# Attacking Bivium with MiniSat

Cameron McDonald, Chris Charnes and Josef Pieprzyk

Centre for Advanced Computing, Algorithms and Cryptography,
Department of Computing, Macquarie University
{cmcdonal,charnes,josef}@ics.mq.edu.au

**Abstract.** Trivium is a stream cipher candidate of the eStream project. It has successfully moved into phase two of the selection process and is currently in the focus group under the hardware category. As of yet there has been no attack on Trivium faster than exhaustive search.
Bivium-A and Bivium-B are simplified versions of Trivium that are built on the same design principles. Their design serves as a tool for investigating Trivium-like ciphers with a reduced complexity. This provides an insight into effective methods of attack that could be extended to Trivium. There have been successful attempts in the cryptanalysis of Bivium ciphers.
This paper focuses on a type of guess and determine attack that utilises the satisfiability solver MiniSat. Given a minimal amount of keystream MiniSat determines the remaining unknown state, leading to complete key recovery.

## 1 Introduction

The eStream project [6] was established with the aim of finding a cryptographic primitive for a stream cipher. There are two main categories in the call - software encryption and hardware encryption. The two main goals stated in the specification criteria are that the cipher is secure and fast. Of the 34 proposals which were received, some have successfully passed the introductory phase one and continued to the second phase. The call is now in phase three of the evaluation process.

The ciphers which exhibited weaknesses in security were dropped in the first two phases. A handful of the proposals which gained special recognition for their design have been placed within a Focus group. One such proposal by Cannière and Preneel [4] is Trivium - a design which was optimized for hardware encryption. The design of Trivium is simple and elegant.

Although this design has attracted much interest from cryptanalysts it remains unbroken. It is well known that breaking a good cipher should require 'as much work as solving a system of simultaneous equations in a large number of unknowns of a complex type'; Shannon [7]. The structure of Trivium can be directly expressed as a system of sparse quadratic equations over $GF(2)$. However, solving systems of quadratic equations - known as the MQ problem, is in general a NP-hard problem.

In this paper we convert the problem of solving a system of non-linear equations over $GF(2)$ into a corresponding SAT-problem. That is, we convert the algebraic equations into a propositional formula in conjunctive normal form (CNF). We use a SAT-solver to solve the SAT-problem, which allows us under certain conditions to recover the key. Our method is based on multiple strategies for converting the algebraic normal form (ANF) of the equations describing the cipher into a propositional formula.

We need to guess a subset of the state variables in order to reduce the complexity of the system, before it can be solved by the SAT-solver MiniSat [9]. The solution returned by MiniSat is the remaining unknown state variables. Once the entire state is known, the cipher is clocked backwards to recover the key. The distinguishing feature of this type of attack is that only minimal amounts of keystream need to be observed in order to recover the key.

In what follows we focus on the cipher Bivium [5] to ascertain the effectiveness of this method, and how it may be applied to the cryptanalysis of Trivium.

## 2 Previous results

Raddum [5] introduced a new method of solving systems of sparse quadratic equations and applied it to the cryptanalysis of Trivium. The complexity of this attack on Trivium is $O(2^{162})$, which is much worse than an exhaustive key search.

Raddum introduced two simplified versions of Trivium: Bivium-A and Bivium-B. The first version was broken 'in about one day', and the second version required approximately $2^{56}$ seconds.

Maximov and Biryukov [2] used a different approach to solve the system of algebraic equations describing Trivium by 'guessing' the value of the state bits (or the products of some state bits). In certain cases guessing reduces the system of quadratic equations to a system of linear equations that can be solved (for example by Gaussian elimination). The complexity of this attack is $O(c \cdot 2^{83.5})$ for Trivium and $O(c \cdot 2^{36.1})$ for Bivium. The constant $c$ is the time taken to solve a system of sparse linear equations. Maximov and Biryukov do not give a complexity estimate of the constant $c$ in [2]. However in [3] an estimate of the constant for Bivium is stated as $O(2^{16.2})$.

