



Tools for Algebraic Cryptanalysis

Martin R. Albrecht

Tools for Cryptanalysis 2010, Egham, UK

Outline



1 Introduction

2 Ciphers

3 Converters

Outline



1 Introduction

2 Ciphers

3 Converters

Introduction



- In this talk we are going to present a collection of scripts for algebraic cryptanalysis.
- All these scripts are available online and require the Sage mathematics software [S⁺08].

```
hg clone http://bitbucket.org/malb/algebraic_attacks
```

Sage I



*Sage is a free open-source mathematics software system licensed under the GPL. It combines the power of many existing open-source packages into a common Python-based interface.
Mission: Creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab.*

– <http://www.sagemath.org>

Sage II



Sage can be used via the command line:

```
malb@road:~$ sage
| Sage Version 4.4.2, Release Date: 2010-05-19
| Type notebook() for the GUI, and license() for information.
|
sage: S = mq.SBox(12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2);
sage: S.polynomials(degree=3,groebner=True)
[x2*y2*y3 + x3*y3 + y1*y3, x0*x1 + x3*y3 + y0*y3 + y2*y3 + x0 + x2 + y2 + y3, x0*x2 + x3*y3 + y0*y3
+ y2*y3 + x1 + x2 + y0 + 1, x1*x2 + x0 + x1 + x3 + y3, x0*x3 + x1*y2 + x2*y2 + x2*y3 + x2 + x3 + y
1 + y3 - 1, x1*x3 + x2*y3 + x3*y3 + x0 + x2 + x3 + y0 + y1 + y3, x2*x3 + x2*y3 + x3*y3 + y0*y3 + y2
*y3 + x1 + x2 + y0 + 1, x0*y0 + x1*y2 + x2*y3 + x3*y3 + y0*y3 + x0 + x1 + x2 + x3 + y2, x1*y0 + x2*
y2 + x3*y3 + y0*y3 + y1*y3 + y2*y3 + x3 + y1 + y2 + y3 + 1, x2*y0 + x1*y2 + x3*y3 + y1*y3 + x1 + x2
+ y0 + y3 + 1, x3*y0 + x3*y3 + y0*y3 + y2*y3 + x0 + x2 + y0 + y1, x0*y1 + x2*y3 + y2*y3 + x0 + x1
+ x2 + y0 + y3 + 1, x1*y1 + x2*y2 + x3*y3 + y0*y3 + x1 + x2 + y1 + y3 + 1, x2*y1 + x2*y2 + y0*y3 +
y1*y3 + x1 + x3 + y0 + y1 + y2 + y3, x3*y1 + x1*y2 + x2*y2 + y0*y3 + x0 + x1 + x3 + y2, y0*y1 + x3*
y3 + y2*y3 + x0 + x2 + x3 + y3 + 1, x0*y2 + x1*y2 + x2*y3 + y0*y3 + y1*y3 + x3 + y1 + 1, x3*y2 + x3
*y3 + y0*y3 + y2*y3 + x0 + x1 + y1 + y3 + 1, y0*y2 + x3 + y1 + y3 + 1, y1*y2 + x3*y3 + y0*y3 + x0 +
x1 + x3 + y2, x0*y3 + y0*y3 + y1*y3 + y2*y3 + x0 + y0 + y1 + y2, x1*y3 + x2*y3 + y0*y3 + y2*y3 + x
0 + y0 + y1 + y2 + y3]
sage:
Exiting Sage (CPU time 0m0.20s, Wall time 0m6.66s).
Exiting spawned Singular process.
malb@road:~$
```

Sage III



Sage can be used via a webbrowser on a local computer and via the Internet.

sage The Sage Notebook admin | Toggle | Home | Published | Log | Settings | Help | Report a Problem | Sign out

