SAT-Solving in Algebraic Cryptanalysis

Bachelor-Thesis von Ahmed Charfi
Juni 2014

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science
CDC

SAT-Solving in Algebraic Cryptanalysis

Vorgelegte Bachelor-Thesis von Ahmed Charfi

1. Gutachten: Prof. Dr. Johannes Buchmann
2. Gutachten: Dr. Mohamed Saied Emam Mohamed

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den June 26, 2014

(A. Charfi)

Abstract

The satisfiability (SAT) problem is one of the most important problems in theoretical computer science. This problem is about determining whether there exists an assignment of variables in a logical formula so that the formula evaluates to true. It is relevant in several application domains such as IT security and algebraic cryptanalysis.

In this bachelor project we used 8 different sat solvers, which are the best available ones according to the International Sat Competition[1] to analyze the security of three block ciphers against algebraic cryptanalysis. More specifically we analyzed AES, CSA and LED.

Regarding AES, we solved systems that represent small scale variants of the AES polynomial system namely (1,4,4,4). Regarding LED, we solved the first round fully.

Furthermore, we were able to break 20 rounds of the CSA block cipher. For our information this is the best algebraic attack on CSA.

Our experiments are based on a very large number of test cases. Consequently, we provide recommendations for selecting the best sat solver for a specific case.

[1] http://www.satcompetition.org/

Contents

1 Introduction

Cryptography is the science of secure communication over public channels. More specifically, this science develops techniques and protocols to secure data and protect it from third parties. Cryptanalysis aims at breaking information systems that use cryptography and gaining access to contents of encrypted data. Over the last few years algebraic cryptanalysis has become an extremely important topic especially with the growth of communication applications and networked applications such as mobile telecommunication and online banking applications. Algebraic Cryptanalysis is a cryptanalysis technique, which tries to break codes by solving systems of multivariate polynomial equations. Another relevant concept in this context is the concept of block ciphers, which are encryption algorithms that operate on data blocks of a fixed size. Block ciphers are an important tool in the design of protocols for shared-key cryptography. On the other hand, there is a need for appropriate tools to test and assess the security level and the power of encryption mechanisms in general such as block ciphers. Sat solvers are such tools and they are often used for algebraic cryptanalysis. Sat solvers take a Boolean formula in CNF as an input and try to find a variable assignment so that the formula evaluates to true.

In this thesis, we focus on algebraic cryptanalysis and study the behaviour of algebraic attacks using the best available sat solvers [8] on three block ciphers: AES, LED, and CSA.

With respect to AES, we solved a small scale variant of the AES polynomial system (1, 4, 4, 4). With regard to LED, we solved a full round and some rounds by correctly guessing a certain number of bits of the key. Concerning CSA, we were able to break 20 rounds of totally 55 rounds in less than 20 hours. To the best of our knowledge, this is the best algebraic attack on CSA.

The remainder of this thesis is organized as follows. Chapter 2 introduces some background knowledge and some preliminaries that are relevant for understanding this work. Chapters 3, 4, 5 respectively present the test results for the block ciphers AES, CSA and LED. Based on the results of these tests recommendations are given for choosing the most suitable sat solver for a given use case. Chapter 6 sums up the results of this work and outline possible directions for future work.

2 Preliminaries

This chapter gives an overview of the preliminaries of this thesis and introduces the relevant concepts for this work. Section 2.1 defines algebraic cryptanalysis. Section 2.2 introduces the sat problem and some related concepts such as the sat algorithm DPLL and eight sat solvers that were used in the tests. Section 2.3 gives an introduction to the block ciphers in general and then presents the three block ciphers that were used in this work. Section 2.4 reports on the tools and hardware used to perform the tests.

2.1 Algebraic Cryptanalysis

According to the Oxford dictionary [1], cryptanalysis is defined as *the art or process of deciphering coded messages without being told the key.* The idea behind algebraic cryptanalysis [3] is to find a relation between the inputs and outputs of a cryptographic functional using a set of polynomial equations. These equations are usually constructed over a finite field GF (2), which is the Galois Field containing two elements (0 and 1). This field is the smallest possible finite field. Algebraic cryptanalysis can be applied in different areas as each cryptographic function can be described by a set of polynomial equations. However, solving a polynomial system is an NP-hard problem that requires a lot of time and resources because the used polynomial systems usually contain many equations and many variables. In many cases attacking such cryptographic functions using brute force is easier than using algebraic cryptanalysis. Brute force is a technique that enumerates all possible solution candidates and checks for each candidate if it satisfies the problem. On the other hand, algebraic cryptanalysis uses techniques, which have an exponential complexity in the worst case. There are several tools that are used for algebraic cryptanalysis such as Gröbner basis computation and sat solvers. A Gröbner basis for a polynomial system is an equivalence systems that has several useful properties. Gröbner basiscomputation is an important practical tool for solving systems of polynomial equations. In addition, sat solvers provide another tool for solving the same problem.

In this work, we use sat solvers to algebraically attack three block ciphers: AES, LED, and CSA. There are other similar works that use sat solvers to attack other cipher such as Bivium [12], Courtois Toy Cipher (CTC) [15], and Data Encryption Standard (DES) [15]. In addition, there are some well-known attack models on block ciphers, which models describe what and how much information the attacker may be able to get. As examples for such models we mention known plain text attack [20] and chosen plain text attack [2].

2.2 The Sat Problem

In this section we first define the Sat problem. Then we present the eight sat solvers that were used in this work to test the hardness of the selected block ciphers. After that we report on the tools and hardware that were used for running the tests.

[1] $http://www.oxforddictionaries.com/$

2.2.1 Definition

In computer science , the Sat problem consists of determining whether there exists an assignment for logical variables that are used in a given formula so that it evaluates to true. If no such assignment exists, the formula is unsatisfiable, otherwise satisifiable. One unique characteristic of Sat is the following: Sat was the first known example of an NP complete problem. This means that there is no known algorithm which solves all instances of sat (i.e., in polynomial time or better). Sat solvers are algorithms that take a formula as an input and try to solve the sat problem for that Formula. Most Sat solvers expect the formula to be in conjunctive normal form (CNF). This normal form is a way of writing logical formulas using only a conjunction of clauses. For example, we consider the following formula in CNF:

$(v_1 \vee \neg v_2) \wedge (v_2 \vee v_3) \wedge (v_1 \vee v_3 \vee v_4)$

This formula is usually encoded in the following matrix format by most sat solvers:

```
c this is a comment
p cnf 4 3
1 −2 0
2 3 0
1 3 4 0
```

The first line in this format is a comment. The second line indicates the form of the logical formula (in this case CNF). The following two integer values indicate the number of variables (here 4) and the number of clauses (here 3). Each variable is encoded as a number (1 for v_1, 2 for v_2, 3 for v_3, and 4 for v_4). Negative numbers encoded negation (e.g., -2 means not v_2). Each clause is encoded as a line in the matrix and the 0 value at the end of each line means that the clause is finished.

In addition to CNF, there is another form of representing logical formulas called the Algebraic Normal Form (ANF). This form is a special way of writing a logical formula. Either the entire formula is purely true or false or the formula consists of a set of terms that use only the AND operated and these terms are combined by the XOR operator. However, all sat solvers that are used in this work are based on CNF.

Most sat solvers that are used in this work are called conflict-driven solvers. These solves are based on the Davus putnam Logemann Loveland (DPLL) algorithm [21]. This algorithm is a backtracking [17] based search algorithm for deciding the satisfiability of propositional logic formulae in CNF. It first chooses a literal and assigns a boolean value to it, which leads to simplifying the formula. Then, this algorithm recursively checks if the simplified formula is satisfiable. Once this is the case the original formula is satisfiable. Otherwise, the literal is assigned the opposite boolean value and the recursive check is done again.

Figure 1 illustrates how the DPLL algorithm works. The algorithm started working and used different assignment values until it came to the first conflict. It then returned using backtracking - [17] to the start position. After that, the algorithm tried with the right side of the tree until a solution was found.

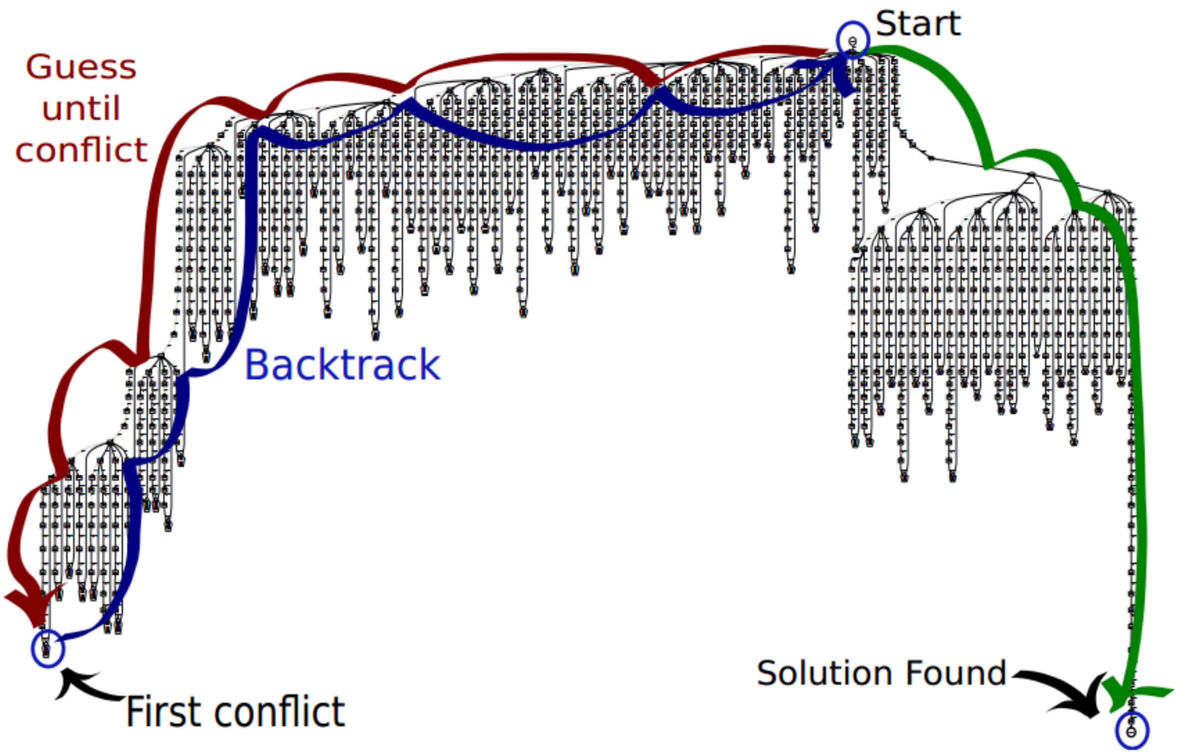

Figure 1.: Visualization of the DPLL algorithm [21]

2.2.2 Sat Solvers

A sat solver [13] is an algorithm or a program that takes a logical formula as an input and returns an assignment so that the formula evaluates to true or it says that no such assignment exists. Sat solvers are being used in several areas such as software and hardware verification, automatic test pattern generation, algebraic planning and scheduling problems, and also in algebraic cryptanalysis. One disadvantage of almost all sat algorithms is their complexity which can be in the worst case exponential. However, in the last few years the performance of sat solvers has improved a lot.

In the following, we present briefly the eight sat solvers that were used in this bachelor work. These are Minisat 2.0, Cryptominist 2.9.5 , Cryptominisat 3.3.0, Lingeling aqw, Glucose 2.3, Riss3g cert, Doug Hains 1.1, and Zenn 0.1.0. These sat solvers were selected based on their excellent results in the international sat competition in 2013 [8]. All of them are implemented either in C or in C++.

Minisat

Minisat [11] is one of the first sat solvers. The other seven sat solvers that we use are improvements and extensions of minisat. According to [11], *MiniSat is a minimalistic, open-source SAT solver, developed to help researchers and developers alike to get started on SAT. Minisat is a minimalistic implementation of a CHAFF-like solver based on the two literal watch schemes for fast boolean constraint propagation (BCP) and clause learning by conflict analysis.* Minisat has several advantages: It is open source, easy to modify, and well-documented. The key features of Minisat are listed in the following:

- unite propagation
- backtracking with restarts
- clause learning through analyze of conflicts
- delete unneeded clauses

In this work we use version 2.0 of Minisat.

Cryptominisat

Cryptominisat [16] is a modern SAT Solver that aims at combining the benefits of four other sat solvers: SATELITE, Precosat, Glucose, and Minisat in order to solve a formula in reasonable time. It was developed by Soos Mate. In this work two versions of cryptominisat are used in the experiments: 2.9.5 and version 3.3.0.

Cryptominisat2 [21] is a DPLL-based sat solver. Some key features of cryptominisat2 are:

- Usage of xor clauses
- Usage of two techniques: phase calculation, saving and random flipping
- Clause cleaning
- Support for 32-bit pointers on 64-bit architectures under Linux

There have been many improvements in version 3 of Cryptominisat compared to version 2. These improvements mainly affect memory use and timeouts in addition to system-wide structural improvements and code cleaning. Another improvement is that Cryptominsiat 3 uses Gaussian Elimination by default whereas Cryptominisat2 uses this elimination at every level of the decision tree.

Lingeling

The Lingeling [5] is a sat solver that was developed at Johannes Kepler University in Linz, Austria. It uses interleaving search and preprocessing to simplify the initial formula. It also uses a garbage collection algorithm to reduce the number of learned clauses. At runtime, this algorithm determines if classic heuristics or glues should be used.

Zenn

Zenn [22] is a sat solver that is based on Minisat 2.2 and which was developed at Kyushu University in Japan. Zenn employs a technique calls Phase Shift, which integrates different search methods. The underlying algorithm performs two or more search phases. Each phase has a predefined and fixed duration. If a certain number of restarts is reached the algorithm switches phases.

Riss3g

Riss3g [19] is a sat solver that is based on Minisat 2.2 and glucose 2.1. This solver was developed at Technical University Dresden. It is used as a research platform and thus provides many parameters and features to enable further techniques that are not present in other general sat Solvers. Some of these features are list below

- *enumeration of all solutions of the input formula,*
- loading and storing learned clauses of a run,
- searching for a solution with a set of assumed literals,
- passing an initial model to the solver that should be tested first

Glucose

Glucose [1] is a sat solver that is built on top of Minisat 2.2. It was developed at University Bordeaux in France. This solver uses an algorithm called Conflict Driven Clause Learning (CDCL) [6], which is inspired from DPLL algorithms. This algorithm first selects a variable and assigns a boolean value to it. Then it applies boolean constraint propagation. After that it builds the implication graph. If there is a conflict the algorithm analyzes it and jumps back non-chronologically. Otherwise it restarts with the variable assignment until all variables are assigned. The name of glucose comes from the importance of glue clauses, which are clauses that allow to sticking a new literal to a book of propagation literals.