## 3 Description of Bivium

Let $x_i^t \in GF(2)$ denote the $i$-th variable of vector $x$ at clock time $t$. Addition and multiplication, denoted by $+$ and $\cdot$ respectively, are done over $GF(2)$.

Bivium consists of a non-linear feedback shift register (NLFSR) coupled with a linear filter function (LF). The NLFSR operates on a 177-bit state, denoted by $(s_1^t, \ldots, s_{177}^t)$. The LF takes a linear combination of the state to produce the keystream. Each clock of the cipher involves updating two bits of the state and outputting one bit of keystream, denoted by $z_i^{(t)}$. The cipher continues to run until the required number of keystream bits are produced.

The following algorithm is a full description of the keystream generation:

**for** $i = 1$ to $N$ **do**
$t_1 \leftarrow s_{66} + s_{93}$
$t_2 \leftarrow s_{162} + s_{177}$
$z_i \leftarrow t_1 + t_2$
$t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$
$t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{69}$
$(s_1, s_2, \ldots, s_{93}) \leftarrow (t_2, s_1, \ldots, s_{92})$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
**end for**

Like Trivium, Bivium incorporates a Key and IV setup stage, where the cipher is clocked a number of times to initialise the state. Our analysis does not use the initialisation process and this will not be discussed further.

## 4 Attacking Bivium

Our attack has two separate stages - a pre-attack computation stage and an attack stage. The pre-attack computation involves the following steps:

1. Produce a system of equations that describe Bivium.
2. Convert the system of ANF equations into a system of clauses in CNF.

Each of the above steps has its own strategy which will be discussed below.

The attack stage comprises the following repeated steps:

1. Guess $m$ state variables and substitute for these values into the propositional formula.
2. Find a solution that satisfies all the clauses in the resulting propositional formula.

Again different strategies will affect the outcome.

The time complexity of the attack depends on the number of state variables guessed, and the time taken to find a satisfiable solution. If the time taken to find a satisifiable solution is $T_m$ clock-cycles for a $m$-bit guess, then on the average the complete state can be recovered in $T_m \cdot 2^{m-1}$ clock-cycles.

## 5 Pre-Attack stage

### 5.1 System of Bivium equations

A description on the system of equations representing Bivium was given in [5]. Where each clock introduces two new variables and three new equations. A minimum of 177 bits of keystream are required to produce a unique solution [5].

It is possible to express the Bivium equations in terms of only one initial state. This has the advantage that there are less variables in the system of equations. However after certain periods the density and the degree increases. A subset of the keystream equations is listed in the Appendix 10.

### 5.2 Converting ANF to CNF

The input to MiniSat is in the DIMACS graph format [8] which specifies that propositional formulae are in CNF.

The running time complexity of MiniSat depends on the number of variables and clauses in the propositional formula. Converting an equation in ANF with $\alpha$ monomials produces $2^{\alpha-1}$ disjunctive clauses in the CNF; see [1].

Different strategies are needed to handle clause expansion introduced by the conversion process:

1. The 'cutting' number - split the equations to reduce the number of monomials per equation (density).
2. Gaussian elimination pre-processing scheme.
3. Replace common groups of monomials by new variables.

The cutting method [1] was used to reduce clause production in the conversion of the equations to CNF. There is a tradeoff between the sparsity of the equations, and the production of new variables since each cut introduces a new variable. Experiments established that the 'optimal' cutting number for the Bivium equations is five.

It was stated in [1], that a preprocessing scheme based on Gaussian elimination reduces the complexity of the system. However in our setting this procedure was superflous, as it increased the density of the equations and degraded the performance of the SAT-solver.

The last method searches for groups of monomials which are repeated in multiple equations. New variables are introduced which represent sums of groups of monomials. Packaging the variables in this way reduces the density of the equations and minimises the introduction of new variables.