Version 4.4.2

Tools

Last edited on June 02, 2010 08:11 PM by admin

File... Action... Data... sage Typeset Print Worksheet Edit Text Undo Share Publish

```
S = mq.SBox(12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2): S
(12, 5, 6, 11, 9, 0, 10, 13, 3, 14, 15, 8, 4, 7, 1, 2)

for f in S.polynomials(degree=3,groebner=True):
    show(f')
```

evaluate

$$x_2y_2y_3 + x_3y_3 + y_1y_3$$

$$x_0x_1 + x_3y_3 + y_0y_3 + y_2y_3 + x_0 + x_2 + y_2 + y_3$$

$$x_0x_2 + x_3y_3 + y_0y_3 + y_2y_3 + x_1 + x_2 + y_0 + 1$$

$$x_1x_2 + x_0 + x_1 + x_3 + y_3$$

Sage: AES & Equation Systems I



We construct SR(1,1,1,4) [CMR06] over \mathbb{F}_{2^4}

```
sage: sr = mq.SR(1,1,1,4); sr
SR(1,1,1,4)
sage: F,s = sr.polynomial_system() #zero inversions
...
<type 'exceptions.ZeroDivisionError'>: A zero inversion occurred ...

sage: F,s = sr.polynomial_system(); F # So we try again.
Polynomial System with 40 Polynomials in 20 Variables
```

Sage: AES & Equation Systems II



We can export F to MAGMA [BCP97]:

```
sage: magma(F)
Ideal of Polynomial ring of rank 20 over GF(2^4)
Graded Reverse Lexicographical Order
Variables: k100, k101, k102, k103, x100, x101, x102, x103,
w100, w101, w102, w103, s000, s001, s002, s003,
k000, k001, k002, k003
Basis:
[
w100 + k000 + $.1^4,
w101 + k001 + $.1^8,
w102 + k002 + $.1,
w103 + k003 + $.1^2,
k000^2 + k001,
....
```

Sage: AES & Equation Systems III



We can export F to SINGULAR [GPS05]:

```
sage: singular(F)
w100+k000+(a+1),
w101+k001+(a^2+1),
w102+k002+(a),
w103+k003+(a^2),
k000^2+k001,
k001^2+k002,
k002^2+k003,
...
...
```

Sage: AES & Equation Systems IV



Or we can use those systems transparently in the background:

```
sage: F.groebner_basis() # Singular in the background
[k002 + (a^3 + 1)*k003 + (a^2),
 k001 + (a^3 + a^2)*k003 + (a^3),
 k000 + (a^2)*k003 + (a^3 + a^2),
 ...]
```

```
sage: F.groebner_basis(algorithm='magma') # Magma in the background
[k002 + (a^3 + 1)*k003 + (a^2),
 k001 + (a^3 + a^2)*k003 + (a^3),
 k000 + (a^2)*k003 + (a^3 + a^2),
 ...]
```

Sage: AES & Equation Systems V



Building blocks are also available individually:

```
sage: sr.Lin
[ a^2 + 1      1      a^3 + a^2      a^2 + 1]
[           a      a      1 a^3 + a^2 + a + 1]
[ a^3 + a      a^2      a^2                  1]
[           1      a^3      a + 1                  a + 1]

sage: sr = mq.SR(1, 1, 1, 8)
sage: R = sr.ring()
sage: xi = Matrix(R, 8, 1, sr.vars('x', 1))
sage: wi = Matrix(R, 8, 1, sr.vars('w', 1))
sage: sr.inversion_polynomials(xi, wi, 8)
[x100*w100 + 1, x101*w101 + 1, x102*w102 + 1, x103*w103 + 1,
 x104*w104 + 1, x105*w105 + 1, x106*w106 + 1, x107*w107 + 1]
```

Sage: AES & Equation Systems VI



Sage also provides AES equation systems over \mathbb{F}_2 .

```
sage: sr = mq.SR(2,1,1,4,gf2=True,polybori=True)
sage: F,s = sr.polynomial_system(); F
Polynomial System with 68 Polynomials in 36 Variables

sage: F.groebner_basis() # PolyBoRi
[k200 + k001 + k003 + 1,
 k201 + k001, k202 + 1,
 k203 + k000, x200 + k003,
 x201 + k000 + k001,
 x202 + k000 + k001 + k003,
 x203 + k000 + k003,
 w200 + k000 + k003 + 1,
 w201 + k001 + k003 + 1,
 w202 + k001 + 1,
 w203,
 ...]
```

