On recent higher-order cryptanalysis techniques

Jean-Philippe Aumasson

FHNW, Switzerland
Agenda

Definitions: higher-order cryptanalysis, cube attacks, cube testers

Applications: Grain-128, Grain-v1, KATAN

Most recent developments: zero-sums and $k$-sums, application to Hamsi, Keccak, Luffa

Conclusion: how to resist higher-order cryptanalysis?
Definitions

=Google(‘‘higher-order differential’’) (???)
Higher-order cryptanalysis (1/2)

Differential cryptanalysis based on **order-1** derivatives:

\[
\text{Ex: } E_k(m) \oplus E_k(m \oplus \Delta)
\]

Higher-order differential cryptanalysis: based on derivatives of **order \( \geq 2 \)**

Order-\( d \) derivative with respect to \( d \) bits is the sum of all \( 2^d \) outputs obtained by varying these \( d \) bits

\[
\frac{\partial^d f}{\partial x_1 \ldots \partial x_d} = \sum_{(x_1, \ldots, x_d) \in \{0, 1\}^d} f(x)
\]

for some \( f : \{0, 1\}^n \to \{0, 1\}, \ d \leq n \)
Higher-order cryptanalysis (2/2)

Why can it work when classical differential cryptanalysis fails?

**Ex:** random \( f : \{0, 1\}^n \rightarrow \{0, 1\} \) of degree \( d \leq n \)

- Its first order derivative looks random
- Its order-\((d – 1)\) derivative is linear
- Its order-\(d\) derivative is a constant

Previous higher-order attacks called “integral cryptanalysis”, “Square attack”, “saturation attack”, etc.

Most recent and refined version: cube attacks/testers
Cube attacks/testers (1/8)

Previous discovery claimed by Vielhaber ("AIDA")...

ePrint 2007/413, 2009/402

[Dinur, Shamir; EUROCRYPT’09] [A., Dinur, Meier, Shamir; FSE’09]
Cube attacks/testers (2/8)

Cube attacks = key-recovery attacks (need a secret)

Offline phase (precomputation)

- Search for public variables (IV, plaintext) whose derivative is a linear combination of key bits
- Linearity detected via probabilistic testing
- Bit-per-bit reconstructions of equations

Online phase

- Evaluate each linear equation detected during precomputation
- Solve the linear system obtained
Cube attacks/testers (3/8)

Ex (coefficient of the max-degree monomial):

\[ f(x_1, x_2, x_3, x_4) = x_1 + x_3 + x_1 x_2 x_3 + x_1 x_2 x_4 = x_1 + x_3 + x_1 x_2 x_3 + x_1 x_2 x_4 + 0 \times x_1 x_2 x_3 x_4 \]

Sum over all values of \((x_1, x_2, x_3, x_4)\):

\[ f(0, 0, 0, 0) + f(0, 0, 0, 1) + f(0, 0, 1, 0) + \cdots + f(1, 1, 1, 1) = 0 \]

= order-4 derivative
Cube attacks/testers (4/8)

Ex (evaluation of linear combination):

\[ f(x_1, x_2, x_3, x_4) = x_1 + x_3 + x_1 x_2 x_3 + x_1 x_2 x_4 \]
\[ = x_1 + x_3 + x_1 x_2 (x_3 + x_4) \]

Fix \( x_3 \) and \( x_4 \), sum over all values of \((x_1, x_2)\):

\[ \sum_{(x_1, x_2) \in \{0,1\}^2} f(x_1, x_2, x_3, x_4) = 4 \times x_1 + 4 \times x_3 + 1 \times (x_3 + x_4) \]
\[ = x_3 + x_4 \]

= order-2 derivative
Cube attacks/testers (5/8)

\( x_3 \) and \( x_4 \) fixed and unknown

\( f(\cdot, \cdot, x_3, x_4) \) queried as a **black box**

**ANF unknown**, except: \( x_1 x_2 \)’s superpoly is \((x_3 + x_4)\)

\[
f(x_1, x_2, x_3, x_4) = \cdots + x_1 x_2 (x_3 + x_4) + \cdots
\]

