Algebraic Fault Attacks

Martin Kreuzer

Universität Passau

martin.kreuzer@uni-passau.de

Webinar "Symbolic Computation and Post-Quantum Cryptography"

The Internet, Apr. 19, 2012

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Collaborators

Prof. Dr. Ilia Polian
Chair of Technical Informatics
Faculty of Computer Science and Mathematics
University of Passau, Germany

Philipp Jovanovic
Faculty of Computer Science and Mathematics
University of Passau, Germany

1 – Algebraic Attacks

A man who tells the truth

1 – Algebraic Attacks

A man who tells the truth

needs a fast horse.

1 – Algebraic Attacks

A man who tells the truth
needs a fast horse.

A **cryptosystem** consists of the following parts:

a set $\mathcal{P}$, called **plaintext space**,

a set $\mathcal{C}$, called **ciphertext space**,

a set $\mathcal{K}$, called **key space**,

for every $k \in \mathcal{K}$ a map $\epsilon_k : \mathcal{P} \longrightarrow \mathcal{C}$, called **encryption map**

and a map $\delta_k : \mathcal{C} \longrightarrow \mathcal{P}$, called **decryption map** such that
$\delta_k \circ \epsilon_k = \mathrm{id}_{\mathcal{P}}$.

1 – Algebraic Attacks

A man who tells the truth
needs a fast horse.

A **cryptosystem** consists of the following parts:

a set $\mathcal{P}$, called **plaintext space**,

a set $\mathcal{C}$, called **ciphertext space**,

a set $\mathcal{K}$, called **key space**,

for every $k \in \mathcal{K}$ a map $\epsilon_k : \mathcal{P} \longrightarrow \mathcal{C}$, called **encryption map**

and a map $\delta_k : \mathcal{C} \longrightarrow \mathcal{P}$, called **decryption map** such that $\delta_k \circ \epsilon_k = \mathrm{id}_{\mathcal{P}}$.

Idea: Reduce the task of breaking a cryptosystem to the task of solving a polynomial system!

Let the plain text space and the cipher text space be of the form $\mathcal{P} = K^n$ and $\mathcal{C} = K^m$ with a finite field K. (Usually $K = \mathbb{F}_2$.)

Let the plain text space and the cipher text space be of the form $\mathcal{P} = K^n$ and $\mathcal{C} = K^m$ with a finite field K. (Usually $K = \mathbb{F}_2$.)

Remark 1.1 Every map $\varphi : K^n \longrightarrow K^m$ is given by polynomials, i.e. there exist polynomials $f_1, \ldots, f_m \in K[x_1, \ldots, x_n]$ such that

$$\varphi(a_1, \ldots, a_n) = (f_1(a_1, \ldots, a_n), \ldots, f_m(a_1, \ldots, a_n))$$

for all $(a_1, \ldots, a_n) \in K^n$.

Let the plain text space and the cipher text space be of the form $\mathcal{P} = K^n$ and $\mathcal{C} = K^m$ with a finite field K. (Usually $K = \mathbb{F}_2$.)

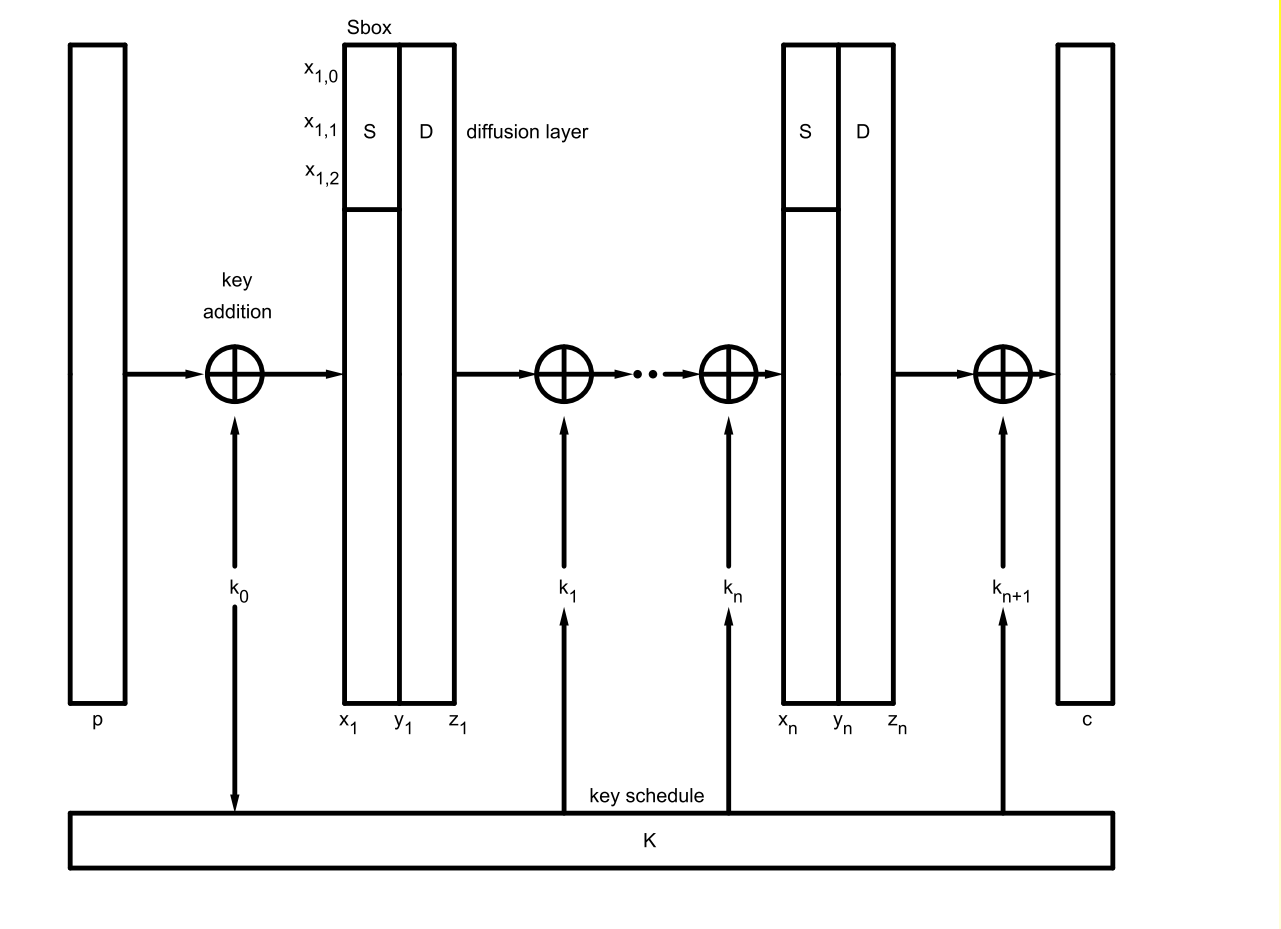
Remark 1.1 Every map $\varphi : K^n \longrightarrow K^m$ is given by polynomials, i.e. there exist polynomials $f_1, \ldots, f_m \in K[x_1, \ldots, x_n]$ such that

$$\varphi(a_1, \ldots, a_n) = (f_1(a_1, \ldots, a_n), \ldots, f_m(a_1, \ldots, a_n))$$

for all $(a_1, \ldots, a_n) \in K^n$.

Remark 1.2 **(Attacking a Symmetric Block Cipher)**
Assume that one or more plaintext – ciphertext pairs are known.

Write the encryption map using polynomials in indeterminates representing the key bits $(k_1, \ldots, k_\ell)$, the plaintext bits $(a_1, \ldots, a_n)$, and possibly intermediate states x_i, y_j.


Sbox
$x_{1,0}$
$x_{1,1}$
$x_{1,2}$
S
D
diffusion layer
S
D
key
addition
k_0
k_1
k_n
k_{n+1}
p
x_1 y_1 z_1
x_n y_n z_n
c
key schedule
K

Substitute the given plaintext unit $(a_1, \ldots, a_n)$ and the corresponding cyphertext unit $(b_1, \ldots, b_m)$ in

$$\begin{aligned} & \epsilon_k(a_1, \ldots, a_n) \\ = & (f_1(k_1, \ldots, k_\ell, x_i, y_j, a_1, \ldots, a_n), \ldots, f_m(k_1, \ldots, k_\ell, x_i, y_j, a_1, \ldots, a_n)) \\ = & \ (b_1, \ldots, b_m) \end{aligned}$$

Then solve the resulting polynomial system for the key indeterminates $(k_1, \ldots, k_\ell)$ and (possibly) the intermediate state bits x_i, y_j.

Remark 1.3 (Attacking a Public Key Cryptosystem)
Algebraic attacks work even better for public key cryptosystems. Let the decryption map be given by

$$\begin{aligned} & \delta_k(b_1, \ldots, b_m) \\ = \; & (f_1(k_1, \ldots, k_\ell, b_1, \ldots, b_m), \ldots, f_n(k_1, \ldots, k_\ell, b_1, \ldots, b_m)) \\ = \; & (a_1, \ldots, a_n) \end{aligned}$$

Since we can generate many ciphertext-plaintext pairs, we can produce a very overdetermined polynomial system for $(k_1, \ldots, k_\ell)$.

Remark 1.3 (Attacking a Public Key Cryptosystem) Algebraic attacks work even better for public key cryptosystems. Let the decryption map be given by

$$\begin{aligned} & \delta_k(b_1, \ldots, b_m) \\ = \; & (f_1(k_1, \ldots, k_\ell, b_1, \ldots, b_m), \ldots, f_n(k_1, \ldots, k_\ell, b_1, \ldots, b_m)) \\ = \; & (a_1, \ldots, a_n) \end{aligned}$$

Since we can generate many ciphertext-plaintext pairs, we can produce a very overdetermined polynomial system for $(k_1, \ldots, k_\ell)$.

Example 1.4 In 2003, J.C. Faugere and A. Joux broke the **Hidden Field Equation** (HFE) cryptosystem for some suggested parameters by solving 80 equations of degree 2 in 80 indeterminates over $\mathbb{F}_2$ using a Gröbner basis approach.

2 – Fault Attacks

To find fault is easy.

2 – Fault Attacks

To find fault is easy.

To do better may be difficult.