Sage: AES & Equation Systems VII



Another example:

```
sage: sr = mq.SR(2,1,1,4,gf2=True,polybori=True)
sage: P = sr.random_element()
sage: K = sr.random_element()
sage: K
[1]
[1]
[1]
[0]
sage: F,s = sr.polynomial_system(P=P,K=K)
sage: s
{k000: 1, k002: 1, k001: 1, k003: 0}
sage: F.subs(s)
Polynomial System with 68 Polynomials in 32 Variables
sage: F.subs(s).groebner_basis()
[k200 + 1, k201, k202, k203 + 1, ...]
```

Sage: AES & Equation Systems VIII



Information Security Group

Interdependencies between polynomials:

```
sage: sr = mq.SR(1,4,4,4,gf2=True,polybori=True, \
                  allow_zero_inversions=True)
sage: F,s = sr.polynomial_system()
sage: F_short = mq.MPolynomialSystem(F.ring(), F.rounds()[:-1])
sage: F_short.connected_components()
[Polynomial System with 80 Polynomials in 64 Variables,
 Polynomial System with 80 Polynomials in 64 Variables,
 Polynomial System with 80 Polynomials in 64 Variables,
 Polynomial System with 80 Polynomials in 64 Variables]

sage: S = mq.SBox(12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2)
sage: F_S = mq.MPolynomialSystem(S.polynomials(degree=3,groebner=True))
sage: F_S.connection_graph()
Graph on 8 vertices
```

S-Boxes I



Sage also supports some operations with S-boxes:

```
sage: S = mq.SBox(12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2); S
(12, 5, 6, 11, 9, 0, 10, 13, 3, 14, 15, 8, 4, 7, 1, 2)
sage: type(S)
<class 'sage.crypto.mq.sbox.SBox'>
sage: S(0)
12
sage: S([0,0,0,1])
[0, 1, 0, 1]
sage: f = S.interpolation_polynomial()
sage: f(0), S(0)
(a^3 + a^2, 12)
```

$$f = (a^3 + a^2 + 1)x^{14} + (a^3 + a^2 + 1)x^{13} + (a^3 + a^2)x^{12} + (a^3 + a^2 + a)x^{11} + (a^3 + 1)x^{10} + (a^3 + 1)x^9 + (a^2 + a + 1)x^8 + a^2x^7 + (a^3 + a^2)x^6 + (a^3 + a)x^5 + (a^3 + a^2 + a)x^4 + (a^2 + a + 1)x^3 + (a^2 + a + 1)x^2 + a^3 + a^2$$



S-Boxes II

```
sage: S = mq.SBox(12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2)
sage: S.linear_approximation_matrix()
[ 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  0  -4  0  -4  0  0  0  0  0  0  -4  0   4]
[ 0  0  2  2  -2  -2  0  0  2  -2  0  4  0  4  -2  2]
[ 0  0  2  2  -2  -4  0  -2  2  -4  0  0  0  0  -2  -2]
[ 0  0  -2  2  -2  -2  0  4  -2  -2  0  -4  0  0  -2  2]
[ 0  0  -2  2  -2  2  0  0  2  2  -4  0  4  0  2  2]
[ 0  0  0  -4  0  0  -4  0  0  -4  0  0  4  0  0  0]
[ 0  0  0  4  4  0  0  0  0  -4  0  0  0  0  4  0]
[ 0  0  2  -2  0  0  -2  2  -2  2  0  0  -2  2  4  4]
[ 0  4  -2  -2  0  0  2  -2  -2  -2  -4  0  -2  2  0  0]
[ 0  0  4  0  2  2  2  -2  0  0  0  -4  2  2  -2  2]
[ 0  -4  0  0  -2  -2  2  -2  -4  0  0  0  2  2  2  -2]
[ 0  0  0  0  -2  -2  -2  -2  4  0  0  -4  -2  2  2  -2]
[ 0  4  4  0  -2  -2  2  2  0  0  0  0  2  -2  2  -2]
[ 0  0  2  2  -4  4  -2  -2  -2  -2  0  0  -2  -2  0  0]
[ 0  4  -2  2  0  0  -2  -2  -2  -2  2  4  0  2  2  0  0]
```