For example, consider the following Bivium equations:

$$s_{66} + s_{93} + s_{162} + s_{177} + z_1 = 0,$$
$$s_{27} + s_{69} + s_{96} + s_{111} + s_{162} + s_{175} \cdot s_{176} + s_{177} + z_{67} = 0.$$

The total number of clauses produced by these equations is $2^4 + 2^7 = 144$. However in Bivium the feedback is directly related to the output function and hence $(s_{162} + s_{177})$ occurs in both equations.

Introducing a new variable $a_1 = s_{162} + s_{177}$ gives a new system of equations:

$$s_{66} + s_{93} + a_1 + z_1 = 0,$$
$$s_{27} + s_{69} + s_{96} + s_{111} + s_{175} \cdot s_{176} + a_1 + z_{67} = 0,$$
$$s_{66} + s_{177} + a_1 = 0.$$

Now the total number of clauses produced by these equations is $2^3+2^6+2^2 = 76$. Clause reduction was achieved at the cost of introducing a new variable. The limit on how many new variables should be introduced, and hence the total reduction of clauses was determined experimentally.

## 6 Attack stage

For each guess of the state variable MiniSat will return either SAT for a satisifiable assignment, or UNSAT for an unsatisfiable assignment. SAT is returned if the guess is correct and an assignment of values to the variables is obtained. At this point the whole state of Bivium is recovered. If UNSAT is returned the guess was incorrect and another guess is tried.

### 6.1 Guessing state variables

The strategies that were used to guess subsets of state bits include:

1. Select a subset of the variables occurring with the highest frequency in the set of equations - reduces the maximum number of monomials per variable guessed.
2. Select a subset of the variables occurring in the quadratic monomials - reduces the non-linearity of the system.
3. Select a subset of the quadratic monomials - (guess the product of two variables [2].).
4. Select alternate variables - all quadratic monomials are the products of two adjacent bits (see example below).

Consider the quadratic monomials in the following subset of Bivium equations:

$$
\begin{aligned}
s_{24} + s_{91} \cdot s_{92} + s_{93} + s_{108} + s_{159} + s_{171} + s_{172} \cdot s_{173} + s_{174} + z_{70} = 0,\\
s_{23} + s_{90} \cdot s_{91} + s_{92} + s_{107} + s_{158} + s_{170} + s_{171} \cdot s_{172} + s_{173} + z_{71} = 0,\\
s_{22} + s_{89} \cdot s_{90} + s_{91} + s_{106} + s_{157} + s_{169} + s_{170} \cdot s_{171} + s_{172} + z_{72} = 0,\\
s_{21} + s_{88} \cdot s_{89} + s_{90} + s_{105} + s_{156} + s_{168} + s_{159} \cdot s_{170} + s_{171} + z_{73} = 0.
\end{aligned}
$$

To reduce all the quadratic terms to linear terms we need only guess the variables $\{s_{89}, s_{91}, s_{170}, s_{172}\}$. Thus eight quadratic terms become linear by guessing four of the variables.

### 6.2 Solving the equations

To solve the system of equations arising from Bivium we used the satisfiability solver MiniSat [9]. Because the complexity of the MiniSat algorithm is difficult to analyse we developed several solution strategies based on experiments. These strategies resulted in a large spread of MiniSat computing times making it difficult to pinpoint the best strategy to use in the average case. (We hope to report more on this in the future.)

**Hamming weights of solutions** We observed that the Hamming weights of the solutions (state vectors) affected the performance of MiniSat. Figure 1 plots the average running time of MiniSat against the Hamming weight of random states when 40 bits of the state are guessed. In the graph we distinguish between times that MiniSat requires to return SAT or UNSAT.