Doug (Minisat Static)

The Doug sat solver [10] was developed at Colorado State University in the USA. This sat solver uses a new preprocessing technique called reduction to simplify the initial formula by fixing truth assignments. Then, it passes the reduced formula to the minisat sat solver.

2.3 Block Ciphers

Block ciphers [18] are encryption algorithms that operate on data blocks of a fixed size. Block ciphers are an important tool in the design of protocols for shared-key cryptography. A block cipher can be represented as a mathematical function E that takes two inputs: the first one is the key which has the length k and the second one is the plain text which has the length n. The output is the cipher text, which also the length n. The key length and the block size vary from one block cipher to another.

In the following, we present the three block ciphers that were tested in this work.

2.3.1 Advanced Encryption Standard (AES)

AES [15] is an encryption standard that is based on a design principle known as substitution permutation network. AES was introduced as a replacement for the Data Encryption Standard (DES), which is slow and has a short key. AES can be implemented in a fast manner in both software and hardware. AES is a variant of the Rijndael algorithm, which has a fixed block size of 128 bits and a key size of either 128,192, or 256 bits. AES takes the following four arguments:

- n ∈ {1,..,10} is the number of encryption rounds
- r ∈ {1,2,4} is the number of rows in the input matrix
- c ∈ {1,2,4} is the number of columns in the input matrix
- e ∈ {4,8} is the degree of the underlying field

AES is totally attacked when $n = 10, r = 4, c = 4, e = 8$. We were able in this work to break AES till the following values: $n = 10, r = 1, c = 1, e = 4$.

2.3.2 Light Encryption Device (LED)

LED [14] is a block cipher that has a small footprint and it is dedicated to being implemented in compact hardware. There were three further goals in the design of that tool: the usage of an ultra-light key schedule, the consideration of resistance against key attacks, and a reasonable performance when implemented in software. This block cipher can handle key sizes from 64 bits up to 128 bits. The key can be changed without modification of the algorithm.

LED provides provable security against classical linear/differential cryptanalysis both in the single-key and related-key models. In this work we were able to break the first round of LED fully and some other rounds (round 2 to 48) by guessing bits less than 16 bits of the key.

2.3.3 Common Scrambling Algorithm (CSA)

CSA [24] is an encryption algorithm which is used in digital video broadcasting (DVB). It is composed of two distinct ciphers: a block cipher and a stream cipher. The Data are first encrypted using the 64 bits block cipher in Cipher Block Chaining (CBC) mode [4]. The stream cipher is applied from packet start. Due to the fact that DVB is commonly used by pay TV, a need to protect transmitted data against not paying viewers is required so that not everybody can receive the broadcasted data. The CSA block cipher operates with 64 input bits, 64output bits, and 64 key bits. This cipher consists of 56 identical rounds. The key is expanded to 448 bits and every round uses 8 bits of this expanded key.

We were able in this work to break CAS till the round 20 in less than 20 hours.

2.4 Tools and Hardware

Several tools were used in this work. To test the different block ciphers we need an equations generator. For that purpose we used the open source mathematics software SAGE [9] version 6.0. The polynomial systems were converted to SAT instances using the implmented class by Michael Birckenstein PolyBoRi CNF converter [7]. Solving the SAT instances was done using the sat solvers presented above.

In addition, several programs and scripts were written in Java, Python, and Shell Script to run the test 100 times, to convert the input files in the required format, to process the output files and extract results and solutions of each run, and to calculate the average time needed.

We run the experiments on a Computer with 4 Six-Core AMD Opteron Processors 8435 operating at 2.6 GHZ. This Computer has 64 GB RAM and runs a 64 bit Linux (Ubuntu 13.10) as operating system.

3 Advanced Encryption Standard (AES)

In this chapter, we first report on the different programs and scripts that were developed to support the different tests of the AES block cipher. Then, we present the results of these tests in several tables and provide a discussion. Finally some curves are presented, which show graphically the stability of the eight different sat solvers that were used.

3.1 Tools for AES tests

Several programs and scripts were necessary to automate the AES tests. First, a sage class will be presented which generates the CNF file that acts as input for the sat solvers. Then, we present as an example the bash script used to call the Lingeling sat solver 100 times. Similar scripts were written for the other sat solvers but for brevity we present only one representative script. After that, we present the Java class Lingelingoutput, which extracts the following data from the output of the sat solver: the results (i.e., the information on whether a formula is satisfiable or not) and the execution time. Finally, we present the Java class Javerage, which calculates the average execution time as each test is run 100 times.

3.1.1 AES Equation Generator

Listing 3.1 shows the source code of the SAGE class that creates the input for the sat solver in CNF out of a polynomial system. In this listing the number of rounds is 3, the number of rows and columns is 2, and the value of the underlying field is 8. This class uses an ANFsat solver, which takes the polynomial system as parameter. Then, the CNF representation of that system is written to a file called AESCNF.txt.

```
load anf2cnf.py
from polybori import *
sr = mq.SR(3,2,2,8,allow_zero_inversions=True,gf2=True) # n=3 , r=c=2 , e=8
print sr.R.repr_long()
F,s = sr.polynomial_system()
B = BooleanPolynomialRing(F.ring().ngens(), F.ring().variable_names())
F = [B(f) for f in F if B(f)]
solver = ANFSatSolver(B)
cf = solver.cnf(F)
c = open('AES_cnf.txt', 'w')
c.write(cf) # write the equations in a file called AES_cnf.txt
c.close()
```

Listing 3.1: SAGE class for generting the CNF file

3.1.2 Bash Script Example

```
#!/bin/bash

rm test_lingeling.out # remove test_lingeling.out
rm test20.tmp            # remove test20.tmp
for i in `seq 1 100;` # for loop 100 times
do
rm test20.tmp         # remove test20.tmp
  lingeling  AES_CNF.txt  > test20.tmp # call lingeling using
  # File AES_CNF.txt as input and write it in test20.tmp
  java LingelingOutput test20.tmp >> test_lingeling.out
  # Call LingelingOutput on file test20.tmp to pick up the
  #time and satisfiafiabile attribut

done
java Javerage test_lingeling.out # call Javerage to get average
echo 'done'
```

Listing 3.2: Bash script for running a test 100 times

Listing 3.2 shows a bash script that is used to call the sat solver Lingeling 100 times. This is motivated by the instability of sat solvers. For example Minisat could take 2 second for a test case run and then take 2 minutes for the same test case in another run. Similar scripts were written for the other sat solvers. At the beginning this script removes any existing output files from previous tests. Then, a for loop is used to call the sat solver 100 times taking the CNF file generated by the AES equation generator as input. The output of the sat solver is then written to the temporary file test20.tmp. After that, a java class LingelingOutput is called to extract a data set consisting of the result (i.e., formula satisfiable or not) and the solver execution time from the solver output file. This data set is then written to the file testlingeling.out (one data set for each iteration). After finishing the for loop the Java class Javerage is called to compute the average execution time based on the results of the 100 tests.

3.1.3 Java Class LingelingOutput

Listing A.1 in the appendix shows the source code of the Java class LingelingOutput. The sat solver produces a comprehensive and long output including program information, information on the input equation (e.g., number of variables, numbers of clauses), etc. The purpose of this class is to extract two specific values out of that output, which are important for our tests. The first value is the execution time and the second value is the satisfiabilty result (i.e., the information on whether the formula is satisfiable or not).

3.1.4 Java Class Javerage

Listing A.2 in the appendix shows the source code of the Java class Javerage. This class takes as input a text file including a line for each run of the test. The line includes the execution time and the satisfiability result for each run. If we run a test 100 times the input file would have 100 lines. This class extracts the execution times values and calculates their average, which will then be printed out.

3.1.5 Java Class ClauseCounter

Listing A.3 in the appendix shows the source code of the Java class ClauseCounter. This class takes as input the CNF file and calculates the length and the number of clauses. These values are needed for presenting the test results in the next section.

3.2 Tests and Results

In this section, we present the results of the tests that were conducted on the AES block cipher. In each subsection the number of rows R and the number of columns C are fixed to some value and we vary the number of rounds N (from 1 to 10) as well as the value of the underlying field E (which can be either 4 or 8). The number or rows and columns take the following values: R=C=1, R=C=2, R=C=4, R=1 and C=2, R=2 and C=4. Next, we present the results of each test.

3.2.1 Test 1: R=C=1

Table 3.1 shows the results of this test, which contains 20 cases. The number of rows and columns was fixed to 1 whereas the number of rounds N varies from 1 to 10 and the value of the underlying field E is either 4 (in the first 10 cases) or 8 (in the last 10 cases).

Columns 1 to 5 respectively show the case number I, the number of rounds N, the number of rows R, the number of columns C, and the underlying field E. Columns 6 to 13 show the time in seconds that is taken by the respective solver for each test case. Column 14 shows the number of variables used to solve the equation system. Columns 25 to 27 show respectively the number of clauses with length 2,3, and 4. Column 28 shows the total number of clauses. The last column states if the equation system was satisfiable or not. In this table the best values are shown in green whereas the worst values are shown in red.

As the number of rows and columns equals one the plain text to be encrypted by AES contains only 1 element. This explains the relatively low time amount taken by the test cases. All tests take less than one hour. Furthermore, the maximum length of clauses is equal to 4. That means that each clause has at most 4 variables and thus it can be solved relatively fast. We notice that 3 cases (7,8 and 20) were unsatisfiable. In these cases, Minisat was the fastest sat solver to tell that the respective equation systems are unsatisfiable.

In the first 5 cases, Minisat static was the fastest sat solver and it immediately tell that the system is satisfiable (in some milliseconds). In contrast, Cryptominast3 was the slowest sat solver in these cases. The other sat solvers had comparable performance. Consequently, if number of rounds is less than 6 it

Table 3.1.: AES Test 1: R=C=1

I	N	R	C	E	Mini	Cr3	Cr2	Ling	Zenn	Riss	Gluc	Doug	#pr	#cl=2	#cl=3	#cl=4	#cl	Sat?
1	1	1	1	4	0.004	0.017	0.0045	0.011	0.0027	0.0023	0.0028	0.0003	118	96	160	432	688	Yes
2	2	1	1	4	0.003	0.033	0.007	0.016	0.005	0.003	0.004	0.003	240	192	336	864	1392	Yes
3	3	1	1	4	0.004	0.041	0.018	0.031	0.005	0.006	0.004	0.002	362	288	512	1296	2096	Yes
4	4	1	1	4	0.007	0.06	0.02	0.044	0.018	0.018	0.017	0.003	484	384	688	1728	2800	Yes
5	5	1	1	4	0.01	0.094	0.39	0.042	0.018	0.017	0.024	0.02	606	480	864	2160	3504	Yes
6	6	1	1	4	0.018	0.085	0.031	0.1	0.015	0.029	0.028	0.011	728	576	1040	2592	4208	Yes
7	7	1	1	4	0.016	0.12	0.06	0.1	0.04	0.04	0.048	0.024	850	672	1216	3024	4912	No
8	8	1	1	4	0.026	0.17	0.09	0.2	0.052	0.048	0.048	0.036	972	768	1392	3456	5616	No
9	9	1	1	4	0.03	0.12	0.06	0.21	0.03	0.02	0.02	0.03	1094	864	1568	3888	6320	Yes
10	10	1	1	4	0.037	0.14	0.06	0.1	0.06	0.02	0.03	0.04	1216	960	1744	4320	7024	Yes
11	1	1	1	8	0.15	0.47	0.12	0.68	0.37	0.35	0.10	0.36	750	308	568	4144	5020	Yes
12	2	1	1	8	1.52	0.45	0.08	3.28	0.18	0.22	0.19	0.82	1500	616	1104	8352	10072	Yes
13	3	1	1	8	5.44	12.44	19.57	16.38	11.38	0.385	12.73	14.35	2250	924	1640	12560	15124	Yes
14	4	1	1	8	10.64	14.79	32.07	49.42	10.46	12.75	15.04	21.97	3000	1232	2176	16768	20176	Yes
15	5	1	1	8	25.24	23.56	14.04	4.75	8.45	7.67	4.39	14.42	3750	1540	2712	20976	25228	Yes
16	6	1	1	8	43.17	67.67	35.35	135.9	53.04	61.86	6.51	13.55	4500	1848	3248	25184	30280	Yes
17	7	1	1	8	69.08	25.25	42.85	59.0	150.48	58.53	4.01	8.05	5250	2156	3784	29392	35332	Yes
18	8	1	1	8	89.22	111.18	232.18	116.92	44.42	93.1	3.75	97.63	6000	2464	4320	33600	40384	Yes
19	9	1	1	8	128.19	185.78	122.29	310.43	86.37	77.22	43	12.85	6750	2772	4856	37808	45436	Yes
20	10	1	1	8	177.03	859.04	501.55	446.7	527.83	483.61	705.49	2443.6	7500	3080	5092	42016	50488	No

is recommended to use Minisat static. In the test cases 6 to 10, we notice that Minisat static is not the fastest sat solver any more. Based on the results we recommend to either use Riss3g or Minisat. The test cases shows that Cryptominisat3 and Lingeling have the worst performance (color red and orange in the table 3.1).

We also notice that at each iteration the number of variables increases by 122, the number of clauses with length 2 increases by 96 , the number of clauses with length 3 increases by 176, the number of clauses with length 4 increases by 432, and the total number of clauses increases by 704. Based on the results of the first 10 cases (E=4) the number of parameters of a system of polynomial equations can be calculated using the following relations:

- number of variables = 118 +122* (number of rounds - 1)
- number of clauses with length 2 = 96 * number of rounds
- number of clauses with length 3 = 160 +176 * (number of rounds - 1)
- number of clauses with length 4 = 432 * number of rounds
- number of total clauses = 688 + 704 * (number of rounds - 1)

In the test cases 11 to 20 E is equal to 8, which means that the key size is 8 bit. This explains why the time required to solve the equation system in this case is more than in the 10 first cases where E is equal to 4.

Concerning test cases 11 to 20 we notice that Lingeling is the slowest sat solver. In the first five cases (N = 11 to 15 and E=8) we recommend to use Glucose. In the next five cases (N=16 to 20 and E=8) we recommend using Glucose and notice that the performance of Lingeling and Cryptominisat3 and Lingeling get worse.