S-Boxes III

```
sage: S = mq.SBox(12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2)
sage: S.difference_distribution_matrix()
[16  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
[ 0  0   0   4   0   0   0   4   0   4   0   0   0   0   4   0]
[ 0  0   0   2   0   4   2   0   0   0   2   0   2   2   2   0]
[ 0  2   0   2   2   0   4   2   0   0   2   2   2   0   0   0]
[ 0  0   0   0   0   4   2   2   0   2   2   0   2   0   2   0]
[ 0  2   0   0   2   0   0   0   0   2   2   2   4   2   0   0]
[ 0  0   2   0   0   0   2   0   2   0   0   4   2   0   0   4]
[ 0  4   2   0   0   0   2   0   2   0   0   0   2   0   0   4]
[ 0  0   0   2   0   0   0   2   0   2   0   4   0   2   0   4]
[ 0  0   2   0   4   0   2   0   2   0   0   0   2   0   4   0]
[ 0  0   2   2   0   4   0   0   2   0   2   0   0   2   2   0]
[ 0  2   0   0   2   0   0   0   4   2   2   2   2   0   2   0]
[ 0  0   2   0   0   4   0   2   2   2   2   0   0   0   2   0]
[ 0  2   4   2   2   0   0   2   0   0   2   2   0   0   0   0]
[ 0  0   2   2   0   0   2   2   2   2   0   0   0   2   2   0]
[ 0  4   0   0   0   4   0   0   0   0   0   0   0   0   0   4]
```

S-Boxes IV



```
sage: S = mq.SBox(12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2)
sage: S.polynomials() #default: degree=2
[x1*x2 + x0 + x1 + x3 + y3,
 x0*x1 + x0*x2 + x0 + x1 + y0 + y2 + y3 + 1,
 x0*x3 + x1*x3 + x1*y0 + x0*y1 + x0*y2 + x1 + x2 + y2,
 x0*x3 + x0*y0 + x1*y1 + x0 + x2 + y2,
 x0*x2 + x0*y0 + x0*y1 + x1*y2 + x1 + x2 + x3 + y2 + y3, ...]

sage: S = mq.SBox(12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2)
sage: P.<y0,y1,y2,y3,x0,x1,x2,x3> = PolynomialRing(GF(2),order='lex')
sage: X = [x0,x1,x2,x3]
sage: Y = [y0,y1,y2,y3]
sage: S.polynomials(X=X,Y=Y,degree=3,groebner=True)
[y0 + x0*x1*x3 + x0*x2*x3 + x0 + x1*x2*x3 + x1*x2 + x2 + x3 + 1,
 y1 + x0*x1*x3 + x0*x2*x3 + x0*x2 + x0*x3 + x0 + x1 + x2*x3 + 1,
 y2 + x0*x1*x3 + x0*x1 + x0*x2*x3 + x0*x2 + x0 + x1*x2*x3 + x2,
 y3 + x0 + x1*x2 + x1 + x3]
```

Outline



1 Introduction

2 Ciphers

3 Converters

DES I



An equation system generator for the full cipher is available:

```
sage: attach des.py
sage: des = DES(Nr=2,sbox_eq='cubic') #sopns, sopns_gb
sage: F,s = des.polynomial_system()
Pre-computing fully cubic S-Box equations.
sage: F
Polynomial System with 1856 Polynomials in 120 Variables
sage: F_easy = F.subs(s); F_easy
Polynomial System with 1856 Polynomials in 64 Variables
sage: %time gb = F_easy.groebner_basis()
CPU times: user 0.20 s, sys: 0.01 s, total: 0.22 s
Wall time: 0.32 s
sage: gb[:5]
[y0100, y0101, y0102, y0103 + 1, y0104]

sage: %time gb = F.groebner_basis()
CPU times: user 0.42 s, sys: 0.00 s, total: 0.42 s
Wall time: 0.47 s
sage: gb[:5]
[k00 + 1, k01, k02, k03 + 1, k04 + 1]
```



