Cube Testers and Key Recovery in Symmetric Cryptography

Willi Meier

n|w University of Applied Sciences Northwestern Switzerland
School of Engineering

Contents

- Describe a new type of algebraic cryptanalysis
- Not based on explicit algebraic description (Black Box Analysis)
- New applications to symmetric crypto systems of inherently low algebraic degree
- Joint work with J.P. Aumasson, I. Dinur, S. Fischer, L. Henzen, S. Khazaei and A. Shamir

Symmetric Crypto Systems

A few Notions

A classical crypto system consists of a parametrized family of transformations.

Let X denote the set of plaintexts, Y the set of ciphertexts. Then encryption is a transformation

$$E_z : X \longmapsto Y$$

with z as a parameter, where $z \in Z$, the set of secret keys.

Decryption is the inverse transformation

$$D_z : Y \longmapsto X.$$

Encryption transformation assumed to be known. Security rests solely on the secrecy of the key.

Several attack scenarios:

- Ciphertext-only: Opponent O knows a number of ciphertexts.
- Known plaintext: O knows pairs (x, y) of plaintexts x and corresponding ciphertexts y.
- Chosen plaintext: O has access to encryption machinery. He can choose plaintexts x and gets ciphertexts y.

Goal of opponent: To determine the secret key.

Condition for design of E: Solving for z in $E_z(x) = y$ for given (x, y) should be a mathematically complex problem.

Well known symmetric crypto systems:

- Block ciphers, e.g., DES, AES
- Stream Ciphers
- Message authentication codes
- Hash functions (have no key)

Stream Ciphers

A (deterministic) stream cipher is a map

$$S : \{0,1\}^n \times \{0,1\}^m \mapsto \{0,1\}^\ell$$

where the input is a pair (k, v), (k: secret key, v: a public initial vector) and that produces a (long) binary string, the keystream.

As in every symmetric crypto system, sender and receiver have to be in possession of the key k (e.g. of 128 bits).

Encryption: Plaintext string x is bitwise added mod 2 to the keystream to get ciphertext string y.

Decryption: Ciphertext string y is bitwise added mod 2 to the keystream to get plaintext x.

Prototype: One-Time-Pad

Keystream: A random binary string

OTP has perfect security.

In a deterministic stream cipher, random string replaced by pseudo random string.

Provable security lost.

Examples of stream ciphers

- RC4, used, e.g., in eBanking
- E0, used in the Bluetooth protocol
- A5/1, used in GSM cellphones

State-of-the-art stream ciphers include Salsa20, Rabbit for software, and Grain and Trivium for hardware.

Hash functions

Hash functions are essential building blocks for digital signatures.

A hash function h is a map

$$\{0,1\}^\star \mapsto \{0,1\}^n$$

of bit strings of arbitrary length to bit strings of length n. Hash functions are often iteratively constructed using compression functions. A compression function is a map

$$h : \{0,1\}^m \mapsto \{0,1\}^n,$$

where $m > n$.

A collision of h is a pair of strings (x, x'), $x \neq x'$, for which $h(x) = h(x')$.

A hash function is collision resistant, if it is ”infeasible” to find a collision (although, mathematically, collisions are abundant).

For any hash function, collision finding based on the birthday paradox can be applied:

Complexity $\approx 2^{n/2}$.

A hash function is ”broken”, if collisions faster than by birthday paradox can be found.

Likewise, a hash function is broken, if a preimage of h faster than with complexity 2^n can be found.

Well known hash functions are MD5 and SHA-1. Both are broken. For MD5, collisions have been found efficiently (Wang, 2005).

Cube attacks

Background

Solving large systems of multivariate polynomial equations over $GF(2)$ is known to be difficult:

Problem is NP-complete even if all equations are of degree only 2.

Best known method for solving this problem: Gröbner bases.

Method becomes inefficient for large number of unknowns, unless system is nonrandom.

Computational complexity hard to assess.

If number of equations is much larger than number of unknowns: Linearisation

For each monomial, a new variable is introduced and system solved by Gaussian elimination.

Observation: Many functions in cryptography come with a secret and a public parameter and are variants derived from a single polynomial.

Problem formalization

Consider a Boolean function

$$f : \{0,1\}^{n+m} \longmapsto \{0,1\},$$

$$f : (k, v) \mapsto z,$$

where k denotes a secret key, and v a public variable.

$k = (k_1, k_2, ..., k_n)$ and $v = (v_1, v_2, ..., v_m)$: Binary vectors of dimensions n and m.

