An Algebraic Analysis of Trivium Ciphers based on the Boolean Satisfiability Problem

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 three of the selection process under the hardware category. No attacks faster than the exhaustive search have so far been reported on Trivium. Bivium-A and Bivium-B are simplified versions of Trivium that are built on the same design principles but with two registers. The simplified design is useful in investigating Trivium type ciphers with a reduced complexity and provides insight into effective attacks which could be extended to Trivium. This paper focuses on an algebraic analysis which uses the boolean satisfiability problem in propositional logic. For reduced variants of the cipher, this analysis recovers the internal state with a minimal amount of keystream observations.

Key words: Algebraic Analysis, Boolean Satisfiability, Trivium, eStream

1 Introduction

The eStream project [7] 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 should be secure and fast. Of the 34 proposals received, some have successfully passed the introductory phase and continued to the second phase.

One such proposal by Cannière and Preneel [5] 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. The structure of Trivium can be directly expressed as a system of sparse quadratic equations over $\mathbb{F}_2$. However, solving systems of quadratic equations - known as the MQ problem, is in general a NP-hard problem.

In this paper we consider the problem of solving a system of non-linear equations over $\mathbb{F}_2$ as a corresponding SAT-problem of propositional logic (see Section 4). That is, we convert the algebraic equations describing the cipher into a propositional formula in conjunctive normal form (CNF). We use a SAT-solver to solve the resulting SAT-problem, which allows us under certain conditions to recover the key.

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 a SAT-solver. The solution returned by the SAT-solver is the remaining unknown state variables. Once the entire state is known, the cipher is clocked backwards to recover the key. The characteristic feature of this type of attack is that only minimal amounts of observed keystream are required in order to recover the key.

2 Previous results

Raddum [6] 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 [3] used a different approach to solve the system of algebraic equations describing Trivium by 'guessing' the value of specific state bits (or the products of 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 equations. Maximov and Biryukov do not give a complexity estimate of the constant c in [3]; in [4] the constant for Bivium was stated as $O(2^{16.2})$.

3 MQ Problem

Let $\mathbb{F}_2$ denote the Galois Field of order two and $P = \mathbb{F}_2[x_1, x_2, \ldots, x_n]$ denote the polynomial ring over $\mathbb{F}_2$ in n variables. Let N be the number of monomials in P. Let $f \in P$, the *algebraic normal form* (ANF) of f is defined as:

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{N} c_i \cdot x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n} = 0,$$

where the coefficients $c_i \in \mathbb{F}_2$. Let $c = (c_1, ..., c_N)$ be the coefficient vector of f. We define the *density* of f as the number of non-zero terms in the ANF (the Hamming weight of c). A system of equations $S = \{f_1, f_2, \ldots, f_m\}$, where $f_i \in P$ defines an instance of the *MQ problem* (the "problem" is to find a solution for S - a set of values that evaluates to 0 for each f_i).

Solving an instance of the MQ problem is regarded as a difficult problem which has motivated ongoing research in Gröbner basis methods. We compared these methods with the SAT approach (Section 4) by conducting a number of experiments with the F4 algorithm implemented in Magma [2]. The outcomes of these experiments are summarised in Table 9.

4 SAT Problem

The SAT-problem in propositional logic is represented by n propositional variables $x_1, x_2, \ldots, x_n$ which are assigned the values 1 or 0, representing true or false respectively. A *literal* is a propositional variable or the negation of a propositional variable. A *propositional formula* is an expression combining propositional variables and the logical operators: *AND, OR, NEGATION*. A *clause* is a disjunction (OR) of literals and a CNF formula is conjunction (AND) of clauses. The problem of determining if all the propositional variables in a propositional formula can be assigned values so that the formula evaluates to *true* is known as the SAT-problem. If the propositional variables can be assigned such values, the formula is *satisfied*. In which case all the clauses in the formula are satisfied.

A SAT solver is employed to solve the SAT problem. If there is no solution to the problem, the solver returns UNSATISFIABLE. If a solution exists, the solver returns SATISFIABLE along with the solution. Depending on the formula, there may be more than one solution. In our experiments we compare three different solvers: MiniSat [10], RSat [11] and PicoSat [12]. These solvers have been chosen because of their performance at the last international SAT competition in 2007 [13] and SAT-Race 2006 [14]. The input to these solvers is in the DIMACS format [9].

