Algebraic Techniques in Cryptanalysis
of Block Ciphers with a bias towards Gröbner bases

Martin R. Albrecht

Team SALSA, UPMC, Paris 6, ...

June 2nd, 2011 @ ECrypt 2 PhD Summer school, Albena, Bulgaria

Outline

Outline

What are Algebraic Attacks?

1. Algebraic attacks model a cryptographic primitive (such as a block cipher) as a system of equations.
2. Then, by applying (algebraic) transformations to these equations they (attempt to) recover information about the secret of the primitive (the key).

Hence, they are quite different in spirit from statistical techniques such as linear and differential cryptanalysis.

A Polemic History of Algebraic Attacks

1959 – the "prophecy"

> *"Thus, if we could show that solving a certain system requires at least as much work as solving a system of simultaneous equations in a large number of unknowns, of a complex type, then we would have a lower bound of sorts for the work characteristic."*
> – Claude Shannon

2002 – the breakthrough

> *Crucial Cipher Flawed, Cryptographers Claim – Two cryptographers say that the new Advanced Encryption Standard, [. . .] has a hole in it. Although some of their colleagues doubt the validity of their analysis, the cryptographic community is on edge, wondering whether the new cipher can withstand a future assault.*
> – Science Magazine

2011 – the disillusion

Not a single proper *block cipher* has been broken using *pure* algebraic techniqu
faster than with other techniques.

So, why bother?

Algebraic techniques

1. have been proven powerful against some stream ciphers and public key schemes,
2. provide a unified attack methodology for various areas of cryptography,
3. may be one of the few choices if very few plaintext-ciphertext pairs are available,
4. may prove useful under more relaxed attack settings (many plaintexts ...),
5. become more relevant as focus shifts toward (very) lightweight constructions,
6. can be combined with other techniques (differential, side-channels, ...),
7. are fun ... well, to some anyway!

Outline

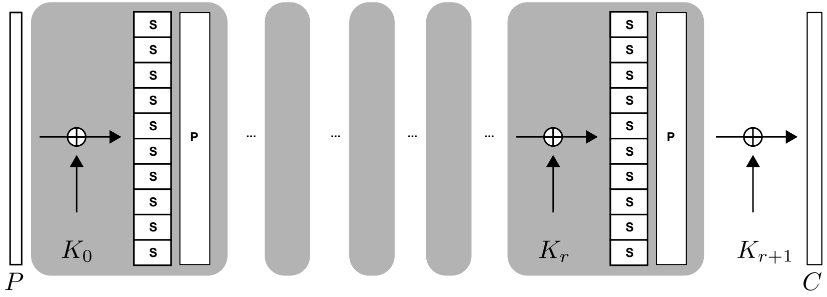
SP-Networks I

We construct an equation system for the block cipher PRESENT, which

- is a substituion-permutation network,
- has a block size of 64 bits,
- either takes 80-bit or 128-bit keys (PRESENT-80 and PRESENT-128 resp.)
- has 31 rounds (shorter variants are denoted by PRESENT-$\{80,128\}$-Nr),
- is conceptually simple, and
- has been extensively studied (differential, linear, side-channels, higher-order differential, algebraic).

Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe.
PRESENT: An ultra-lightweight block cipher.
In *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 7427 of *Lecture Notes in Computer Science*, pages 450–466, Berlin, Heidelberg, New York, 2007. Springer Verlag.

SP-Networks II



Key Addition and the Permutation Layer

- *Key addition* is easy, if X_i is a bit before key addition and Y_i is a bit after key addition, we write:

$$Y_i + X_i + K_i (= 0).$$

- the *Permutation layer* is just a permutation of wires given by the rule

$$s \cdot j + i \Rightarrow B \cdot i + j \text{ for } 0 \leq j < 16 \text{ and } 0 \leq i < 4,$$

hence we simply rename variables.

In general the permuation layer gives rise to linear equations.

S-Box I

The S-box is a non-linear operation. However, finding equations is still easy.