Threat model: An adversary sends a public variable v of his choice to the oracle, and gets back the value (i.e., the output) z, according to a fixed unknown key k chosen by the oracle.

Goal: Determine the key efficiently (i.e., with computational complexity lower than exhaustive search over all 2^n values of k).

Cube attacks: the idea

Requirements of the attacker:

- only **black-box access** to the function
- negligible memory

Cube attacks work in 2 phases

- **precomputation**: chosen keys and chosen IVs
- **online**: fixed unknown key and chosen IVs

Observation 1

Computation of coefficient of monomial of largest degree

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= x_1 + x_3 + x_1x_2x_3 + x_1x_2x_4 \\ &= x_1 + x_3 + x_1x_2x_3 + x_1x_2x_4 + 0 \times x_1x_2x_3x_4 \end{aligned}$$

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

$$f(0,0,0,0)+f(0,0,0,1)+f(0,0,1,0)+\cdots+f(1,1,1,1) = 0$$

Observation 2

Evaluation of factor polynomials

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= x_1 + x_3 + x_1 x_2 x_3 + x_1 x_2 x_4 \\ &= x_1 + x_3 + x_1 x_2 (x_3 + x_4) \end{aligned}$$

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

$$\begin{aligned} \sum_{(x_1, x_2) \in \{0,1\}^2} f(x_1, x_2, x_3, x_4) &= 4 \times x_1 + 4 \times x_3 + 1 \times (x_3 + x_4) \\ &= x_3 + x_4 \end{aligned}$$

Observation 2

Evaluation of factor polynomials

$$f(x_1, x_2, x_3, x_4) \quad = \quad \cdots + x_1 x_2 (x_3 + x_4)$$

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

$$\sum_{(x_1, x_2) \in \{0,1\}^2} f(x_1, x_2, x_3, x_4) \quad = \quad x_3 + x_4$$

Terminology

$$f(x_1, x_2, x_3, x_4) = x_1 + x_3 + x_1 x_2 (x_3 + x_4)$$

$(x_3 + x_4)$ is called the **superpoly** of the **cube** $x_1 x_2$

Evaluation of a superpoly

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 cryptosystem with key k and public parameter v

$$f : (k, v) \mapsto \text{first keystream bit}$$

Offline: find cubes with linear superpolys

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

(reconstruct the superpolys with linearity tests)

Online: evaluate the superpolys, solve the system

Cube attacks (more formally)

Ignore distinction between secret and public variables.

Variables $x_1, ..., x_n$.

$p(x_1, ..., x_n)$ a multivariate polynomial of total degree d.

As $x_i^2 = x_i$ mod 2, monomials t_I in ANF of p can be identified with subset $I \subseteq \{1, ..., n\}$ of the variables x_i, $i \in I$, that are multiplied.

Given a polynomial p and a index subset I, can factor common monomial t_I out of some of the monomials in p:

Represent p as sum of monomials which are supersets of I, and monomials which are not supersets of I:

Superpoly

$$p(x_1, ..., x_n) \equiv t_I \cdot p_{S(I)} + q(x_1, ..., x_n).$$

$p_{S(I)}$: superpoly of I in p.

For any p and I, $p_{S(I)}$ is polynomial that does not contain a common variable with t_I, and each monomial in $q(x_1, ..., x_n)$ misses at least one variable from I.

A maxterm of p is a monomial t_I such that the degree of the superpoly $p_{S(I)}$ is 1, (i.e., linear, and not a constant).

Cubes

A subset I of size k defines k-dimensional binary cube of 2^k vectors C_I:

Assign all possible combinations of 0/1 values to variables in I. Leave all other variables undetermined.

Any vector $v \in C_I$ defines new derived polynomial $p_{|v}$ with $n - k$ variables.

Sum these derived polynomials over all 2^k vectors in C_I: New polynomial, denoted by

$$p_I = \sum_{v \in C_I} p_{|v}.$$

Determining the superpoly

For any polynomial p and subset I of variables I, $p_I \equiv p_{S(I)}$ mod 2.

Proof: Write $p(x_1, ..., x_n) = t_I \cdot p_{S(I)} + q(x_1, ..., x_n)$.

First case: Consider an arbitrary monomial t_J of $q(x_1, ..., x_n)$ (i.e., J is the subset containing the variable indexes that are multiplied in t_J).

t_J misses at least one of the variables in I. Hence it is added an even number of times: For the two values 0/1 of any of the missed variables, whereas all other values of the variables are the same. Thus it cancels mod 2 in $\sum_{v \in C_I} p_{|v}$.