(Plutarch)

2 – Fault Attacks

To find fault is easy.
To do better may be difficult.
(Plutarch)

A **fault attack** is a particular kind of **side-channel attack**. The attacker does not directly target the encryption (or decryption) map, but rather its implementation.

2 – Fault Attacks

To find fault is easy.
To do better may be difficult.
(Plutarch)

A **fault attack** is a particular kind of **side-channel attack**. The attacker does not directly target the encryption (or decryption) map, but rather its implementation.

Some standard techniques for fault attacks use **fault injection** to manipulate logical values being processed by an electronic circuit, e.g.

- manipulation of the power-supply voltage to cause miscalculations,
- manipulation of a circuit's clock, and
- parasitic charge-carrier generation by a laser beam.

For a successful fault attack, the following capabilities of the attacker will be assumed:

For a successful fault attack, the following capabilities of the attacker will be assumed:

Sufficient temporal resolution: For instance, by measuring the electromagnetic field, the attacker should to be able to determine which part of the algorithm is currently carried out. For instance, for a multi-round block cipher, the attacker should know which round is currently executed.

For a successful fault attack, the following capabilities of the attacker will be assumed:

Sufficient temporal resolution: For instance, by measuring the electromagnetic field, the attacker should to be able to determine which part of the algorithm is currently carried out. For instance, for a multi-round block cipher, the attacker should know which round is currently executed.

Sufficient spacial resolution: The attacker should have some approximate knowledge in which register or memory cells a certain intermediate state of the computation is stored. While, for example, a laser may not have the precision to target individual memory cells, it may be able to target a register.

Example 2.1 **(The Bellcore Attack)**

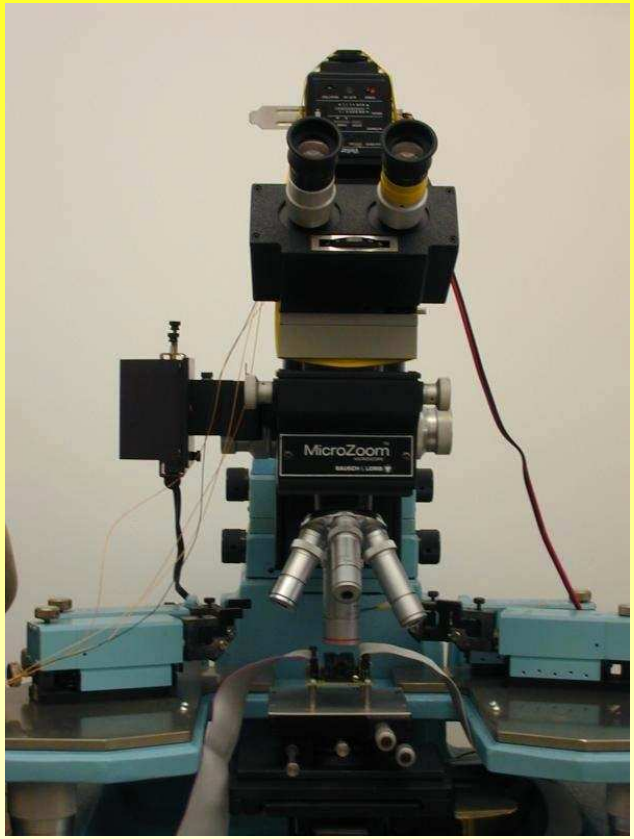
In 1996, three employees of **Bell Communications Research** found an attack that breaks a CRT-RSA decryption device by injecting a single fault.

To decode $y = x^e$, the CRT method computes $m_1 = y^d (\mathrm{mod}\, p)$ and $m_2 = y^d (\mathrm{mod}\, q)$ and recombines $x = a\, m_1 + b\, m_2 (\mathrm{mod}\, n)$ with suitable $a = \ell\, q$ and $b = k\, p$. If we disturb the computation of m_1, then $\gcd(x - \hat{x}, n) = q$.

Example 2.1 (The Bellcore Attack)
In 1996, three employees of **Bell Communications Research** found an attack that breaks a CRT-RSA decryption device by injecting a single fault.

To decode $y = x^e$, the CRT method computes $m_1 = y^d \pmod{p}$ and $m_2 = y^d \pmod{q}$ and recombines $x = a\,m_1 + b\,m_2 \pmod{n}$ with suitable $a = \ell\,q$ and $b = k\,p$. If we disturb the computation of m_1, then $\gcd(x - \hat{x}, n) = q$.

Example 2.2 (Optical Fault Induction)
In 2002, Skorobogatov and Anderson suggested a very simple and effective way to induce faults in smartcards and microcontrollers using a flashgun (30$) and a laser pointer ($5).


MicroZoom

Fault Attacks on DES

Example 2.3 The first fault attack on DES was suggested by Biham and Shamir in 1997. They assume that it is possible to flip exactly one bit at an unknown point in the DES algorithm, and they need to repeat this with 50-200 encryptions of the same plain text.

Fault Attacks on DES

Example 2.3 The first fault attack on DES was suggested by Biham and Shamir in 1997. They assume that it is possible to flip exactly one bit at an unknown point in the DES algorithm, and they need to repeat this with 50-200 encryptions of the same plain text.

This was improved many times. For instance, in 2010, Courtois, Jackson and Ware showed that it suffices to create **one faulty ciphertext** by flipping two bits in round 13 and guessing 24 key bits.

(There are 56 key bits and 16 rounds.)

Fault Attacks on DES

Example 2.3 The first fault attack on DES was suggested by Biham and Shamir in 1997. They assume that it is possible to flip exactly one bit at an unknown point in the DES algorithm, and they need to repeat this with 50-200 encryptions of the same plain text.

This was improved many times. For instance, in 2010, Courtois, Jackson and Ware showed that it suffices to create **one faulty ciphertext** by flipping two bits in round 13 and guessing 24 key bits.

(There are 56 key bits and 16 rounds.)

Also variants such as **Triple-DES** have been attacked successfully.

Fault Attacks on AES

Example 2.4 The first fault attack on AES was reported by Blömer and Seifert in 2003. They assume that an attacker is able to perform 128 encryptions of the same plain text and set a specific bit to zero at a specific point in the algorithm each time.

Fault Attacks on AES

Example 2.4 The first fault attack on AES was reported by Blömer and Seifert in 2003. They assume that an attacker is able to perform 128 encryptions of the same plain text and set a specific bit to zero at a specific point in the algorithm each time.

Example 2.5 The initial fault attack on AES has been improved in several ways. A state-of-the-art variant was presented by Tunstall, Mukhopadhyay and Ali in 2011, using the following **fault model**: the attacker can create a fault at the first byte of the input state matrix of round 8. (Standard AES performs 10 rounds.) Fault induction is achieved by changing the clock frequency or the power supply voltage at a specific point during the AES encryption algorithm. The key space is reduced to about 2^8 candidate keys.

Fault Attacks on PRESENT

PRESENT is an ultra lightweight block cipher introduced in 2007. It has 64 bit states, 80 or 128 bit keys, and performs 31 encryption rounds.

Fault Attacks on PRESENT

PRESENT is an ultra lightweight block cipher introduced in 2007. It has 64 bit states, 80 or 128 bit keys, and performs 31 encryption rounds.

Example 2.6 In 2010, G. Wang and S. Wang presented a fault attack on the key schedule of PRESENT. They assume that the attacker is able to create a fault in one nibble (4 bit) of the round key K_{31}. Using 64 pairs of correct and faulty ciphertexts, they can reduce the search space to 2^{29} key candidates.

Fault Attacks on PRESENT

PRESENT is an ultra lightweight block cipher introduced in 2007. It has 64 bit states, 80 or 128 bit keys, and performs 31 encryption rounds.

Example 2.6 In 2010, G. Wang and S. Wang presented a fault attack on the key schedule of PRESENT. They assume that the attacker is able to create a fault in one nibble (4 bit) of the round key K_{31}. Using 64 pairs of correct and faulty ciphertexts, they can reduce the search space to 2^{29} key candidates.

Moreover, reduced-round versions of PRESENT have been subjected to differential cryptoanalysis by M. Wang.

3 – A Fault Attack on LED

If an experiment works

3 – A Fault Attack on LED

If an experiment works

something has gone wrong.

(Finagle's Eighth Law)

3 – A Fault Attack on LED

If an experiment works
something has gone wrong.
(Finagle's Eighth Law)

The **LED (Light Encryption Device)** cipher is a block cipher introduced in 2011 having the following properties:

3 – A Fault Attack on LED

If an experiment works
something has gone wrong.
(Finagle's Eighth Law)

The **LED (Light Encryption Device)** cipher is a block cipher introduced in 2011 having the following properties:

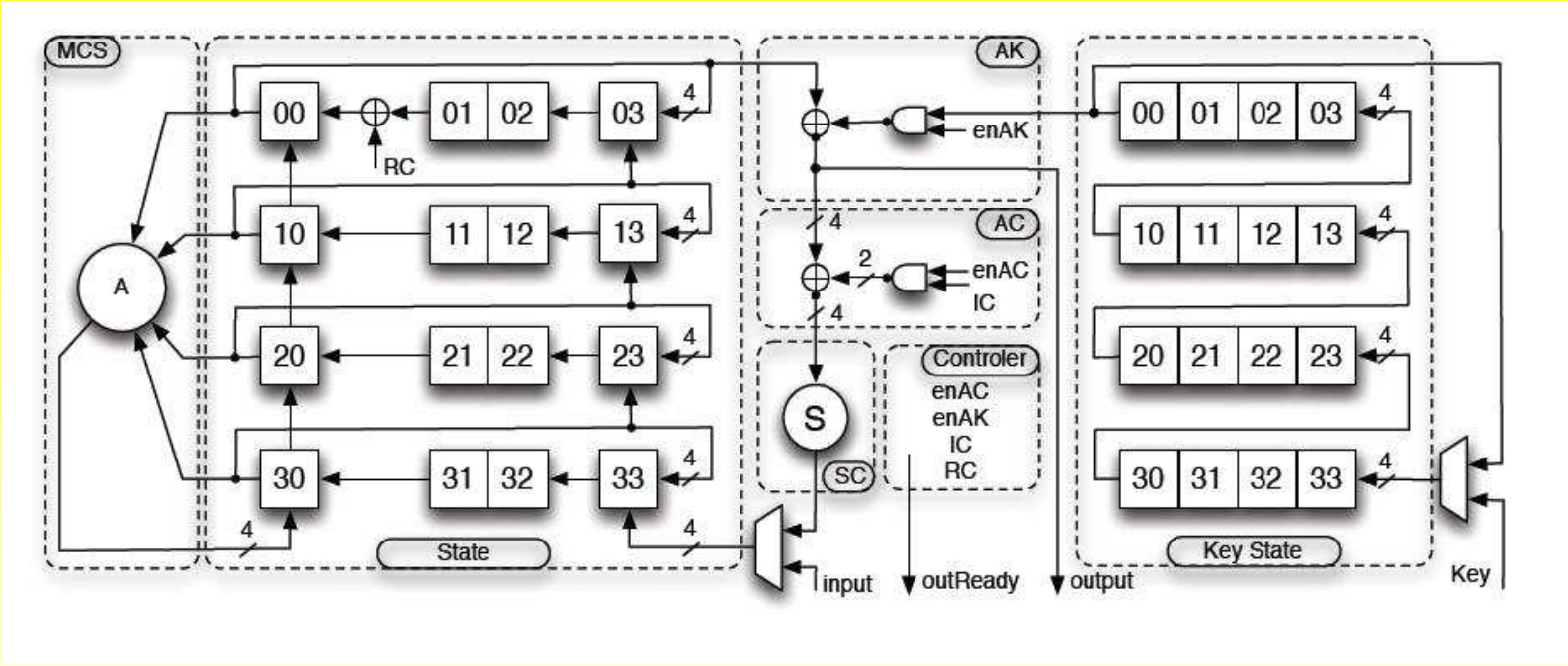
- 64-bit block cipher, small and fast like PRESENT
- uses one or two 64 bit keys (LED-64, LED-128), **no key schedule**
- uses 32 rounds for LED-64 and 48 rounds for LED-128

3 – A Fault Attack on LED

If an experiment works
something has gone wrong.
(Finagle's Eighth Law)

The **LED (Light Encryption Device)** cipher is a block cipher introduced in 2011 having the following properties:

- 64-bit block cipher, small and fast like PRESENT
- uses one or two 64 bit keys (LED-64, LED-128), **no key schedule**
- uses 32 rounds for LED-64 and 48 rounds for LED-128
- provably secure against classical cryptoanalysis
- extremely small silicone footprint


MCS
A
00
01
02
03
RC
10
11
12
13
20
21
22
23
30
31
32
33
4
State
AK
enAK
AC
2
enAC
IC
Controler
enAC
enAK
IC
RC
S
SC
input
outReady
output
Key State
Key

Description of LED

The current state of LED is represented by a 4x4 matrix of nibbles

$$S = \begin{pmatrix} s_1 & s_2 & s_3 & s_4 \\ s_5 & s_6 & s_7 & s_8 \\ s_9 & s_{10} & s_{11} & s_{12} \\ s_{13} & s_{14} & s_{15} & s_{16} \end{pmatrix}$$

which corresponds to the 64-bit tuple $s_1 \parallel s_2 \parallel \cdots \parallel s_{16}$. The key is similarly represented by $K = (k_i)$.

Description of LED

The current state of LED is represented by a 4x4 matrix of nibbles

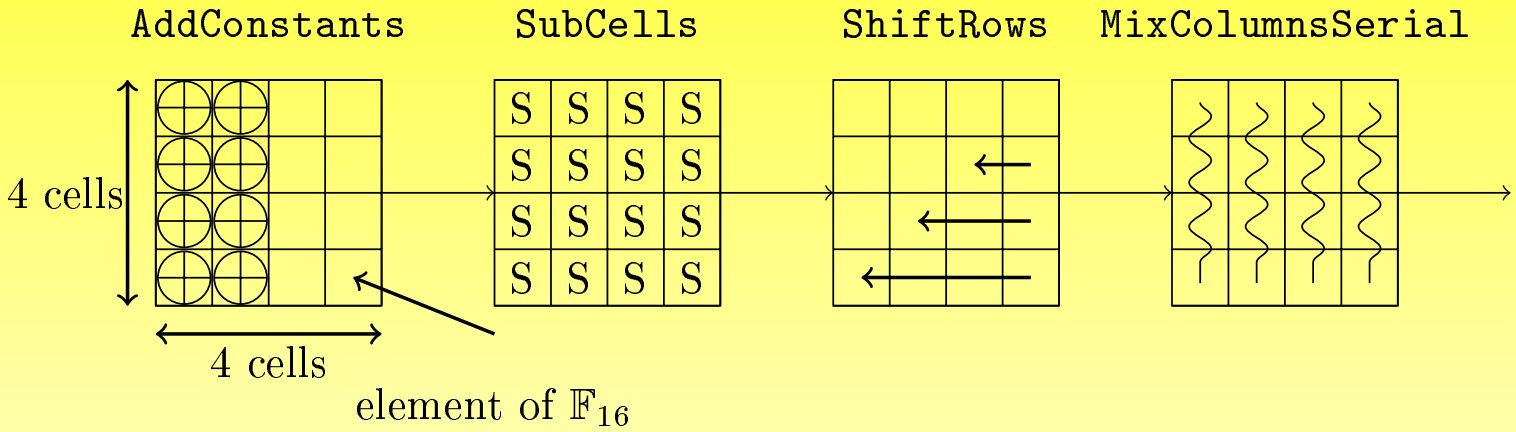
$$S = \begin{pmatrix} s_1 & s_2 & s_3 & s_4 \\ s_5 & s_6 & s_7 & s_8 \\ s_9 & s_{10} & s_{11} & s_{12} \\ s_{13} & s_{14} & s_{15} & s_{16} \end{pmatrix}$$

which corresponds to the 64-bit tuple $s_1 \parallel s_2 \parallel \cdots \parallel s_{16}$. The key is similarly represented by $K = (k_i)$.

The entries s_i correspond to elements of $\mathbb{F}_{16} = \mathbb{F}_2[x]/\langle x^4 + x + 1 \rangle$ and are represented by coefficient tuples in hexadecimal form:

$$x^3 + x + 1 \longmapsto 1\,0\,1\,1 \longmapsto \texttt{B}$$

Every round of LED consists of four operations:



Every four rounds there is a key addition.

AddConstants (AC): A round constant consisting of a tuple of six bits $(b_5, b_4, b_3, b_2, b_1, b_0)$ is defined as follows.

Before the first round, we start with the zero tuple. In consecutive rounds, we start with the previous round constant. Then we shift the six bits one position to the left. The new value of b_0 is computed as $b_5 + b_4 + 1$. This results in the following round constants.

Rounds	Constants
1-24	01,03,07,0F,1F,3E,3D,3B,37,2F,1E,3C,39,33,27,0E,1D,3A,35,2B,16,2C,18,30
25-48	21,02,05,0B,17,2E,1C,38,31,23,06,0D,1B,36,2D,1A,34,29,12,24,08,11,22,04

Form $\mathtt{x} = b_5 \mid\mid b_4 \mid\mid b_3$ and $\mathtt{y} = b_2 \mid\mid b_1 \mid\mid b_0$ and add $\begin{pmatrix} 0 & \mathtt{x} & 0 & 0 \\ 1 & \mathtt{y} & 0 & 0 \\ 2 & \mathtt{x} & 0 & 0 \\ 3 & \mathtt{y} & 0 & 0 \end{pmatrix}$ to the state matrix.

SubCells (SC): Each entry x of the state matrix is replaced by the element $S[x]$ from the **SBox** given in the following table. (This is the SBox used by PRESENT.)

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

SubCells (SC): Each entry x of the state matrix is replaced by the element $S[x]$ from the **SBox** given in the following table. (This is the SBox used by PRESENT.)

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

ShiftRows (SR): For $i = 1, 2, 3, 4$, the i-th row of the state matrix is shifted cyclically to the left by $i - 1$ positions.

SubCells (SC): Each entry x of the state matrix is replaced by the element $S[x]$ from the **SBox** given in the following table. (This is the SBox used by PRESENT.)

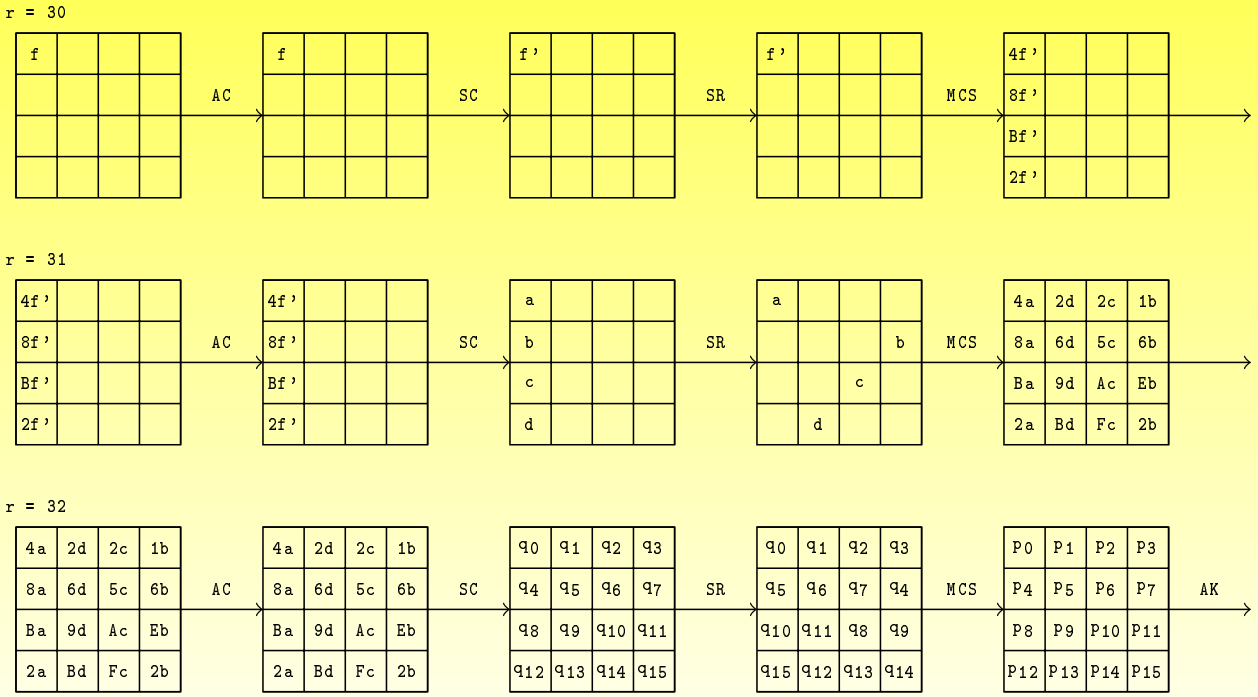
x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

ShiftRows (SR): For $i = 1, 2, 3, 4$, the i-th row of the state matrix is shifted cyclically to the left by $i - 1$ positions.

