New Low-Memory Algebraic Attacks on LowMC in the Picnic Setting

Fukang Liu[1], Willi Meier[4], Santanu Sarkar[5], Takanori Isobe[1,2,3]

[1]University of Hyogo, Japan

[2]NICT, Japan,

[3]PRESTO, Japan

[4]FHNW, Switzerland

[5]Indian Institute of Technology Madras, India

liufukangs@gmail.com

FSE 2023

The LowMC Primitive

- Proposed at Eurocrypt 2015
- Designed to be MPC/FHE/ZK-friendly
- Flexible parameters (affine layers, KSF, #S-boxes per round)

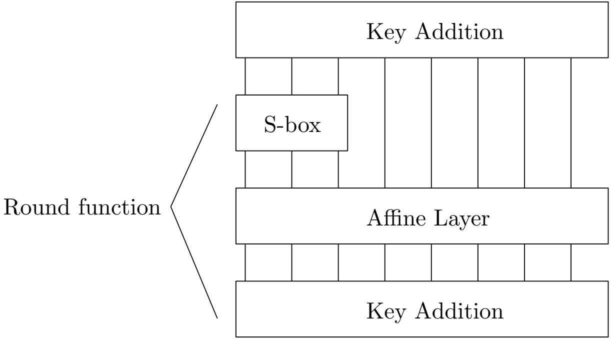


Figure: The round function of LowMC

The Picnic Setting

Problem

Given 1 known plaintext-ciphertext pair denoted by (p, c), how to recover the secret key k such that

$$c = \texttt{LowMC}(p, k)$$

- Extreme case
  - 1 S-box per round
- Picnic2
  - 10 S-boxes per round
- Picnic3
  - full S-box layer

Cryptanalysis of LowMC

- >3 chosen plaintext-ciphertext pairs
  - Higher-order differential attack (ICISC 2015)
  - Interpolation attack (Asiacrypt 2015)
- $=3$ chosen plaintext-ciphertext pairs
  - Difference enumeration attack (ToSC 2018)
- $=2$ chosen plaintext-ciphertext pairs (Security proof of Picnic)
  - Difference enumeration + algebraic method (CRYPTO 2021)
  - Algebraic MITM method (Asiacrypt 2022)
- $=1$ known plaintext-ciphertext pair (Security of Picnic)
  - Guess-and-determine (GnD) attack (ToSC 2020, Asiacrypt 2021)
  - Polynomial method (EUROCRYPT 2021)
  - Polynomial method + GnD (ToSC 2022)

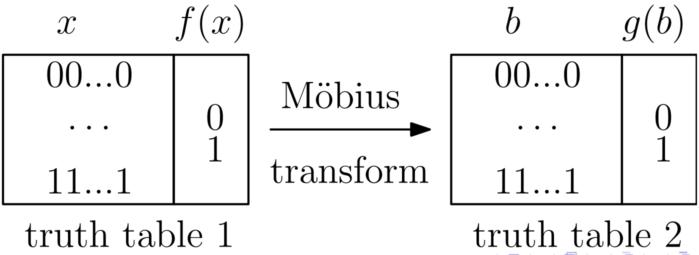
On Möbius Transform

Recovering the ANF

Given the truth table for a function $f(x) : \mathbb{F}_2^u \mapsto \mathbb{F}_2$, we can recover the Algebraic Norm Form (ANF) of

$$f(x) = \bigoplus_{b=(b_1,b_2,\ldots,b_u)\in\mathbb{F}_2^u} g(b) \prod_{i=1}^{u} x_i^{b_i},$$

i.e, recovering the truth table of $(b, g(b))$.



On Möbius Transform

Evaluating $f(x)$ over all $x \in \mathbb{F}_2^u$

Given the ANF of $f(x) : \mathbb{F}_2^u \mapsto \mathbb{F}_2$ of algebraic degree d, i.e. the truth table of $(b, g(b))$ is known, we can recover the truth table $(x, f(x))$ with the Möbius transform.