Based on the data of the last 10 test cases we derive the following relations between number of rounds and the number of variables and the number of clauses of a given length.

- number of variables = 750 * number of rounds.
- number of clauses length 2 = 308 * number of rounds.
- number of clauses length 3 = 568 +236 * (number of rounds - 1)
- number of clauses length 4 = 4144 + 4208 (number of rounds - 1)
- number of total clauses = 5020 + 5052 * (number of rounds -1)

3.2.2 Stability Case 13

Figure 2 shows the stability curve of the used eight sat solvers when R=C=1, N=3, and E=8. This figure shows that Minisat is very unstable. In addition, Cryptominisat2 and Lingeling are unstable. In opposite, the Riss3g was the most stable sat solver. The respective curve is almost a horizontal line as shown on the bottom (color blue sky).

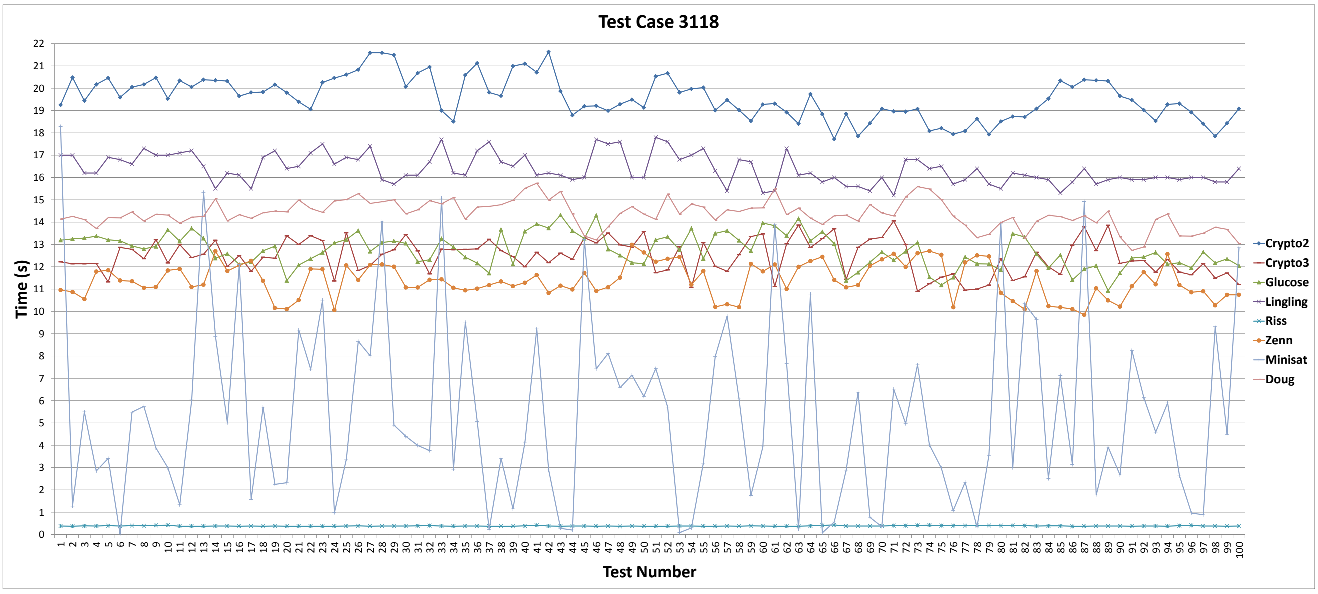

Figure 2.: AES Stability Case 13

3.2.3 Test 2: R= 2 , C=1

Table 3.2 shows the results of the second test, which also contains 20 cases. The number of rows was fixed to 2 and the number of columns was fixed to 1. Like in the previous test the number of rounds N varies from 1 to 10 and the value of the underlying field E is either 4 (in the cases 21 to 30) or 8 (in the cases 31 to 40). The maximum length of clauses remained 4 as in the previous test. In this table the best values are shown in green whereas the worst values are shown in red.

As the number of rows equals two and the number of columns equals one the plain text to be encrypted by AES in this case contains 2 elements. This explains the relatively higher time amount taken by the test cases compared to the previous test, in which we had only one element. In this second test, the longest test case takes 378 hours. In addition, we notice that for some of the last cases, sat solvers took more than 2 weeks and were still unable to deliver a solution. This applies for Minisat (N=10 to 10), Cryptominisat2 (N=7 to 10), Lingeling (N= 8 and 10), Zenn (N= 9 and 10), Minisat Static(N= 8 to 10).

We notice that 6 test cases (24,25,27,28,29 and 30) were unsatisfiable. In the test cases 21 to 30 Glucose and Riss3g are the fastest sat solvers as they require less than 1 second to deliver their result. On the other hand Minisat static and Lingeling were the slowest and took 8 seconds to solve the equation system. Based on that we recommend using either Glucose or Riss3g when N=1 to 10, E=4 , r=2 and c=1.

Table 3.2.: AES Test 2: R=2 , C=1

I	N	R	C	E	Minisat	Cr3	Cr2	Ling	Zenn	Riss	Gluc	Doug	# pr	#cl=2	#cl=3	#cl=4	#cl	Sat?
21	1	2	1	4	0.003	0.027	0.009	0 .021	0.005	0.003	0.004	0.002	248	192	368	864	1424	Yes
22	2	2	1	4	0.041	0.061	0.037	0.097	0.031	0.011	0.022	0.031	498	384	720	1776	2880	Yes
23	3	2	1	4	0.109	0.179	0.119	0.3	0.12	0.188	0.084	0.063	748	576	1072	2688	4336	Yes
24	4	2	1	4	0.128	0.26	0.26	0.4	0.16	0.192	0.172	0.18	998	768	1424	3600	5792	No
25	5	2	1	4	0.276	0.38	0.37	0.6	0.312	0.184	0.18	0.268	1248	960	1776	4512	7248	No
26	6	2	1	4	0.595	0.454	0.363	1.215	0.295	0.225	0.281	0.323	1498	1152	2128	5424	8704	Yes
27	7	2	1	4	1.468	0.73	1.13	1.0	0.752	0.4	0.448	0.516	1748	1344	2480	6336	10160	No
28	8	2	1	4	0.872	1.12	1.24	1.9	0.724	0.722	0.62	3.14	1998	1536	2832	7248	11616	No
29	9	2	1	4	1.02	1.18	4.38	1.4	1.348	1.088	0.46	5.02	2248	1728	3184	8160	13072	No
30	10	2	1	4	5.77	2.08	3.23	3.2	1.332	0.564	0.42	8.2	2498	1920	3536	9072	14528	No
31	1	2	1	8	2.11	0.43	8.01	11.29	1.05	1.0	9.09	3.33	1520	616	1136	8448	10200	Yes
32	2	2	1	8	126.24	128.36	215.25	41.93	11.98	168.45	52.80	91.2	3040	1232	2208	17024	20464	Yes
33	3	2	1	8	0.79h	0.18h	0.92h	0.08h	0.88h	0.59h	0.86h	1.12h	4560	1848	3280	25600	30728	Yes
34	4	2	1	8	2.52h	0.39h	6.82h	0.55h	9.04h	2.63h	0.51h	4.22h	6080	2464	4352	34176	40992	Yes
35	5	2	1	8	9.15h	0.5h	378.7h	7.6h	1.87h	0.45h	3.35h	2.71h	7600	3080	5424	42752	51256	Yes
36	6	2	1	8	-	0.94h	23.77h	6.91h	6.74h	1.62h	1.7h	33.56h	9120	3696	6496	51328	61520	Yes
37	7	2	1	8	-	1.83h	-	15.24h	20.97h	1.45h	1.0h	97.98h	10640	4312	7568	59904	71784	Yes
38	8	2	1	8	-	2.87h	-	54.02h	19.29h	1.67h	3.34h	-	12160	4928	8640	68480	82048	Yes
39	9	2	1	8	-	2.74h	-	-	-	3.52h	3.19h	-	13680	5544	9712	77056	92312	Yes
40	10	2	1	8	-	4.24h	-	-	-	3.62h	2.93h	-	15200	6160	10784	85632	102576	Yes

Like in the previous test we derived the following relations between the number of variables and the length of clauses and the number of rounds:

- number of parameters = 248 + 250 * (number of rounds -1)
- number of clauses length 2 = 192 * number of rounds.
- number of clauses length 3 = 368 +352*(number of rounds - 1)
- number of clauses length 4 = 864 + 912 (number of rounds - 1)
- number of total clauses = 1424+ 1456 * (number of rounds -1)

Concerning test cases 31 to 40 (E=8) we notice that Riss3g and Glucose are still the fastest sat solvers. They were able to solve all systems in less than 4 hours. Furthermore, we notice that the performance of Cryptominisat3 improved a lot compared to the test cases 21 to 20. In fact, Cryptominisat3 was the third fastest sat solver and it was able to solve all systems unlike the other five sat solvers.

In the test cases 31 to 36 Cryptominisat2 and Minisat−Static are the worst sat solvers from the performance point of view. For instance, in test case 35 Cryptominisat2 took 378 hours which is more than 2 weeks whereas Riss3g took less than half an hour for solving the same system.

From test case 37, Minisat , Cryptominisat2, Lingeling, Zenn and Minisat−Static may require up to several weeks. Therefore, we set a time limit of 4 days for each test case. When a sat solver is not able to deliver a result within that period we put the symbol − in the respective entry in the table.

Based on the results of the test cases 31 to 40 (E=8), we recommend using Cryptominisat2, Riss3g or Glucose. In general, when R=2 and C=1 we recommend using either Glucose or Riss3g.

Next, we present the mathematical relations between the number of variables and the length of clauses and the number of rounds:

- number of parameters = 1520 * (number of rounds)
- number of clauses length 2 = 616 * number of rounds
- number of clauses length 3 = 1136 +1072*(number of rounds - 1)
- number of clauses length 4 = 8448 + 8576 (number of rounds - 1)
- number of total clauses = 10200+ 10264 * (number of rounds - 1)

3.2.4 Stability Case 31

Figure 3 shows the stability curve of the used eight sat solvers when R=2, C=1, N=1, and E=8. This figure shows that Minisat is very unstable as also in Figure 2. As shown in this figure the most stable sat solvers are Riss3g (as also in Figure 2) , Minisat−static and Zenn.

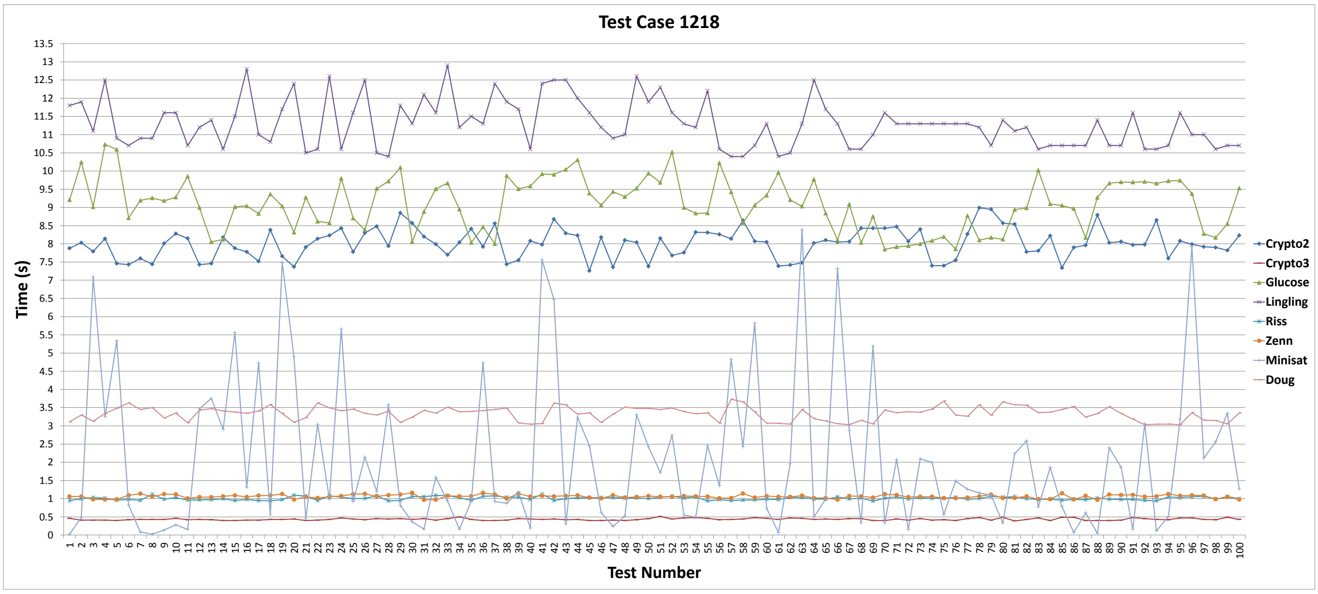

Figure 3.: AES Stability Case 31

3.2.5 Test 3: R=C=2 , R=4,C=2 and R=C=4

Table 3.3 shows the results of the third test, which contains 16 cases. The maximum length of clauses remained 4 as in the previous two tests. In this table the best values are shown in green whereas the worst values are shown in red. The blue color is used to highlights the values of the second best sat solver as blocks.

In the test cases 41 to 50 the number of rows R is set to 2 as well as the number of columns C. This means that we now have 4 elements in the matrix. Furthermore, the underlying field E is set to 4. Only the first two test cases are satisfiable. The other eight test cases are unsatisfiable.