Proof (contd.)

<u>Second case:</u> Consider polynomial $t_I \cdot p_{S(I)}$.

For all $v \in C_I$ the monomial t_I takes value 0, except for $v = (1, .., 1)$.

As the polynomial $p_{S(I)}$ has no variables with indexes in I, it is independent of the values that are summed over.

Hence $p_{S(I)}$ is summed only once, when t_I has value 1.

A consequence

Result states that the sum of the 2^k polynomials derived from the polynomial p by assigning all values to the k variables in I, eliminates all monomials, except those which are contained in the superpoly of I in p.

Summation reduces the total degree of p by at least k.

If t_I is any maxterm in p, this sum yields a <u>linear</u> equation in remaining variables.

In this procedure, only 0/1 values are added, not (huge) symbolic expressions.

Preprocessing Phase

Given an explicit description of polynomial p, splitting p into $p(x_1, ..., x_n) = t_I \cdot p_{S(I)} + q(x_1, ..., x_n)$ is feasible for any monomial t_I.

In Cryptography, no mathematical description of polynomial p is assumed. Instead, p is given as a black box polynomial:

$$p : (k, v) \longmapsto z = p(k, v)$$

Access of function values z for chosen public vector $v = (v_1, ..., v_m)$, and fixed unknown secret vector $k = (k_1, ..., k_n)$.

Assume total degree of p is known to be d.

Question: How to find $p_{S(I)}$ for given maxterm t_I, if p given as black box polynomial?

Solution: Use a separate preprocessing phase, in which both, public and secret variables are accessible.

Variables of superpoly $p_{S(I)}$ are secret, variables in set I are public.

Find linear superpoly

Let t_I be a maxterm in a black box polynomial p. Then:

1. Compute the constant in $p_{S(I)}$ by summing mod 2 the values of p over all inputs of the $n + m$ variables which are 0 everywhere, except on the $d - 1$ variables in the summation cube C_I.
2. Compute coefficient of k_j in linear expression $p_{S(I)}$ by summing mod 2 all values of p for input vectors which are 0 everywhere except on the summation cube C_I and all the values of p for input vectors which are 0 everywhere except on the summation cube and at k_j which ist set to 1.

Proof: In a linear expression, the coefficient of any variable k_j is 1 if and only if flipping the value of k_j flips the value of the expression. The constant is computed by setting all the variables to 0.

Cube attack: Complexity

Need about n linear equations to determine n unknowns k_j, $j = 1, ..., n$.

Assume black box polynomial has total degree d.

Generating each linear equation (linear superpoly) requires $2^{d-1}n$ computations.

If matrix determined by n equations is nonsingular, compute its inverse once. (Probability that matrix nonsingular: ≈ 0.3.)

Preprocessing complexity: $2^{d-1}n^2 + n^3$

Online complexity: $2^{d-1}n + n^2$.

An application

In practice, total degree d of black box polynomial unknown in advance, and polynomials often nonrandom.

Need linearity test to check whether superpoly is indeed linear (e.g., Blum-Luby-Rubinfeld test).

Stream cipher Trivium (reduced to 771 rounds):

Recover 80-bit key in $\approx 2^{36}$

Trivium is eSTREAM finalist, designed by De Cannière and Preneel in 2005.

Trivium

- 80-bit key and initial value IV (public)
- 3 quadratic NFSRs, of different lenghts
- 1152 initialization rounds before output is produced
- best practical attack on 771 rounds (cube attack)

Trivium (description)

Recall that a stream cipher is as a map

$$S : \{0,1\}^n \times \{0,1\}^m \mapsto \{0,1\}^\ell$$

In practice, this map is effected in two phases, and uses the mechanism of a state (of size at least $m + n$, due to time-memory-data tradeoffs):

- Initialization of a state
- Generation of output by state update and output function

In Trivium, $m = n = 80$.

State size is 288 bit.

Update function nonlinear, to counter algebraic attacks.

Output function is linear.

At each update, one output bit is produced.