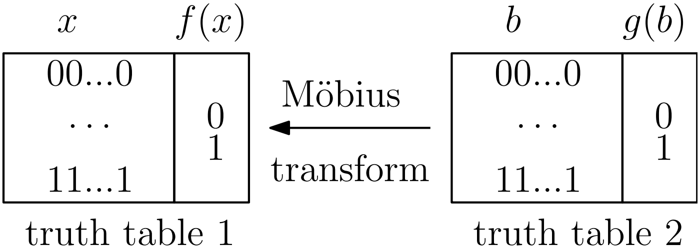


Figure: Evaluating a polynomial

On Möbius Transform

- standard Möbius transform:
  - time: $u \cdot 2^u$ bit operations.
  - memory: 2^u bits
- optimized Möbius transform (credit to Dinur):
  - time: $u \cdot 2^u$ bit operations
  - memory: $u \cdot \binom{u}{\leq d}$ bits (EUROCRYPT 2021)
- Evaluating a quadratic ($d = 2$) polynomial $f(x)$ with Gray code:
  - time: $u \cdot 2^u$ bit operations
  - memory: $\binom{u}{\leq 2}$ bits

On Crossbred Algorithm

Core idea 1

Given a Boolean polynomial $f(x)$, we aim to split $x = (x_1, \ldots, x_u)$ into two parts y, z of length $u - u_1$ and u_1, respectively, i.e.

$$\{y_1, \ldots y_{u-u_1}, z_1, \ldots, z_{u_1}\} = \{x_1, \ldots, x_u\}$$

such that $f(x)$ can be rewritten as

$$f(x) = \sum q_i(y)\ell_i(z)$$

where ℓ_i is a linear function in z. In this case, we simply say $f(x)$ is linear in z.

On Crossbred Algorithm

Core idea 2

Given m Boolean polynomial equations

$$f_1(x) = 0,\ f_2(x) = 0,\ \ldots,\ f_m(x) = 0$$

we aim to find a possible way to divide x into (y, z) such that m' polynomials $f_i(x)$ are linear in z.

In this way, we can exhaust all possible values of $y \in \mathbb{F}_2^{u-u_1}$ and solve the corresponding m' linear equations in z.

On Crossbred Algorithm for Quadratic Equation Systems

The original crossbred algorithm

Let

$$f_1(x) = 0,\ f_2(x) = 0,\ \ldots,\ f_m(x) = 0$$

be m quadratic Boolean equations in u variables.

For each f_i, we can generate some degree-3 and degree-4 equations:

$$x_j f_i(x) = 0,\ x_j x_k f_i(x) = 0.$$

Then, we obtain a much overdefined system of high-degree equations and expect to find as many linear equations in z from these equations by splitting x into y and z.

The simplified crossbred algorithm

Let

$$f_1(x) = 0,\ f_2(x) = 0,\ \ldots,\ f_m(x) = 0$$

be m quadratic Boolean equations in u variables where $m > u$.

Randomly choose u_1 variables such that

$$m \geq u_1 + \binom{u_1}{2}$$

and set them as z. Then, we can always expect to obtain

$$m - \binom{u_1}{2}$$

linear equations in z by eliminating all quadratic terms $z_i z_j$.

On Crossbred Algorithm for Quadratic Equation Systems

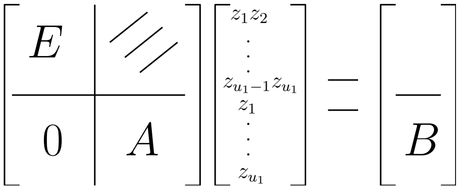
The simplified crossbred algorithm

In this way, we obtain the following equation system:

$$A \cdot (z_1, z_2, \ldots, z_{u_1})^T = B,$$

where **each element in A and B is linear and quadratic in y, resp.**

Finally, with the polynomial evaluation, traverse y over $\mathbb{F}_2^{u-u_1}$ and compute the corresponding matrices A and B. Solve the linear equation system in z and recover z.



Let

$$\epsilon + u_1 = m - u_1(u_1 - 1)/2, \epsilon > 0.$$

The total time complexity is

$$m^2 \cdot \binom{u}{\leq 2} + 2^{u-u_1} \cdot (u_1 + \epsilon) \cdot (u_1^2 + u_1 \cdot \epsilon + u)$$

bit operations.