MixColumnsSerial (MCS): Each column v of the state matrix is replaced by the product $M \cdot v$, where M is the matrix $M = \begin{pmatrix} 4 & 1 & 2 & 2 \\ 8 & 6 & 5 & 6 \\ B & E & A & 9 \\ 2 & 2 & F & B \end{pmatrix}$

Fault Propagation in LED

We inject a random fault in the first entry of the state matrix at the beginning of round 30.



Fault Equations

We consider the correct cipher text c and the faulty cipher text c' as starting points and invert every operation of the encryption until the beginning of round 30. The 4-bit sized elements $k_0, \ldots, k_{15}$ of the key are viewed as indeterminates.

Fault Equations

We consider the correct cipher text c and the faulty cipher text c' as starting points and invert every operation of the encryption until the beginning of round 30. The 4-bit sized elements $k_0, \ldots, k_{15}$ of the key are viewed as indeterminates.

`AddKey`$^{-1}$**:** Replace c_i by $c_i + k_i$ and c'_i by $c'_i + k_i$.

Fault Equations

We consider the correct cipher text c and the faulty cipher text c' as starting points and invert every operation of the encryption until the beginning of round 30. The 4-bit sized elements $k_0, \ldots, k_{15}$ of the key are viewed as indeterminates.

`AddKey`$^{-1}$**:** Replace c_i by $c_i + k_i$ and c'_i by $c'_i + k_i$.

`MixColumnSerial`$^{-1}$**:** Multiply by the inverse matrix

$$M^{-1} = \begin{pmatrix} \mathtt{C} & \mathtt{C} & \mathtt{D} & \mathtt{4} \\ \mathtt{3} & \mathtt{8} & \mathtt{4} & \mathtt{5} \\ \mathtt{7} & \mathtt{6} & \mathtt{2} & \mathtt{E} \\ \mathtt{D} & \mathtt{9} & \mathtt{9} & \mathtt{D} \end{pmatrix}$$

and get the expressions

$$\mathtt{C} \cdot (c_0 + k_0) + \mathtt{C} \cdot (c_4 + k_4) + \mathtt{D} \cdot (c_8 + k_8) + \mathtt{4} \cdot (c_{12} + k_{12}) \text{ resp.}$$

$$\mathtt{C} \cdot (c'_0 + k_0) + \mathtt{C} \cdot (c'_4 + k_4) + \mathtt{D} \cdot (c'_8 + k_8) + \mathtt{4} \cdot (c'_{12} + k_{12}).$$

`ShiftRows`$^{-1}$**:** The expressions are permuted, but not changed.

`ShiftRows`$^{-1}$: The expressions are permuted, but not changed.

`SubCells`$^{-1}$: If S^{-1} is the inverse of the LED SBox, we get

$S^{-1}(\mathtt{C} \cdot (c_0 + k_0) + \mathtt{C} \cdot (c_4 + k_4) + \mathtt{D} \cdot (c_8 + k_8) + \mathtt{4} \cdot (c_{12} + k_{12}))$ resp.

$S^{-1}(\mathtt{C} \cdot (c'_0 + k_0) + \mathtt{C} \cdot (c'_4 + k_4) + \mathtt{D} \cdot (c'_8 + k_8) + \mathtt{4} \cdot (c'_{12} + k_{12}))$.

$\texttt{ShiftRows}^{-1}$: The expressions are permuted, but not changed.

$\texttt{SubCells}^{-1}$: If S^{-1} is the inverse of the LED SBox, we get

$$S^{-1}(\texttt{C}\cdot(c_0+k_0)+\texttt{C}\cdot(c_4+k_4)+\texttt{D}\cdot(c_8+k_8)+\texttt{4}\cdot(c_{12}+k_{12})) \text{ resp.}$$

$$S^{-1}(\texttt{C}\cdot(c'_0+k_0)+\texttt{C}\cdot(c'_4+k_4)+\texttt{D}\cdot(c'_8+k_8)+\texttt{4}\cdot(c'_{12}+k_{12})).$$

Fault Indeterminates: Now equate the difference of the traces of c and c' with indeterminates coming from the fault propagation.

$$\begin{aligned}\texttt{4}a = {} & S^{-1}(\texttt{C}\cdot(c_0+k_0)+\texttt{C}\cdot(c_4+k_4)+\texttt{D}\cdot(c_8+k_8)+\texttt{4}\cdot(c_{12}+k_{12})) \\ & + S^{-1}(\texttt{C}\cdot(c'_0+k_0)+\texttt{C}\cdot(c'_4+k_4)+\texttt{D}\cdot(c'_8+k_8)+\texttt{4}\cdot(c'_{12}+k_{12}))\end{aligned}$$

and similar equations for the other 15 entries of the state matrix result. Altogether, we find 16 equations over $\mathbb{F}_{16}$ in the indeterminates a, b, c, d representing the fault propagation and the indeteminates $k_1, \ldots, k_{16}$ representing the secret key.

Key Tuple Filtering

Next we use the fault equations to reduce the set of candidate keys until it is small enough for an exhaustive search.

Key Tuple Filtering

Next we use the fault equations to reduce the set of candidate keys until it is small enough for an exhaustive search.

Let x be one of the **fault indeterminates** $\{a, b, c, d\}$ and let $i \in \{1, 2, 3, 4\}$.

Key Tuple Filtering

Next we use the fault equations to reduce the set of candidate keys until it is small enough for an exhaustive search.

Let x be one of the **fault indeterminates** $\{a, b, c, d\}$ and let $i \in \{1, 2, 3, 4\}$.

Each fault equation $E_{x,i}$ depends only on 4 key indeterminates.

Key Tuple Filtering

Next we use the fault equations to reduce the set of candidate keys until it is small enough for an exhaustive search.

Let x be one of the **fault indeterminates** $\{a, b, c, d\}$ and let $i \in \{1, 2, 3, 4\}$.

Each fault equation $E_{x,i}$ depends only on 4 key indeterminates.

Compute a list $S_{x,i}$ of length 16. Its j-th entry is the set of all 4-tuples of values of key indeterminates for which $E_{x,i}$ evaluates to the j-th element of $\mathbb{F}_{16}$. (These are 16^5 evaluations of simple polynomials.)

Key Tuple Filtering

Next we use the fault equations to reduce the set of candidate keys until it is small enough for an exhaustive search.

Let x be one of the **fault indeterminates** $\{a, b, c, d\}$ and let $i \in \{1, 2, 3, 4\}$.

Each fault equation $E_{x,i}$ depends only on 4 key indeterminates.

Compute a list $S_{x,i}$ of length 16. Its j-th entry is the set of all 4-tuples of values of key indeterminates for which $E_{x,i}$ evaluates to the j-th element of $\mathbb{F}_{16}$. (These are 16^5 evaluations of simple polynomials.)

Now find all values j_x such that the j-th entry of $S_{x,1}(j_x)$ and $S_{x,2}(j_x)$ and $S_{x,3}(j_x)$ and $S_{x,4}(j_x)$ are all non-empty. A field element j_x is called a possible **fault value** for x.

Finally, for all possible **fault tuples** (j_a, j_b, j_c, j_d) intersect those sets $S_{x,i}$ which correspond to equations involving the same key indeterminates:

$$
\begin{aligned}
(k_0, k_4, k_8, k_{12}) : &\quad S_{a,0}(j_a) \cap S_{d,1}(j_d) \cap S_{c,2}(j_c) \cap S_{b,3}(j_b) \\
(k_1, k_5, k_9, k_{13}) : &\quad S_{a,3}(j_a) \cap S_{d,0}(j_d) \cap S_{c,1}(j_c) \cap S_{b,2}(j_b) \\
(k_2, k_6, k_{10}, k_{14}) : &\quad S_{a,2}(j_a) \cap S_{d,3}(j_d) \cap S_{c,0}(j_c) \cap S_{b,1}(j_b) \\
(k_3, k_7, k_{11}, k_{15}) : &\quad S_{a,1}(j_a) \cap S_{d,2}(j_d) \cap S_{c,3}(j_c) \cap S_{b,0}(j_b)
\end{aligned}
$$

Finally, for all possible **fault tuples** (j_a, j_b, j_c, j_d) intersect those sets $S_{x,i}$ which correspond to equations involving the same key indeterminates:

$$
\begin{aligned}
(k_0, k_4, k_8, k_{12}) : &\quad S_{a,0}(j_a) \cap S_{d,1}(j_d) \cap S_{c,2}(j_c) \cap S_{b,3}(j_b) \\
(k_1, k_5, k_9, k_{13}) : &\quad S_{a,3}(j_a) \cap S_{d,0}(j_d) \cap S_{c,1}(j_c) \cap S_{b,2}(j_b) \\
(k_2, k_6, k_{10}, k_{14}) : &\quad S_{a,2}(j_a) \cap S_{d,3}(j_d) \cap S_{c,0}(j_c) \cap S_{b,1}(j_b) \\
(k_3, k_7, k_{11}, k_{15}) : &\quad S_{a,1}(j_a) \cap S_{d,2}(j_d) \cap S_{c,3}(j_c) \cap S_{b,0}(j_b)
\end{aligned}
$$

The resulting tuples $(k_0, \ldots, k_{15})$ form the **key candidate set**. Each of the intersections contains typically $2^4 - 2^8$ elements. The key candidate set contains typically $2^{19} - 2^{25}$ key tuples.