Initialization of Trivium

$(s_1, s_2, ..., s_{93}) \leftarrow (k_1, ..., k_{80}, 0, 0, ..,)$
$(s_{94}, s_{95}, ..., s_{177}) \leftarrow (v_1, v_2, ..., v_{80}, 0., , , , 0)$
$(s_{178}, s_{179}, ..., s_{288}) \leftarrow (0, 0, ..., 0, 1, 1, 1)$
for $i = 1$ to $4 \cdot 288$ **do**

$t_1 \leftarrow s_{66} + s_{93}$

$t_2 \leftarrow s_{162} + s_{177}$

$t_3 \leftarrow s_{243} + s_{288}$

$t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$

$t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$

$t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$

$(s_1, s_2, ..., s_{93}) \leftarrow (t_3, s_1, ..., s_{92})$

$(s_{94}, s_{95}, ..., s_{177}) \leftarrow (t_1, s_{94}, ..., s_{176})$

$(s_{178}, ..., s_{288}) \leftarrow (t_2, s_{178}, ..., s_{287})$

end for

Output generation of Trivium

for $i = 1$ to ℓ **do**

$t_1 \leftarrow s_{66} + s_{93}$

$t_2 \leftarrow s_{162} + s_{177}$

$t_3 \leftarrow s_{243} + s_{288}$

$z_i \leftarrow t_1 + t_2 + t_3$

$t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$

$t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$

$t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$

$(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$

$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$

$(s_{178}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$

end for

Remarks

If in iterations, state variables $s_1, ..., s_{288}$ are expressed by $k_1, ..., k_{80}$ and $v_1, ..., v_{80}$, degree of polynomials increases only slowly.

System of equations in state variables for given output sequence $z_1, ..., z_\ell$ is of low degree for $\ell = 288$, and has only few nonlinear monomials.

Best attack on full Trivium for given output sequence by Maximov-Biryukov.

Involves guessing of certain state bits and products of state bits that reduce nonlinear system of equations to linear one.

Complexity: $c \cdot 2^{84}$ for some constant c.

Cube testers

Cube testers in brief

Like cube attacks:

- need only black-box access
- target primitives with secret and public variables and
- built on low-degree components

Unlike cube attacks:

- give **distinguishers** rather than key-recovery
- don’t require low-degree functions
- need **no precomputation**

Basic idea

Detect structure (nonrandomness) in the superpoly,

using **algebraic property testers**

A tester for property $\mathcal{P}$ on the function f:

- makes (adaptive) queries to f
- accepts when f satisfies $\mathcal{P}$
- rejects with bounded probability otherwise

Examples of efficiently testable properties

- balance
- linearity
- low-degree
- constantness
- presence of linear variables
- presence of neutral variables

General characterization by Kaufman/Sudan, *STOC' 08*

Superpolys attackable by testing...

...**low-degree** (6)

$$\cdots + x_1x_2x_3(x_5x_6 + x_7x_{21} + x_6x_9x_{20}x_{30}x_{40}x_{50}) + \cdots$$

...**neutral variables** (x_6)

$$\cdots + x_1x_2x_3x_4x_5 \cdot g(x_7, x_8, \ldots, x_{80}) + \cdots$$

...**linear variables** (x_6)

$$\cdots + x_1x_2x_3x_4x_5 \cdot (x_6 + g(x_7, x_8, \ldots, x_{80})) + \cdots$$

Results

MD6

Presented by Rivest at CRYPTO 2008

Submitted to the SHA-3 competition

- quadtree structure
- construction RO-indifferentiable
- low-degree compression function
- at least **80 rounds**
- best attack by the designers: 12 rounds

Compression function of MD6

$$\{0,1\}^{64\times 89} \mapsto \{0,1\}^{64\times 16}$$

Input: 64-bit words $A_0.A_1,\ldots,A_{88}$

Compute the A_i's with the recursion

$$\begin{aligned}
x &\leftarrow S_i \oplus A_{i-17} \oplus A_{i-89} \oplus (A_{i-18} \wedge A_{i-21}) \oplus (A_{i-31} \wedge A_{i-67}) \\
x &\leftarrow x \oplus (x \gg r_i) \\
A_i &\leftarrow x \oplus (x \ll \ell_i)
\end{aligned}$$

- round-dependent constant S_i
- quadratic step, at least 1280 steps

Results on MD6

Cube attack (key recovery)

- on the **14-round** compression function
- recover any 128-bit key
- in time $\approx 2^{22}$

Cube testers (testing balance)

- detect nonrandomness on **18 rounds**
- detect nonrandomness on **66 rounds** when $S_i = 0$
- in time $\approx 2^{17}$, 2^{24}, resp.