We notice that Cryptominisat3 and Lingeling have a relatively bad performance as they require up to 70 seconds. On the other hand, Glucose , Minisat and Zenn are the fastest sat solvers and they require

Table 3.3.: AES Test 3: R=C=2 , R=4,C=2, R=C=4

I	N	R	C	E	Minisat	Cr3	Cr2	Ling	Zenn	Riss	Gluc	Doug	#pr	#cl=2	#cl=3	#cl=4	#cl	Sat?
41	1	2	2	4	0.011	0.052	0.038	0.049	0.017	0.04	0.035	0.019	422	312	672	1360	2344	Yes
42	2	2	2	4	1.737	5.052	4.373	1.034	1.486	3.304	3.869	2.345	848	624	1312	2816	4752	Yes
43	3	2	2	4	10.08	20.22	6.63	11.7	8.048	8.484	10.856	10.492	1274	936	1952	4272	7160	No
44	4	2	2	4	14.736	25.77	10.03	34.8	19.973	12.828	12.876	11.692	1700	1248	2592	5728	9568	No
45	5	2	2	4	13.008	36.06	11.49	21.1	15.233	14.648	14.460	10.824	2126	1560	3232	7184	11976	No
46	6	2	2	4	15.773	30.05	20.09	40.0	16.533	20.989	15.213	16.613	2552	1872	3872	7184	14384	No
47	7	2	2	4	17.03	34.48	26.67	34.8	23.03	21.58	17.69	20.05	2978	2184	4512	10096	16792	No
48	8	2	2	4	23.645	30.6	16.04	69.8	22.889	21.797	20.037	25.501	3404	2496	5152	11552	19200	No
49	9	2	2	4	53.955	39.81	63.28	74.3	41.722	20.241	21.041	46.130	3830	2808	5792	13008	21680	No
50	10	2	2	4	50.383	34.33	71.92	67.2	67.236	22.253	24.109	84.393	4256	3120	6432	14464	24016	No
51	1	2	2	8	145.06	260.65	87.55	214.5	27.71	36.124	124.53	242.61	2378	972	1768	13136	15876	Yes
52	2	2	2	8	10.43h	41.48h	179.13h	6.35h	9.91h	1.34h	7.56h	72.94h	4756	1944	3408	26528	31880	Yes
53	1	4	2	4	15.98	10.047	5.266	9.88	1.181	0.625	2.183	0.06	948	624	1408	3424	5456	Yes
54	2	4	2	4	1.38h	1.62h	0.72h	1.19h	0.51h	0.54	1.6h	0.4h	1896	1248	2688	7104	11040	Yes
55	1	4	2	8	-	-	-	-	-	33.17h	-	-	5076	1944	3536	28832	34312	Yes
56	1	4	4	4	4.1h	5.92h	8.17h	1.68h	19.54h	4.24h	>1 week	2.46h	1716	1072	2496	6368	9936	Yes

less than 25 seconds. When considering these 10 cases, Cryptominsat3 was at the last place (8 times) and then Lingeling (7 times).

Based on the data of these 10 test cases we derive the following relations between number of rounds and the number of variables and the number of clauses of a given length.

- number of variables = 422 + 426 * (number of rounds - 1)
- number of clauses length 2 = 312 * number of rounds
- number of clauses length 3 = 672 + 640 * (number of rounds - 1)
- number of clauses length 4 = 1360 + 1456 (number of rounds - 1)
- number of total clauses = 2344 + 2408 * (number of rounds -1)

In the test cases 51 and 52 E is equal to 8. Based on the result data it is recommended to use Riss3g. Minisat−Static and Cryptominisat2 are not recommended.

We also run test cases with N=3, R=C=2, and E=8. However, after 10 days, all sat solvers were unable to deliver a result.

In the test cases 53 and 54 E=4 , R=4 , C= 2. This means that we now have eight elements to be encrypted by AES. We notice that Minisat-Static is the fastest sat solver whereas Minisat and Cryptominisat3 have the worst performance.

In test case 55 E is set to 8. Only Riss3g was able to solve the equation system. It took 33 hours for delivering a solution. All other sat solvers were not able to deliver a result within 1 week.

We also run test cases with N=2, R=4, C=2, and E=8. However, after 10 days, all sat solvers were unable to deliver a result.

In the last test case N=1 and R=C=E=4 Lingeling was the fastest solver and it delivered a solution in 1,68 hour followed by Minisat-Static that delivered a solution in 2,46 hours. Even after 10 days Glucose was not able to deliver a solution.

We also run test cases with N=1, R=C=4, and E=8. Even after 14 days, all sat solvers were unable to deliver a result.

Based on these results we can state that AES (N=10, R=C=4, E=8) is unbreakable with the state of art computing resources.

3.2.6 Stability Cases 42 and 53

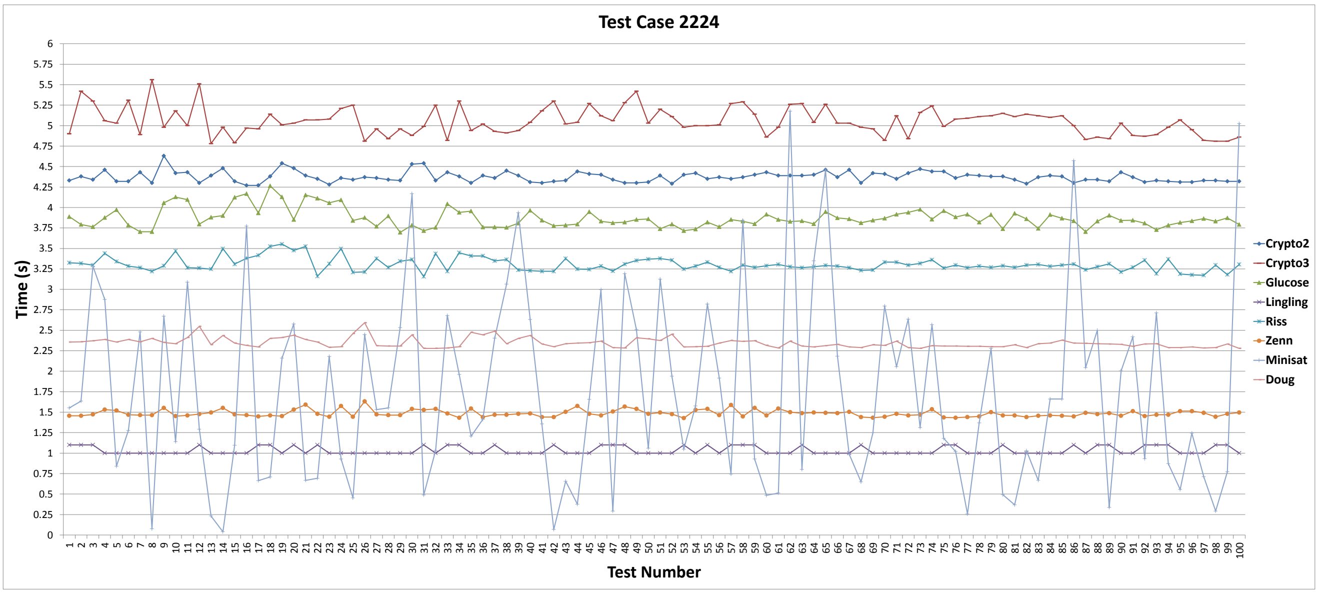

Figure 4.: AES Stability Case 42

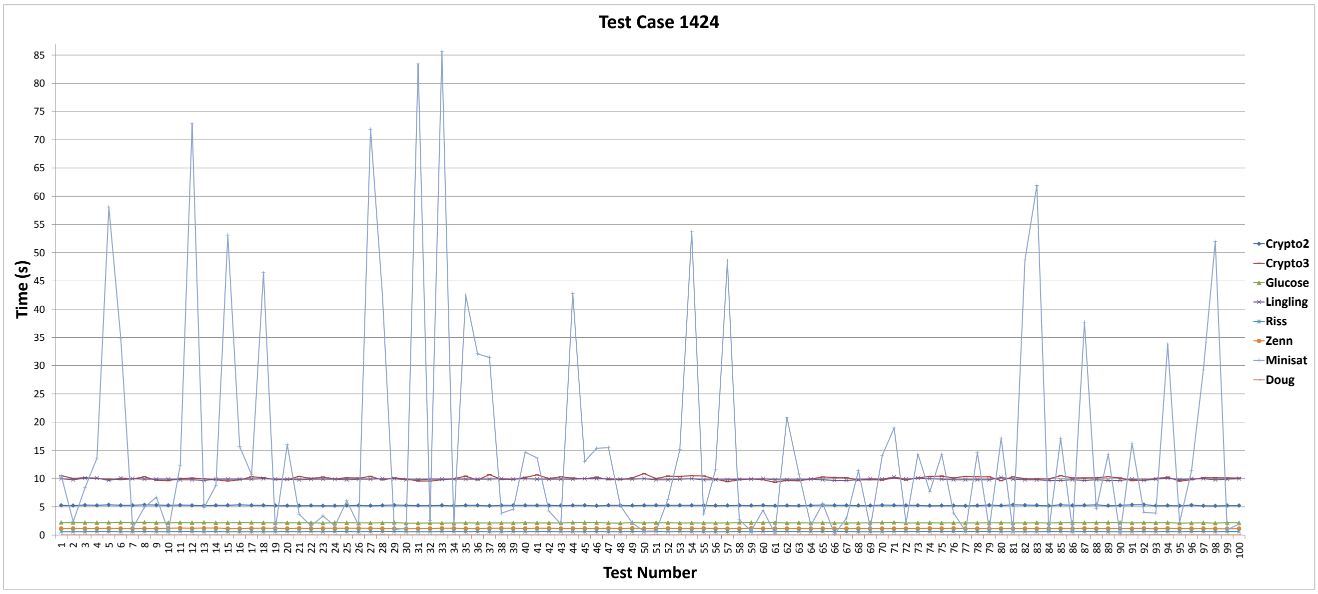

Figure 5.: AES Stability Case 53

Figure 4 shows the stability curve of the used eight sat solvers when R=C=N=2 and E=4. This figure shows that Minisat is very unstable as already shown in Figures 2 and 3. The curve shows that Lingeling and Zenn are quite stable.

Figure 5 shows the stability curve of the used eight sat solvers when N=1,R=4,C=2 and E=4. As usual, Minisat is very unstable whereas all other seven sat solvers are stable. Furthermore, for delivering the same solution Minisat may take up to 85 seconds, Lingeling may take up to 10 seconds, and the other deliver a result in less than 5 seconds.

4 Light Encryption Device (LED)

In this chapter, we first report on the different programs and scripts that were developed to perform the tests of the LED block cipher. Then, we present the results of these tests using several tables and we discuss these results. Finally some curves are presented, which show graphically the stability of the eight different sat solvers that were used.

4.1 Tools for LED Tests

As for the AES and CSA tests we had to write several programs to automate and support the LED tests.

For the LED equation generator we reused the code developed in the context of Julian Wälde bachelor thesis [23] with some minor modifications to the method `getPolynomialSystem` as shown in Listing 4.1.

```
r = int(sys.argv[1])
key = random.randint(0,1<<64)
p = random.randint(0,1<<64)
c = encrypt(p,key,rounds=r)
if int(sys.argv[3])!= 0:
 key = random.randint(0,1<<64)
system = getPolynomialSystem(p,c,r,key,int(sys.argv[2]))
solver = anf2cnf.ANFSatSolver(R,6)
print(solver.cnf(system))
```

Listing 4.1: Changes to the LED Equation generator

In addition, we used the eight bash scripts to run the experiments 100 times as in the previous two chapters. Furthermore, we used the java class `ClauseCounter` to get the length of each clause and the java class `Javerage` to get the average time taken by each sat solver.

4.2 Tests and Results

In this section, we present the results of the tests that were conducted on the LED block cipher. In each subsection we vary the number of rounds R (from 1 to 48), the number of guessed bits (60 to 0) as well as the value of truly or randomly guessing bits G (which can be either 0 or 1). The results of all tests are shown in five tables, which all have the following column structure.

Columns 1 to 3 respectively show the number of rounds R, the number of guessed bits N, and if the guessing was correct or randomly. Columns 4 to 11 show the time in seconds that is taken by the respective solver for each test case. Column 12 shows the complexity of the equation system based on the number of guessed bits multiplied by the least needed time amount for the test case. Column 13 states if the equation system is satisfiable or not. Column 14 ($\#vars$) shows the number of variables used to solve the equation system. Column 15 shows the total number of clauses. Columns 16 to 20 show respectively the number of clauses with length 1, 2,3,4,5 and 6. In this table the best values are shown in green whereas the worst values are shown in red.

4.2.1 Test 1: R =1

Table 4.1 shows the results of this test, which contains 13 test cases. The number of rounds R was fixed to 1 whereas the number of guessed bits N varies from 0 to 48 and the value of the correct guessing G is either 0 (in the first 6 cases) or 1 (in the last 7 cases). All sat solvers were able to solve the first round fully.

Table 4.1.: LED Test 1: R=1

I	R	N	G	Mini	Cr3	Cr2	Li	Ze	Glu	doug	Riss	C	S?	#pr	#cl	l1	l2	l3	l4	l5	l6
1	1	48	0	0.06	0.05	0.03	0.003	0.06	0.09	0.03	0.03	2^{40}	Y	1514	19266	176	1906	360	760	2176	13888
2	1	32	0	0.06	0.044	0.045	0.021	0.063	0.093	0.011	0.019	2^{25}	Y	1514	19250	160	1906	360	760	2176	13888
3	1	16	0	0.065	0.076	0.005	0.029	0.068	0.096	0.014	0.02	2^{8}	Y	1514	19234	144	1906	360	760	2176	13888
4	1	4	0	2.268	2.522	3.726	10.4	3.516	8.295	10.91	1.657	2^{5}	Y	1514	19222	132	1906	360	760	2176	13888
5	1	1	0	3.81	6.95	134.27	5.362	0.414	1.611	6.685	0.702	-2.5	Y	1514	19219	129	1906	360	760	2176	13888
6	1	0	0	53.77	48.59	76.54	31.075	33.76	36.74	54.71	33.5	31	Y	1514	19218	128	1906	360	760	2176	13888
7	1	48	1	0.02	0.03	0.02	0.001	0.017	0.03	0.02	0.03	2^{38}	N	1514	19282	192	1906	360	760	2176	13888
8	1	8	1	1.39	1.54	0.83	3.0	1.57	1.15	1.1	1.35	212.5	N	1514	19226	136	1906	360	760	2176	13888
9	1	4	1	23.35	14.11	16.7	11.1	16.67	15.07	35.17	27.5	177.6	N	1514	19222	132	1906	360	760	2176	13888
10	1	3	1	21.25	24.16	17.25	26.8	24.8	25	36.9	10.67	85.36	N	1514	19221	131	1906	360	760	2176	13888
11	1	2	1	2.3	10.61	76.4	5.2	36.4	1.66	10.11	34	6.64	Y	1514	19220	130	1906	360	760	2176	13888
12	1	1	1	38.81	62.81	175.02	77.2	27.41	81.7	49	9.2	54.8	Y	1514	19219	129	1906	360	760	2176	13888
13	1	0	1	2.24	0.41	4.26	42.7	17.83	111	60	51	0.41	Y	1514	19218	128	1906	360	760	2176	13888

The first 6 test cases were always satisfiable by all used sat solvers. Based on the time required by each sat solver we recommend using Lingeling and Riss3g, which were the fastest sat solver to solve the problem. In contrast, Glucose and Cryptominisat2 were the slowest sat solvers and therefore we recommend avoiding them in situations such as those in test cases 1 to 6.

