on them. For the register whose sequence bit is written in terms of keystream and internal state bits, let D_l denote the largest index among the stages used as input to the output function. The size of the set of solutions, N_A obtained when the system of equations is solved is:

$$N_A = 2^{S_R} \times 2^{D_l} \tag{16}$$

To illustrate this, consider the system of equations for Trivium constructed using our approach and analysed in Section 3.2. For Trivium, S_R will be the combined size of registers B and C, $S_R = 84 + 111 = 195$ and D_l is the index of the largest stage of register A (A_{27}) whose contents is used as input to Trivium's output function and whose sequence bits are expressed using Berbain et al.'s approach. That is, $D_l = 27$. The total set of solutions obtained when the respective values for S_R and D_l are substituted into Equation 16 is $2^{195} \times 2^{27} = 2^{222}$. This is the same size as the set of solutions obtained in our previous analysis in Section 3.2.

A summary of the results of our analyses of certain Trivium-like ciphers with regards to the number of equations, variables, and solutions obtained when their system of equations are formed and solved using our approach, and their relationship with S_R and D_l, is given in Table 7.

Cipher	Parameters		Step 1			Step 2		
	S_R	D_l	Eqn	Var	Soln (N_A)	Eqn	Var	Soln
Bivium-A	0	15	177	192	2^{15}	216	201	1
Bivium-B	84	27	198	309	2^{111}	N.A	N.A	N.A
Trivium	195	27	264	486	2^{222}	N.A	N.A	N.A
Trivium-50	34	5	44	83	2^{39}	N.A	N.A	N.A

Table 7. Details on systems of equations in our approaches for certain Trivium-like ciphers

4.4 Relationship between inputs to output function and N_A

Our new attack on Bivium-A can be prevented by making changes to the keystream output function. Recall the keystream generation equation for Bivium-A shown in Equation 7 and the equation representing the sequence bit B_{84}, shown in Equation 8. As was discussed in our previous analyses of Bivium-A in Section 3.1, the number of solutions obtained when the first system of equations is solved has an important effect on the effectiveness of our attack. If the number of solutions obtained is less than the total number of possible keys, it may be possible to recover the initial state of the cipher faster than exhaustive keysearch, otherwise it will be worse than exhaustive keysearch. In our approach, a keystream equation is used to represent a feedback bit, preventing us from adding keystream equation at some future point due to it being redundant. In Equation 16, we showed how it is possible to calculate the size of the set of solutions when a system of equations is solved for Trivium-like ciphers. To make an attack on such a cipher worse than exhaustive keysearch we need to modify the output function of Trivium-like ciphers such that this happens early on in the first system of equations. For example, suppose the output function of Bivium-A was, instead:

$$z_{\alpha-1} = B_{\alpha-1} \oplus B_{\alpha+82} \ , \ \alpha \geq 1$$

The sequence bit B_{84}, rewritten in terms of linear combinations of keystream and internal state bits is now $B_{84} = z_1 \oplus B_1$. We start to form the first system of equations with 84 variables. For the first iteration, two equations and one variable are added. From this point onwards, the number of variables will always be 83 more than the number of equations. If this system of equations is solved, it will give 2^{83} possible solutions. Equation 16 can also be used to determine the size of N_A. In this example, the value $D_l = 83$ and $S_R = 0$, as the contents of register A are not used during the construction of the first system of equations. Therefore, substituting the values of D_l and S_R into Equation 16 will also yield $N_A = 2^{S_R} \times 2^{D_l} = 2^0 \times 2^{83} = 2^{83}$. Since the set of solutions for the first system of equations is already larger than the number of all possible keys, it is not worthwhile to generate and solve the second system of equations. In general, for Trivium-like ciphers where $j < q$, if $D_l > 80$, the number of solutions obtained when the first system of equations is solved will be more than the total possible number of secret keys and the complexity of the entire attack will be worse than exhaustive keysearch. It should also be noted that Equation 16 only gives an indication of how large the set of solutions will be when the system of equations is solved. It does not take into account the complexity of solving the system of equations. As was shown in Equation 15, the time T_1 and T_2 needs to be taken in account when calculating the total time complexity T_f of solving the system of equation. If T_f has a complexity which is worse than exhaustive keysearch, the cipher is still considered secure against our algebraic analyses.

Conversely, assume D_l is a small value. We use Trivium to illustrate how this change reduces the number of solutions obtained when the system of equations was solved. Assume that the output function for Trivium is instead:

$$z_{\alpha-1} = A_\alpha \oplus A_{\alpha-1} \oplus B_{\alpha+14} \oplus B_{\alpha-1} \oplus C_{\alpha+44} \oplus C_{\alpha-1} \ , \ \alpha \geq 1$$

We use our approach to represent the new sequence bit equations for register A. In this case, $D_l = 1$ and $S_R = 111 + 84 = 195$. Substituting these values into Equation 16 gives the following value $N_A = 2^{S_R} \times 2^{D_l} = 2^{195} \times 2^1 = 2^{196}$. Recall from Section 3.2 that the size of the set of solutions when the system of equations formed using our approach is solved was 2^{222}. By changing the inputs to the output function, the size of the set of solutions obtained is 2^{196}, a decrease by a factor of 2^{26} in the number of possible solutions. However, examining all 2^{196} candidate solutions to determine the correct initial state is worse than exhaustive keysearch.

