Software and Algebraic Cryptanalysis Lab,
Optional Lab for Cryptanalysis COMPGA18

Draft, v0.9.1.
17 March 2016

In theory, there is no difference between theory and practice.
But, in practice, there is…
[de Snepscheut, computer scientist and educator].

Algebraic Cryptanalysis Philosophy:

1. Write Equations
2. Modify/expand/convert them [in this lab we will AVOID costly expansion!!!]
3. Solve them

The main goal of this lab is to show that the apparent complexity of systems of non-linear equations can be tackled by software and will totally disappear at times.

Some algebraic solver software like SAT solvers and converters will either:

- solve equations by logical methods [SAT solvers]
- or like XL or F4 methods expand (2.) to higher degree to solve at great cost and lots of memory.

Miracle: In contrast, yes,

- the ElimLin method can sometimes make disappear TOTALLY what is the main and only problem in breaking most ciphers: non-linear monomials.
- We will see how this is achieved by a form of “combinatorial explosion” (there is a question about this later).
- Step 2. in the attack “philosophy” 123 above can be eliminated. We call this “fast algebraic attacks”.

Part 0 - Setup.

In this lab we use ready exe files to run under command line in Windows 7/8/10 version x64, the most commonly used versions of Windows.

The main 3 files are:

- simon.exe
  http://www.nicolascourtois.com/software/simon.exe
  - Source:
    https://github.com/GSongHashrate/SimonSpeck
  - Documentation – section Simon inside:
    http://www.cryptosystem.net/aes/toyciphers.html

- ax64.exe
  http://www.nicolascourtois.com/software/ax64.exe
  - Documentation:
    http://www.cryptosystem.net/aes/tools.html

- cryptominisat-2.9.6-win64.exe
- http://www.nicolascourtois.com/software/cryptominisat-2.9.6.-win64.exe
  - Sources and documentation:
    http://www.msoos.org/cryptominisat2/

We open a command line window ("cmd.exe") and copy these 3 files inside our home directory, or any other directory like one created below:

Press "Windows Key" + F => type cmd.exe => run cmd.Exe
md "C:\Lab1\".
cd "C:\Lab1\"
Open windows explorer and copy 3 exe file to the same directory of your choice.

Exercise 2 - Inspection.

Here is a picture of a toy SPN cipher CTC2.
In this block cipher each round is composed of Nr rounds of:

- A XOR with a key [permutation of bits from the global key]
- A layer of parallel 3-bit S-boxes S
- A diffusion layer D: here linear,
  for example each bit is the XOR of 2 bits from 2 previous S-boxes.

- ….
- repeat the above three steps times Nr rounds
- ….

- At the end we have another key XORed,
  We have Nr+1 keys for Nr rounds.

Note: This is implemented inside ax64.exe 777 option:

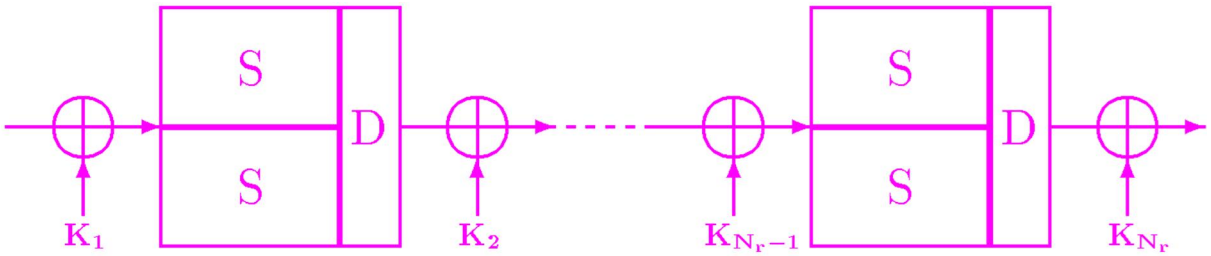


Fig. 1. A toy cipher with $B = 2$ S-boxes per round

Diffusion Layer D is as follows [please inspect the equations generated]:

$$\begin{cases} Z_{i\ (j\cdot 1987+257 \mod Bs)} = Y_{i\ j} \oplus Y_{i\ (j+137 \mod Bs)} \oplus Y_{i\ (j+274 \mod Bs)} \\ \qquad\qquad \text{if } j = 257 \mod Bs \\ Z_{i\ (j\cdot 1987+257 \mod Bs)} = Y_{i\ j} \oplus Y_{i\ (j+137 \mod Bs)} \\ \qquad\qquad \text{otherwise} \end{cases}$$