Finally, for all possible **fault tuples** (j_a, j_b, j_c, j_d) intersect those sets $S_{x,i}$ which correspond to equations involving the same key indeterminates:

$$
\begin{aligned}
(k_0, k_4, k_8, k_{12}) : &\quad S_{a,0}(j_a) \cap S_{d,1}(j_d) \cap S_{c,2}(j_c) \cap S_{b,3}(j_b) \\
(k_1, k_5, k_9, k_{13}) : &\quad S_{a,3}(j_a) \cap S_{d,0}(j_d) \cap S_{c,1}(j_c) \cap S_{b,2}(j_b) \\
(k_2, k_6, k_{10}, k_{14}) : &\quad S_{a,2}(j_a) \cap S_{d,3}(j_d) \cap S_{c,0}(j_c) \cap S_{b,1}(j_b) \\
(k_3, k_7, k_{11}, k_{15}) : &\quad S_{a,1}(j_a) \cap S_{d,2}(j_d) \cap S_{c,3}(j_c) \cap S_{b,0}(j_b)
\end{aligned}
$$

The resulting tuples $(k_0, \ldots, k_{15})$ form the **key candidate set**. Each of the intersections contains typically $2^4 - 2^8$ elements. The key candidate set contains typically $2^{19} - 2^{25}$ key tuples.

The key candidate sets are pairwise disjoint. Only one of them contains the correct key. It remains to find out which.

Key Set Filtering

Let x_0, x_4, x_8, x_{12} be the entries of the first column of the state matrix at the beginning of round 31. Fault propagation yields

$$x'_0 = x_0 + \mathtt{4}f' \qquad x'_8 = x_8 + \mathtt{B}f'$$
$$x'_4 = x_4 + \mathtt{8}f' \qquad x'_{12} = x_{12} + \mathtt{2}f'$$

for the faulty entries x'_i.

Key Set Filtering

Let x_0, x_4, x_8, x_{12} be the entries of the first column of the state matrix at the beginning of round 31. Fault propagation yields

$$x'_0 = x_0 + \mathtt{4}f' \qquad x'_8 = x_8 + \mathtt{B}f'$$
$$x'_4 = x_4 + \mathtt{8}f' \qquad x'_{12} = x_{12} + \mathtt{2}f'$$

for the faulty entries x'_i. Let y_0, y_4, y_8, y_{12} be the values in the first column after **AddColumns** and **SubCells**. We get

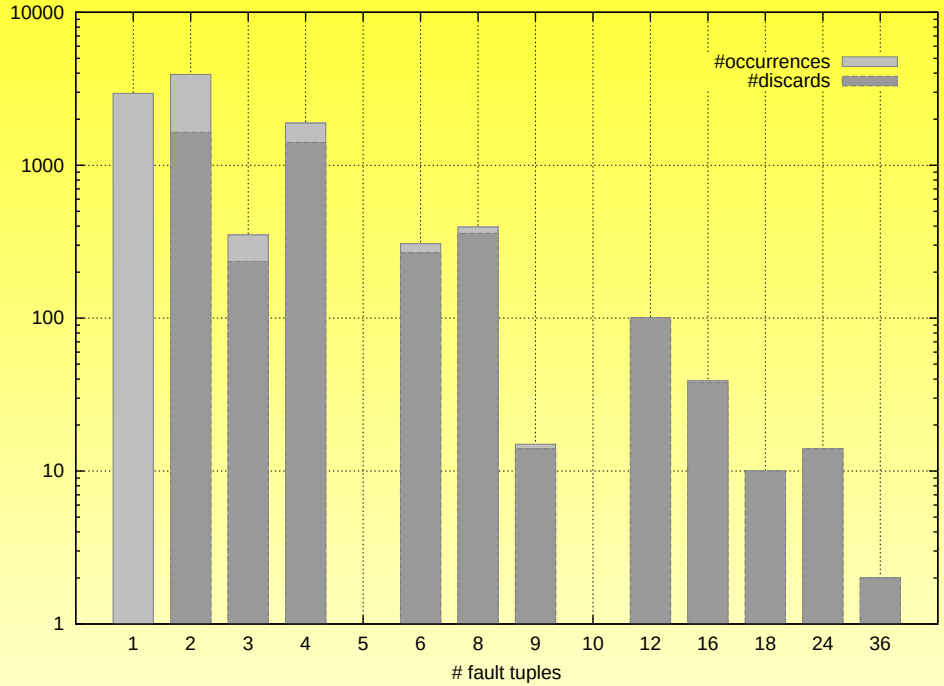
$$S(x_0 + \mathtt{0}) = y_0 \qquad S(x'_0 + \mathtt{0}) = y_0 + a$$
$$S(x_4 + \mathtt{1}) = y_4 \qquad S(x'_4 + \mathtt{1}) = y_4 + b$$
$$S(x_8 + \mathtt{2}) = y_8 \qquad S(x'_8 + \mathtt{2}) = y_8 + c$$
$$S(x_{12} + \mathtt{3}) = y_{12} \qquad S(x'_{12} + \mathtt{3}) = y_{12} + d$$

This yields the equations

$$\begin{aligned}
\texttt{4}f' &= S^{-1}(y_0) + S^{-1}(y_0 + a) \\
\texttt{8}f' &= S^{-1}(y_4) + S^{-1}(y_4 + b) \\
\texttt{B}f' &= S^{-1}(y_8) + S^{-1}(y_8 + c) \\
\texttt{2}f' &= S^{-1}(y_{12}) + S^{-1}(y_{12} + d)
\end{aligned}$$

For every fault tuple (j_a, j_b, j_c, j_d), check whether this system has a solution. In this way we can discard many fault tuples and the corresponding key sets, as the following table and figure show.

# fault tuples	2	3	4	5	6	8	9	10	12	16	18	24	36
ø discarded	0.4	0.9	1.4	2.0	2.5	3.6	3.7	5.0	6.1	8.4	8.4	12.6	24.0


10000
1000
100
10
1
#occurrences
#discards
1
2
3
4
5
6
8
9
10
12
16
18
24
36
fault tuples

4 – Algebraic Fault Attacks

People will accept your ideas much more readily

4 – Algebraic Fault Attacks

People will accept your ideas much more readily
if you tell them Grothendieck said it first.
(Algebraics Anonymous)

4 – Algebraic Fault Attacks

People will accept your ideas much more readily
if you tell them Grothendieck said it first.
(Algebraics Anonymous)

Idea: Combine algebraic attacks and fault attacks!

4 – Algebraic Fault Attacks

People will accept your ideas much more readily
if you tell them Grothendieck said it first.
(Algebraics Anonymous)

Idea: Combine algebraic attacks and fault attacks!

More precise idea: Describe a cipher algebraically using multivariate polynomials. Describe the fault equations using multivariate polynomials. Then solve the resulting poylnomial system for the key indeterminates.

An Algebraic Fault Attack for LED

First Step: Determine the polynomials describing the steps of an LED round. Describe an LED state $s_1 \parallel s_2 \parallel \cdots \parallel s_{16}$ by representing $s_i \in \mathbb{F}_{16}$ using $s_i = c_{4i-3}x^3 + c_{4i-2}x^2 + c_{4i-1}x + c_{4i}$ by elements $c_1, \ldots, c_{64} \in \mathbb{F}_2$ where we use $\mathbb{F}_{16} \cong \mathbb{F}_2[x]/\langle x^4 + x + 1\rangle$.

An Algebraic Fault Attack for LED

First Step: Determine the polynomials describing the steps of an LED round. Describe an LED state $s_1 \parallel s_2 \parallel \cdots \parallel s_{16}$ by representing $s_i \in \mathbb{F}_{16}$ using $s_i = c_{4i-3}x^3 + c_{4i-2}x^2 + c_{4i-1}x + c_{4i}$ by elements $c_1, \ldots, c_{64} \in \mathbb{F}_2$ where we use $\mathbb{F}_{16} \cong \mathbb{F}_2[x]/\langle x^4 + x + 1\rangle$.

Let p_i be the plaintext bits, k_i the key bits, $x_i^{(r)}$ the input state bits of the SBox in round r, $y_i^{(r)}$ intermediate state bits of round r, $z_i^{(r)}$ output state bits of round r, and c_i the ciphertext bits.

An Algebraic Fault Attack for LED

First Step: Determine the polynomials describing the steps of an LED round. Describe an LED state $s_1 \parallel s_2 \parallel \cdots \parallel s_{16}$ by representing $s_i \in \mathbb{F}_{16}$ using $s_i = c_{4i-3}x^3 + c_{4i-2}x^2 + c_{4i-1}x + c_{4i}$ by elements $c_1, \ldots, c_{64} \in \mathbb{F}_2$ where we use $\mathbb{F}_{16} \cong \mathbb{F}_2[x]/\langle x^4 + x + 1\rangle$.

Let p_i be the plaintext bits, k_i the key bits, $x_i^{(r)}$ the input state bits of the SBox in round r, $y_i^{(r)}$ intermediate state bits of round r, $z_i^{(r)}$ output state bits of round r, and c_i the ciphertext bits.

Thus we are constructing polynomials in the ring $\mathbb{F}_2[p_i, k_i, x_i^{(r)}, y_i^{(r)}, z_i^{(r)}, c_i]$ having 6336 indeterminates.