In the last 6 cases guessing was done randomly (G=1). The first four cases were unsatisfiable, which can be explained by the high number of randomly guessed bit. The last three cases were satisfiable, which can be explained by the small number of guessed bits (0 to 2). Based on this we recommend to guess at most 2 bits of the key randomly in order to get a satisfiable system.

In the last 6 test cases, the slowest sat solvers are Cryptominisat2 and Minisat Static. In contrast, Riss3g and Minisat are the fastest ones.

Concerning the length of clauses, we noticed that only the number of clauses with length 1 changes in each iteration. For example, in the first test case N is equal to 48 and in the second case N is equal to 32. The difference in the total number of clauses (column 16) between the values for these two cases is equal to 16. This rule applies in all five tables. However, the number of clauses with length 2, 3,4,5,6 and the number of variables do not change. Concerning the complexity which in presented in Column 11, it is rather constant $\Theta(1)$ (in the last 6 cases).

All eight used sat solvers were able to solve the first round fully (i.e., the number of guessed bits equals zero).

4.2.2 Stability Case 1160

Figure 6 shows the stability curve of the used eight sat solvers when R=1, N=16 and G=0. This figure shows that Glucose is very unstable whereas Lingeling is quite stable.

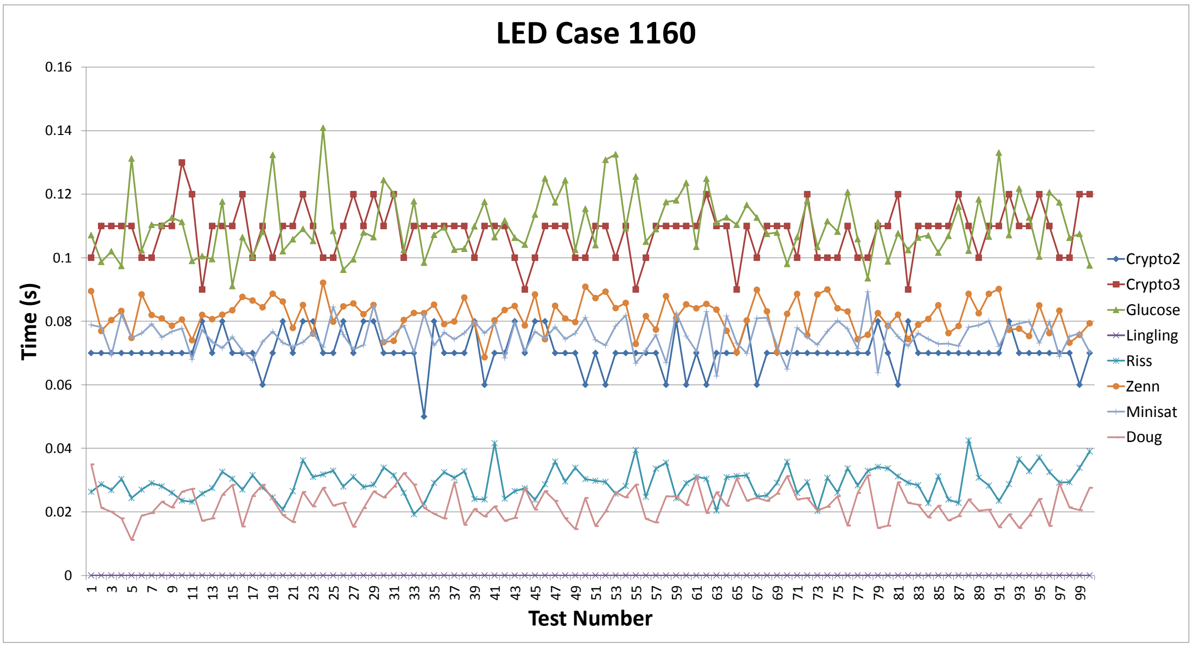

Figure 6.: LED Stability Case 1160

In the next test, we increment the number of rounds to 2 and repeat the same tests.

4.2.3 Test 2: R=2

Table 4.2 shows the results of this second test, which contains 9 test cases. The number of rounds R was fixed to 2 whereas the number of guessed bits N varies from 12 to 48 and the value of the correct guessing G is either 0 (in the first 5 cases) or 1 (in the last 4 cases). All eight sat solvers were not able to solve fully this round (N=0).

Table 4.2.: LED Test 2: R=2

I	R	N	G	Mini	Cr3	Cr2	Li	Ze	Glu	doug	Riss	C	S?	#pr	#cl	l1	l2	l3	l4	l5	l6
1	2	48	0	0.6	1.46	0.95	0.32	0.61	0.84	0.11	0.13	2^{41}	Y	7939	147034	176	9466	880	3152	3472	129888
2	2	32	0	1.085	4.134	1.9	2.6	1.45	1.88	1.19	1.01	2^{32}	Y	7939	147018	160	9466	880	3152	3472	129888
3	2	28	0	1.43	6.75	2.13	11.9	4.49	1.15	1.15	2.54	2^{28}	Y	7939	147014	156	9466	880	3152	3472	129888
4	2	16	0	217.5	946.7	217.5	219.1	1436.2	258.4	1458.6	193	2^{23}	Y	7939	147002	144	9466	880	3152	3472	129888
5	2	12	0	4.18h	11.36h	8.75h	6.2h	4.05h	2.11h	1.88h	2.82h	2^{25}	Y	7939	146998	140	9466	880	3152	3472	129888
6	2	48	1	0.6	1.15	0.85	0.32	0.62	0.92	0.08	0.12	2^{44}	N	7939	147034	176	9466	880	3152	3472	129888
7	2	32	1	2.75	7.21	4.28	6	3.04	3.46	2.15	3.23	2^{33}	N	7939	147018	160	9466	880	3152	3472	129888
8	2	16	1	619.5	0.55h	240.6	0.33h	582	266	0.74h	312	2^{24}	N	7939	147002	144	9466	880	3152	3472	129888
9	2	15	1	7.56h	5.69h	-	2.71h	4.3h	-	7.9h	8.55h	2^{28}	N	7939	147001	143	9466	880	3152	3472	129888

The first 5 cases are satisfiable. We used the strategy bottom down, which means that we start with guessing 64 bits and then decrease at each case the number of guessed bits till the sat solver is unable to solve the problem within 1 week. The minimal number of guessing bits is equal to 12, i.e., a solution was found when guessing 12 bits in 2 to 11 hours.

Based on the results of the first 5 cases it is recommended to use Riss3g and Minisat static, which were the fastest when R=2 and G=0. In addition, one should avoid using Cryptominisat3 and Cryptominisat2 due to their bad performance.

The last 4 test cases were not satisfiable, which can be explained by the high number of randomly guessed bits). We tried the case R=2,N=0,G=1 but even after 2 weeks the eight sat solvers were not able to deliver a result. Based on that we recommended using Minisat static or Lingeling and avoiding Cryoptominisat3 . In addition, it is noteworthy that Cryptominisat2 and Glucose were not able to solve the last case.

We notice also that the complexity in the last 4 test cases is very big 2^{32}, which may explain why we are not able to solve round 2 fully. Concerning the number of parameters and clauses, the rules mentioned for Table 4.1 are still true.

We also run test cases with R=2, N=0, and G=0. However, after 14 days, all sat solvers were unable to deliver a result.

Next, we will increment number of rounds to 3.

4.2.4 Test 3: R=3

Table 4.3 shows the results of this third test, which contains 9 test cases. The number of rounds R was fixed to 3 whereas the number of guessed bits N varies from 32 to 56 and the value of the correct guessing G is either 0 (in the first 5 cases) or 1 (in the last 4 cases).

Table 4.3.: LED Test 3: R=3

I	R	N	G	Mini	Cr3	Cr2	Li	Ze	Glu	doug	Riss	C	S?	#pr	#cl	l1	l2	l3	l4	l5	l6
1	3	56	0	1.23	3.64	1.6	0.6	1.27	1.66	0.22	0.25	2^{52}	Y	14365	274826	184	17030	1404	5520	4768	245920
2	3	48	0	6.52	11.47	4.32	27.01	7.24	23.4	4.442	3.03	2^{49}	Y	14365	274818	176	17030	1404	5520	4768	245920
3	3	40	0	127.35	301.53	101.24	359.3	194	186.05	136.7	235.6	2^{47}	Y	14365	274810	168	17030	1404	5520	4768	245920
4	3	36	0	2824.9	2170.6	493.7	2213.2	433.8	546.7	959.7	2822.5	2^{45}	Y	14365	274806	164	17030	1404	5520	4768	245920
5	3	32	0	3.73h	6.74h	5.04h	11.87h	2.07h	8.67h	1.19h	1.72h	2^{44}	Y	14365	274802	160	17030	1404	5520	4768	245920
6	3	56	1	1.24	3.64	1.72	1.4	1.28	1.74	0.22	0.34	2^{54}	N	14365	274826	184	17030	1404	5520	4768	245920
7	3	48	1	13.71	64.9	25.13	49.1	22.9	37.4	17.6	33.6	2^{52}	N	14365	274818	176	17030	1404	5520	4768	245920
8	3	36	1	1.47h	3.45h	0.16h	0.57h	1.21h	0.67h	2.67h	0.44h	2^{45}	N	14365	274806	164	17030	1404	5520	4768	245920
9	3	34	1	4.4h	2.06h	0.82h	1.37h	1.16h	2.05h	8.65h	1.85h	2^{46}	N	14365	274804	162	17030	1404	5520	4768	245920

The 5 first test cases were satisfiable due to the high number of correctly guessed bits. On the other hand, the last 4 test cases were unsatisfiable due to the high number of randomly guessed bits. In addition, we notice that Cryptominisat2, Minisat Static and Riss3g are the fastest sat solvers whereas Lingeling and Glucose are the slowest ones. Concerning the complexity, when the number of guessing bits is too high, the complexity is huge $\Theta(2^n)$. For example, when N equals to 56 the complexity is 2^{52}.

The last 4 test cases are all unsatisfiable. Cryptominisat3 is the slowest one. Based on that, we suggest using Cryptominisat2 and Riss3g. The same rule for complexity is still applicable: When the number of guessed bits is huge, the complexity is very big. Nevertheless, it is noteworthy that when N is set to 0 or 1 the sat solvers need more than one month to find the solution.

In this test we were able to guess half of the key bits and find a solution for the formula. In fact, in test case 5, we guessed 32 bits out of 64 bits of the key. Compared to the previous test, in which we guessed only 12 bits to find a solution (cf. test case 5 in Section 4.2.3).

We also run test cases with R=3, N=0, and G=0. However, even after 14 days all sat solvers were still unable to deliver a result. Hence, round 3 of CSA is not fully breakable with the currently available computing resources.

Next, we increment the number of rounds to 4 and 5 because the number of rounds increases and therefore it is difficult to find solution. In fact, we were not able to fully solve rounds 2 and 3. Therefore, it is unlikely that rounds 4 and 5 can be broken fully.

4.2.5 Stability Case 3400

Figure 7 shows the stability curve of the used eight sat solvers when R=3, N=34 and G=0. This figure shows that Riss3g is quite stable. On the other hand Lingeling is not stable any more. Also the other six sat solvers are not stable.

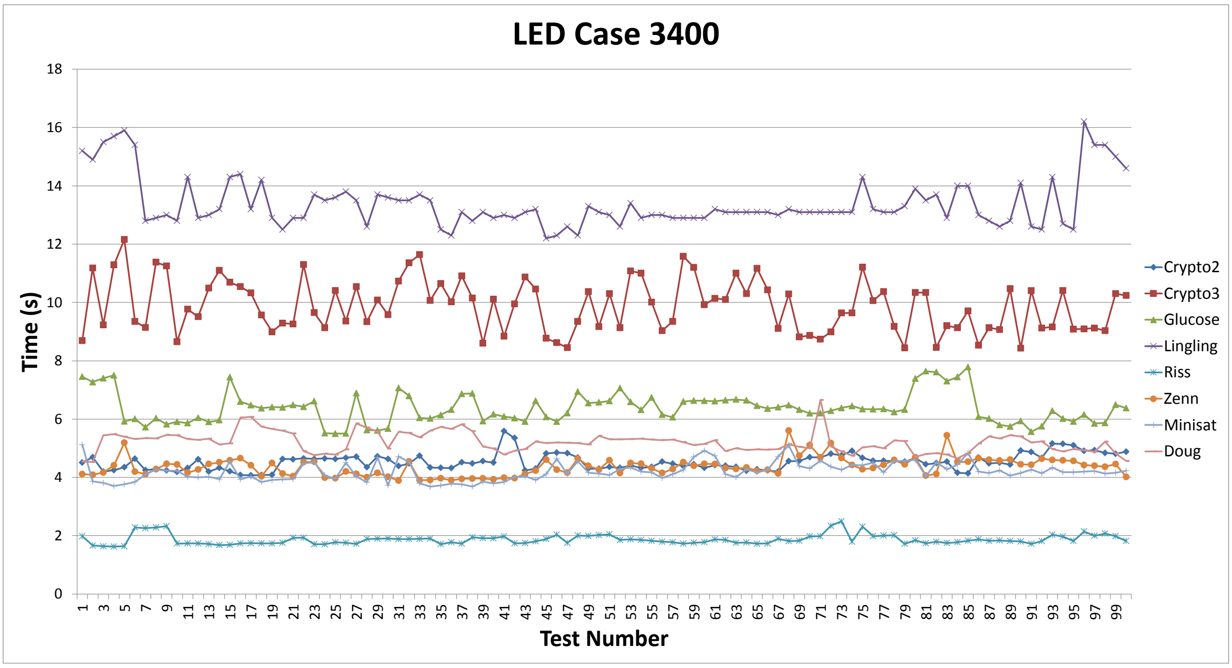

Figure 7.: LED Stability Case 3400

4.2.6 Test 4: R= 4 and R=5

Table 4.4 shows the results of this test, which contains 15 test cases. The number of rounds R was fixed to 4 and then 5 whereas the number of guessed bits R varies from 44 to 56 and the value of the correct guessing G is either 0 (in the first 4 cases and in cases 9 to 12) or 1 (in the cases 5 to 8 and in the last 3 cases).