As an example consider the 3-bit (since it fits on the slides) S-box

$$[7, 6, 0, 4, 2, 5, 1, 3].$$

Construct the matrix on the right and perform fraction-free Gaussian elimination on it (fitting a linear model).

$$\begin{array}{ccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \\
\end{array}$$
$$\left(\begin{array}{ccccccccr}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & x_0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & x_1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & x_2 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & y_0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & y_1 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & y_2 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & x_0x_1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & x_0x_2 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & x_0y_0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & x_0y_1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & x_0y_2 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & x_1x_2 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & x_1y_0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x_1y_1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & x_1y_2 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & x_2y_0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & x_2y_1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & x_2y_2 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & y_0y_1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & y_0y_2 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & y_1y_2
\end{array}\right)$$

S-Box II

$$\left(\begin{array}{cccccccc r}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_0y_0 + x_1 + x_2 + y_0 + y_1 + 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & x_0y_0 + x_0 + x_1 + y_2 + 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & x_0y_0 + x_0 + y_0 + 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & x_0y_0 + x_0 + x_2 + y_1 + y_2 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & x_0y_0 + x_0 + x_1 + x_2 + y_0 + y_1 + y_2 + 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & x_0y_0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & x_0y_0 + x_2 + y_0 + y_2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x_0y_0 + x_1 + y_1 + 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{x_0x_2 + x_1 + y_1 + 1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{x_0x_1 + x_1 + x_2 + y_0 + y_1 + y_2 + 1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{x_0y_1 + x_0 + x_2 + y_0 + y_2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{x_0y_0 + x_0y_2 + x_1 + x_2 + y_0 + y_1 + y_2 + 1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{x_1x_2 + x_0 + x_1 + x_2 + y_2 + 1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{x_0y_0 + x_1y_0 + x_0 + x_2 + y_1 + y_2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{x_0y_0 + x_1y_1 + x_1 + y_1 + 1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{x_1y_2 + x_1 + x_2 + y_0 + y_1 + y_2 + 1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{x_0y_0 + x_2y_0 + x_1 + x_2 + y_1 + 1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{x_2y_1 + x_0 + y_1 + y_2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{x_2y_2 + x_1 + y_1 + 1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{y_0y_1 + x_0 + x_2 + y_0 + y_1 + y_2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{y_0y_2 + x_1 + x_2 + y_0 + y_1 + 1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{y_1y_2 + x_2 + y_0}
\end{array}\right)$$

S-Box III

If you cannot be bothered to do that yourself, use Sage (http://www.sagemath.org):

```
sage: S = mq.SBox(7,6,0,4,2,5,1,3)
sage: S.polynomials()
[x0*x2 + x1 + y1 + 1,
 x0*x1 + x1 + x2 + y0 + y1 + y2 + 1,
 x0*y1 + x0 + x2 + y0 + y2,
 x0*y0 + x0*y2 + x1 + x2 + y0 + y1 + y2 + 1,
 x1*x2 + x0 + x1 + x2 + y2 + 1,
 x0*y0 + x1*y0 + x0 + x2 + y1 + y2,
 x0*y0 + x1*y1 + x1 + y1 + 1,
 x1*y2 + x1 + x2 + y0 + y1 + y2 + 1,
 x0*y0 + x2*y0 + x1 + x2 + y1 + 1,
 x2*y1 + x0 + y1 + y2,
 x2*y2 + x1 + y1 + 1,
 y0*y1 + x0 + x2 + y0 + y1 + y2,
 y0*y2 + x1 + x2 + y0 + y1 + 1,
 y1*y2 + x2 + y0]
```

If we post-process these polynomials (`groebner=True`), we get 21 quadratic equations and one cubic equation for the S-Box which have a nice algebraic structure.

Putting it all together

- We have equations for the S layer, P layer and the key addtion.
- The key schedule is similar and has one/two S-boxes.
- For each round we introduce $2 \cdot 64$ new state variables for the S layer.
- Adding key schedule and key variables we get $132 \cdot Nr + 80$ variables
- On ther other hand, we get $(22 \cdot 16 + 22 + 64)Nr + 64$ equations

```
# from http://bitbucket.org/malb/algebraic_attacks/present.py
sage: attach present.py
sage: p = PRESENT(Nr=31)
sage: F,s = p.polynomial_system(); F
Polynomial System with 13642 Polynomials in 4172 Variables
```

Solving this system means recovering the key ...