DES II

S-boxes are available independently:

```
sage: S1 = DESSBox(1)
sage: S1.polynomials(degree=3)
... # a long long list
sage: print S1.difference_distribution_matrix().str()
[64  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[ 0  0  0  6  0  2  4  4  4  0  10 12  4  10  6  2  4]
[ 0  0  0  8  0  4  4  4  4  0  6  8  6  12  6  4  2]
[14  4  2  2  10  6  4  2  6  4  4  0  2  2  2  0]
[ 0  0  0  6  0  10 10  6  0  4  6  4  2  8  6  2]
[ 4  8  6  2  2  4  4  2  0  4  4  0  12  2  4  6]
[ 0  4  2  4  8  2  6  2  8  4  4  2  4  2  0 12]
[ 2  4  10  4  0  4  8  4  2  4  8  2  2  2  2  4]
[ 0  0  0 12  0  8  8  4  0  6  2  8  8  2  2  4]
[10  2  4  0  2  4  6  0  2  2  8  0 10  0  2 12]
[ 0  8  6  2  2  8  6  0  6  4  6  0  4  0  2 10]
[ 2  4  0 10  2  2  4  0  2  6  2  6  6  4  2 12]
[ 0  0  0  8  0  6  6  0  0  6  6  4  6  6 14  2]
...
...
```

DES III



Some tools for combining algebraic and differential techniques are also available:

PRESENT I



```
sage: attach present.py
sage: p = PRESENT(Nr=1)
sage: F,s = p.polynomial_system(); F
Polynomial System with 502 Polynomials in 212 Variables
sage: R = F.ring()
sage: R.gens()[:16]
(K0000, K0001, K0002, K0003, K0004, K0005, K0006, ...)
sage: K_guess = R.gens()[:16] # 80 - 64 = 16
sage: guess = dict([(k,GF(2).random_element()) for k in K_guess])
sage: F_guessed = F.subs(guess); F_guessed
Polynomial System with 502 Polynomials in 196 Variables
sage: F_target = F_guessed.eliminate_linear_variables() # ElimLin
sage: F_target
Polynomial System with 260 Polynomials in 42 Variables
sage: F_target.groebner_basis()
[1]

sage: guess = dict([(k,s[k]) for k in K_guess])
sage: F_guessed = F.subs(guess)
sage: F_target = F_guessed.eliminate_linear_variables();
sage: gb = F_target.groebner_basis()
sage: gb[-1]
K0070
```

PRESENT II



Attack-B from [AC09]:

```
sage: for i in range(16):
    F = present_dc(PRESENT(Nr=2), r=1, \
                    characteristic=True, \
                    return_system=True)
    F_target = F.eliminate_linear_variables(maxlength=2)
    gb = F_target.groebner_basis(faugere=False, \
                                  linear_algebra_in_last_block=False)
    print gb != [1]
....:
False
False
...
```

KATAN, CTC & SEA I



We also provide equation system generators for

- all KATAN and KTANTAN variants,
- all SEA variants and
- CTC 1.

```
sage: F,s = katan_ia_factory(KTANTAN32(Nr=50))
sage: settings = {'faugere':False,'linear_algebra_in_last_block':False}
sage: %time gb = F.groebner_basis(**settings)
CPU times: user 6.00 s, sys: 0.04 s, total: 6.04 s
Wall time: 6.42 s

sage: F_target = F.eliminate_linear_variables(maxlength=2)
sage: %time gb = F_target.groebner_basis(**settings)
CPU times: user 2.34 s, sys: 0.00 s, total: 2.34 s
Wall time: 2.42 s
```