AddConstants: For the first round, we combine the key addition and the addition of the matrix corresponding to the first round constant $(b_5, b_4, \ldots, b_0) = (0, \ldots, 0, 1)$ and get

$$\begin{aligned} x_i^{(1)} &= p_i + k_i + 1 \qquad \text{for } i = 20, 24, 35, 51, 52, 56, \\ x_i^{(1)} &= p_i + k_i \qquad \text{otherwise.} \end{aligned}$$

AddConstants: For the first round, we combine the key addition and the addition of the matrix corresponding to the first round constant $(b_5, b_4, \ldots, b_0) = (0, \ldots, 0, 1)$ and get

$$\begin{aligned} x_i^{(1)} &= p_i + k_i + 1 \qquad \text{for } i = 20, 24, 35, 51, 52, 56, \\ x_i^{(1)} &= p_i + k_i \qquad\qquad \text{otherwise.} \end{aligned}$$

For later rounds, we let $(b_5^{(r)}, \ldots, b_0^{(r)})$ be the r-th round constants. We obtain

$$x_i^{(r)} = z_i^{(r-1)} + 1 \text{ for } i = 20, 35, 51, 52 \qquad x_i^{(r)} = z_i^{(r-1)} + b_5^{(r)} \text{ for } i = 6, 38$$

$$x_i^{(r)} = z_i^{(r-1)} + b_4^{(r)} \text{ for } i = 7, 39 \qquad x_i^{(r)} = z_i^{(r-1)} + b_3^{(r)} \text{ for } i = 8, 40$$

$$x_i^{(r)} = z_i^{(r-1)} + b_2^{(r)} \text{ for } i = 22, 54 \qquad x_i^{(r)} = z_i^{(r-1)} + b_1^{(r)} \text{ for } i = 23, 55$$

$$x_i^{(r)} = z_i^{(r-1)} + b_0^{(r)} \text{ for } i = 24, 56 \qquad x_i^{(r)} = z_i^{(r-1)} \text{ otherwise}$$

SubCells: The input (x_1, x_2, x_3, x_4) and output (y_1, y_2, y_3, y_4) of the 4-bit SBox are related by the polynomials

$$y_1 = x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4 + x_2x_3 + x_1 + x_3 + x_4 + 1$$

$$y_2 = x_1x_2x_4 + x_1x_3x_4 + x_1x_3 + x_1x_4 + x_3x_4 + x_1 + x_2 + 1$$

$$y_3 = x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4 + x_1x_2 + x_1x_3 + x_1 + x_3$$

$$y_4 = x_2x_3 + x_1 + x_2 + x_4$$

SubCells: The input (x_1, x_2, x_3, x_4) and output (y_1, y_2, y_3, y_4) of the 4-bit SBox are related by the polynomials

$$y_1 = x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4 + x_2x_3 + x_1 + x_3 + x_4 + 1$$

$$y_2 = x_1x_2x_4 + x_1x_3x_4 + x_1x_3 + x_1x_4 + x_3x_4 + x_1 + x_2 + 1$$

$$y_3 = x_1x_2x_4 + x_1x_3x_4 + x_2x_3x_4 + x_1x_2 + x_1x_3 + x_1 + x_3$$

$$y_4 = x_2x_3 + x_1 + x_2 + x_4$$

ShiftRows: We combine the action of SubCells and ShiftRows into one set of polynomials. The permutation representing ShiftRows is

$$\begin{aligned}\sigma = {} & (17\ 29\ 25\ 21)(18\ 30\ 26\ 22)(19\ 31\ 27\ 23)(20\ 32\ 28\ 24) \\ & (33\ 41)(34\ 42)(35\ 43)(36\ 44)(37\ 45)(38\ 46)(39\ 47)(40\ 48) \\ & (49\ 53\ 57\ 61)(50\ 54\ 58\ 62)(51\ 55\ 59\ 63)(52\ 56\ 60\ 64)\end{aligned}$$

Letting $i_\ell = 4i - 4 + \ell$, we obtain, for $i = 1, \ldots, 16$, polynomials

$$\begin{aligned}
Y^{(r)}_{\sigma(i_1)} &= X^{(r)}_{i_1} X^{(r)}_{i_2} X^{(r)}_{i_4} + X^{(r)}_{i_1} X^{(r)}_{i_3} X^{(r)}_{i_4} + X^{(r)}_{i_2} X^{(r)}_{i_3} X^{(r)}_{i_4} + \\
&\qquad X^{(r)}_{i_2} X^{(r)}_{i_3} + X^{(r)}_{i_1} + X^{(r)}_{i_3} + X^{(r)}_{i_4} + 1 \\
Y^{(r)}_{\sigma(i_2)} &= X^{(r)}_{i_1} X^{(r)}_{i_2} X^{(r)}_{i_4} + X^{(r)}_{i_1} X^{(r)}_{i_3} X^{(r)}_{i_4} + X^{(r)}_{i_1} X^{(r)}_{i_3} + \\
&\qquad X^{(r)}_{i_1} X^{(r)}_{i_4} + X^{(r)}_{i_3} X^{(r)}_{i_4} + X^{(r)}_{i_1} + X^{(r)}_{i_2} + 1 \\
Y^{(r)}_{\sigma(i_3)} &= X^{(r)}_{i_1} X^{(r)}_{i_2} X^{(r)}_{i_4} + X^{(r)}_{i_1} X^{(r)}_{i_3} X^{(r)}_{i_4} + X^{(r)}_{i_2} X^{(r)}_{i_3} X^{(r)}_{i_4} + \\
&\qquad X^{(r)}_{i_1} X^{(r)}_{i_2} + X^{(r)}_{i_1} X^{(r)}_{i_3} + X^{(r)}_{i_1} + X^{(r)}_{i_3} \\
Y^{(r)}_{\sigma(i_4)} &= X^{(r)}_{i_2} X^{(r)}_{i_3} + X^{(r)}_{i_1} + X^{(r)}_{i_2} + X^{(r)}_{i_4}
\end{aligned}$$

MixColumnsSerial: Each column v of the state matrix has to be replaced by $M \cdot v$, where $M = \begin{pmatrix} 4 & 1 & 2 & 2 \\ 8 & 6 & 5 & 6 \\ B & E & A & 9 \\ 2 & 2 & F & B \end{pmatrix}$. This yields the equations

$$
\begin{aligned}
z_1^{(r)} &= y_3^{(r)} + y_{17}^{(r)} + y_{34}^{(r)} + y_{50}^{(r)} \\
z_2^{(r)} &= y_1^{(r)} + y_4^{(r)} + y_{18}^{(r)} + y_{35}^{(r)} + y_{51}^{(r)} \\
z_3^{(r)} &= y_1^{(r)} + y_2^{(r)} + y_{19}^{(r)} + y_{33}^{(r)} + y_{36}^{(r)} + y_{49}^{(r)} + y_{52}^{(r)} \\
z_4^{(r)} &= y_2^{(r)} + y_{20}^{(r)} + y_{33}^{(r)} + y_{49}^{(r)} .
\end{aligned}
$$

and further 60 equations of this type.

MixColumnsSerial: Each column v of the state matrix has to be replaced by $M \cdot v$, where $M = \begin{pmatrix} 4 & 1 & 2 & 2 \\ 8 & 6 & 5 & 6 \\ \mathtt{B} & \mathtt{E} & \mathtt{A} & 9 \\ 2 & 2 & \mathtt{F} & \mathtt{B} \end{pmatrix}$. This yields the equations

$$
\begin{aligned}
z_1^{(r)} &= y_3^{(r)} + y_{17}^{(r)} + y_{34}^{(r)} + y_{50}^{(r)} \\
z_2^{(r)} &= y_1^{(r)} + y_4^{(r)} + y_{18}^{(r)} + y_{35}^{(r)} + y_{51}^{(r)} \\
z_3^{(r)} &= y_1^{(r)} + y_2^{(r)} + y_{19}^{(r)} + y_{33}^{(r)} + y_{36}^{(r)} + y_{49}^{(r)} + y_{52}^{(r)} \\
z_4^{(r)} &= y_2^{(r)} + y_{20}^{(r)} + y_{33}^{(r)} + y_{49}^{(r)} .
\end{aligned}
$$

and further 60 equations of this type.

Final key addition: We have $c_i = z_i^{(32)} + k_i$ for $i = 1, \ldots, 64$.

Second Step: Find polynomials representing the fault equations.

Up to now, the fault equations involve S^{-1}, the inverse of the SBox.

Second Step: Find polynomials representing the fault equations.
Up to now, the fault equations involve S^{-1}, the inverse of the SBox.

Using the identification $\mathbb{F}_{16} \cong \mathbb{F}_2[x]/\langle x^4 + x + 1\rangle$, the map
$S^{-1} : \mathbb{F}_{16} \longrightarrow \mathbb{F}_{16}$ is given by the interpolation polynomial

$$\begin{aligned} f(y) = {} & (x^2+1) + (x^2+1)y + (x^3+x)y^2 + (x^3+x^2+1)y^3 + xy^4 \\ & + (x^3+1)y^5 + (x^3+1)y^7 + (x+1)y^9 + (x^2+1)y^{10} + (x^3+1)y^{11} \\ & + (x^3+x)y^{12} + (x+1)y^{13} + (x^3+x^2+1)y^{14} \end{aligned}$$

Instead of applying S^{-1}, we can now substitute the right-hand sides of the fault equations for y in this polynomial.

Third Step: Recall that the fault equations are of the form $E_{a,0}$:

$$a = \mathtt{D} \cdot (S^{-1}(\mathtt{C}\cdot(c_1 + k_1) + \mathtt{C}\cdot(c_5 + k_5) + \mathtt{D}\cdot(c_9 + k_9) + \mathtt{4}\cdot(c_{13} + k_{13}))$$
$$+ S^{-1}(\mathtt{C}\cdot(c'_1 + k_1) + \mathtt{C}\cdot(c'_5 + k_5) + \mathtt{D}\cdot(c'_9 + k_9) + \mathtt{4}\cdot(c'_{13} + k_{13})))$$

By subtracting the right-hand sides of $E_{a,0}, E_{a,1}, E_{a,2}, E_{a,3}$ we get three polynomials over $\mathbb{F}_{16}$. Proceeding similarly for $E_{b,i}, E_{c,i}, E_{d,i}$ we get 12 polynomials over $\mathbb{F}_{16}$.

Third Step: Recall that the fault equations are of the form $E_{a,0}$:

$$a = \mathtt{D} \cdot (S^{-1}(\mathtt{C}\cdot(c_1 + k_1) + \mathtt{C}\cdot(c_5 + k_5) + \mathtt{D}\cdot(c_9 + k_9) + \mathtt{4}\cdot(c_{13} + k_{13}))$$
$$+ S^{-1}(\mathtt{C}\cdot(c'_1 + k_1) + \mathtt{C}\cdot(c'_5 + k_5) + \mathtt{D}\cdot(c'_9 + k_9) + \mathtt{4}\cdot(c'_{13} + k_{13})))$$

By subtracting the right-hand sides of $E_{a,0}, E_{a,1}, E_{a,2}, E_{a,3}$ we get three polynomials over $\mathbb{F}_{16}$. Proceeding similarly for $E_{b,i}, E_{c,i}, E_{d,i}$ we get 12 polynomials over $\mathbb{F}_{16}$.

The coefficient polynomials of $1, x, x^2, x^3$ then yield **48 polynomials** over $\mathbb{F}_2$ representing the fault equations.