Outline

Solver Families

In cryptography there are four families of algorithms which are usually used for solving systems of equations.

In order of popularity for block ciphers:

1. SAT solvers: MiniSat2, CryptoMiniSat, (Raddum-Semaev, MRHS), …
2. Gröbner basis methods: Buchberger's algorithm, F_4, F_5, …
3. Mixed Integer (Linear) Solvers: SCIP, CPLEX, Gurobi, …
4. Algebraic higher-order differential: AIDA, Cube attack, Cube Tester, …

It is very useful to understand a bit how these solvers work.

"We put our equations into MAGMA and it ran out of memory" is not a valid analysis.

Gröbner Bases I
for Cryptographers

- $P = \mathbb{F}_q[x_0, \dots, x_{n-1}]$.
- $\mathbb{F}_q$ is a finite field of order q.
- $\mathcal{I}$ is an ideal $\subset P$. That is,
  - $f, g \in \mathcal{I} \longrightarrow f + g \in \mathcal{I}$ and
  - $f \in P, g \in \mathcal{I} \longrightarrow f \cdot g \in \mathcal{I}$.
- $\langle f_0, \dots, f_{m-1} \rangle$ is the ideal spanned by $f_0, \dots, f_{m-1}$.

```
sage: P.<x,y,z> = PolynomialRing(GF(127),order='deglex')
sage: I = ideal(x*y + z, y^3 + 1, z^2 - x*5 - 1)
sage: (x*y + z) + (y^3 + 1) in I
True
sage: x*z*(z^2 - x*5 - 1) in I
True
```

Gröbner Bases II
for Cryptographers

- A term order decides how we compare monomials, e.g., is xy or y^3 bigger (degree or variable first)?

```
sage: P.<x,y,z> = PolynomialRing(GF(127),order='lex')
sage: x*y > y^3
True
sage: P.<x,y,z> = PolynomialRing(GF(127),order='deglex')
sage: x*y > y^3
False
```

- $\mathrm{M}(f)$ is the set of all monomials in f.
- $\mathrm{LM}(f)$ is the leading or largest monomial in f.

```
sage: P.<x,y,z> = PolynomialRing(GF(127),order='deglex')
sage: f = x*y + x + 3
sage: f.lm()
x*y
sage: f.monomials()
[x*y, x, 1]
```

Gröbner Bases III

for Cryptographers

Definition (Gröbner Basis)

Let $\mathcal{I}$ be an ideal of $\mathbb{F}[x_0, \dots, x_{n-1}]$ and fix a monomial ordering. A finite subset

$$G = \{g_0, \dots, g_{m-1}\} \subset \mathcal{I}$$

is said to be a *Gröbner basis* of $\mathcal{I}$ if for any $f \in \mathcal{I}$ there exists $g_i \in G$ such that

$$\mathrm{LM}(g_i) \mid \mathrm{LM}(f).$$

Among other things Gröbner bases allow to solve (non-linear) systems of equations. However, they are much more powerful objects than just that, as we will discuss at the end of this talk.

Note

Gröbner bases generalise greatest common divisors over $\mathbb{F}[x]$ and row echelon forms over $\mathbb{F}^n$.

Gröbner Bases IV
for Cryptographers

As a warm-up, consider a linear system of equations over $\mathbb{F}_{127}[x,y,z]$.

$$\begin{aligned} f &= 26y + 52z + 62 = 0 \\ g &= 54y + 119z + 55 = 0 \\ h &= 41x + 91z + 13 = 0 \end{aligned} \qquad \begin{pmatrix} 0 & 26 & 52 & 62 \\ 0 & 54 & 119 & 55 \\ 41 & 0 & 91 & 13 \end{pmatrix}$$

$$\begin{aligned} f' &= x + 29 = 0 \\ g' &= y + 38 = 0 \\ h' &= z + 75 = 0 \end{aligned} \qquad \begin{pmatrix} 1 & 0 & 0 & 29 \\ 0 & 1 & 0 & 38 \\ 0 & 0 & 1 & 75 \end{pmatrix}$$

Thus, $x = -29$, $y = -38$ and $z = -75$ is a solution.