Table 4.4.: LED Test 4: R=4 and R=5

I	R	N	G	Mini	Cr3	Cr2	Li	Ze	Glu	doug	Riss	C	S?	#pr	#cl	l1	l2	l3	l4	l5	l6
1	4	56	0	24.5	15.83	3.17	17.7	19.23	15.36	4.38	7.7	2^{58}	Y	20786	402436	184	24596	1928	7888	6048	361792
2	4	52	0	68.38	56.74	7.22	64.5	12.73	23.21	119.05	12.51	2^{55}	Y	20786	402432	180	24596	1928	7888	6048	361792
3	4	48	0	433.5	569.4	199	325.2	93.25	442.5	2121.3	306.84	2^{55}	Y	20786	402428	176	24596	1928	7888	6048	361792
4	4	44	0	1.54h	6.04h	4.15h	2.33h	0.65h	2.02h	3.13h	4.56h	2^{55}	Y	20786	402424	172	24596	1928	7888	6048	361792
5	4	56	1	8.18	15.23	3.95	6	9.11	14.2	18.41	3.4	2^{58}	N	20786	402436	184	24596	1928	7888	6048	361792
6	4	48	1	522	638	197	357	638	320	3456	379	2^{55}	N	20786	402428	176	24596	1928	7888	6048	361792
7	4	46	1	807.6	2543.2	1081.4	20255	1838	2534.6	4863.7	2297	2^{56}	N	20786	402426	174	24596	1928	7888	6048	361792
8	4	44	1	-	-	2.3	8.01h	-	-	-	21.75h	2^{57}	N	20786	402424	172	24596	1928	7888	6048	361792
9	5	56	0	53.78	26.66	10.7	2.8	89.34	20.16	25.27	20.1	2^{57}	Y	36707	664415	184	49388	3299	15448	7424	588672
10	5	52	0	49.79	152.4	28.79	385.8	280.24	92.59	119.2	187.5	2^{57}	Y	36707	664411	180	49388	3299	15448	7424	588672
11	5	48	0	205.3	598.3	88.5	1142.7	142.48	546.7	661.1	234.7	2^{54}	Y	36707	664407	176	49388	3299	15448	7424	588672
12	5	44	0	2.59h	9.03h	6.08h	1.57h	2.23h	7.84h	3.4h	9.77h	2^{56}	Y	36707	664403	172	49388	3299	15448	7424	588672
13	5	56	1	30.89	32.62	7.77	154.2	35.86	11.14	489.9	38.24	2^{59}	N	36707	664415	184	49388	3299	15448	7424	588672
14	5	48	1	0.19h	0.55h	0.2h	0.47h	0.29h	0.38h	3.05h	0.39h	2^{57}	N	36707	664407	176	49388	3299	15448	7424	588672
15	5	46	1	0.91h	3.05h	1.27h	1.26h	11.74h	1.31h	0.97h	2.01h	2^{58}	N	36707	664405	174	49388	3299	15448	7424	588672

When R equals to 4 the first 4 test cases were satisfiabe and the next four cases (cases 5 to 8) were unsatisfiable. Based on the results, when G = 0 we recommend using Zenn and Cryptominisat2 . Cryptominisat3 and Minisat static were the slowest sat solver and therefore it should be avoided.

In the test cases 5 to 8, the guessing was done randomly and this could explain why the equation system was unsatisfiale for all sat solvers. In test case 8, only Cryptominisat2 (2,3h), Lingeling(8h), and Riss3g (21h) were able to deliver a solution in less than a day. The other solvers were not able to deliver a result even after one week. In these test cases Cryptominisat2 is still the fastest sat solver and Cryptominisat3 and Minisat static are still the slowest one.

When R equals to 5, we notice that Lingeling and Cryptominisat2 are the fastest sat solvers in the test cases 9 to 12. On the other hand, the guessing was done randomly in the test cases 13 to 15 and in these cases Minisat static was the worst sat solver.

Concerning complexity the same rules as in Test 2 and Test 3 is still true. When looking globally at table 4.4, we notice that Lingeling and Cryptominsiat2 are the fastest sat solvers whereas Cryptominsiat3 is the slowest one.

Next, we summarize the results for 5 test cases in which R varies from 6 to 10 in Table 4.5.

4.2.7 Test 5: R= 6 to 10

Table 4.5 shows the results of this test, which contains 30 test cases. The number of rounds R varies from 6 to 10 whereas the number of guessed bits N varies from 44 to 58 and the value of the correct guessing G is either 0 or 1.

Table 4.5.: LED Test 5: R= 6 to 10

I	R	N	G	Mini	Cr3	Cr2	Li	Ze	Glu	doug	Riss	C	S?	#pr	#cl	l1	l2	l3	l4	l5	l6
1	6	56	0	35.13	325.23	10.88	449.5	59.4	17.73	60.83	6.85	2^{59}	Y	43145	792567	184	56948	3835	17824	8688	705088
2	6	50	0	138.04	335.77	202.15	1273.58	190.36	120.817	110.95	424.35	2^{57}	Y	43145	792561	178	56948	3835	17824	8688	705088
3	6	44	0	31.34h	44.34h	4.89h	9.08h	4.07h	8.57h	11.91h	29.54h	2^{58}	Y	43145	792555	172	56948	3835	17824	8688	705088
4	6	56	1	62.14	295.7	24.55	149.8	82.32	42.07	41.03	37.76	2^{61}	N	43145	792567	184	56948	3835	17824	8688	705088
5	6	54	1	125.24	538.27	46.1	468.5	100.24	95.31	118.3	72.03	2^{60}	N	43145	792565	182	56948	3835	17824	8688	705088
6	6	47	1	2.34h	2.45h	0.64h	4.27h	1.66h	1.3h	3.96h	1.34h	2^{58}	N	43145	792558	175	56948	3835	17824	8688	705088
7	7	57	0	26.4	114.5	16.57	45.3	17.31	21.05	0.83	73.79	2^{57}	Y	49575	920502	185	64506	4355	20224	9984	821248
8	7	52	0	6.97	172.41	140.77	166.7	7.1	135.95	190.54	60.35	2^{55}	Y	49575	920497	180	64506	4355	20224	9984	821248
9	7	46	0	1.97h	6.33h	2.16h	4.78h	0.96h	0.17h	0.73h	2.44h	2^{55}	Y	49575	920491	174	64506	4355	20224	9984	821248
10	7	57	1	41.98	277	65.6	64	50.85	84.73	37.9	7.13	2^{60}	N	49575	920502	185	64506	4355	20224	9984	821248
11	7	53	1	116.25	210.96	74.89	1105	171.07	165.14	2405	130.1	2^{59}	N	49575	920498	181	64506	4355	20224	9984	821248
12	7	47	1	1.73h	3.41h	1.1h	4.8h	1.64h	1.33h	1.9h	1.85h	2^{59}	N	49575	920492	175	64506	4355	20224	9984	821248
13	8	57	0	6.51	80.51	30.24	77.9	7.59	38.53	14.46	1.04	2^{57}	Y	56002	1048314	185	72066	4879	22624	11248	937312
14	8	54	0	37.62	372.74	67.62	238.6	42.68	331.2	10.78	24.7	2^{57}	Y	56002	1048311	182	72066	4879	22624	11248	937312
15	8	46	0	0.74h	1.18h	1.11h	19.67h	0.05h	1.15h	0.60h	3.06h	2^{53}	Y	56002	1048303	174	72066	4879	22624	11248	937312
16	8	57	1	17.07	87.21	13.37	59.8	17.7	37.19	12.24	7.91	2^{60}	N	56002	1048314	185	72066	4879	22624	11248	937312
17	8	55	1	63.24	533.18	40.07	390.5	58.05	59.07	43.37	67.51	2^{60}	N	56002	1048312	183	72066	4879	22624	11248	937312
18	8	47	1	13.38h	3.45h	1.37h	16.45h	2.46h	1.86h	2.45h	2.34h	2^{59}	N	56002	1048304	175	72066	4879	22624	11248	937312
19	9	58	0	9.49	64.64	13.48	34.5	11.05	11.17	7.19	14.43	2^{61}	Y	71929	1310634	186	96848	6256	30160	12608	1164576
20	9	55	0	49.22	203.03	24.37	77.8	60.2	95.8	13.97	323.46	2^{59}	Y	71929	1310631	183	96848	6256	30160	12608	1164576
21	9	47	0	1.86h	3.79h	1.15h	7.22h	1.46h	3h	0.32h	3.08h	2^{57}	Y	71929	1310623	175	96848	6256	30160	12608	1164576
22	9	58	1	16.43	73.36	12.37	33.1	16.72	37.07	6.75	6.43	2^{61}	N	71929	1310634	186	96848	6256	30160	12608	1164576
23	9	55	1	112.29	314.41	40.82	258.9	128.74	82.61	52.1	200.08	2^{60}	N	71929	1310631	183	96848	6256	30160	12608	1164576
24	9	47	1	21.62h	4.53h	1.33h	16.1h	2.6h	2.83h	10.08h	3.15h	2^{59}	N	71929	1310623	175	96848	6256	30160	12608	1164576
25	10	58	0	21.23	331.9	14.47	7.5	17.72	74.24	15.98	4.35	2^{60}	Y	78357	1438466	186	104412	6788	32520	13888	1280672
26	10	55	0	36.1	884.3	68.87	524.3	130.23	28.91	33.89	35.73	2^{60}	Y	78357	1438463	183	104412	6788	32520	13888	1280672
27	10	47	0	1.87h	0.33h	0.47h	12.32h	3.47h	2.03h	0.08h	2.28h	2^{55}	Y	78357	1438455	175	104412	6788	32520	13888	1280672
28	10	58	1	12.45	64.21	23.5	25.6	13.47	17.01	5.11	5.4	2^{60}	N	78357	1438466	186	104412	6788	32520	13888	1280672
29	10	56	1	32.2	155.36	44.29	173	37.83	221.62	108.41	100.47	2^{61}	N	78357	1438464	184	104412	6788	32520	13888	1280672
30	10	47	1	4.03h	5.77h	1.98h	19.36h	2.81h	2.57h	3.7h	3.71h	2^{60}	N	78357	1438455	175	104412	6788	32520	13888	1280672

When looking at this table globally, we notice that Cryptominsat2, Lingeling and Minisat Static are the fastest sat solvers whereas Cryptominisat3, Zenn and Glucose are the slowest ones.

The first three test cases (1 to 3) are satisfiable and following three test cases (4 and 6) are unsatisfiable. In such cases (i.e., when R equals 6), we recommend using either Minisat Static or Cryptominisat2. The sat solvers Cryptominisat3 and Lingeling should be avoided in these cases.

In the test cases 7 to 12 R is equal to 7. The table shows that 3 test cases were satisfiable (when G=0) and 3 were unsatisfiable (when G=1). Based on the results,we notice that Cryptominisat2 stilles the best solver and we suggest avoiding cryptominisat3 and Minisat static.

In the test cases 13 to 18 R is equal to 8. As in the cases 7 to 12, three test cases were satisifable (when G=0) and three test cases were not satisfiable. The tables shows that Cryptominisat3 (presented with color Red) is the slowliest sat solver . Based on the results we recommend using Minisat Static and cryptominisat2.

In the test cases 19 to 24 R is equal to 9. As in the cases 13 to 15, three test cases were satisifable (when G=0) and three test cases were not satisfiable. The same applies also for the test cases 25 to 30, in which R is equal to 10. Based on the results of the test cases 19 to 30 Minisat Static is recommended as it is the fastest sat solver whereas Cryptominisat2 and Ligneling should be avoided as they are the slowest sat solvers.

Next, we summarize the results for 3 test cases in which R varies from 16 to 48 in Table 4.6.

4.2.8 Test 6: R= 16, 32 and 48

Table 4.6 shows the results of this test, which contains 18 cases. The number of rounds R was fix to 16, 32 then 48 whereas the number of guessed bits N varies from 46 to 60 and the value of the correct guessing G is either 0 or 1 .

Table 4.6.: LED Test 6: R= 16, 32 and 48

I	R	N	G	Mini	Cr3	Cr2	Li	Ze	Glu	doug	Riss	C	S?	#pr	#cl	l1	l2	l3	l4	l5	l6
1	16	59	0	10.58	129.2	23.15	72.7	11.73	22.94	5.64	6.05	2^{61}	Y	126419	2340023	187	166998	10806	51872	21680	2088480
2	16	56	0	74.29	224.73	62.41	78.1	76.83	170.63	43.67	30.57	2^{61}	Y	126419	2340020	184	166998	10806	51872	21680	2088480
3	16	46	0	3.54h	10.36h	1.82h	32.74h	0.08h	10.27h	3.44h	7.08h	2^{54}	Y	126419	2340010	174	166998	10806	51872	21680	2088480
4	16	59	1	17.29	122.6	29.43	140.7	16.49	26.1	7.43	6.47	2^{62}	N	126419	2340023	187	166998	10806	51872	21680	2088480
5	16	56	1	217.01	706.61	144.6	143.6	99.43	74.48	502.5	58.41	2^{62}	N	126419	2340020	184	166998	10806	51872	21680	2088480
6	16	47	1	6.76h	9.31h	6.58h	17.91h	5.87h	4.71h	6.3h	4.66h	2^{61}	N	126419	2340011	175	166998	10806	51872	21680	2088480
7	32	60	0	27.72	193.66	40	77.4	30.16	40.26	3.13	6.54	2^{62}	Y	267256	4923655	188	356834	22601	110560	42720	4390752
8	32	56	0	204.5	550.4	163.2	361.7	176.07	106.46	160.29	92.06	2^{62}	Y	267256	4923651	184	356834	22601	110560	42720	4390752
9	32	48	0	1.99h	3.33h	0.38h	24.11h	3.05h	3.76h	3.27h	0.42h	2^{52}	Y	267256	4923643	176	356834	22601	110560	42720	4390752
10	32	60	1	43.2	172	37.3	190.5	43.48	51.41	8.36	51.41	2^{63}	N	267256	4923655	188	356834	22601	110560	42720	4390752
11	32	56	1	315.9	832.7	237.01	204	327.2	239.5	182.8	518	2^{63}	N	267256	4923651	184	356834	22601	110560	42720	4390752
12	32	50	1	5.19h	2.54h	1.45h	5h	2.08h	1.48h	1.92h	1.85h	2^{62}	N	267256	4923645	178	356834	22601	110560	42720	4390752
13	48	60	0	31.43	244.3	38	107.5	34.16	66.57	14.97	9.23	2^{63}	Y	408046	7506237	188	546646	34419	169048	64000	6691936
14	48	55	0	371.24	918.02	530.21	466.8	484.7	261.1	282.3	63.9	2^{61}	Y	408046	7506232	183	546646	34419	169048	64000	6691936
15	48	50	0	2.61h	2.35h	0.44h	1.61h	0.12h	2.34h	3.66h	1.8h	2^{59}	Y	408046	7506227	178	546646	34419	169048	64000	6691936
16	48	60	1	48.54	328.8	33.72	42.6	48.65	61.39	12.42	18.3	2^{64}	N	408046	7506237	188	546646	34419	169048	64000	6691936
17	48	55	1	410.5	1253.2	572.1	707.3	458.8	835.8	330.07	1417.1	2^{63}	N	408046	7506232	183	546646	34419	169048	64000	6691936
18	48	50	1	7.56h	4.05h	2.97h	9.2h	2.94h	2h	6.21h	3.9h	2^{63}	N	408046	7506227	178	546646	34419	169048	64000	6691936

When looking at this table globally, we notice that Riss3g and Minisat Static are the fastest sat solvers whereas Cryptominisat3 and Lingeling are the slowest ones.