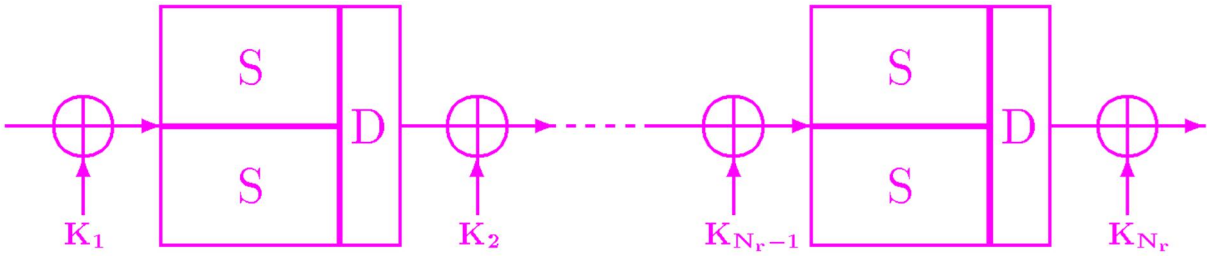


Fig. 1. A toy cipher with $B = 2$ S-boxes per round

A block cipher is encoded as concatenation of equations.

Many equations are linear: variables are added mod 2.

1. Quadratic (for each S-box)

```
X[0][1]*X[0][2]+Z[0][1]+X[0][3]+X[0][2]+X[0][1]+1
X[0][1]*X[0][3]+Z[0][2]+X[0][2]+1
X[0][1]*Z[0][1]+Z[0][2]+X[0][2]+1
X[0][1]*Z[0][2]+Z[0][2]+Z[0][1]+X[0][3]
X[0][2]*X[0][3]+Z[0][3]+Z[0][2]+Z[0][1]+X[0][2]+X[0][1]+1
X[0][2]*Z[0][1]+Z[0][3]+Z[0][2]+Z[0][1]+X[0][2]+X[0][1]+1
X[0][2]*Z[0][2]+X[0][1]*Z[0][3]+X[0][1]
X[0][2]*Z[0][3]+X[0][1]*Z[0][3]+Z[0][1]+X[0][3]+X[0][2]+1
X[0][3]*Z[0][1]+X[0][1]*Z[0][3]+Z[0][3]+Z[0][1]
X[0][3]*Z[0][2]+Z[0][3]+Z[0][1]+X[0][3]+X[0][1]
X[0][3]*Z[0][3]+X[0][1]*Z[0][3]+Z[0][2]+X[0][2]+X[0][1]+1
Z[0][1]*Z[0][2]+Z[0][3]+X[0][1]
Z[0][1]*Z[0][3]+Z[0][3]+Z[0][2]+X[0][2]+X[0][1]+1
Z[0][2]*Z[0][3]+Z[0][3]+Z[0][2]+Z[0][1]+X[0][3]+X[0][1]

X[1][1]*X[1][2]+Z[1][1]+X[1][3]+X[1][2]+X[1][1]+1
X[1][1]*X[1][3]+Z[1][2]+X[1][2]+1
X[1][1]*Z[1][1]+Z[1][2]+X[1][2]+1
X[1][1]*Z[1][2]+Z[1][2]+Z[1][1]+X[1][3]
X[1][2]*X[1][3]+Z[1][3]+Z[1][2]+Z[1][1]+X[1][2]+X[1][1]
X[1][2]*Z[1][1]+Z[1][3]+Z[1][2]+Z[1][1]+X[1][2]+X[1][1]
X[1][2]*Z[1][2]+X[1][1]*Z[1][3]+X[1][1]
X[1][2]*Z[1][3]+X[1][1]*Z[1][3]+Z[1][1]+X[1][3]+X[1][2]
X[1][3]*Z[1][1]+X[1][1]*Z[1][3]+Z[1][3]+Z[1][1]
X[1][3]*Z[1][2]+Z[1][3]+Z[1][1]+X[1][3]+X[1][1]
X[1][3]*Z[1][3]+X[1][1]*Z[1][3]+Z[1][2]+X[1][2]+X[1][1]
Z[1][1]*Z[1][2]+Z[1][3]+X[1][1]
Z[1][1]*Z[1][3]+Z[1][3]+Z[1][2]+X[1][2]+X[1][1]+1
Z[1][2]*Z[1][3]+Z[1][3]+Z[1][2]+Z[1][1]+X[1][3]+X[1][1]
:
:
```