Note

Gaussian elimination iteratively reduces the leading terms: $\mathrm{LM}(h) = x \Rightarrow \mathrm{LM}(h') = z$.

Gröbner Bases V
for Cryptographers

Now consider two polynomials in $\mathbb{F}_{127}[x, y, z]$ with term ordering **deglex**.

$$f = x^2 + 2xy - 2y^2 + 14z^2 + 22z$$
$$g = x^2 + xy + y^2 + z^2 + x + 2z$$

$$\begin{pmatrix} 1 & 2 & -2 & 14 & 0 & 22 \\ 1 & 1 & 1 & 1 & 1 & 2 \end{pmatrix}$$

$$f = x^2 + 4y^2 - 12z^2 + 2x - 18z$$
$$g' = xy + -3y^2 + 13z^2 - x + 20z$$

$$\begin{pmatrix} 1 & 0 & 4 & -12 & 2 & -18 \\ 0 & 1 & -3 & 13 & -1 & 20 \end{pmatrix}$$

Gaussian elimination still “reduces” the system.

Gröbner Bases VI
for Cryptographers

This approach fails for

$$\begin{aligned} f &= \mathbf{x^2} - 2\mathbf{xy} - 2y^2 + 14z^2, \\ g &= \mathbf{x} + y + 2z. \end{aligned}$$

since x is not a monomial of f.

However, x divides two monomials of f: x^2 and xy.

To account for those include multiples $m \cdot g$ of g such that

$$\mathrm{LM}(m \cdot g) = m \cdot \mathrm{LM}(g) \in \mathrm{M}(f).$$

Gröbner Bases VII
for Cryptographers

$$\begin{aligned} f &= x^2 - 2xy - 2y^2 + \dots \\ x \cdot g &= \mathbf{x^2} + xy \dots \\ y \cdot g &= \mathbf{xy} + y^2 + \dots \\ g &= x + y + 2z \end{aligned}$$

$$\begin{pmatrix} 1 & -2 & -2 & 0 & 0 & 14 & 0 & \dots \\ 1 & 1 & 0 & 2 & 0 & 0 & 0 & \dots \\ 0 & 1 & 1 & 0 & 2 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \dots \end{pmatrix}$$

$$\begin{aligned} f' &= x^2 + 4yz + 14z^2, \\ h_1 &= \mathbf{xy} + 2xz + -4yz - \dots, \\ h_2 &= \mathbf{y^2} - 2xz + 6yz + \dots, \\ g &= x + y + 2z \end{aligned}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 & \dots \\ 0 & 1 & 0 & 2 & \dots & 0 & \dots \\ 0 & 0 & 1 & -2 & \dots & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots & 1 & \dots \end{pmatrix}$$

Let's call the preprocessing we performed "symbolic preprocessing" … but that alone is still not enough to solve the system.

Gröbner Bases VIII
for Cryptographers

Consider $f = yx + 1$ and $g = zx + 2$. Neither $\mathrm{LM}(f)$ nor $\mathrm{LM}(g)$ divides any monomial in the other polynomial. However, we have

$$\begin{aligned} zf - yg &= z(yx+1) - y(zx+2), \\ &= \mathbf{xyz} + z - \mathbf{xyz} - 2y, \\ &= z - 2y. \end{aligned}$$

We constructed multiples of f and g such that when we add them their leading terms cancel out. In other words, we constructed an *S-polynomial*.

Gröbner Bases IX
for Cryptographers

Definition (S-Polynomial)

Let $f, g \in \mathbb{F}[x_0, \dots, x_{n-1}]$ be non-zero polynomials.