Third Step: Recall that the fault equations are of the form $E_{a,0}$:

$$a = \mathtt{D} \cdot (S^{-1}(\mathtt{C}\cdot(c_1 + k_1) + \mathtt{C}\cdot(c_5 + k_5) + \mathtt{D}\cdot(c_9 + k_9) + \mathtt{4}\cdot(c_{13} + k_{13}))$$
$$+ S^{-1}(\mathtt{C}\cdot(c'_1 + k_1) + \mathtt{C}\cdot(c'_5 + k_5) + \mathtt{D}\cdot(c'_9 + k_9) + \mathtt{4}\cdot(c'_{13} + k_{13})))$$

By subtracting the right-hand sides of $E_{a,0}, E_{a,1}, E_{a,2}, E_{a,3}$ we get three polynomials over $\mathbb{F}_{16}$. Proceeding similarly for $E_{b,i}, E_{c,i}, E_{d,i}$ we get 12 polynomials over $\mathbb{F}_{16}$.

The coefficient polynomials of $1, x, x^2, x^3$ then yield **48 polynomials** over $\mathbb{F}_2$ representing the fault equations.

Most of these polynomials have degree 2 and 50-170 terms. After substituting plaintext-ciphertext pairs, usually a few of them become linear (with about 15-20 terms).

Experiments and Timings

The polynomial system consisting of the polynomials describing the encryption and the fault polynomials was solved using the SAT-solvers **MiniSat 2.2** (MS) and **Cryptominisat 2.9.3** (CMS) on an 8 core Xeon processor running at 3.5 GHz. The following timings are averages of 10 runs using random plaintexts, keys and faults.

Solver	MS (1 thread)	CMS (1 thread)	CMS (4 threads)
AVG (sec / h)	83947 / 23.31	48961 / 13.60	20859 / 5.79

Experiments and Timings

The polynomial system consisting of the polynomials describing the encryption and the fault polynomials was solved using the SAT-solvers **MiniSat 2.2** (MS) and **Cryptominisat 2.9.3** (CMS) on an 8 core Xeon processor running at 3.5 GHz. The following timings are averages of 10 runs using random plaintexts, keys and faults.

Solver	MS (1 thread)	CMS (1 thread)	CMS (4 threads)
AVG (sec / h)	83947 / 23.31	48961 / 13.60	20859 / 5.79

This attack is slower than the direct fault attack, but may be applicable to more cryptosystems.

5 – Defences Against Fault Attacks

42.7% of all statistics

5 – Defences Against Fault Attacks

42.7% of all statistics
are made up on the spot.
(Statistics Anonymous)

5 – Defences Against Fault Attacks

42.7% of all statistics
are made up on the spot.
(Statistics Anonymous)

First Defence: **Never encrypt the same plaintext twice using the same key!**

5 – Defences Against Fault Attacks

42.7% of all statistics
are made up on the spot.
(Statistics Anonymous)

First Defence: Never encrypt the same plaintext twice using the same key!

• If the device is connected to the internet or a phone network, a new key can be generated for every encryption.

5 – Defences Against Fault Attacks

42.7% of all statistics
are made up on the spot.
(Statistics Anonymous)

First Defence: Never encrypt the same plaintext twice using the same key!

- If the device is connected to the internet or a phone network, a new key can be generated for every encryption.
- **(Serial Re-Keying)** For offline devices, a **session key** can be generated incrementally using a counter. (For instance, the application transaction counter on a bank card **(ATC)** is used in this way.)

Second Defence: Change your keys regularly!

- Compute session keys from a master key **(parallel re-keying)** or use a key schedule.
- The fault attack may also target the key generation algorithm.

Second Defence: Change your keys regularly!

• Compute session keys from a master key **(parallel re-keying)** or use a key schedule.

• The fault attack may also target the key generation algorithm.

Third Defence: Detect fault attacks!

• Redo (part of) the computation and compare the results to detect faults. **(100% overhead!)**

• If a fault is detected, suppress the output of the ciphertext or output a modified, incorrect ciphertext.

Fourth Defence: Tolerate and correct faults!

- Use error correcting codes to remove faults that have been injected.
- If the error surpasses the error correction capacity of the code, output modified, incorrect ciphertext.

Fourth Defence: Tolerate and correct faults!

• Use error correcting codes to remove faults that have been injected.

• If the error surpasses the error correction capacity of the code, output modified, incorrect ciphertext.

Fifth Defence: Protect your hardware!

• Cover the chip with embedded sensors and then resin. If the sensors detect a physical attack, they disable the chip.

• Drive up the price tag of a physical attack.

Sixth Defence: Don't leak information!

• For passive side-channel attacks, one can measure and minimize the amount of information leaked.

• For active side-channel attacks (such as fault attacks) a lot depends on the capabilities of the attacker **(fault model)**, in particular his spacial and temporal resolution.

• Create a hardware and software design which maximizes the required resources of the attacker.

Sixth Defence: Don't leak information!

• For passive side-channel attacks, one can measure and minimize the amount of information leaked.

• For active side-channel attacks (such as fault attacks) a lot depends on the capabilities of the attacker **(fault model)**, in particular his spacial and temporal resolution.

• Create a hardware and software design which maximizes the required resources of the attacker.

Seventh Defence: Improve your algorithms!

• Include a security analysis against fault attacks in the algorithm design.

• Insert data into the computation which can be used by the system to protect against fault attacks.

6 – Solving Methods for Fault Equations

I think the next best thing to solving a problem

6 – Solving Methods for Fault Equations

I think the next best thing to solving a problem
is finding some humour in it.
(Frank A. Clark)

6 – Solving Methods for Fault Equations

I think the next best thing to solving a problem
is finding some humour in it.
(Frank A. Clark)

Algebraic attacks and algebraic fault attacks require the solution of a system of polynomial equations

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_m(x_1, \ldots, x_n) &= 0 \end{aligned}$$

with polynomials $f_1, \ldots, f_s \in K[x_1, \ldots, x_n]$ defined over a finite field K.

Special properties of the polynomial system:

Special properties of the polynomial system:

• The system has usually a **unique solution** or a small number of solutions.

Special properties of the polynomial system:

• The system has usually a **unique solution** or a small number of solutions.

• The solution of the system is defined over K.

Special properties of the polynomial system:

• The system has usually a **unique solution** or a small number of solutions.

• The solution of the system is defined over K.

• The field K is usually of **characteristic 2**, i.e. we have $K = \mathbb{F}_{2^e}$ with $e > 0$.

Special properties of the polynomial system:

- The system has usually a **unique solution** or a small number of solutions.
- The solution of the system is defined over K.
- The field K is usually of **characteristic 2**, i.e. we have $K = \mathbb{F}_{2^e}$ with $e > 0$.
- We may add the **field equations** $x_i^q - x_i = 0$ to the system, where $q = \#K$.

Special properties of the polynomial system:

• The system has usually a **unique solution** or a small number of solutions.

• The solution of the system is defined over K.

• The field K is usually of **characteristic 2**, i.e. we have $K = \mathbb{F}_{2^e}$ with $e > 0$.

• We may add the **field equations** $x_i^q - x_i = 0$ to the system, where $q = \#K$.

• In particular, if $K = \mathbb{F}_2$, we may reduce all terms in the polynomials f_i until they are **squarefree**.

Solving Using Gröbner Bases

If the system $f_1(x_1, \ldots, x_n) = \cdots = f_m(x_1, \ldots, x_n) = 0$ has a unique solution $(a_1, \ldots, a_n) \in K^n$, then the reduced Gröbner basis with respect to any term ordering of the ideal $I = \langle f_1, \ldots, f_m \rangle$ is given by

$$\{\, x_1 - a_1,\ x_2 - a_2,\ \ldots,\ x_n - a_n \,\}$$

Thus, in principle, it suffices to compute the Gröbner basis of I. Unfortunately, this becomes very hard if the number of indeterminates increases.

Solving Using Gröbner Bases

If the system $f_1(x_1, \ldots, x_n) = \cdots = f_m(x_1, \ldots, x_n) = 0$ has a unique solution $(a_1, \ldots, a_n) \in K^n$, then the reduced Gröbner basis with respect to any term ordering of the ideal $I = \langle f_1, \ldots, f_m \rangle$ is given by

$$\{\, x_1 - a_1,\ x_2 - a_2,\ \ldots,\ x_n - a_n \,\}$$

Thus, in principle, it suffices to compute the Gröbner basis of I. Unfortunately, this becomes very hard if the number of indeterminates increases.

Many variations and improvements have been suggested, for instance **mutants** or **linearisation**.

Solving Using Linear Algebra

Suppose that $K = \mathbb{F}_2$ and that the system has a unique solution $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$. Then we have

$$x_i - a_i \;=\; g_1 f_1 + \cdots + g_m f_m$$

for certain polynomials $g_1, \ldots, g_m \in \mathbb{F}_2[x_1, \ldots, x_n]$. This is called a **Hilbert Nullstellensatz certificate**. We may proceed as follows.

Solving Using Linear Algebra

Suppose that $K = \mathbb{F}_2$ and that the system has a unique solution $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$. Then we have

$$x_i - a_i \;=\; g_1 f_1 + \cdots + g_m f_m$$

for certain polynomials $g_1, \ldots, g_m \in \mathbb{F}_2[x_1, \ldots, x_n]$. This is called a **Hilbert Nullstellensatz certificate**. We may proceed as follows.

• Assume that $g_1, \ldots, g_m$ are all linear and write them down using indeterminate coefficients.

Solving Using Linear Algebra

Suppose that $K = \mathbb{F}_2$ and that the system has a unique solution $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$. Then we have

$$x_i - a_i \;=\; g_1 f_1 + \cdots + g_m f_m$$

for certain polynomials $g_1, \ldots, g_m \in \mathbb{F}_2[x_1, \ldots, x_n]$. This is called a **Hilbert Nullstellensatz certificate**. We may proceed as follows.

- Assume that $g_1, \ldots, g_m$ are all linear and write them down using indeterminate coefficients.
- Compare coefficients in $x_i = \sum_{j=1}^m g_j f_j$ and in $x_i + 1 = \sum_{j=1}^m g_j f_j$ and obtain two linear systems for the indeterminate coefficients.

Solving Using Linear Algebra

Suppose that $K = \mathbb{F}_2$ and that the system has a unique solution $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$. Then we have

$$x_i - a_i \;=\; g_1 f_1 + \cdots + g_m f_m$$

for certain polynomials $g_1, \ldots, g_m \in \mathbb{F}_2[x_1, \ldots, x_n]$. This is called a **Hilbert Nullstellensatz certificate**. We may proceed as follows.