On Dinur's Algorithm

Let

$$E(x): P_1(x) = P_2(x) = 0 = \ldots = P_m(x) = 0$$

be m Boolean equations in u variables and the degree is d.

The core idea:

1. Split x into $y \in \mathbb{F}_2^{u-u_1}$ and $z \in \mathbb{F}_2^{u_1}$.
2. Randomly pick $\ell = u_1 + 1$ equations from the m equations and denote them by

$$E_1(y, z): R_1(y, z) = R_2(y, z) = \cdots = R_\ell(y, z) = 0$$

3. Each solution to $E(x)$ must be a solution to $E_1(y, z)$, but the inverse does not hold. The goal is efficiently enumerate the solutions to $E_1(y, z)$ and check their correctness against $E(x)$.

On Dinur's Algorithm

Assumption

We assume that when the value of y is specified, there is at most 1 solution of z satisfying $E_1(y, z)$, and the corresponding (y, z) is called the isolated solution to $E_1(y, z)$.

[Reason: after y is specified, we have $\ell = u_1 + 1$ equations in u_1 variables.]

How to efficiently solve $E_1(x)$?

On Dinur's Algorithm

Polynomial method

Let

$$F_1(y,z) = (R_1(y,z) \oplus 1)(R_2(y,z) \oplus 1) \ldots (R_\ell(y,z) \oplus 1).$$

Then, $E_1(y,z)$ is equivalent to the following equation

$$F_1(y,z) = 1.$$

Hence, **the problem becomes how to enumerate all possible (y,z) such that $F_1(y,z) = 1$.**

On Dinur's Algorithm

New representations of $F_1(y, z)$ (similar to cube attack):

$$
\begin{aligned}
F_1(y, z) &= z_1 z_2 \dots z_{u_1} U_0(y) \oplus Q_0(y, z), \\
F_1(y, z) &= z_1 z_2 \dots z_{i-1} z_{i+1} \dots z_{u_1} U_i(y) \oplus Q_i(y, z) \text{ where } z_i = 0.
\end{aligned}
$$

Then, we have

$$
\begin{aligned}
U_0(y) &= \bigoplus_{z \in \mathbb{F}_2^{u_1}} F_1(y, z), \\
U_i(y) &= \bigoplus_{(z_1, z_2, \dots, z_{i-1}, z_{i+1}, \dots, z_{u_1}) \in \mathbb{F}_2^{u_1 - 1}, z_i = 0} F_1(y, z) \text{ where } 1 \leq i \leq u_1, \\
d_{U_0} &= Deg(U_0) \leq d_{F_1} - u_1, \\
d_{U_i} &= Deg(U_i) \leq d_{F_1} - u_1 + 1 \text{ where } 1 \leq i \leq u_1.
\end{aligned}
$$

On Dinur's Algorithm

Properties under the previous assumption

If

$$U_0(y) = 0,$$

there will be no solution to z.

If

$$U_0(y) = 1,$$

there is a solution to z and it can be computed as follows:

$$z_i = U_i(y) \oplus 1, \ i \in [1, u_1].$$

On Dinur's Algorithm

The overall procedure:

1. Find the ANFs of $U_i(y)$ where $i \in [0, u_1]$.
2. Evaluate $U_i(y)$ over all $y \in \mathbb{F}_2^{u-u_1}$ with the optimized Möbius transform.
3. For each obtained value of $U_i(y)$, use the above property to recover z and hence $x = (y, z)$ is known.
4. Check the correctness of $x = (y, z)$ against $E(x)$.

On Dinur's Algorithm

Costs:

- Costs in Step 1 to recover $U_i(y)$.
- Costs in Step 2 to evaluate the polynomials over all $y \in \mathbb{F}_2^{u-u_1}$.
- Amortize the costs to check the correctness by considering 4 such smaller systems: $E_1(y,z)$, $E_2(y,z)$, $E_3(y,z)$, $E_4(y,z)$.