The first three test cases (1 to 3) are satisfiable and following three test cases (4 and 6) are unsatisfiable. In such cases (i.e., when R equals 16), we recommend using Minisat Static. The sat solvers Cryptominisat3 and Lingeling should be avoided in these cases.

In the test cases 7 to 12 R is equal to 32. The table shows that 3 test cases were satisfiable (when G=0) and 3 were unsatisfiable (when G=1). Based on the results, we suggest using Minisat static and Riss3g and avoiding Cryptominisat3. In the test cases 13 to 18 R is equal to 48. As in the cases 7 to 12, three test cases were satisifable (when G=0) and three test cases were not satisfiable. Based on the results we recommend using Minisat static and suggest avoiding Cryptominisat3

To sum up, we were able to solve fully the first round of LED and some of the other rounds (2 to 48) by guessing an appropriate number of key bits. In round 48 we guessed 50 bits out of 64 and were able to solve LED. Without guessing any bit breaking LED is impossible with the currently available hardware resources.

5 Common Scrambling Algorithm (CSA)

In this chapter, we first report on the different programs and scripts that were developed to support the different tests of the CSA block cipher. Then, we present the results of these tests using several tables and discuss these results. Finally some curves are presented, which show graphically the stability of the eight different sat solvers that were used in the CSA tests.

5.1 Tools for CSA tests

Several programs were necessary to automate the CSA tests. First, a modified part of a sage class [23] will be presented which generates the CNF file that acts as input for the sat solvers. Then we present another sage class called converter, which transforms nonlinear equations to linear equations.

5.1.1 CSA Equation Generator

Listing 5.1 shows the source code of the modified SAGE class that was developed in the context of the bachelor thesis of Julian Wälde [23] . We had to change the method `encodelinear` of that class so that the generated file can be used by the eight sat solvers. The original version of that method works only with the sat solver Cryptominisat2. Without this modification only Cryptominisat2 can read the CNF file and the other seven sat solvers produce a syntax error message.

```
def encodelinear(p):
        if p.deg() != 1: raise ValueError("polynomial must be of degree 1")
        if len(p) == 1: #raise ValueError("polynomial must have length > 1")
                v = p.vars_as_monomial().variables()[0]
                r = str(v.index()+1)+" 0"
                if not p.has_constant_part():
                        r = "-"+r
                return r
        var = list(p.vars_as_monomial().variables())
        idx = [v.index()+1 for v in var]
        if not p.has_constant_part(): #idx[-1] = -idx[-1]
                return "x"+("+x".join(map(lambda x:str(x),idx)))
        else:
                return "x"+("+x".join(map(lambda x:str(x),idx)))+"+1"

        return p
```

Listing 5.1: CSA Equation generator

We modified only the method `encodelinear` so that nonlinear equations are converted to linear equations. Listing 5.2 shows as an example with a part of the generated file.

```
152 159 −151 150 −149 146 147 −145 0
−152 −159 151 −150 −148 −147 145 0
159 −151 149 148 146 147 −145 0
−152 −159 151 −150 149 −148 146 −145 0
x18+x65+x129
x57+x66+x130
x50+x67+x131
x8+x68+x132
```

Listing 5.2: Example CSA Output

.

5.1.2 Converter of Non-linear Equations to Linear Equations

An additional advantage of the modification to the function `encodelinear` is the reduction of the length of clauses. In the original version the length of clauses can reach the value 30 and the file size can exceed 10 GB (e.g., in round 20). In our modified version the maximum length value is 9 and the maximum file size is 2 MB (e.g, in round 20) . This is achieved by setting max_vars_sparse to 2.

Listing 5.3 shows a simplified example showing the conversion of nonlinear equations to linear one. First, a sufficient number of variables is declared so that the original names of variables does not change. In this example, 10 variables are declared but this can be up to 40.000 variables. Then, all nonlinear equations are added to a list L using the method `append`. After that, the CNF encoder is called once to operate on the whole list L and the result is written to the output file `result.txt`.

```
B.<x1,x2,x3,x4,x5,x6,x7,x8,x9,x10> = BooleanPolynomialRing()
from sage.sat.converters.polybori import CNFEncoder
from sage.sat.solvers.dimacs import DIMACS
fn = 'tmp.txt'
nf = open('result.txt','w')
L = []
L.append(x1+x6+x2)
L.append(x5+x6+x8)
L.append(x5+x7+x4)
solver = DIMACS(filename=fn)
e=CNFEncoder(solver, B, max_vars_sparse=2)
e(L)
solver.write(filename=fn)
nf.write(open(fn).read())
nf.close
```

Listing 5.3: Converter

5.1.3 Other Tools

As in the previous chapter we used the bash scripts presented in Section for running a given test 100 times. We also used the java classes `Javerage` and `ClauseCounter` for the same purposes.

5.2 Tests and Results: CSA round 1 to 20

In this section, we present the results of the tests that were conducted on the CSA block cipher. In these test we vary the number of rounds R from 1 to 20.

Table 5.1 shows the results of this test, which contains 20 cases. Column 1 shows the number of rounds R. Columns 2 to 9 show the time in seconds that is taken by the respective sat solver for each test case. Column 10 states if the equation system was satisfiable or not. Column 11 shows the number of variables used to solve the equation system. Column 12 shows the total number of clauses. Columns 13 to 21 show respectively the number of clauses with the length 2,3,4,5,6,7,8 and 9.

In this table the best values are shown in green whereas the worst values are shown in red. The orange color is used to highlights the values of the second slowest sat solvers as blocks.

Table 5.1.: CSA Test: Round 1 to 20

R	Min	CR3	CR2	Lin	Zenn	GLu	Riss	Doug	S?	#V	#Cl	l1	l2	l3	l4	l5	l6	l7	l8	l9
1	0.004	0.009	0.001	0.001	0.005	0.007	0.003	0.003	Y	112	1141	120	80	32	0	0	90	578	234	7
2	0.009	0.009	0.004	0.001	0.001	0.013	0.004	0.003	Y	128	2170	96	96	96	64	0	180	1156	468	14
3	0.013	0.012	0.004	0.001	0.014	0.02	0.004	0.003	Y	144	3271	96	32	160	256	0	270	1734	702	21
4	0.017	0.013	0.006	0.0015	0.018	0.026	0.004	0.003	Y	184	4548	96	48	192	192	384	360	2312	936	28
5	0.025	0.023	0.017	0.004	0.026	0.036	0.006	0.003	Y	216	5744	96	16	192	384	512	450	2890	1170	35
6	0.032	0.025	0.024	0.007	0.033	0.048	0.006	0.003	Y	264	7070	96	48	128	448	896	540	3468	1404	42
7	0.04	0.029	0.0374	0.011	0.04	0.06	0.009	0.006	Y	320	8587	96	80	128	256	1664	630	4046	1638	49
8	0.05	0.034	0.047	0.014	0.051	0.07	0.012	0.006	Y	376	10183	96	96	160	256	1792	1232	4624	1872	56
9	0.064	0.046	0.045	0.016	0.066	0.095	0.026	0.014	Y	416	11909	96	48	256	256	2048	1834	5202	2106	63
10	0.087	0.063	0.068	0.015	0.086	0.128	0.047	0.03	Y	456	13650	96	16	288	320	2304	2436	5780	2340	70
11	0.159	0.153	0.162	0.014	0.149	0.222	0.144	0.09	Y	512	15759	96	32	256	384	2432	3550	6358	2574	77
12	0.33	0.728	0.692	0.1	0.301	0.430	0.364	0.224	Y	568	17900	96	48	192	512	2560	4664	6936	2808	84
13	0.526	1.437	1.002	0.101	0.479	0.693	0.620	0.378	Y	616	19913	96	64	192	320	3072	5522	7514	3042	91
14	1.089	2.77	1.719	1.388	1.165	1.243	1.571	1.29	Y	680	22070	96	96	256	192	3072	6892	8092	3276	98
15	9.345	16.59	8.16	13.88	3.13	4.38	6.15	7.87	Y	736	24179	96	80	256	384	3072	8006	8670	3510	105
16	120.38	107.83	115.72	416.9	1.03h	2479.9	0.9h	29.4	Y	792	26368	96	80	256	384	3328	9120	9248	3744	112
17	54.9	585.42	0.36h	336.43	297.7	349.5	1.03h	0.4h	Y	856	28717	96	80	288	384	3456	10490	9826	3978	119
18	0.125h	1.11h	0.97h	0.59h	1.9h	0.95h	5.05h	0.9h	Y	920	31162	96	80	288	384	3712	11860	10404	4212	126
19	0.33h	-	15.5h	6.63h	102.8h	18.64h	251.67h	-	Y	992	33670	96	112	256	448	3712	13486	10982	4446	133
20	-	-	12.14h	-	136.5h	-	-	22.6h	Y	1064	36276	96	112	288	448	3840	15112	11560	4680	140

In the first 10 test cases, the eight sat solvers require less than 1 second to solve the equation system. Lingeling (4 first cases) and Minisat static (cases 5 to 9) were the fastest sat solver and it immediately tells that the system is satisfiable (in some milliseconds). On the other hand, Glucose and Zenn were the slowest sat solver in these cases (results are presented with color read and orange) . In these test cases we notice that the number of variables and the number of clauses did not increase a lot compared to the work of [23], in which the number of clauses exceed 140000 in round 10. In our work the number of variables in round 1 is 112 and the total number of clauses is 1141 whereas in round 10 the number of variables is 456 and the number of clauses is 13650.

In the test cases 11 to 15, we notice that Lingeling and Minisat static are the fastest sat solvers. Based on that we recommend using these sat solvers in such situations. In contrast, the performance of Cryptominisat2 and Cryptominisat3 was the worst. Therefore, both versions of Cryptominisat should be avoided when the number of rounds is between 11 and 15. Furthermore, we noticed that the eight used sat solvers require less than 17 seconds in round 15 to solve the CNF file, which is an encouraging sign to try breaking bigger rounds.

In the test cases 16 to 19, Minisat was the fastest sat solver and Riss3g was the slowest one. For instance in test case 18, Minisat took just 400 seconds for providing a solution whereas Riss3g took 5 hours for solving the same system. In test case 19, Cryptominisat3 and Minisat static were not able to deliver a result within 1 week while Zenn required 4 days. In Opposite, Minisat needed just 0.33 hour to solve the equation system.

In test case20, we notice that Minisat was not able to solve the problem even within 2 weeks. This was also the case for Cryptominisat3, Lingeling, Glucose and Riss3g. On the other hand Cryptominisat2 was the fastest sat solver and it needs only 12 hours.

We also run test cases with R=21. However, even after 14 days, all sat solvers were unable to deliver a result.

Based on the results presented above we derive the following consequences:

- It is recommended to use Lingeling or Minisat Static when number of rounds less than 14 . In these cases Zenn, and Glucose should be avoided.
- It is recommended to use Minisat static, Minisat, or Cryptominisat2 when number of rounds bigger than 14. Riss3g and cryptominisat3 should be avoided in such cases.

Like for the previous block cipher (AES), we derived some mathematical relations between the number of variables and the length of clauses and the number of rounds as explained in the following:

- number of clauses length 1 = 92, if P > 1
- number of clauses length 7 = 578 * number of rounds.
- number of clauses length 8 = 234 * number of rounds
- number of clauses length 9 = 7 * number of rounds

Based on these results we can state that CSA (R=55) is unbreakable with the state of art computing resources. In fact, we were able in our tests to solve only 20 rounds out of 55 and all 20 cases were satisfiable.

5.3 Stability Cases 12 and 15

Figure 8 shows the stability curve of the used eight sat solvers when R=12. This figure shows that Cryptominisat3 is very unstable. Furthermore, the curve shows that Lingeling is the most stable sat solver.

Figure 9 shows the stability curve of the used eight sat solvers when R=15. The curve shows that Riss3g and Zenn are the most stable sat solvers. Lingeling is unstable. Cryptominisat3 is very unstable as when R=12.