Outline



1 Introduction

2 Ciphers

3 Converters

ANF to CNF Conversion I



Consider the PRESENT S-Box:

```
sage: S = mq.SBox(12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2); S  
(12, 5, 6, 11, 9, 0, 10, 13, 3, 14, 15, 8, 4, 7, 1, 2)
```

A truth table based conversion to CNF will have $4 \cdot 2^4 = 64$ clauses and will introduce no new variables.

```
sage: S.cnf(format='dimacs').count("0\n")  
64
```

ANF to CNF Conversion II



POLYB_OR_I provides a conversion to CNF which is essentially also truth table based but performs minimisation. It also does not introduce new variables. To illustrate it, we start with a degree lexicographical Gröbner basis.

```
sage: P.<y0,y1,y2,y3,x0,x1,x2,x3> = BooleanPolynomialRing(order='deglex')
sage: X,Y = [x0,x1,x2,x3],[y0,y1,y2,y3]
sage: F = S.polynomials(X=X,Y=Y,degree=3,groebner=True)
sage: ce = CNFEncoder(P)
sage: ce.dimacs_cnf(F).count("\n")
561
```

Using the Bard-Courtois method [CB07] for conversion (our implementation) we get a much bigger CNF representation and new variables:

```
sage: a2 = ANFSatSolver(P)
sage: a2.cnf(F).count("\n")
1577
```

ANF to CNF Conversion III



We try a lexicographical Gröbner basis since it has only four polynomials:

```
sage: P.<y0,y1,y2,y3,x0,x1,x2,x3> = BooleanPolynomialRing(order='lex')
sage: X,Y = [x0,x1,x2,x3],[y0,y1,y2,y3]
sage: F = S.polynomials(X=X,Y=Y,degree=3,groebner=True)
sage: ce = CNFEncoder(P)
sage: ce.dimacs_cnf(F).count("0\n")
43
```

Bard-Courtois:

```
sage: a2 = ANFSatSolver(P)
sage: a2.cnf(F).count("0\n")
261
```

ANF to CNF Conversion IV



We try the principal ideal generator of the ideal generated by the S-Box polynomials:

```
sage: P.<y0,y1,y2,y3,x0,x1,x2,x3> = BooleanPolynomialRing(order='lex')
sage: X,Y = [x0,x1,x2,x3],[y0,y1,y2,y3]
sage: F = S.polynomials(X=X,Y=Y,degree=3,groebner=True)
sage: F = prod([f+1 for f in F]) + 1
sage: ce = CNFEncoder(P)
sage: ce.dimacs_cnf([F]).count("0\n")
41
```

For this representation Bard-Courtois is very inefficient:

```
sage: a2 = ANFSatSolver(P)
sage: a2.cnf([F]).count("0\n")
1167
```

ANF to CNF Conversion V



For polynomials with many variables and for linear polynomials
POLYBURI's approach is not very efficient:

```
sage: B = BooleanPolynomialRing(10, 'x')
sage: l = [B.random_element(terms=10, degree=1) for _ in range(10)]
sage: ce = CNFEncoder(B)
sage: ce.dimacs_cnf(l).count("0\n")
2560
```

This is the case for which Bard-Courtois' conversion was designed:

```
sage: a2 = ANFSatSolver(B)
sage: a2.cnf(l).count("0\n")
631
```

A hybrid approach remains to be implemented.

ANF to MIP I



In [BKS09] a method for solving boolean polynomial systems of equations was proposed. Two conversion routines were proposed

- Standard Conversion, similar to the Bard-Courtois conversion to SAT
- Integer Adapted Standard Conversion, using the fact that the equations only need to hold over the integers.