Time complexity:

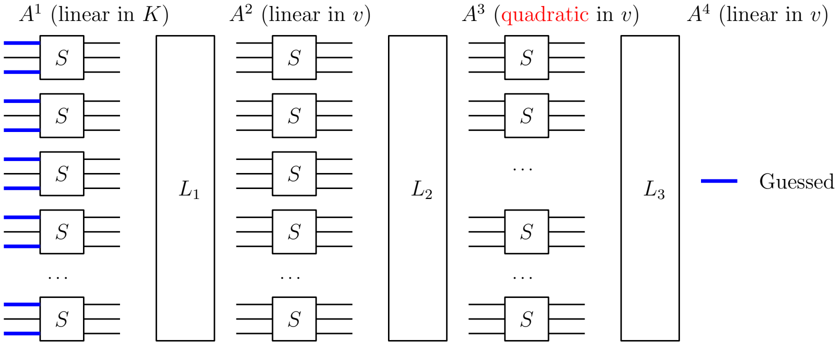
$$4 \cdot (2d \cdot \log_2 u \cdot 2^{u_1} \cdot \binom{u-u_1}{\leq d_{F_1} - u_1 + 1}) + 4 \cdot (u_1 + 1) \cdot (u - u_1) \cdot 2^{u-u_1}$$

Memory complexity:

$$4 \cdot (u_1 + 1) \cdot \binom{u-u_1}{\leq d_{F_1} - u_1 + 1}$$

Attack on 3-round LowMC:

- GnD + crossbred algorithm (m variables; $3m$ quadratic equations)

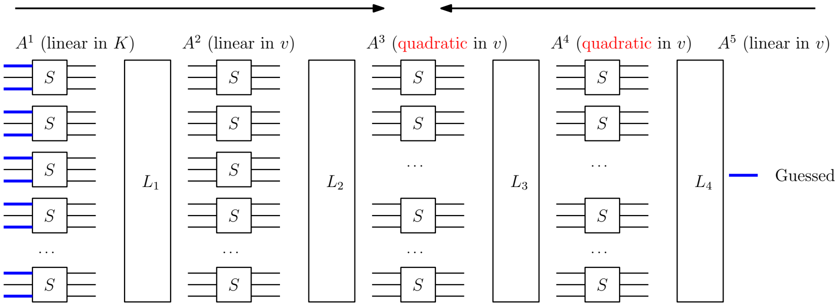


Results for 3-Round LowMC

Methods	n	k	s	r	Time	Memory
Fast exhaustive search	129	129	43	3	$2^{134.8}$	2^{21}
Dinur's algorithm					2^{125}	2^{104}
Our attack					$2^{127.2}$	$2^{16.9}$
Fast exhaustive search	192	192	64	3	$2^{197.9}$	$2^{22.7}$
Dinur's algorithm					2^{180}	2^{150}
Our attack					$2^{186.2}$	$2^{18.6}$
Fast exhaustive search	255	255	85	3	2^{261}	2^{24}
Dinur's algorithm					2^{235}	2^{197}
Our attack					$2^{246.8}$	$2^{19.8}$

Attack on 4-round LowMC:

- GnD + polynomial method (m variables; $14m$ degree-4 equations)