2. Linear (connecting S-boxes via key vars)

```
1+X[0][1]=k_0
1+X[0][2]=k_1
1+X[0][3]=k_2
1+X[1][1]=k_3
1+X[1][2]=k_4
1+X[1][3]=k_5
Z[0][3]+X[2][1]=k_1
Z[1][1]+X[2][2]=k_2
Z[1][2]+X[2][3]=k_3
Z[1][3]+X[3][1]=k_4
Z[0][1]+X[3][2]=k_5
Z[0][2]+X[3][3]=k_0
Z[2][3]+1=k_2
Z[3][1]+1=k_3
Z[3][2]+1=k_4
Z[3][3]+1=k_5
Z[2][1]+0=k_0
Z[2][2]+1=k_1
```

Each S-box is characterized by a "block of equations" which are non-linear.

One paper about CTC2 (there are several):
https://eprint.iacr.org/2007/152.pdf

2.1.1.

Run command line:

ax64 777 1 1 /genonly

Inspect the equations generated.

- How many S-boxes? ……
- What is the degree of equations? …..
- How many non-linear equations? ……
- How many non-linear monomials? ……
- What is the key size? ……
- What is the block size? ….
- How many rounds? …..

P.S. Below is a more “human readable” version of our “non-linear block”.

$$
\begin{cases}
0 = x_1x_2 + y_1 + x_3 + x_2 + x_1 + 1 \\
0 = x_1x_3 + y_2 + x_2 + 1 \\
0 = x_1y_1 + y_2 + x_2 + 1 \\
0 = x_1y_2 + y_2 + y_1 + x_3 \\
0 = x_2x_3 + y_3 + y_2 + y_1 + x_2 + x_1 + 1 \\
0 = x_2y_1 + y_3 + y_2 + y_1 + x_2 + x_1 + 1 \\
0 = x_2y_2 + x_1y_3 + x_1 \\
0 = x_2y_3 + x_1y_3 + y_1 + x_3 + x_2 + 1 \\
0 = x_3y_1 + x_1y_3 + y_3 + y_1 \\
0 = x_3y_2 + y_3 + y_1 + x_3 + x_1 \\
0 = x_3y_3 + x_1y_3 + y_2 + x_2 + x_1 + 1 \\
0 = y_1y_2 + y_3 + x_1 \\
0 = y_1y_3 + y_3 + y_2 + x_2 + x_1 + 1 \\
0 = y_2y_3 + y_3 + y_2 + y_1 + x_3 + x_1
\end{cases}
$$

2.1.2.

We want to break a larger cipher

ax64 777 8 5 /ins1 /genonly

Again:

- How many S-boxes? ……
- What is the degree of equations? …..
- How many non-linear equations? ……
- How many non-linear monomials? ……
- What is the key size? ……
- What is the block size? ….
- How many rounds? …..

2.1.3. Solving attempt.

Then run again without /genonly option.
This will call ElimLin by default with “ax64 4000 filename”. Example:
ax64 777 8 5 /fix0 /ins1

We see on the screen sth like: “192+ 0”.