For Trivium-like ciphers where $j = q$, Equation 16 allows us to determine the number of solutions obtained when the system of equations is solved. For Trivium, the size of each register is greater than 80 bits. Therefore, changing the position of the stages used as input to the output function such that D_l is as small as possible may not be sufficient for our algebraic analysis to succeed, as 2^{S_R} will always be larger than the number of possible secret keys. Similar to the case when $j < q$, Equation 16 does not take into account the complexity of solving the system of equations. Here the time T_s needs to be taken in account when calculating the total time complexity T_f of solving the system of equation.

4.5 Relationship between size of registers and N_A

The size of the registers used in the formation of the first system of equations can have an effect on the number of solutions obtained for the first system of equations. For example, suppose that Trivium's three registers, A, B, and C had lengths 198, 45 and 45 bits, where the output function used is the same as the original Trivium proposal. During the construction of the system of equations, we rewrite the feedback bits of register A in terms of sequence bits and keystream bits. Therefore, $D_l = 27$. For this particular case, $S_R = 45 + 45$, and $N_A = 2^{90} \times 2^{27} = 2^{117}$. Recall from Section 3.2 that the expected size of the set of solutions obtained when the original system of equations using our approach is solved was 2^{222}. By changing the size of the registers, the expected size of the set of solutions obtained is 2^{117}, a 2^{105} factor decrease in the number of possible solutions. However, going through all 2^{117} possible solutions to determine which is the correct initial state remains worse than exhaustive keysearch.

5 Conclusion

This paper analysed Trivium-like ciphers using the approach of Berbain et al. Our analysis answers their open question regarding the possibility of extending the technique to keystream generators which update $q > 1$ bits of internal state at each iteration but only output $q' < q$ linear combinations of the state bits. We have also presented an alternative approach of representing the output sequence from a register in Trivium-like ciphers as a linear combination of initial state bits and keystream bits.

In particular, we demonstrated a new algebraic attack on Bivium-A. Our approach requires less time and memory than previous techniques which use the F4 algorithm to recover Bivium-A's initial state. However, applying the new approach to Bivium-B, Trivium-N and Trivium is worse than exhaustive key search. We also demonstrated that if $j < q$, it may be possible to mount a divide-and-conquer algebraic attack which can recover the initial state of the keystream generator, with less complexity than exhaustive keysearch over the entire keyspace.

For Trivium-like ciphers, we showed that the size of the registers used in the construction of the first system of equations, and the selection of stages used as input to the output function can affect the number of solutions. For Bivium-A, changing the value of D_l can change the complexity of our algebraic attack. This can be worse than exhaustive keysearch. In the case of Bivium-B, Trivium and Trivium-N, even if the value of D_l is small, the complexity of our algebraic attack is still worse than exhaustive keysearch as the value of S_R is larger than the keysize.

Acknowledgments. The authors would like to thank the anonymous reviewers from two conferences for their helpful comments. Computational resources and services used in this work were provided by the HPC and Research Support Unit, Queensland University of Technology.

References

1. Akers, S.B.: Binary Decision Diagrams. IEEE Transactions on Computers 27(6), 509–516 (1978)
2. Aumasson, J.P., Dinur, I., Meier, W., Shamir, A.: Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In: Dunkelman, O. (ed.) Fast Software Encryption (FSE 2009). Lecture Notes in Computer Science, vol. 5665, pp. 1–22. Springer (2009)