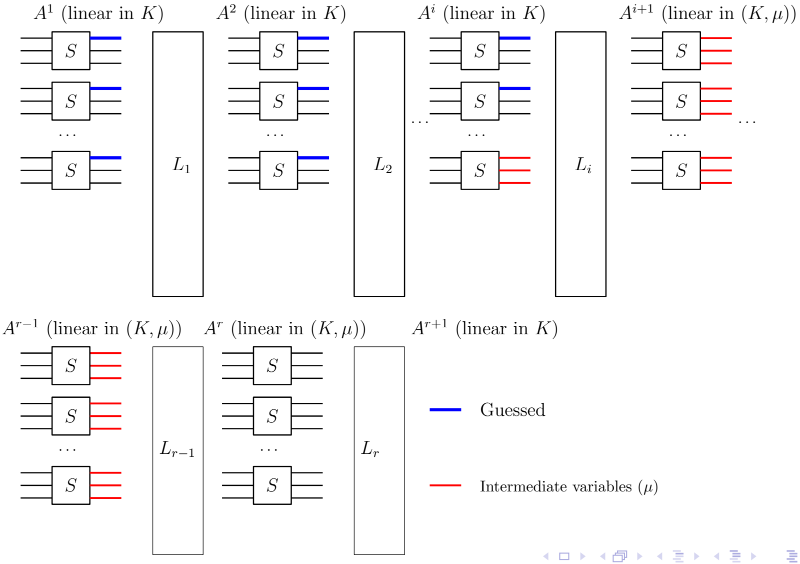


Results for 4-Round LowMC

Methods	n	k	s	r	Time	Memory
Fast exhaustive search	129	129	43	4	$2^{134.8}$	2^{21}
Dinur's algorithm					2^{130}	2^{113}
Our attack					$2^{133.8}$	$2^{36.7}$
Fast exhaustive search	192	192	64	4	$2^{197.9}$	$2^{22.7}$
Dinur's algorithm					2^{188}	2^{164}
Our attack					$2^{195.0}$	$2^{53.4}$
Fast exhaustive search	255	255	85	4	2^{261}	2^{24}
Dinur's algorithm					2^{245}	2^{218}
Our attack					$2^{255.8}$	$2^{68.0}$

Trivial time-memory trade offs for Dinur's algorithm:
Time: not higher than ours;
Memory: $> 2^{84.6}$, $> 2^{108.2}$ and $> 2^{134.2}$ for $k = 129, 192, 255$, resp.

Attack on LowMC with partial nonlinear layers:



Analyzing LowMC in the Picnic Setting (ToSC 2022)

Attack on LowMC with partial nonlinear layers:

- GnD + crossbred algorithm (h variables; αh quadratic equations)
- Guess 1 quadratic equation $\rightarrow$ 3 quadratic equations
- intermediate variables $\rightarrow$14 quadratic equations per S-box

Linearization:

$$\begin{aligned} z_0 &= x_0 \oplus x_1x_2 = a^\star, \\ z_1 &= (x_1x_2 \oplus a^\star) \oplus x_1 \oplus (x_1x_2 \oplus a^\star)x_2 = a^\star \oplus x_1 \oplus a^\star x_2, \\ z_2 &= (x_1x_2 \oplus a^\star) \oplus x_1 \oplus x_2 \oplus (x_1x_2 \oplus a^\star)x_1 = a^\star \oplus x_1 \oplus x_2 \oplus a^\star x_1. \end{aligned}$$

3 additional quadratic equations:

$$\begin{aligned} z_0 &= x_0 \oplus x_1x_2 = a^\star, \\ x_0x_1 \oplus x_1x_2 &= x_1a^\star, \\ x_0x_2 \oplus x_1x_2 &= x_2a^\star. \end{aligned}$$

Results for Partial Nonlinear Layers

Methods	n	k	s	r	Time (#bit operations)	Time (#calls)	Memory (in bits)
MITM Our attack	128	128	1	128	2^{147} $2^{142.3}$	2^{125} $2^{120.3}$	2^{22} $2^{18.9}$
MITM Our attack	192	192	1	192	$2^{212.8}$ $2^{205.8}$	2^{189} $2^{182.1}$	2^{22} $2^{19.9}$
MITM Our attack	256	256	1	256	2^{278} $2^{268.7}$	2^{253} $2^{243.7}$	2^{22} $2^{20.5}$

Results for Partial Nonlinear Layers

Methods	n	k	s	r	Time (#bit operations)	Time (#calls)	Memory (in bits)
MITM Our attack	128	128	10	12	$2^{129.6}$ $2^{134.6}$	2^{111} $2^{116.0}$	2^{38} $2^{18.8}$
MITM Our attack	192	192	10	19	$2^{199.4}$ $2^{203.7}$	2^{179} $2^{183.2}$	2^{38} $2^{20.0}$
MITM Our attack	256	256	10	25	$2^{259.6}$ $2^{262.8}$	2^{238} $2^{241.2}$	2^{38} $2^{20.6}$

Summary

1. Efficient attacks on LowMC when memory is costly.
2. New guess strategies combined with advanced techniques to solve nonlinear equations
3. Can we improve the polynomial method for overdefined systems?

Thank you