Explanation:

1. It is important to see that the first step of ElimLin is to remove the so called “NATURAL linear equations”. Those already present in the input file.
2. A report of type “192+ 0” means that ElimLin finds NOTHING at all.
   Zero non-trivial linear equations which have not been there already.
3. However the process is NOT NEUTRAL. First of all ElimLin is called with the file and by default it will modify this file.
4. We can run it again, ElimLin alone:
   ax64.exe 4000 CTC2_8_5_24_24.txt
   and see that linear equations have NOT been lost, they are preserved. Also the initial system of equations is preserved: usually more sparse, smaller size on hard drive.
5. The reduced and substituted system of equations with LESS variables can be found as *.modified.txt
   here * = CTC2_8_5_24_24.txt
   This file *.modified.txt is typically a lot larger than the original system of equations. This is why by default the system reverts to the original VERY SPARSE and human-readable GF(2) MQ system *.
6. If the ElimLin algorithm reports sth like: 120/ 120 => this means that 120 variables are left at the end. 192 variables were eliminated and they do not appear in *.modified.txt but they still appear and are appended in the modified main file *.
7. To monitor the progress of ElimLin: see ElimLinSteps.log and ElimLin.log files.

2.1.4.

Definition 1:
We call “DEGREE of REGULARITY” for XX rounds of cipher YY and KK P/C pair the MINIMUM degree of algebraic equations over GF(2) generated such that equations are solved.

- Not done so far with ElimLin, ElimLin stays at degree 2.
- By default we could try at degree 2, then EXPAND to degree 3 by XL etc…
    - One method to compute this is to use Gröbner bases algorithms such as XL, XL2, Magma F4 or F5.
    - Another method to estimate this is to try this:

ax64 777 3 5 /ins1 /up4

***OPTIONAL:
Observe the expanded equations [very slow].

Observe the solving process.

Try with other software [Magma].

2.1.4. Second Solving attempt – 2 KP

ax64 777 8 5 /ins2 /genonly

Again:

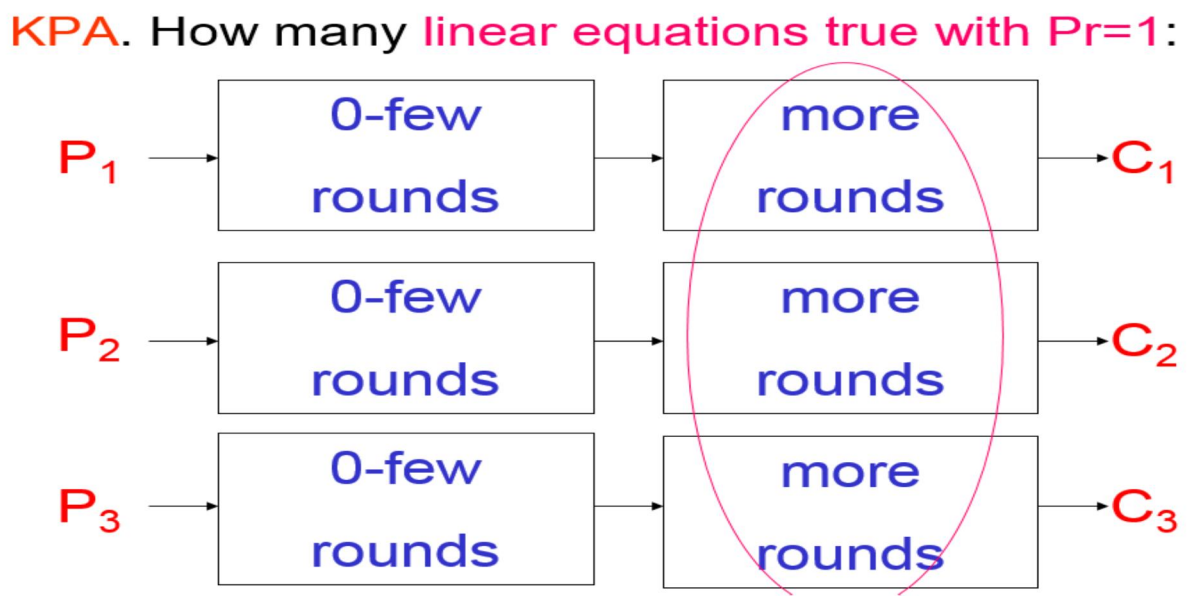
- How many S-boxes? ……
- What is the degree of equations? …..
- How many non-linear equations? ……
- How many non-linear monomials? ……
- What is the key size? ……
- What is the block size? ….
- How many rounds? …..