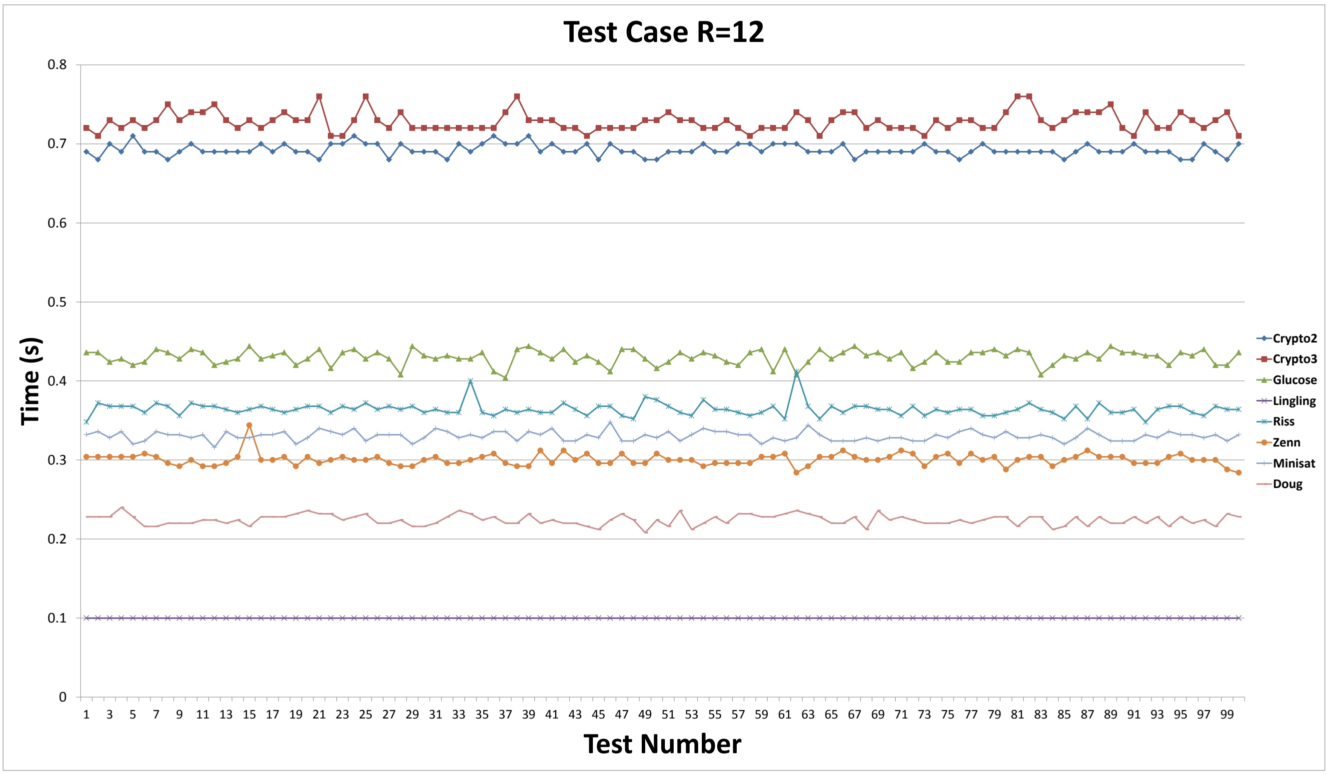

Figure 8.: CSA Stability Case 12

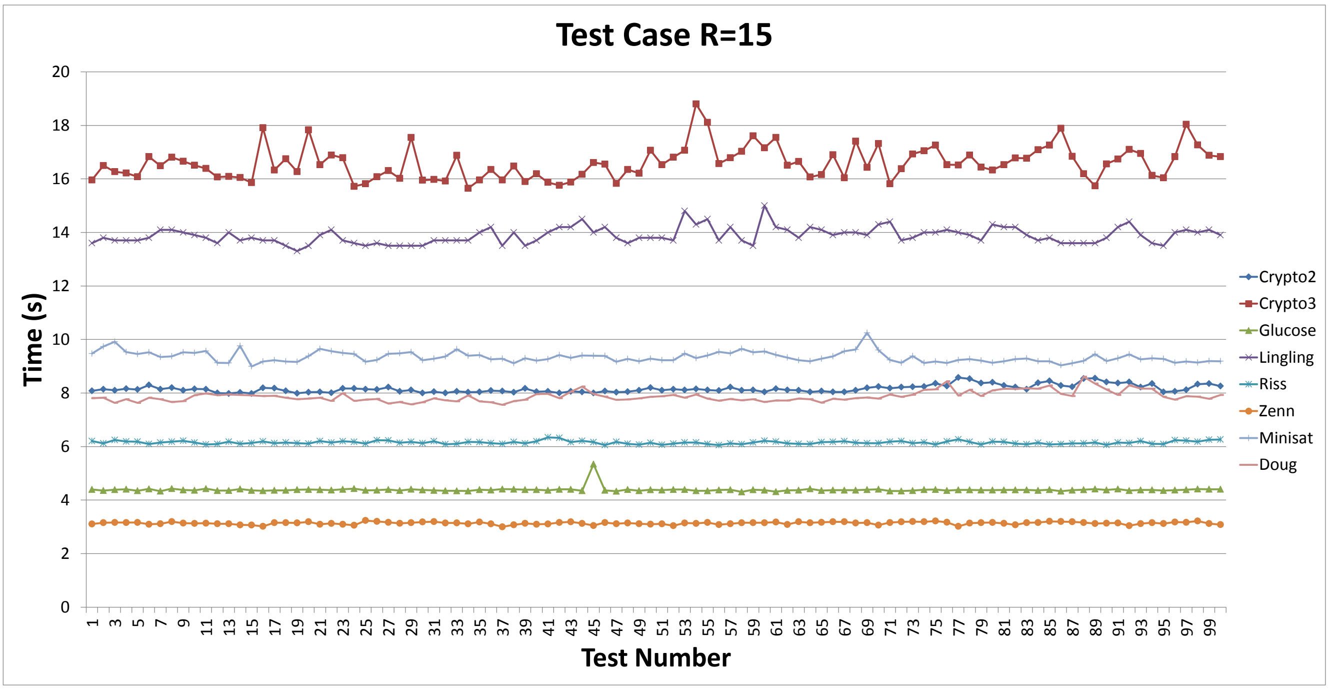

Figure 9.: CSA Stability Case 15

6 Conclusion and General Discussion

In this chapter first a summary of this work is given and then a general discussion is provided.

6.1 Summary

Block ciphers play an important role in the design of protocols for shared-key cryptography. Appropriate tools are needed to test and assess the security level and the power of encryption mechanisms in general such as block ciphers.

In this work we analyzed the security of three block ciphers using algebraic cryptanalysis. More specifically we used eight of the best available sat solvers to study the security behavior of AES, LED, and CSA. The choice of these sat solvers is motivated by their excellent performance at the international sat competition[1]. Several tests have been performed for each block cipher. The results of these tests were presented in the previous three chapters. Based on these results we give in the following recommendations on the best sat solver for a given situation.

6.2 General Discussion and Recommendations

In the following, we first mention some general observations on the sat solvers. Then, we give some recommendations based on this work.

It is noteworthy that Minisat and Minisat Static do not provide a concrete solution when the system is satisfiable. They just tell that it is satisfiable. The six other sat solvers display the solution at the end of the generated file. Furthermore, we notice one limitation of Glucose regarding the size of the generated output file, which exceeded 20 GB in some of our tests. Glucose does not provide a configuration option to hinder printing out all possibilities that this sat solver tries until a result is found.

Based on the test results we suggest avoiding Minisat. In fact, Minisat is the parent of all other seven sat solvers that we used. The rationale behind this is the unstability of this sat solver. As we saw in the previous chapters Minisat can solve a given test case in 10 minutes. Later, it can require one hour for the same test case. In contrast, the seven other sat solvers were always stable during the different experiments.

In general, we recommend using either Zenn or Minisat Static when the polynomial systems are small and very sparse. The good performance of Zenn in such a case can be explained by the fact that Zenn employs a technique called phase shift, which integrates different search methods. As the system is small, the result can be found quickly. On the other hand the good performance of Minisat static can be explained by the used reduction technique which simplifies the initial formula by fixing truth assignments. In contrast, we suggest avoiding Cryptominisat3 when the polynomial systems are small and very sparse.

[1] http://www.satcompetition.org/

Furthermore, we recommend using Cryptominisat2 when the systems are big and dense. The good performance of Cryptominisat2 in such cases can be explained by the usage of clause cleaning and the DPLL algorithm. In contrast we suggest in such cases avoiding Cryptominisat3 - which should be actually an improvement of Cryptominisat2 - but our experiments show that the performance of Cryptominisat3 is worse than that of Cryptominisat2.

In cases where the number of variables in the polynomial system is bigger than 43000, we recommend not using Cryptominisat3 and Lingeling. In such cases, it is better to use Riss3g or Minisat Static. In cases where number of variable is less than 1000, we recommend avoiding Cryptominisat3. Instead, we suggest using Minisat Static.

When the maximum length of clauses is less than 5, we recommend avoiding Cryptominisat3. Instead, Riss3g should be used. When the maximum length of clauses is between 5 and 7, Cryptominisat3 should be avoided as before. Instead, Minisat Static and Riss3g should be chosen. When the maximum length of clauses is bigger than 7, both Glucose and Cryptominisat3 have to be avoided. In such a case, Riss3g and Minisat Static are the best alternatives.

To sum up, Minisat Static can be considered as the most suitable sat solver in general. Despite that it does not provide the concrete solution, it is very efficient and fast. In the second place, we recommend Riss3g, which enumerates all solutions of the input formula in addition to being fast and efficient. On the other hand Cryptominisat3 has the worst performance and should therefore be avoided.

The most important results of this work can be summarized as follows: With respect to AES, we were able to break a small scale variant of the AES polynomial system namely (1, 4, 4, 4). With respect to LED, we were able to break the first round fully and some rounds by guessing a specific number of bits of the Key. With regard to CSA, we were able to break 20 rounds of totally 55 rounds. However, it is probably possible to reach round 22 by guessing some bits of the key. This is definitely an interesting direction for future research.

A Appendix

```java
import java.io.*;
import java.util.*;
public class LingelingOutput
{       public static void main (String args[])
        {
        String thisLine;
        ArrayList<String> lines = new ArrayList();
        int c =0;
        int temp =0;
        for (int i=0; i < args.length; i++)
        {
                int k =0;
                try
                {
                BufferedReader br =
                new BufferedReader(new FileReader(args[i]));
                while((thisLine = br.readLine()) != null)
                {
                        lines.add(thisLine);
                        if(thisLine.startsWith("s"))
                                temp=c;
                        c++;
                }
            String resultline1 = lines.get(temp);
                String resultline2 = lines.get(lines.size()-1);
                String result = resultline1.split(" ")[1];
                String time   = resultline2.split(" ")[1];
                System.out.println(result + " " + time);
                }
                catch (IOException e)
                {
                   System.err.println("Error: " + e);
                }
        }
 }
}
```

Listing A.1: Java Class LingelingOutput

```
import java.io.*;
import java.util.*;
public class Javerage
        {
        public static void main(String [] args) throws FileNotFoundException{
                if (args.length == 0)
                {
                        System.out.println("Usage java javerage filename");
                        return;
                }
                String n = args[0];
            File f = new File(n);
            while(!f.exists()){
                System.out.print("Doesn't exist. Enter a valid filename: ");
                return;
            }
            Scanner input = new Scanner(f);
            double countDouble = 0;
            double averageDouble = 0;
            double sum = 0;
            input.useLocale(Locale.US);
            while(input.hasNext()){
                if(input.hasNextFloat()){
                    double next2 = input.nextDouble();
                    sum = sum + next2;
                    countDouble++;
                }
                else
                        input.next();
            }
            averageDouble = sum/countDouble;
            System.out.println("The results for the integers in the file:");
            System.out.printf(" Count = %f\n", countDouble);
            System.out.printf(" average = %f\n", averageDouble);
              }
        }
```

Listing A.2: Java Class Javerage

```
import java.io.*;
import java.util.*;
public class ClauseCounter {
        public static void main(String args[]) {
        String thisLine;
        ArrayList<String> lines = new ArrayList();
        int c,temp,k = 0;
        int[] counters = new int[20];
        for (int i = 0; i < args.length; i++) {
        try {
        BufferedReader br = new BufferedReader(new FileReader(args[i]));
        while ((thisLine = br.readLine()) != null) {
        if (!thisLine.startsWith("c") && !thisLine.startsWith("p"))
                {
                lines.add(thisLine);
                for (int l = 0; l < thisLine.length(); l++)
                {
                        if (Character.toString(thisLine.charAt(l))
                        .equals(" "))
                                        c++;
                }
                                counters[c]++;
                                System.err.println(c);
                                c = 0;
                        }
                }
                for (int x = 0; x < counters.length; x++) {
                        System.out.println("there are " + counters[x]
                                + "Clauses with " + x + "Parameters");
                }
        } catch (IOException e) {
                System.err.println("Error: " + e);
        }
        }
        }
}
```

Listing A.3: Java Class ClauseCounter

[1] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence (IJCAI)*, pages 399–404. Morgan Kaufmann Publishers Inc., 2009.

[2] Gregory V. Bard. A challenging but feasible blockwise-adaptive chosen-plaintext attack on ssl. Cryptology ePrint Archive, Report 2006/136, 2006. `http://eprint.iacr.org/`.

[3] Gregory V. Bard. *Algebraic Cryptanalysis*. Springer, 1st edition, 2009.

[4] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362 – 399, 2000.

[5] Armin Biere. Lingeling and friends entering the SAT challenge 2012. In *SAT Competition 2012*, pages 33–34, 2012.

[6] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[7] Michael Brickenstein and Alexander Dreyer. Polybori: A framework for gröbner-basis computations with boolean polynomials. *J. Symb. Comput.*, 44(9):1326–1345, 2009.

[8] SAT 2013 Organizing Committee. Sat competition 2013, July 2013.

[9] The SAGE developement team. Sage mathematics software. `http://www.sagemath.org/`, May 2014.

[10] Adele Howe Doug Hains, Darell Whitely. Hyperplane guided minisat. In *SAT Competition 2013*, 2013.

[11] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *SAT Competition 2013*, LNCS, pages 502–518, 2013.

[12] Tobias Eibach, Enrico Pilz, and Gunnar Völkel. Attacking bivium using sat solvers. In *SAT Competition 2008*, pages 63–76, May 2008.

[13] Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. In *Handbook of Knowledge Representation*, pages 89–134. Elsevier, 2008.

[14] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The LED block cipher. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2011)*, volume 6917 of *LNCS*, pages 326–341, September 2011.

[15] Philipp Jovanovic and Martin Kreuzer. Algebraic attacks using sat-solvers. *Groups Complexity Cryptology*, 2(2):247–259, 2010.

[16] Wu Kehui, Wang Tao, Zhao Xinjie, and Liu Huiying. Cryptominisat solver based algebraic side-channel attack on present. In *Proceedings of the 2011 First International Conference on Instrumentation, Measurement, Computer, Communication and Control*, IMCCC '11, pages 561–565, Washington, DC, USA, 2011. IEEE Computer Society.

[17] Grzegorz Kondrak and Peter van Beek. A theoretical evaluation of selected backtracking algorithm. *Artificial Intelligence*, 89(1–2):365 – 387, 1997.

[18] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In *22nd Annual International Cryptology Conference (CRYPTO 2012)*, volume 2442 of *LNCS*, pages 31–46, August 2002.

[19] Nobert Manthey. The sat solver RISS3G at sc 2013. In *SAT Competition 2013*, July 2013.

[20] David A. McGrew. Impossible plaintext cryptanalysis and probable-plaintext collision attacks of 64-bit block cipher modes. *IACR Cryptology ePrint Archive*, 2012:623, 2012.

[21] Mate Soos. Limits of sat solvers in cryptography, guest lecture at cased, July 2011.

[22] Takumi Okugawa Takeru Yasumoto. Zenn. In *SAT Competition 2013*, July 2013.

[23] Julian Wälde. Algebraic cryptanalysis of round reduced versions of CSA and the LED family of block ciphers. Bachelor thesis, TU Darmstadt, January 2013.

[24] Ralf-Philipp Weinmann and Kai Wirt. Analysis of the DVB common scrambling algorithm. In *Proc. of the 8th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security*, volume 175, New York, NY, September 2005. IFIP.