- Let x^γ be the least common multiple of $\mathrm{LM}(f)$ and $\mathrm{LM}(g)$, written as
$$x^\gamma = \mathrm{LCM}(\mathrm{LM}(f), \mathrm{LM}(g)).$$
- The S-polynomial of f and g is defined as
$$S(f, g) = \frac{x^\gamma}{\mathrm{LT}(f)} \cdot f - \frac{x^\gamma}{\mathrm{LT}(g)} \cdot g.$$

It is *sufficient* to consider *only* multiples coming from S-polynomials since *any* reduction of leading terms can be attributed to S-polynomials.

Gröbner Bases X
for Cryptographers

```
Input: $F = [f_0, \dots, f_{m-1}]$ – list of polynomials
Output: a Gröbner basis for $\langle f_0, \dots, f_{m-1} \rangle$
1 begin
2     while True do
3         $\overline{F} \leftarrow$ multiply all pairs $f_i, f_j \in F$ by $m_i, m_j$ such that
            $\mathrm{LM}(m_i f_i) = \mathrm{LM}(m_j f_j)$;
4         $\overline{F} \leftarrow$ perform “symbolic preprocessing” on $\overline{F} \cup F$;
5         $\tilde{F} \leftarrow$ peform Gaussian elimination on $\overline{F}$ ;
6         $F \leftarrow F \cup \{f \in \tilde{F}$ with $\forall g \in F$ we have $\mathrm{LM}(g) \nmid \mathrm{LM}(f)\}$;
7         if $F$ didn't change in the last iteration then
8             return $F$;
```

Algorithm 1: simplified F_4

Gröbner Bases XI
for Cryptographers

Buchberger	select one pair in line 3 and use polynomial division instead of Gaussian elimination in line 5; implemented everywhere
F_4	use Buchberger's criteria in line 3 to avoid useless pairs (= zero rows in the matrix); implemented in MAGMA, POLYBORI, FGB
F_5	use criteria in lines 3 and 4 such that all matrices have full rank under some assumption; implementation worked on in SINGULAR
(Mutant)XL	multiply by everything up to some degree in line 3 and skip line 4 (worse than Algorithm 1 because of redundancies)
XSL	make some choice in line 3 and line 4 (worse than Algorithm 1 because of wrong choice)
ElimLin	always stay at degree 2 using change of ordering (exact relationship to Algorithm 1 unclear)

SAT Solvers I

```
# from http://bit.ly/h0022y
sage: attach "anf2cnf.py"
sage: B.<a,b> = BooleanPolynomialRing()
sage: aa = ANFSatSolver(B)
sage: print aa.cnf([a*b + b + 1])
p cnf 3 5
c ------------------------------
c Next definition: a*b + b + 1
-3 -2 0
3 2 0
c ------------------------------
c Next definition: a*b
1 -3 0
2 -3 0
3 -1 -2 0
```

```
1 begin
2     while True do
3         simplify clauses;
4         if contradiction then
5             backtrack;
6         if solution then
7             return;
8         guess something;
```

Gregory V. Bard, Nicolas T. Courtois, and Chris Jefferson.
Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers.
Cryptology ePrint Archive, Report 2007/024, 2007.

SAT Solvers II

- SAT solvers decide satisfiability, hence they will terminate once *one* solution is found.
- SAT solvers are randomised: one success does not constitute an average running time.
- Run hundreds/thousands of experiments with lots of re-randomisation.
- The conversion from ANF to CNF can make a huge difference, try Mate Soos' `http://gitorious.org/anfconv`.
- Different solvers behave differently, try Mate Soos' CryptoMiniSat.

Mixed Integer Programming I

- MIP minimises (or maximises) a linear function $c^T x$ subject to linear equality and inequality constraints given by linear inequalities $Ax \leq b$.
- We restrict some variables to integer values while others may take any real values.
- The main advantage of MIP solvers compared to other branch-and-cut solvers (SAT solvers etc.) is that they can relax the problem to an (easy) floating point problem.
- This allows to obtain lower and upper bounds for $c^T x$ which can be used to cut search branches.
- The non-linear generalisation is called Constraint Integer Programming (CIP).
- CPLEX & Gurobi (accademic licenses available), SCIP ($\approx$ open-source)