The difference is that we generated TWO copies of the same system of equations for TWO DIFFERENT encryptions
(/ins2 or 2KP = 2 P/C pairs = 2 Plaintext/Ciphertext pairs).

ax64 777 8 5 /ins2

Here we should see ElimLin do something NOT TRIVIAL:
Discover new equations which relate variables within several encryptions.
This could be observed by looking at log files [TBD: make this more visible].

Tricky Question:
Why Linear Cryptanalysis (LC) does NOT find these linear equations?
Why in LC we do NOT have equations true with probability 1????
……..
……..



2.1.4. Third Solving attempt – 4 KP and combinatorial explosion.

What is which happens at /ins8 which we do NOT observe at /ins4?

……

What else?
Now we should observe A COMBINATORIAL EXPLOSION in the number of NEW EQUATIONS of non-trivial sort generated at stage 1 of ElimLin.

- ⇨ THE NUMBER OF NEW LINEAR EQUATIONS should grow as a ……. of the number of instances K.
  (K=2,4,8 etc with /ins2 /ins4 etc).
- ⇨ At the same time the number of variables grows ……. with K.

Provide numerical evidence for the 1st claim above. Note: one older numerical example for a different cipher can be found at slide 153 in algatt_all_teach_2015.pdf

What is a preliminary conclusion?? ……. ……..

……

*Advanced: What is the limit for this combinatorial explosion?
Does it continue for arbitrarily large K??……

……

2.1.5. Final Solving attempt – X KP

How many KP are needed to break our cipher with 40 S-boxes by ElimLin?

……

What is the “Degree of Regularity” here?
…….

What happens to the “Degree of Regularity” when we INCREASE the data complexity?
……..

2.1.6. Alternative Solving attempt – 1 KP, SAT solver

Try this:

ax64 777 8 5 /ins1 /sat

Which method has LOWER data complexity: ElimLin or SAT Solver?

……..

Which method was faster, ElimLin or SAT Solver?

……

How many solutions for the key we expect for 1 KP?

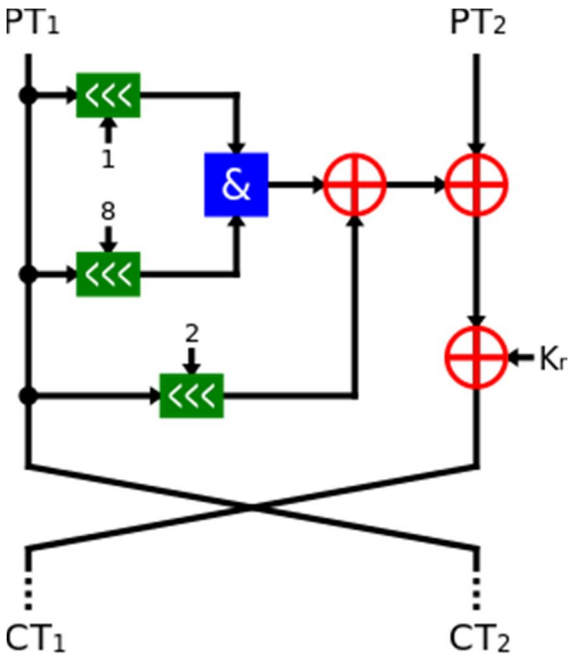
……..

Is the solution obtained with ElimLin correct?

…..

Exercise 3.

Now we study a real-life cipher. It is the very recent NSA cipher Simon.
Here each round modifies ONLY half of the state. It is a Feistel cipher.
It has 32 or more rounds (there exist many versions). Our code implements 1 version for now.



Ex 3.1.1.

In order to run the same attack on Simon we type:
Simon.exe 8 /fixk128 /ins1 /xl0
This is for now a “pathological” run in which 128/128 key variables are printed for the attacker, it is very easy to solve. The student should try to decrease this number down to 0.

Simon.exe 8 /fixk0 /ins1 /xl0

Ex 3.1.2.

What is the key size and block size in these block ciphers?