5 Converting MQ to SAT

The conversion process is explained in [1]. There are two main steps:

1. Convert the polynomial system to a linear system. This involves the substitution of non-linear monomials by new variables.
2. Convert the linear system to an expression in CNF. The CNF includes clauses representing the substitutions made in step 1.

A linear equation with α variables (α monomials), converts to a CNF containing $2^{(\alpha-1)}$ clauses: in the resulting form each clause contains an odd number of negated literals. In a nonlinear equation each unique quadratic monomial introduces one new variable. This substitution creates 3 extra clauses in the CNF. In general, an ANF equation with α variables and β monomials (γ of which are quadratic) converts to a CNF with $2^{(\beta-1)} + 3 \cdot \gamma$ clauses and $\alpha + \gamma$ literals. The total number of clauses in the CNF is determined by:

- Number of variables.
- Density of equations.
- Number of unique quadratic monomials.

Example: To convert the quadratic equation $x_1 + x_2 + x_1x_2 = 0$, to CNF. First replace the quadratic monomial (x_1x_2) by a new variable (x_3), resulting in:

$$x_1 + x_2 + x_1x_2 = 0 \Leftrightarrow \begin{cases} x_1 + x_2 + x_3 & = 0 \\ x_1x_2 & = x_3 \end{cases}$$

The first equation converts to a CNF containing four clauses. Three of the clauses arise from combinations of the three literals with one of the literals negated. The remaining clause contains the negation of all three literals. That, is

$$x_1 + x_2 + x_3 = 0 \Leftrightarrow (x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_1 \vee \bar{x_2} \vee x_3) \\ \wedge (\bar{x_1} \vee x_2 \vee x_3) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3})$$

The quadratic equation converts to the following CNF containing three clauses. This equivalence is easily established by constructing a truth table.

$$x_1 x_2 = x_3 \Leftrightarrow (x_1 \vee \bar{x_3}) \wedge (x_2 \vee \bar{x_3}) \wedge (\bar{x_1} \vee \bar{x_2} \vee x_3)$$

To solve a CNF using the SAT solvers listed it is required that the CNF be in the DIMACS graph format. Writing the above CNF in DIMACS format produces:

```
p cnf 5 9
1 0
-2 0
-3 4 5 0
3 -4 5 0
3 4 -5 0
-3 -4 -5 0
-5 3 0
-5 4 0
5 -3 -4 0
```

In the appendix we present an example of this method as it applies to Bivium-A Section 8. This cipher can be described by a system of 396 equations in 396 variables. Of these we included 6 representative equations, the first three from the linear filter function and the second three from the non-linear feedback shift register. We also give the CNF for these six equations in DIMACS form. These equations should provide the reader with enough detail to recreate our results.

6 Attack Description

Our attack involves the following steps.

1. Construct a system of ANF equations describing the cipher.
2. Convert the system of ANF equations to CNF and express in DIMACS format.
3. Use a SAT solver to find a solution to the problem (if it exists).

A system of ANF equations describing the cipher is obtained by translating each component of the algorithm into an algebraic form over a chosen field (eg. $\mathbb{F}_2$ or $\mathbb{F}_{2^8}$).

6.1 Solving the SAT problem

Typically some factors which influence the running time of a SAT solver are: the total number of literals; the total number of clauses; and the number of literals in each clause. The number of clauses depends on the density of the ANF equations. Cutting [1] and taking linear combinations of equations are two methods which alter the densities of the equations and hence the number of clauses in the resulting CNF.

In certain situations, the time required to find a satisfiable instance to the SAT problem can be optimised by assigning values to a subset of the CNF variables prior to invoking the solver. We refer to this assignment of variables as a *guess*. The SAT solver returns SAT or UNSAT depending on whether the guess was correct or not. If no solution was found, the guess was incorrect and another guess is tried.

This "guess and determine" component is summarised by the following algorithm:

```
guess = 0
Do
    instance = Substitute(guess, system)
    result = MINISAT(instance)
    if result is SATISFIABLE
        return guess
    else
        guess = guess + 1
While guess < 2^m
```

If a guess involves an assignment of m literals and the SAT solver requires on average time T to return, the overall running time to find a solution is $T \cdot 2^{m-1}$.

In our experiments we achieved a significant improvement in the performance of the SAT solver when an assignment was made to subsets of literals which occurred with the highest frequency in the CNF.