Query \( f \) to evaluate the superpoly:

\[
\sum_{(x_1, x_2) \in \{0,1\}^2} f(x_1, x_2, x_3, x_4) = x_3 + x_4
\]
Key recovery attack on a stream cipher

\( f : (k, v) \mapsto 1\text{st keystream bit:} \)

**Offline:** find cubes with linear superpolys

\[
\begin{align*}
  f(k, v) &= \cdots + v_1 v_3 v_5 v_7 (k_2 + k_3 + k_5) + \cdots \\
  f(k, v) &= \cdots + v_1 v_2 v_6 v_8 v_{12} (k_1 + k_2) + \cdots \\
  \cdots &= \cdots \\
  f(k, v) &= \cdots + v_3 v_4 v_5 v_6 (k_3 + k_4 + k_5) + \cdots
\end{align*}
\]

**Online:** evaluate the superpolys, solve the system
Cube testers = distinguishers

Detect a structure in the derivative which is not expected for an ideal algorithm

Ex: linearity, low degree, sparsity, imbalance

Compared to cube attacks
  ▶ At least as powerful (wrt # rounds attacked)
  ▶ Need less precomputation
  ▶ Do not require linear or low-degree derivative
Cube attacks/testers (8/8)

Problem: finding good sets of public variables (bottleneck)

**Analytical** approach:
- Analyze internals of the algorithm to determine variables with “lesser” interaction in the computation
  - Ex: study of recurrence relations in Luffa by Hatano and Watanabe

**Empirical** approach:
- Use tools such a discrete optimization algorithms
  - Ex: genetic algorithms for attacking Grain-128...

In practice, combine the two approaches
Applications

Why haven't cube attacks broken anything?

The talk and the paper

Hundreds of cryptographers were sitting in a dark lecture room at the University of California at Santa Bar "How to solve it: new techniques in algebraic cryptanalysis."

Shamir had already advertised his talk as introducing "cube attacks," a powerful new attack technique that describing a stream cipher with an extremely large key, many S-boxes, etc. David Wagner later wrote that laugh -- since it seemed ridiculous to imagine an attack on the design, yet I knew if he was describing this...
Grain-128 (1/2)

State-of-the-art stream cipher developed within ECRYPT’s eSTREAM Project (04-08)

- Designed by Hell, Johansson, Maximov, Meier (2007)
- 128-bit version of the eSTREAM cowinner Grain-v1
- 128-bit key, 96-bit IV, 256-bit state
- Previous DPA and related-key attacks
- Standard-model attack on 192-round version (of 256)
Grain-128 (2/2)

\[
\begin{align*}
\text{deg } f &= 1, \quad \text{deg } g = 2, \quad \text{deg } h = 3 \\
\text{Initialization: key in NFSR, IV in LFSR, clock 256 times} \\
\text{Then 1 keystream bit per clock}
\end{align*}
\]
Cube testers on Grain-128 (1/4)

[A., Dinur, Henzen, Meier, Shamir; SHARCS’09]

**Method:**

1. Select $n$ variables (IV bits)
2. Set the remaining IV bits to zero
3. Set the key bits randomly
4. Run Grain-128 for all $2^n$ values to evaluate derivative
5. Repeat steps 3-4 $N$ times and make statistics

Try to detect **imbalance**

Ex: if derivatives look like $x_0x_1x_2 + x_1x_2x_3x_4x_5$
Cube testers on Grain-128 (2/4)

Hardware implementation:

- Xilinx Virtex-5 FPGA
- 256 instances of $32 \times$ Grain-128 in parallel
- Efficient VHDL implementation of cube testers
- Attacks involving more than $2^{54}$ clocks in $\approx 1$ day
Cube testers on Grain-128 (3/4)

Bitsliced C program:

- Run 64 instances of Grain-128 in parallel
- Used for parameters optimization (evolutionary algos)

```c
u64 grain88_bitsliced64(u64 * key, u64 * lv, int rounds)
{
    u64 l[80+rounds], n[80+rounds], z=0;
    int i;

    /* initialize registers */
    for(i=0; i<64; i++)
    {
        n[3]= key[i];
        l[3]= lv[i];
    }

    for(i=64; i<80; i++)
    {
        n[3]= key[i];
        l[3]= 0xFFFFFFFFFULL;
    }

    for(i=0; i<rounds; i++)
    {
        /* clock */


        l[1+40] = z; n[1+88] = z;
    }

    /* return 1 keystream bit */

    return z;
}
```
Cube testers on Grain-128 (4/4)

Distinguisher for 237 rounds (of 256) in $2^{40}$

Extrapolation:

Suggests existence of distinguishers in $2^{77}$

$\Rightarrow$ 128-bit security unlikely
The case of Grain-v1

eSTREAM cowinner, original version of Grain
2×80-bit state, 160 initialization rounds
Seems to resist cube testers (81 rounds in $2^{24}$), why?

▶ NFSR feedback of degree 6 (vs. 2 for Grain-128)
▶ Filter function of degree 3 (vs. 2)
▶ Denser feedback and filter functions
▶ Shorter feedback delay (16 vs. 32)
▶ Smaller registers (80 vs. 128)
⇒ converges faster towards ideal ANF
The case of Grain-v1

eSTREAM cowinner, original version of Grain
2×80-bit state, 160 initialization rounds
Seems to resist cube testers (81 rounds in $2^{24}$), why?

- NFSR feedback of degree 6 (vs. 2 for Grain-128)
- Filter function of degree 3 (vs. 2)
- Denser feedback and filter functions
- Shorter feedback delay (16 vs. 32)
- Smaller registers (80 vs. 128)

⇒ converges faster towards ideal ANF
The case of KATAN (1/2)

[De Cannière, Dunkelman, Knezevic; CHES’09]

Lightweight block cipher

80-bit key; 32-, 48-, or 64-bit blocks

Compact in HW: 802 GE, for 6.25 GE/flip-flop (KATAN-32)
The case of KATAN (2/2)

NFSR’s properties:

- Degree-2 and sparse feedback function
- Short feedback delay (3)

Paper: “after 160 rounds, the degree of each internal state bit can reach 32” (for KATAN-32)

Our observations: degree 20 reached after 55 rounds, thus degree-32 probably reached after 87 rounds only!
The case of KATAN (2/2)

NFSR’s properties:
  ▶ Degree-2 and sparse feedback function
  ▶ Short feedback delay (3)

Paper: “after 160 rounds, the degree of each internal state bit can reach 32” (for KATAN-32)

Our observations: degree 20 reached after 55 rounds, thus degree-32 probably reached after 87 rounds only!

⇒ sparse and degree-2 function okay when feedback delay is short.

... but combinatorial logic is cheap (a few NAND’s), while memory (FSR’s) is expensive in hardware.

⇒ better increase degree and density as a safety net?
Most recent developments

[A., Knudsen, Meier]
Consider a permutation $F : \{0, 1\}^n \to \{0, 1\}^\ell$

**k-sum**: set $\{x_1, \ldots, x_k\}$ such that

$$\bigoplus_{i=1}^{k} F(x_i) = 0$$

Generalized birthday attack in time and space

$$O \left( k \cdot 2^{\ell/(1+\log k)} \right)$$
Zero-sums (1/2)

Consider a permutation $P : \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell$

**Zero-sum**: set \{x_1, \ldots, x_k\} such that

$$\bigoplus_{i=1}^{k} x_i = \bigoplus_{i=1}^{k} P(x_i) = 0$$

Generic probabilistic methods:
- Generalized birthday attack
- XHASH attack (linear algebra)

How to exploit the algebraic structure to find zero-sums?
Zero-sums (2/2)

Cube attack and $k$-sums need

\[ \text{low degree} \]

Zero-sums need

\[ \text{low degree} \quad \text{low degree} \]

Inside-out strategy

- Fix state in the middle, vary $k$ bits
- If degree $< k$ for both halves, $2^k$ values sum to zero

Need only evaluate at most half the algorithm
Application to Keccak (1/3)