Mixed Integer Programming II

We can convert a polynomial $f \in \mathbb{F}_2[x_0, \ldots, x_{n-1}]$ to MIP and then use an off-the-shelf MIP solver (in this example SCIP).

```
sage: from sage.libs.scip.scip import SCIP
sage: B.<a,b,c> = BooleanPolynomialRing()
sage: f = a*c + a + b + c + 1
sage: s = SCIP(maximization=False,name="ecrypt2")
sage: s
SCIP Constraint Integer Program "ecrypt2"
( minimization, 0 variables, 0 constraints )
sage: s.read_polynomial_system_mod2(Sequence([f])); s
down: {b: 1, c: 0, a: 2}
  up: {0: c, 1: b, 2: a}
SCIP Constraint Integer Program "ecrypt2"
( minimization, 4 variables, 2 constraints )
```

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.

Cube Attacks and Friends I

2008 – the buzz:

> *“At this moment, Adi Shamir is giving an invited talk at the Crypto 2008 conference about a new type of cryptanalytic attack called ‘cube attacks.’ He claims very broad applicability to stream and block ciphers. My personal joke – at least I hope it’s a joke – is that he’s going to break every NIST hash submission without ever seeing any of them.”*
>
> – Bruce Schneier

2009 – the criticism:

> *“Why haven’t cube attacks broken anything? Is there some secret reason that every real-world cipher resists cube attacks? It turns out that the answer is yes.”*
>
> – Dan Bernstein

2011 – it is not all bad:

It seems these kind of techniques perform well for some hash functions and stream ciphers.

Cube Attacks and Friends II

The Cube attack is essentially a higher-order differential attack:

```
sage: B.<v1,v2,v3,x1,x2,x3> = PolynomialRing(GF(2))
sage: f = v1*v2*v3 + v1*v2*x1 + v1*v3*x1 + v2*v3*x1 + v1*v2*x3 \
        + v1*v3*x2 + v2*v3*x2+ v1*v3*x3 + v1*x1*x3 + v3*x2*x3 \
        + x1*x2*x3 + v1*v2 + v1*x3 + v3*x1 + x1*x2 + x2*x3 \
        + x2 + v1 + v3 + 1
sage: f.derivative(x1,x3)
v1 + x2
```

However, the derivation is performed “numerically” instead of symbolically:

```
sage: g = 0
...   for v in VectorSpace(GF(2),2):
...       g += f.subs(x1=v[0],x3=v[1])
sage: g
v1 + x2
```

This family of techniques seems to perform reasonably well for some hash functions and some stream ciphers.

Performance

Cipher	Method	System	RAM	Wall time
4r 4-bit AES	MRHS	???	1 GB	0.032s
10r 4-bit AES	MRHS	???	1 GB	0.32s
10r 4-bit AES	GB	Opteron 2.2 Ghz	16 GB	0.02s
10r 8-bit AES	GB	Opteron 2.2 Ghz	16 GB	0.2s
10r 16-bit AES	GB	Opteron 2.2 Ghz	16 GB	1205s
4r DES	ElimLin	Centrino 1.6 Ghz	??? GB	$2^{19} \cdot 8$s
5r DES	ElimLin	Centrino 1.6 Ghz	??? GB	$2^{23} \cdot 173$s
6r DES	SAT	Centrino 1.6 Ghz	??? GB	$2^{20} \cdot 68$s
PRESENT-80-2	SCIP	i7 2.6 Ghz	4 GB	3100s
PRESENT-80-5	ElimLin	??? 2.0 Ghz	1 GB	7200s

Table: Reported runtimes of various algorithms against reduced ciphers.

Outline

Beyond Solving I

- Consider an arbitrary function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ and its polynomial representation $f_0, \dots, f_{m-1}$
- Let $x_0, \dots x_{n-1}$ be the input variables and $y_0, \dots, y_{m-1}$ the output variables
- Consider the ideal $I = \langle f_0, \dots, f_{m-1} \rangle$:
  - Every member g of this ideal is a combination of $f_0, \dots, f_{m-1}$.
  - If $f_0, \dots, f_{m-1}$ vanish, so does g.
  - This can be read as: $f_0, \dots, f_{m-1}$ *implies* g.