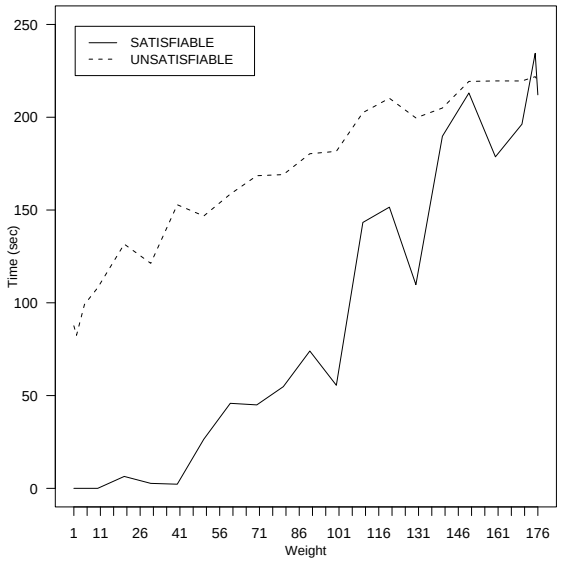


**Fig. 1.** MiniSat Time vs. State Weight

From this graph we derive two important conclusions which affect the average running time of MiniSat:

1. MiniSat returns faster when the guess is correct.
2. MiniSat running time increases with weight.

Therefore during the attack stage a limit should be placed on the running time of MiniSat. If this limit is reached the guess is most likely incorrect and another guess should be tried.

Furthermore, in order to evaluate an average running time for MiniSat we observed how the probability distribution of the state varies with respect to the weight. The weights of random states over multiple clock times are given in Figure 2.

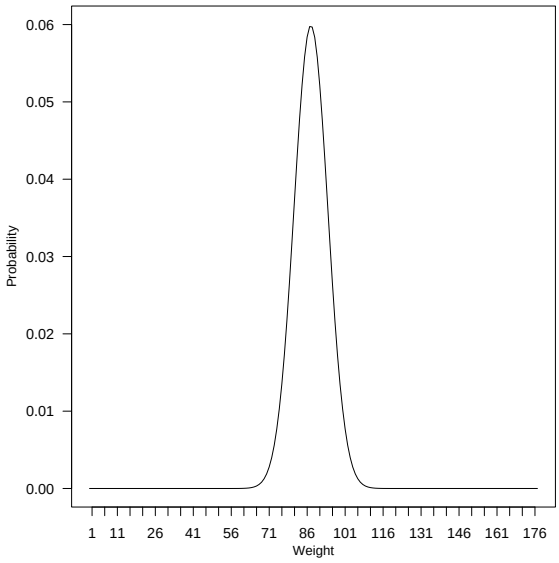


**Fig. 2.** Probability distribution on state weight

## 7 Total attack time

Figure 3 displays the total attack times for different values of $m$ observed experimentally.

The optimal attack time on Bivium occurs when 34-bits of the state are guessed. The probability that a guess is correct is $2^{-34}$, thus around $2^{34}$ values will be tried before finding the correct guess. The average MiniSat running time on this propositional formula is $440 \approx 2^{8.7}$ seconds. The overall average attack time on Bivium is $2^{34} \cdot 2^{8.7} = 2^{42.7}$ seconds. Raddum states [5] that approximately $2^{13.3}$ keys can be searched exhaustively in 1 second, which gives $2^{56}$ as the approximate complexity of our attack.

### 7.1 Minimum running time of MiniSat

In our experiments we encountered many instances where MiniSat completed in a fraction of the average time. In 10% of the tests there was at least one case which was solved in only 5% of the average time. Hence if we run parallel MiniSat processes on a minimum of 10 different blocks of keystream, we can expect to encounter an instance that is solved in 5% of the average time. The best complexity achieved by this strategy for a 34-bit guess is $2^{52}$.