- CTC2? … block=…….key=…… .
- Simon? … block=…….key=…… .
- Russian GOST? … block=…….key=…… .
- DES? … block=…….key=……
- 2-key triple -DES? … block=…….key=…… .
- AES? … block=…….key=…… .

More questions:

Please write all answers in form X out Y or X/Y where Y is the total key size.

- How many key bits per round used CTC2 with B=8 Nr=5 parameters? …….
- How many key bits per round are used in Simon w=32 k=128? …. ……
- How many key bits per round are used GOST? …….
- How many key bits per round are used in DES? …….
- How many key bits per round are used in 2-key 3-DES? …….
- How many key bits per round are used in AES? …….

Ex 3.1.3.

More questions:

- How bijectivity is achieved in CTC2? ……
- How bijectivity is achieved in Simon? ……
- How bijectivity is achieved in DES? ……
- How bijectivity is achieved in GOST? ……
- How bijectivity is achieved in AES? ……

Ex 3.2.1.

Definition 2:
We call “SAT immunity” for X rounds of cipher YY and K P/C pairs the MINIMUM number of bits Z that need to be fixed or guessed [in a way freely chosen by the adversary] such that if we guess these variables CORRECTLY the key will be computed in less than 1000 seconds by a SAT solver.

Typically what we can compute is an upper bound, there may a better choice of variables. But maybe not a lot better.

- In CTC2 variables are spread “evenly” in each round, and we expect that there is NO advantage to choose a particular subset of key variables which are guessed by the attacker.
- Similar for DES.
- Same for AES/Simon because key bits are derived in a complex way from a small set.
- For other ciphers, e.g. GOST, it is OTHERWISE, there exist some VERY GOOD choices. Home reading:
  https://www.sav.sk/journals/uploads/0114113604CuGaSo.pdf

Ex. 3.2.2:

Consider the CTC2 with B=16 Nr=6 /ins1 case.

- What is the key size? …..
- What is the SAT immunity? …….

Command line:
ax64 777 16 6 /ins1 /fix10
ax64 4444 CTC2_16_6_48_48_f10.txt

Definition 3:

We call “UNSAT immunity” or also “Contradiction Immunity” for X rounds of cipher YY and K P/C pairs the MINIMUM number of bits Z that need to be fixed or guessed [in a way freely chosen by the adversary] such that IF THE GUESS IS INCORRECT the SAT solver will output UNSAT in less than 1 second and with probability of 50% over the keys.

Question: Why 1 second? Why not 1000 seconds????

……
……

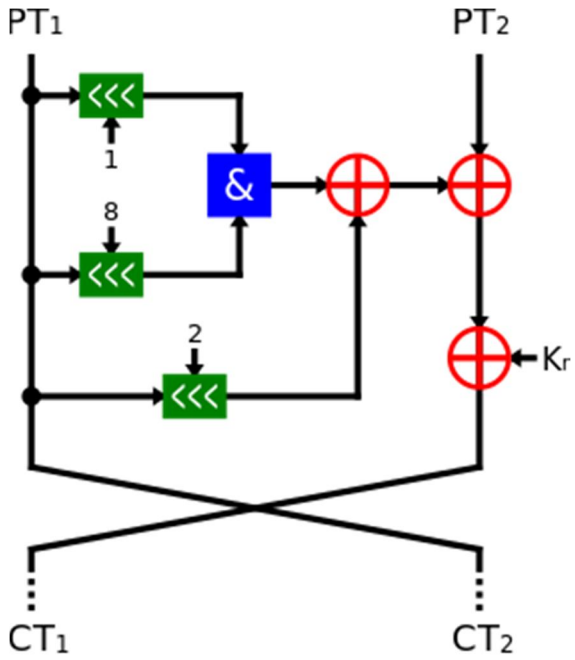
Ex. 3.2.3:

Estimate the UNSAT Immunity for the same CTC2 case.
B=16 Nr=6 /ins1.

Command line:

```
ax64 777 16 6 /ins1 /fix10 /wrong1 /sat
```

Ex. 3.3.1:



Estimate the SAT immunity of Simon w=32 k=128 Nr=8.

Simon.exe 8 /fixk40 /ins1 /sat