"If $f_0, \dots, f_{m-1}$ hold, so does g".

Beyond Solving II

- Let c be a condition on the input variables (in polynomial form).
- Calculate a Gröbner basis for $\langle c, f_0, \dots, f_{m-1} \rangle$ in an elimination ordering which eliminates input variables first.
- The smallest elements of this Gröbner basis will be polynomials with a minimum number of input variables (if possible, none). Call them $g_0, \dots, g_{r-1}$.
- These polynomials are **implied** by the polynomials $f_0, \dots, f_{m-1}$ and the condition c.

"If $f_0, \dots, f_{m-1}$ and the condition c hold, so do $g_0, \dots, g_{r-1}$"

Beyond Solving III

- **All** on the output bits that are implied by f under condition c are **combinations** of $g_0, \dots, g_{r-1}$
- If we pick the term ordering right, $g_0, \dots, g_{r-1}$ have minimal degree.

For a given function f under a precondition c we can calculate **all** conditions on the output bits that **must** hold.

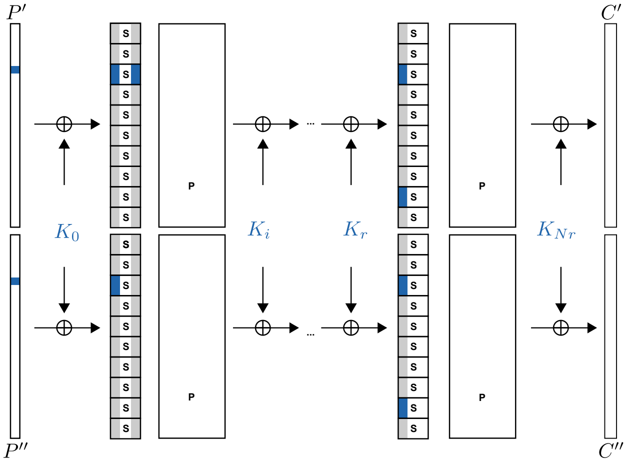
Beyond Solving IV

Some example applications:

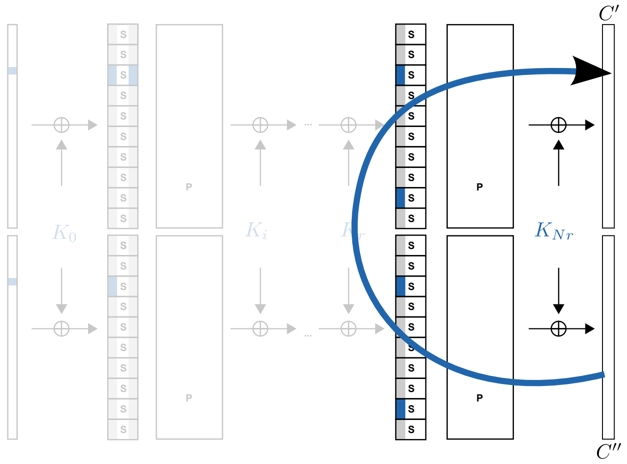
Differential:	algebraic description of all possible output differences under some input difference.
Cond. Diff.:	conditional relations on the plaintext and the key bits.
Integral:	algebraic descriptions on the output bits after r rounds.

Martin Albrecht, Carlos Cid, Thomas Dullien, Jean-Charles Faugère, and Ludovic Perret.
Algebraic precomputations in Differential and Integral Cryptanalysis.
In *INSCRYPT 2010 – Information Security and Cryptology 6th International Conference, Lecture Notes in Computer Science*, 18 pages, October 2010.

Algebraic Techniques and Differential Cryptanalysis I



Algebraic Techniques and Differential Cryptanalysis II



Algebraic Techniques and Differential Cryptanalysis III

1. Pick your favourite differential characteristic which holds with probability p.
2. Construct an equation system for two pairs $P' \Rightarrow C'$ and $P'' = P' + \Delta P \Rightarrow C''$.
3. Add linear equations of the form $X'_{i,j} + X''_{i,j} = \Delta X_{i,j}$ and $Y'_{i,j} + Y''_{i,j} = \Delta Y_{i,j}$
4. Attempt to solve $\mathcal{O}(1/p)$ such systems to get one that has a solution.