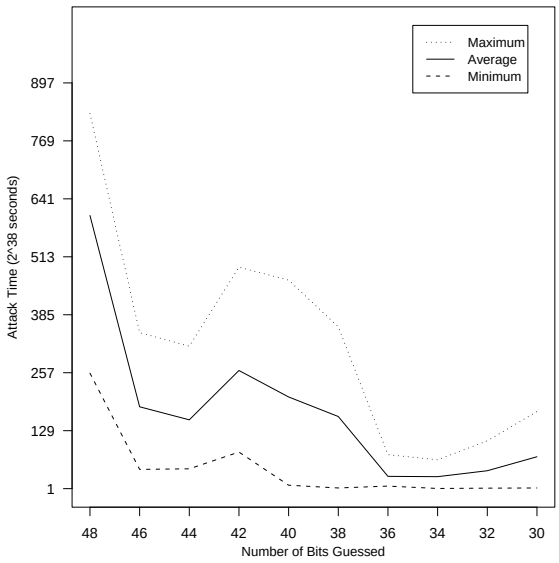


**Fig. 3.** Attack time vs. Number of bits guessed

## 8 Bivium-A

Bivium-A is a simplified version of Trivium which was initially analysed by Raddum [5]. The design of Bivium-A differs from Bivium (Bivium-B) in the linear output filter. The output is

$$z_i \leftarrow t_1,$$

a combination of only two state variables. This leads to keystream equations of the type:

$$z_i + s_j + s_k = 0,$$

where $z_i \in \{0, 1\}$.

After substituting the observed keystream these simple keystream relations between the two state variables give a large reduction in the number of variables in the propositional formula. Compared to Bivium, this propositional formula can be solved directly by MiniSat without the need for guessing state variables.

The system of algebraic equations for Bivium-A is converted to CNF by cutting the equations into blocks of five. MiniSat was run on a sample of 100 random states and the results obtained are plotted in Figure 4.

The average running time to recover the state (and hence the key) of Bivium-A is 21 seconds. The minimum observed running time was 0.5 seconds and the maximum was 60 seconds. This can be compared to Raddum's algorithm which recovers the state of Bivium-A in 'about a day' [5].

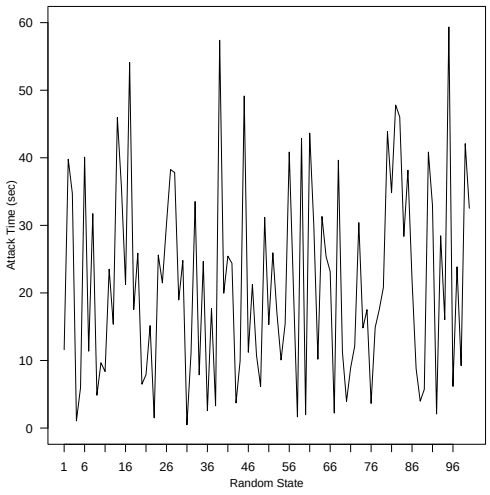


**Fig. 4.** Attack time for Bivium-A

## 9 Summary of our results

Due to the unpredictable behaviour and complexity of the MiniSat algorithm, the results obtained in this paper are derived from experiments. We estimate the best complexity for the attack on Bivium to be $O(2^{52})$. This algebraic attack recovers the private key after observing only 1770 bits of keystream. Bivium-A is completely broken within 21 seconds on the average requiring only 177 bits of keystream.

## References

1. G. Bard, N. Courtois and C. Jefferson. *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.
2. A. Maximov and A. Biryukov. *Two Trivial Attacks on Trivium*. Cryptology ePrint Archive, Report 2007/021, 2007.
3. A. Maximov. private communication 14/3/07.
4. C. De Cannière and B. Preneel. *TRIVIUM - a stream cipher construction inspired by block cipher design principles*. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/030, 2005. `http://www.ecrypt.eu.org/stream/trivium.html`
5. H. Raddum. *Cryptanalytic results on TRIVIUM*. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/039, 2006. `http://www.ecrypt.eu.org/stream`
6. eSTREAM: ECRYPT Stream Cipher Project `http://www.ecrypt.eu.org/stream/`
7. C. Shannon. *Communication theory of secrecy systems*. Bell System Technical Journal 28, 1949.