SC linearises and splits sums into subsums of maximum length four, thus it introduces many new variables:

```
sage: P.<y0,y1,y2,y3,x0,x1,x2,x3> = BooleanPolynomialRing(order='lex')
sage: X,Y = [x0,x1,x2,x3],[y0,y1,y2,y3]
sage: F = S.polynomials(X=X,Y=Y,degree=3,groebner=True)

sage: bc = BooleanPolynomialMIPConverter()
sage: bc.standard_conversion(F)
Mixed Integer Program ( minimization, 53 variables, 113 constraints )
```

ANF to MIP II



IASC uses less variables and constraints for many typical block cipher examples, however the conversion is exponential in the number of variables per equation:

```
sage: bc.integer_adapted_standard_conversion(F)
Mixed Integer Program ( minimization, 20 variables, 31 constraints )
```

ANF to MIP III



We also provide a probabilistic polynomial system class (cf. [AC10]). As an example, consider two parallel one round encryptions of PRESENT under some fixed input difference. We encode that a certain output difference is likely to hold.

```
sage: p = PRESENT(Nr=1,sbox_representation='lex')
sage: F = present_dc(,r=1,return_system=True,characteristic=True)
sage: H = F.gens()[:-64]
sage: S = F.gens()[-64:]
sage: S[:9]
(Y00100 + Y10100, Y00101 + Y10101, Y00102 + Y10102,
Y00103 + Y10103, Y00104 + Y10104, Y00105 + Y10105,
Y00106 + Y10106, Y00107 + Y10107 + 1, Y00108 + Y10108)
```

ANF to MIP IV



We construct a probabilistic polynomial system, where the equations describing the encryption must hold and the output difference holds with high probability.

```
sage: F_prob = ProbabilisticMPolynomialSystem(F.ring(), H, S)
sage: s,t = F_prob.solve_mip(solver='SCIP')
Writing problem data to '/home/malb/.sage//temp/road/16007//tmp_1.mps',
6605 records were written
CPU Time: 0.20  Wall time: 25.95, Obj: 3.00
```

Let's check which output differences did not hold:

```
sage: all(f.subs(s)==0 for f in H)
True
sage: [f for f in S if f.subs(s[0])]
[Y00104 + Y10104, Y00105 + Y10105, Y00107 + Y10107 + 1]
```



Thank you for your attention.

Literature I



Martin Albrecht and Carlos Cid.

Algebraic Techniques in Differential Cryptanalysis.

In *Fast Software Encryption 2009*, Lecture Notes in Computer Science, Berlin, Heidelberg, New York, 2009. Springer Verlag.



Martin Albrecht and Carlos Cid.

Cold boot key recovery using polynomial system solving with noise.

In *Proceedings of the 2nd International Conference on Symbolic Computation and Cryptography*, 2010.



Wieb Bosma, John Cannon, and Catherine Playoust.

The MAGMA Algebra System I: The User Language.

In *Journal of Symbolic Computation* 24, pages 235–265. Academic Press, 1997.

Literature II



Julia Borghoff, Lars R. Knudsen, and Mathias Stolpe.

Bivium as a Mixed-Integer Linear programming problem.

In Matthew G. Parker, editor, *Cryptography and Coding – 12th IMA International Conference*, volume 5921 of *Lecture Notes in Computer Science*, pages 133–152, Berlin, Heidelberg, New York, 2009.
Springer Verlag.



Nicolas T. Courtois and Gregory V. Bard.

Algebraic Cryptanalysis of the Data Encryption Standard.

In Steven D. Galbraith, editor, *Cryptography and Coding – 11th IMA International Conference*, volume 4887 of *Lecture Notes in Computer Science*, pages 152–169, Berlin, Heidelberg, New York, 2007.

Springer Verlag.

pre-print available at <http://eprint.iacr.org/2006/402>.

Literature III



Carlos Cid, Sean Murphy, and Matthew Robshaw.

Algebraic Aspects of the Advanced Encryption Standard.

Springer Verlag, Berlin, Heidelberg, New York, 2006.



Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann.

Singular 3.0.

A Computer Algebra System for Polynomial Computations, Centre
for Computer Algebra, University of Kaiserslautern, 2005.

Available at: <http://www.singular.uni-kl.de>.



William Stein et al.

SAGE Mathematics Software.

The Sage Development Team, 2008.

Available at <http://www.sagemath.org>.