Cipher	System	RAM	pairs	time
PRESENT-80-14	C2D 2.33 Ghz	4 GB	$\approx 2^{44}$	$2^{72.60}$ CPU cycles
PRESENT-80-15	2.4 Ghz	64 GB	$\approx 2^{59}$	$2^{73.79}$ encryptions
PRESENT-128-14	2.4 Ghz	64 GB	$\approx 2^{55}$	$2^{112.83}$ encryptions
PRESENT-128-17	C2D 2.33 Ghz	4 GB	$\approx 2^{62}$	$2^{43.70} \cdot t$ CPU cycles*
KTANTAN32-113	C2D 2.33 Ghz	4 GB	$\approx 2^{31}$	2^{64} CPU cycles

Algebraic Techniques and Differential Cryptanalysis IV

* this is a successful attack if $t < 2^{89}$. There is no consensus whether this is plausible.

A more general perspective

We can view the algebraic-differential approach as a special case of:

- Find some property which holds with some probability p after r rounds (under some chosen inputs)
- Setup a smaller equation system which relates the property to the output
- Attempt to solve the smaller system $\mathcal{O}(1/p)$ times.
- For this smaller equation system we have very few "plaintext-ciphertext" pairs, hence algebraic techniques seem to fit well.

Nicolas T. Courtois, Gregory V. Bard and David Wagner.
Algebraic and Slide Attacks on KeeLoq
In *Fast Software Encryption – FSE 2008*, pages 97–115, Berlin, Heidelberg, New York, 2008. Springer Verlag

Nicolas T. Courtois.
Security Evaluation of GOST 28147-89 In View Of International Standardisation,
In *Cryptology ePrint Archive*, Report 2011/211, 2011

Algebraic Techniques and Integral Cryptanalysis

In Integral or Higher-Order Differential Cryptanalysis the attacker encrypts plaintexts with some structure such that the output (after some rounds) also has some (algebraic) structure.

We can use algebraic techniques find such algebraic relations (cf. Beyond Solving).

Cipher	Method	#P	Wall time
PRESENT-80-5	HODC	$5 \cdot 2^4$	$\approx 2^{25.7}$ CPU cycles
PRESENT-80-5	AHODC	2^4	$\approx 2^{23.3}$ CPU cycles
PRESENT-80-6	HODC	$2^{22.4}$	$\approx 2^{41.7}$ CPU cycles
PRESENT-80-6	AHODC	2^{20}	$\approx 2^{39.3}$ CPU cycles
PRESENT-80-7	HODC	$2^{24.4}$	$\approx 2^{100.1}$ CPU cycles
PRESENT-80-7	AHODC	$2^{21.9}$	$\approx 2^{97.8}$ CPU cycles
KTANTAN32-65	AHODC	2^5	59004.10 s

Algebraic Techniques and Side-Channel Cryptanalysis

- Side-channel attacks provide information about the internal state of an encryption operation to the attacker.
- This information can then be used to recover key information.
- Algebraic techniques seem to be natural candidates for this task, because they are good for tracking/propagating dependencies.

Mathieu Renauld and Francois-Xavier Standaert.
Algebraic Side-Channel Attacks.
In *INSCRYPT 2009 – Information Security and Cryptology 5th International Conference*, volume 6151 of *Lecture Notes in Computer Science*, pages 393-410, Berlin, Heidelberg, New York, 2009. Springer Verlag.

Martin Albrecht and Carlos Cid.
Cold Boot Key Recovery by Solving Polynomial Systems with Noise
To appear in *ACNS 2011 – 9th International Conference on Applied Cryptography and Network Security*, in *Lecture Notes in Computer Science*, Berlin, Heidelberg, New York, 2011. Springer Verlag.

Questions?

Thank You!