8. DIMACS http://www.cs.ubc.ca/ hoos/SATLIB/Benchmarks/SAT/satformat.ps
9. MiniSat 2.0 http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/MiniSat.html

## 10 Appendix

$$s_{66} + s_{93} + s_{162} + s_{177} + z_1 = 0,$$
$$s_{65} + s_{92} + s_{161} + s_{176} + z_2 = 0,$$
...
$$s_1 + s_{28} + s_{97} + s_{112} + z_{66} = 0,$$
$$s_{27} + s_{69} + s_{96} + s_{111} + s_{162} + s_{175} \cdot s_{176} + s_{177} + z_{67} = 0,$$
$$s_{26} + s_{68} + s_{95} + s_{110} + s_{161} + s_{174} \cdot s_{175} + s_{176} + z_{68} = 0,$$
$$s_{25} + s_{67} + s_{94} + s_{109} + s_{160} + s_{173} \cdot s_{174} + s_{175} + z_{69} = 0,$$
$$s_{24} + s_{91} \cdot s_{92} + s_{93} + s_{108} + s_{159} + s_{171} + s_{172} \cdot s_{173} + s_{174} + z_{70} = 0,$$
$$s_{23} + s_{90} \cdot s_{91} + s_{92} + s_{107} + s_{158} + s_{170} + s_{171} \cdot s_{172} + s_{173} + z_{71} = 0,$$
...
$$s_{10} + s_{77} \cdot s_{78} + s_{79} + s_{94} + s_{145} + s_{157} + s_{158} \cdot s_{159} + s_{160} + z_{84} = 0,$$
$$s_9 + s_{66} + s_{76} \cdot s_{77} + s_{78} + s_{91} \cdot s_{92} + s_{93} + s_{144} + s_{156} + s_{157} \cdot s_{158} + s_{159} + s_{171} + z_{85} = 0,$$
$$s_8 + s_{65} + s_{75} \cdot s_{76} + s_{77} + s_{90} \cdot s_{91} + s_{92} + s_{143} + s_{155} + s_{156} \cdot s_{157} + s_{158} + s_{170} + z_{86} = 0,$$
...
$$s_1 + s_{58} + s_{68} \cdot s_{69} + s_{70} + s_{83} \cdot s_{84} + s_{85} + s_{136} + s_{148} + s_{149} \cdot s_{150} + s_{151} + s_{163} + z_{93} = 0,$$
$$s_{57} + s_{67} \cdot s_{68} + s_{82} \cdot s_{83} + s_{84} + s_{135} + s_{147} + s_{148} \cdot s_{149} + s_{150} + s_{175} \cdot s_{176} + s_{177} + z_{94} = 0,$$
...
$$s_{16} + s_{26} \cdot s_{27} + s_{41} \cdot s_{42} + s_{43} + s_{94} + s_{106} + s_{107} \cdot s_{108} + s_{109} + s_{134} \cdot s_{135} + s_{136} + z_{135} = 0,$$
$$s_{15} + s_{25} \cdot s_{26} + s_{40} \cdot s_{41} + s_{42} + s_{66} + s_{91} \cdot s_{92} + s_{93} + s_{105} + s_{106} \cdot s_{107} + s_{108} + s_{133} \cdot s_{134} + s_{135} + s_{171} + z_{136} = 0,$$
$$s_{14} + s_{24} \cdot s_{25} + s_{39} \cdot s_{40} + s_{41} + s_{65} + s_{90} \cdot s_{91} + s_{92} + s_{104} + s_{105} \cdot s_{106} + s_{107} + s_{132} \cdot s_{133} + s_{134} + s_{170} + z_{137} = 0,$$
...
$$s_4 + s_{14} \cdot s_{15} + s_{29} \cdot s_{30} + s_{31} + s_{55} + s_{80} \cdot s_{81} + s_{82} + s_{94} + s_{95} \cdot s_{96} + s_{97} + s_{122} \cdot s_{123} + s_{124} + s_{160} + z_{147} = 0,$$
$$s_3 + s_{13} \cdot s_{14} + s_{28} \cdot s_{29} + s_{30} + s_{54} + s_{66} + s_{79} \cdot s_{80} + s_{81} + s_{91} \cdot s_{92} + s_{93} + s_{94} \cdot s_{95} + s_{96} + s_{121} \cdot s_{122} + s_{123} + s_{159} + s_{171} + z_{148} = 0$$