7 Description of Trivium

Let $x_i^t \in \mathbb{F}_2$ denote the value of the variable x_i at clock time t. Trivium consists of a non-linear feedback shift register (NLFSR) coupled with a linear filter function (LF). The NLFSR operates on a 288-bit state, denoted by $(s_1^t, \ldots, s_{288}^t)$, which is divided into three registers. The LF produces the keystream by taking a linear combination of the state. At each clock the cipher updates three bits of the state and outputs 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**

$t_1 \leftarrow s_{66} + s_{93}$

$t_2 \leftarrow s_{162} + s_{177}$

$t_3 \leftarrow s_{243} + s_{288}$

$z_i \leftarrow t_1 + t_2 + t_3$

$t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$

$t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$

$t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$

$(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$

$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$

$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$

end for

The ANF equations follow directly from this description where each clock introduces three new variables and four new equations (three from the NLFSR and one from the LF).

Trivium incorporates a Key and IV setup stage, where the cipher is clocked $4 \cdot 288$ times to initialise the state. Our analysis does not depend on the initialisation process and will not be discussed further.

8 Reduced Variants

8.1 Bivium

Bivium-A and Bivium-B are two reduced variants of Trivium with only two registers and a total state size of 177 bits. The complete description is given in [6]. Each clock of these ciphers introduces two new variables and three new equations.

8.2 Trivium-n

We introduce Trivium-n, a reduced version of Trivium to a state length of n bits. In constructing these reduced variants we take the following considerations into account.

- The tap bits used from each register should be separated by the same proportions as the original Trivium. In addition, the tap bits should be separated by at least one bit.
- One tap bit in each register that contributes to the LF is the last bit in the register.
- The tap bits in each register that contribute to the AND in the NLFSR are the two adjacent bits preceeding the last bit in the register.

Trivium incorporates many design principles which do not affect our analysis, hence the above conditions are sufficient for our purposes.

An example of a 177-bit state version of Trivium is:

for $i = 1$ to N **do**

$t_1 \leftarrow s_{39} + s_{58}$

$t_2 \leftarrow s_{98} + s_{109}$

$t_3 \leftarrow s_{148} + s_{177}$

$z_i \leftarrow t_1 + t_2 + t_3$

$t_1 \leftarrow t_1 + s_{56} \cdot s_{57} + s_{104}$

$t_2 \leftarrow t_2 + s_{107} \cdot s_{108} + s_{161}$

$t_3 \leftarrow t_3 + s_{175} \cdot s_{176} + s_{42}$

$(s_1, s_2, \ldots, s_{58}) \leftarrow (t_3, s_1, \ldots, s_{58})$

$(s_{59}, s_{60}, \ldots, s_{109}) \leftarrow (t_1, s_{59}, \ldots, s_{108})$

$(s_{110}, s_{111}, \ldots, s_{177}) \leftarrow (t_2, s_{110}, \ldots, s_{176})$

end for

9 Results

We ran multiple experiments with the SAT-solvers on each of the ciphers described in Section 8. Our test machine is an Intel Core2 1.86GHz with 1Gb RAM. The attack times for each variant is listed in the following table.

		ANF		CNF		Time (seconds)			
Cipher	Keystream	Variables	Equations	Literals	Clauses	MiniSat	RSat	PicoSati	F4
Bivium-A	177	396	396	617	4517	$2^{4.4}$	$2^{8.9}$	$2^{5.9}$	2^{24}
Bivium-B	177	396	396	617	5579	$2^{42.7}$	$2^{46.6}$	$2^{45.9}$	2^{68}
Trivium-98	98	331	331	566	7565	$2^{45.3}$	$2^{45.9}$	$2^{44.9}$	2^{67}
Trivium-177	177	590	590	1005	13513	$2^{98.2}$	$2^{101.9}$	$2^{101.8}$	2^{119}
Trivium-288	288	951	951	1616	21815	$2^{159.9}$	$2^{161.9}$	$2^{161.6}$	2^{189}

10 Summary

Due to the unpredictable behaviour and complexity of the SAT-solvers, the results obtained in this paper are derived from experiments.

Both attacks on Bivium require only 177 bits of keystream. The average attack time on Bivium-A is 21 seconds. The average complexity of the attack on Bivium-B is $2^{42.7}$ seconds. Both of these attack are faster than an exhaustive key search.

The attack complexity on Trivium is worse than an exhaustive search.