3. Berbain, C., Gilbert, H., Joux, A.: Algebraic and Correlation Attacks against Linearly Filtered Non Linear Feedback Shift Registers. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) Selected Areas in Cryptography (SAC 2008). Lecture Notes in Computer Science, vol. 5381, pp. 184–198. Springer (2009)
4. Bernstein, D.: A reformulation of TRIVIUM. Submission to Phorum: ECRYPT forum (2006), Available from `http://www.ecrypt.eu.org/stream/phorum/read.php?1,448`
5. Borghoff, J., Knudsen, L.R., Matusiewicz, K.: Hill Climbing Algorithms and Trivium. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) Selected Areas in Cryptography (SAC 2010). Lecture Notes in Computer Science, vol. 6544, pp. 57–73. Springer (2011)
6. Bosma, W., Cannon, J.J., Playoust, C.: The Magma algebra system. I. The user language. Journal of Symbolic Computation 24(3-4), 235–265 (1997), Computational algebra and number theory (London, 1993)
7. Canniére, C.D., Preneel, B.: Trivium. In: Robshaw, M.J.B., Billet, O. (eds.) New Stream Cipher Designs: The eSTREAM Finalists. Lecture Notes in Computer Science, vol. 4986, pp. 244–266. Springer (2008)
8. Courtois, N.T., Meier, W.: Algebraic Attacks on Stream Ciphers with Linear Feedback. In: Biham, E. (ed.) Advances in Cryptology — EUROCRYPT 2003. Lecture Notes in Computer Science, vol. 2656, pp. 345–359. Springer (2003)
9. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: Joux, A. (ed.) Advances in Cryptology — EUROCRYPT 2009. Lecture Notes in Computer Science, vol. 5479, pp. 278–299. Springer (2009)
10. Eibach, T., Pilz, E., Völkel, G.: Attacking Bivium Using SAT Solvers. In: Büning, H.K., Zhao, X. (eds.) Theory and Applications of Satisfiability Testing — SAT 2008. Lecture Notes in Computer Science, vol. 4996, pp. 63–76. Springer (2008)
11. Eibach, T., Pilz, E., Völkel, G.: Optimising Gröbner Bases on Bivium. Mathematics in Computer Science 3(2), 159–172 (2010)
12. European Network of Excellence for Cryptology: The eSTREAM Project, Available from `http://www.ecrypt.eu.org/stream/index.html`
13. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases F4. Journal of Pure and Applied Algebra 139, 61–88 (1999)
14. Fouque, P.A., Vannet, T.: Improving Key Recovery to 784 and 799 rounds of Trivium using Optimized Cube Attacks. In: Moriai, S. (ed.) Fast Software Encryption (FSE 2013). To appear.
15. McDonald, C., Charnes, C., Pieprzyk, J.: An Algebraic Analysis of Trivium Ciphers based on the Boolean Satisfiability Problem. Cryptology ePrint Archive, Report 2007/129 (2007), Available from `http://eprint.iacr.org/2007/129`
16. Raddum, H.: Cryptanalytic Results on Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/039 (2006), Available from `http://www.ecrypt.eu.org/stream/papersdir/2006/039.ps`
17. Schilling, T.E., Raddum, H.: Analysis of Trivium Using Compressed Right Hand Side Equations. In: Kim, H. (ed.) Information Security and Cryptology (ICISC 2011). Lecture Notes in Computer Science, vol. 7259, pp. 18–32. Springer (2012)
18. Schilling, T.E., Raddum, H.: Solving Compressed Right Hand Side Equation Systems with Linear Absorption. In: Helleseth, T., Jedwab, J. (eds.) Sequences and Their Applications (SETA 2012). Lecture Notes in Computer Science, vol. 7280, pp. 291–302. Springer (2012)
19. Simonetti, I., Perret, L., Faugr̀e, J.C.: Algebraic Attack Against Trivium. In: First International Conference on Symbolic Computation and Cryptography, SCC 2008. pp. 95–102. LMIB (2008), available from `http://www-salsa.lip6.fr/~jcf/Papers/SCC08c.pdf`

A Analysis of Bivium-B

In our analysis of Bivium-B, we express the sequence bits of register A using our approach and the sequence bits produced by register B using the equation described in Equation 6. We add three equations and two variables at each iteration into the system of equations for the first 66 iterations. A set of these equations consists of:

$$\begin{aligned}
&A_{\alpha+92} \oplus z_{\alpha+65} \oplus A_{\alpha+65} \oplus B_{\alpha+80} \oplus B_{\alpha+65} = 0, \ , \alpha \geq 1 \\
&B_{\alpha+83} \oplus B_{\alpha+5} \oplus A_{\alpha+26} \oplus A_{\alpha-1} \oplus A_{\alpha}A_{\alpha+1} = 0 \ , \alpha \geq 1 \\
&z_{\alpha-1} \oplus A_{\alpha+26} \oplus A_{\alpha-1} \oplus B_{\alpha+14} \oplus B_{\alpha-1} = 0 \ , \alpha \geq 1
\end{aligned}$$

Similar to our approach used in solving Bivium-A in Section 3.1, we can only add equations describing keystream equations up to z_{65} as the equations describing the keystream equations from z_{66} are redundant. Therefore, after 66 iterations, we have formed our system of equations for Bivium-B. This system consist of 198 equations in 309 variables. Solving this system of equations using the F4 algorithm will give 2^{111} possible solutions, which is worse than exhaustive keysearch. The comparison of both systems of equations is shown in Table 8. Column headings are the same as Table 3. Our system of equations has less quadratic equations compared to Raddum's technique: Raddum's technique gives 222 quadratic equations, compared to 66 quadratic equations in ours. The drawback of our system of

Technique	Linear eqn.	Quadratic eqn.	Total eqn.	K.S. (bits)	Variables	Expected solns
Raddum	177	222	399	177	399	1
Our approach	132	66	198	132	309	2^{111}

Table 8. Details for the system of equations in Bivium-B

equations however, is that we have a greater excess of variables over equations. In Raddum's technique, solving a system consisting of 399 variables in 399 equations will yield a unique solution. In contrast, solving the system of equations in our approach, which consists of 309 variables in 288 equations, using the F4 algorithm will yield 2^{111} possible solutions (initial states), which is worse than exhaustive keysearch.