- Assume that $g_1, \ldots, g_m$ are all linear and write them down using indeterminate coefficients.
- Compare coefficients in $x_i = \sum_{j=1}^m g_j f_j$ and in $x_i + 1 = \sum_{j=1}^m g_j f_j$ and obtain two linear systems for the indeterminate coefficients.
- If both linear systems have no solution, assume that all g_i have degree 2 and repeat the procedure. Then try degree 3, etc.

This algorithm is called **NLA algorithm**.

The worst-case complexity is exponential, but in practice, the Hilbert Nullstellensatz certificate has frequently a low degree.

This algorithm is called **NLA algorithm**.

The worst-case complexity is exponential, but in practice, the Hilbert Nullstellensatz certificate has frequently a low degree.

Problem:

The size of the linear system of equations increases rapidly.

Special **sparse linear algebra** techniques have to be used.

Solving Using Integer Programming

For simplicity, let us assume that $K = \mathbb{F}_2$. We are looking for a tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ such that

$$\begin{aligned} F_1(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \\ &\vdots \\ F_m(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \end{aligned}$$

where $F_1, \ldots, F_m \in \mathbb{Z}[x_1, \ldots, x_n]$ are **liftings** of the polynomials $f_1, \ldots, f_m \in \mathbb{F}_2[x_1, \ldots, x_n]$.

Solving Using Integer Programming

For simplicity, let us assume that $K = \mathbb{F}_2$. We are looking for a tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ such that

$$\begin{aligned} F_1(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \\ &\vdots \\ F_m(a_1, \ldots, a_n) &\equiv 0 \pmod 2 \end{aligned}$$

where $F_1, \ldots, F_m \in \mathbb{Z}[x_1, \ldots, x_n]$ are **liftings** of the polynomials $f_1, \ldots, f_m \in \mathbb{F}_2[x_1, \ldots, x_n]$.

Idea: Formulate these congruences as a system of linear equalities or inequalities over $\mathbb{Z}$ and solve it using an IP-solver.

Integer Liftings of Congruences: Suppose we are given a congruence

$$F(a_1, \ldots, a_n) \equiv 0 \pmod 2$$

with $F \in \mathbb{Z}[x_1, \ldots, x_n]$ and we are looking for solutions with $0 \leq a_i \leq 1$. For simplicity, let us assume that $\deg(F) = 2$. By reducing F modulo the field equations $x_i^2 + x_i$, we may assume that all terms in the support of F are square-free.

Integer Liftings of Congruences: Suppose we are given a congruence

$$F(a_1, \ldots, a_n) \equiv 0 \pmod 2$$

with $F \in \mathbb{Z}[x_1, \ldots, x_n]$ and we are looking for solutions with $0 \leq a_i \leq 1$. For simplicity, let us assume that $\deg(F) = 2$. By reducing F modulo the field equations $x_i^2 + x_i$, we may assume that all terms in the support of F are square-free.

(1) Let k be a new indeterminate. Form the inequality $k \leq \lfloor \#\operatorname{Supp}(F)/2 \rfloor$.

Integer Liftings of Congruences: Suppose we are given a congruence

$$F(a_1, \ldots, a_n) \equiv 0 \pmod 2$$

with $F \in \mathbb{Z}[x_1, \ldots, x_n]$ and we are looking for solutions with $0 \leq a_i \leq 1$. For simplicity, let us assume that $\deg(F) = 2$. By reducing F modulo the field equations $x_i^2 + x_i$, we may assume that all terms in the support of F are square-free.

(1) Let k be a new indeterminate. Form the inequality $k \leq \lfloor \, \#\,\mathrm{Supp}(F)/2 \, \rfloor$.

(2) For every term $x_i x_j$ in the support of F, introduce a new indeterminate y_{ij}. Let ℓ be the linear part of F. Form the equation $\sum_{i,j} y_{ij} + \ell - k = 0$.

Integer Liftings of Congruences: Suppose we are given a congruence

$$F(a_1, \ldots, a_n) \equiv 0 \pmod 2$$

with $F \in \mathbb{Z}[x_1, \ldots, x_n]$ and we are looking for solutions with $0 \leq a_i \leq 1$. For simplicity, let us assume that $\deg(F) = 2$. By reducing F modulo the field equations $x_i^2 + x_i$, we may assume that all terms in the support of F are square-free.

(1) Let k be a new indeterminate. Form the inequality $k \leq \lfloor \, \#\,\mathrm{Supp}(F)/2 \, \rfloor$.

(2) For every term $x_i x_j$ in the support of F, introduce a new indeterminate y_{ij}. Let ℓ be the linear part of F. Form the equation $\sum_{i,j} y_{ij} + \ell - k = 0$.

(3) Form the inequalities $x_i \leq 1$ and $y_{ij} \leq x_i$ and $y_{ij} \leq x_j$ and $y_{ij} \geq x_i + x_j - 1$.

Proposition 6.1 *The non-negative solutions of the system*

$$x_i \leq 1 \qquad y_{ij} \leq x_i \qquad y_{ij} \leq x_j$$

$$y_{ij} \leq x_i + x_j - 1 \quad k \leq \lfloor \, \# \operatorname{Supp}(F)/2 \, \rfloor \quad \sum_{i,j} y_{ij} + \ell - k = 0$$

correspond 1–1 to the solutions $(a_1, \ldots, a_n)$ *of the congruence* $F(a_1, \ldots, a_n) \equiv 0 \pmod 2$ *such that* $0 \leq a_i \leq 1$.

Proposition 6.1 *The non-negative solutions of the system*

$$x_i \le 1 \qquad y_{ij} \le x_i \qquad y_{ij} \le x_j$$

$$y_{ij} \le x_i + x_j - 1 \quad k \le \lfloor \, \#\,\mathrm{Supp}(F)/2 \, \rfloor \quad \sum_{i,j} y_{ij} + \ell - k = 0$$

correspond 1–1 to the solutions $(a_1, \ldots, a_n)$ *of the congruence* $F(a_1, \ldots, a_n) \equiv 0 \pmod 2$ *such that* $0 \le a_i \le 1$.

In this way (and many similar ways) we can translate the given polynomial system to a system of linear equalities and inequalities over $\mathbb{Z}$ whose non-negative solutions we can find using an IP-solver.

Solving Using SAT-Solvers

Suppose we are given an equality $f(x_1, \ldots, x_n) = 0$ with $f \in \mathbb{F}_2[x_1, \ldots, x_n]$. We want to find a **logical representation of f**, i.e. a (propositional) logical formula F (preferably in CNF) such that

$$f(x_1, \ldots, x_n) = 0 \qquad \text{iff} \qquad F(X_1, \ldots, X_n) = \texttt{true}$$

Solving Using SAT-Solvers

Suppose we are given an equality $f(x_1, \ldots, x_n) = 0$ with $f \in \mathbb{F}_2[x_1, \ldots, x_n]$. We want to find a **logical representation of f**, i.e. a (propositional) logical formula F (preferably in CNF) such that

$$f(x_1, \ldots, x_n) = 0 \qquad \text{iff} \qquad F(X_1, \ldots, X_n) = \texttt{true}$$

(1) The logical representation of $x_1x_2 + x_3$ is

$$(X_1 \vee \neg X_3) \wedge (X_2 \vee \neg X_3) \wedge (\neg X_1 \vee \neg X_2 \vee X_3)$$

Solving Using SAT-Solvers

Suppose we are given an equality $f(x_1, \ldots, x_n) = 0$ with $f \in \mathbb{F}_2[x_1, \ldots, x_n]$. We want to find a **logical representation of f**, i.e. a (propositional) logical formula F (preferably in CNF) such that

$$f(x_1, \ldots, x_n) = 0 \qquad \text{iff} \qquad F(X_1, \ldots, X_n) = \texttt{true}$$

(1) The logical representation of $x_1x_2 + x_3$ is

$$(X_1 \vee \neg X_3) \wedge (X_2 \vee \neg X_3) \wedge (\neg X_1 \vee \neg X_2 \vee X_3)$$

(2) The logical representation of $x_1 + x_2 + x_3$ is

$$(X_1 \vee X_2 \vee \neg X_3) \wedge (X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_2 \vee \neg X_3)$$

Solving Using SAT-Solvers

Suppose we are given an equality $f(x_1, \ldots, x_n) = 0$ with $f \in \mathbb{F}_2[x_1, \ldots, x_n]$. We want to find a **logical representation of f**, i.e. a (propositional) logical formula F (preferably in CNF) such that

$$f(x_1, \ldots, x_n) = 0 \qquad \text{iff} \qquad F(X_1, \ldots, X_n) = \texttt{true}$$

(1) The logical representation of $x_1x_2 + x_3$ is

$$(X_1 \vee \neg X_3) \wedge (X_2 \vee \neg X_3) \wedge (\neg X_1 \vee \neg X_2 \vee X_3)$$

(2) The logical representation of $x_1 + x_2 + x_3$ is

$$(X_1 \vee X_2 \vee \neg X_3) \wedge (X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee \neg X_2 \vee \neg X_3)$$

By combining **(1)** and **(2)**, we can find a logical representation of a polynomial system over $\mathbb{F}_2$.

The resulting set of CNF clauses can be tested for satisfiability using a SAT-solver.

The resulting set of CNF clauses can be tested for satisfiability using a SAT-solver.

The timings for the algebraic fault attack at LED-64 were obtained in this way.

The resulting set of CNF clauses can be tested for satisfiability using a SAT-solver.

The timings for the algebraic fault attack at LED-64 were obtained in this way.

Example 6.2 One full round of AES corresponds to **8704** polynomial equations in **4352** indeterminates.

A (slightly optimised) logical conversion consists of **367049** clauses in **49497** logical variables. The SAT-solver **MiniSat** was able to solve this set of clauses in **716 seconds**.

The resulting set of CNF clauses can be tested for satisfiability using a SAT-solver.

The timings for the algebraic fault attack at LED-64 were obtained in this way.

Example 6.2 One full round of AES corresponds to **8704** polynomial equations in **4352** indeterminates.

A (slightly optimised) logical conversion consists of **367049** clauses in **49497** logical variables. The SAT-solver **MiniSat** was able to solve this set of clauses in **716 seconds**.

Thank you for your attention!