References

1. G. Bard, N. Courtois and C. Jefferson. *Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over $\mathbb{F}_2$ via SAT-Solvers.* Cryptology ePrint Archive, Report 2007/024, 2007.
2. University of Sydney Computational Algebra Group *The Magma Computational Algebra System, 2004* `http://magma.maths.usyd.edu.au/magma/`
3. A. Maximov and A. Biryukov. *Two Trivial Attacks on Trivium.* Cryptology ePrint Archive, Report 2007/021, 2007.
4. A. Maximov. private communication 14/3/07.
5. 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`
6. H. Raddum. *Cryptanalytic results on TRIVIUM.* eStream, ECRYPT Stream Cipher Project, Report 2006/039, 2006. `http://www.ecrypt.eu.org/stream`
7. eStream: ECRYPT Stream Cipher Project `http://www.ecrypt.eu.org/stream/`
8. C. Shannon. *Communication theory of secrecy systems.* Bell System Technical Journal 28, 1949.
9. DIMACS http://www.cs.ubc.ca/ hoos/SATLIB/Benchmarks/SAT/satformat.ps
10. MiniSat 2.0 http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/MiniSat.html
11. RSat http://reasoning.cs.ucla.edu/rsat/
12. PicoSat http://fmv.jku.at/picosat/
13. The international SAT competition. http://www.satcompetition.org/
14. SAT-Race 2006 http://fmv.jku.at/sat-race-2006/

11 Appendix

The following system of equations are a subset of the 396 ANF equations that describe Bivium-A. The internal state of Bivium-A consists of two registers (a and b). The first three equations describe the relationship between the internal state and the keystream (z_i). This follows from the description of the LF. The next three equations arise from the NLFSR.

```
a_66 + a_93 + z_1,
a_65 + a_92 + z_2,
a_64 + a_91 + z_3,
a_94 + a_69 + b_69 + b_82*b_83 + b_84,
a_95 + a_68 + b_68 + b_81*b_82 + b_83,
a_96 + a_67 + b_67 + b_80*b_81 + b_82
```

The corresponding CNF of the above equations, displayed in DIMACS form is:

```
p cnf 28 71
1 0
-2 0
-3 4 5 0
3 -4 5 0
3 4 -5 0
-3 -4 -5 0
-6 7 8 0
6 -7 8 0
6 7 -8 0
-6 -7 -8 0
-9 10 11 0
9 -10 11 0
9 10 -11 0
-9 -10 -11 0
-12 13 14 15 18 0
12 -13 14 15 18 0
12 13 -14 15 18 0
-12 -13 -14 15 18 0
12 13 14 -15 18 0
-12 -13 14 -15 18 0
-12 13 -14 -15 18 0
12 -13 -14 -15 18 0
12 13 14 15 -18 0
-12 -13 14 15 -18 0
-12 13 -14 15 -18 0
12 -13 -14 15 -18 0
-12 13 14 -15 -18 0
12 -13 14 -15 -18 0
12 13 -14 -15 -18 0
-12 -13 -14 -15 -18 0
-19 20 21 22 17 0
19 -20 21 22 17 0
19 20 -21 22 17 0
-19 -20 -21 22 17 0
19 20 21 -22 17 0
-19 -20 21 -22 17 0
-19 20 -21 -22 17 0
19 -20 -21 -22 17 0
19 20 21 22 -17 0
-19 -20 21 22 -17 0
-19 20 -21 22 -17 0
19 -20 -21 22 -17 0
-19 20 21 -22 -17 0
19 -20 21 -22 -17 0
19 20 -21 -22 -17 0
-19 -20 -21 -22 -17 0
-24 25 26 27 16 0
24 -25 26 27 16 0
24 25 -26 27 16 0
-24 -25 -26 27 16 0
24 25 26 -27 16 0
-24 -25 26 -27 16 0
-24 25 -26 -27 16 0
24 -25 -26 -27 16 0
24 25 26 27 -16 0
-24 -25 26 27 -16 0
-24 25 -26 27 -16 0
24 -25 -26 27 -16 0
-24 25 26 -27 -16 0
24 -25 26 -27 -16 0
24 25 -26 -27 -16 0
-24 -25 -26 -27 -16 0
-15 16 0
-15 17 0
15 -16 -17 0
-22 23 0
-22 16 0
22 -23 -16 0
-27 28 0
-27 23 0
27 -28 -23 0
```