- Second-round SHA-3 candidate
- 1600-bit state
- 18 nonidentical rounds
Application to Keccak (1/3)

- Second-round SHA-3 candidate
- 1600-bit state
- 18 nonidentical rounds
- One round has degree 2
- One inverse round has degree 3

10 rounds: degree upper bound $2^{10} = 1024$ (suboptimal)
⇒ higher-order distinguisher in $2^{1024}$

13 rounds: degree upper bound $2^{13} \gg 1599$ (optimal)
∉ higher-order distinguisher
Application to Keccak (2/3)

Consider 257 variables in the state after 5 rounds
- Preimage = degree-243 mapping
- Image = degree-256 mapping
- Compute order-257 derivative in both directions

<table>
<thead>
<tr>
<th>5 rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>degree ( \leq 3^5 = 243 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8 rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>degree ( \leq 2^8 = 256 )</td>
</tr>
</tbody>
</table>

Obtain \((x_1, \ldots, x_{2^{257}})\) such that \(\bigoplus_{i=1}^{2^{257}} x_i\) is the order-257 derivative of a degree-256 polynomial: must be \textbf{zero}

\(\Rightarrow\) zero-sum on 13 rounds in \(2^{257} \times \) first 5 rounds
Application to Keccak (3/3)

Optimizations: exploit structure of the inverse permutation

<table>
<thead>
<tr>
<th>#rounds</th>
<th>complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>$2^{17}$</td>
</tr>
<tr>
<td>10</td>
<td>$2^{60}$</td>
</tr>
<tr>
<td>12</td>
<td>$2^{128}$</td>
</tr>
<tr>
<td>14</td>
<td>$2^{256}$</td>
</tr>
<tr>
<td>16</td>
<td>$2^{1024}$</td>
</tr>
</tbody>
</table>

(18 rounds in full version)

Tweak for the second round: #rounds set to 24, rate modified
Application to Luffa

- Second-round SHA-3 candidate
- AND/XOR algorithm (like Keccak)
- Tweaked for the second round

Q permutation of Luffa (256-bit)

- Distinguisher on full version (8 rounds) in $2^{81}$
- Distinguisher on 7 rounds in $2^{27}$
- Not relevant for the hash algorithm
Application to Hamsi (1/3)

- Second-round SHA-3 candidate
- Two main instances: **Hamsi-256** and Hamsi-512
- Serpent-like algorithm (4-bit Sbox + linear layer)

Davies-Meyer compression function

3 rounds (6 for the last compression)
Application to Hamsi (2/3)

Observations:

- 3 rounds have degree 3 only, instead of ideally 27 (with respect to carefully chosen variables)
- Distribution of monomials and binomials is sparse
Application to Hamsi (2/3)

Observations:

- 3 rounds have degree 3 only, instead of ideally 27 (with respect to carefully chosen variables)
- Distribution of monomials and binomials is sparse

Consequences:

16-, 8-, 4-sums can be found efficiently

Example found for the default IV of Hamsi...

Zero-sums can be found efficiently for the permutation
Previous near collisions (or 2 sums):

- (256 – 25)-bit collision from 14 bit differences [Nikolic]
- (256 – 23)-bit collision from 16 bit differences [Wang et al.]
Previous near collisions (or 2 sums):

- (256 – 25)-bit collision from 14 bit differences [Nikolic]
- (256 – 23)-bit collision from 16 bit differences [Wang et al.]

We found a differential characteristic of probability $2^{-26}$

Consequence:

(256 – 25)-bit collision from 6 bit differences

Easier for the default IV than for a random one...
Conclusion
Conclusion

Higher-order methods are diverse, simple, powerful…

But only on certain designs, based on
- AND/XOR
- Small Sboxes

AXR and AES-based designs immune (even for low #rounds)

Recommendations for new designs
- If possible, use ADD (or other highly nonlinear op.)
- Maximum degree achieved with 25% of the #rounds
- Benchmark with cube testers
On recent higher-order cryptanalysis techniques

Jean-Philippe Aumasson

FHNW, Switzerland