Cube testers on Trivium

Test the presence of **neutral variables**

Distinguishers (only choose IVs)

- 2^{24}: 772 rounds
- 2^{30}: 790 rounds

Nonrandomness (assumes some control of the key)

- 2^{24}: 842 rounds
- 2^{27}: 885 rounds

Full version: 1152 rounds

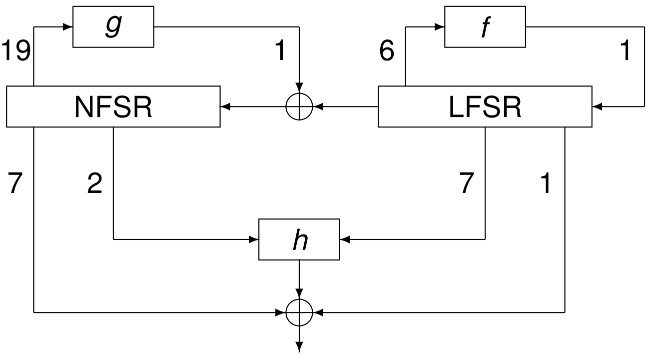
Grain-128

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 (2005)
- 128-bit key, 96-bit IV, 256-bit state
- previous DPA and related-key attacks
- standard-model attack on round-reduced version (192/256)

Grain-128

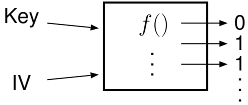


$\deg f = 1$, $\deg g = 2$, $\deg h = 3$

Initalization: key in NFSR, IV in LFSR, clock 256 times

Then 1 keystream bit per clock

Cube testers (simple version)



1. pick a random key and fix $(96 - n)$ IV bits
2. vary n IV bits to obtain the evaluation of **order-n derivative**

$$\bigoplus_{(x_0,\ldots,x_{n-1})\in\{0,1\}^n} f(x) = \frac{\partial^n f}{\partial x_0 \ldots \partial x_{n-1}}$$

for **well-chosen cube** (=variables), statistical bias detectable

ex: f of degree $n \Rightarrow$ constant derivative

How to determine variable bits?

Complexity bottleneck, and main distinction with previous high-order differential attacks

Analytically: find “weak” variables by analyzing the algorithm

Ex: Trivium

$$
\begin{aligned}
&t_1 \leftarrow s_{66} + s_{91} \cdot s_{92} + s_{93} + s_{171}\\
&t_2 \leftarrow s_{162} + s_{175} \cdot s_{176} + s_{177} + s_{264}\\
&t_3 \leftarrow s_{243} + s_{286} \cdot s_{287} + s_{288} + s_{69}\\
&(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})\\
&(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})\\
&(s_{178}, s_{279}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})
\end{aligned}
$$

Empirically: explore the search space to find good sets of variables with discrete optimization tools

Going against the Grain

Method:

1. select n variable IV bits
2. set the remaining IV bits to zero
3. set the key bits randomly
4. run Grain-128 for all the 2^n values and collect results
5. repeat steps 3-4 N times and make statistics

we try to detect for imbalance in the distribution of the results
e.g., if derivatives look like $x_0 x_1 x_2 + x_1 x_2 x_3 x_4 x_5$

Going against the Grain

Method:

1. select n variable IV bits
2. set the remaining IV bits to zero
3. set the key bits randomly
4. run Grain-128 for all the 2^n values and collect results
5. repeat steps 3-4 N times and make statistics

we try to detect for imbalance in the distribution of the results
e.g., if derivatives look like $x_0 x_1 x_2 + x_1 x_2 x_3 x_4 x_5$

Problem 1: finding good cubes/variables (SW: C code + `gcc *.c`)

Going against the Grain

Method:

1. select n variable IV bits
2. set the remaining IV bits to zero
3. set the key bits randomly
4. run Grain-128 for all the 2^n values and collect results
5. repeat steps 3-4 N times and make statistics

we try to detect for imbalance in the distribution of the results
e.g., if derivatives look like $x_0 x_1 x_2 + x_1 x_2 x_3 x_4 x_5$

Problem 1: finding good cubes/variables (SW: C code + `gcc *.c`)
Problem 2: implementing the attack (HW: VHDL + FPGA)

Software precomputation

Bitsliced implementation

- 64 instances in parallel with different keys and IVs
- tester using order-30 derivatives in $\approx$ 45min

Evolutionary algorithm

- generic discrete optimization tool
- search variables that maximize the number of rounds attackable
- huge search space, e.g. $\binom{96}{32} \geq 2^{84}$
- quickly converges into local optima

Cube dimension	6	10	14	18	22	26	30	...	**?**
Rounds	180	195	203	208	215	222	227	...	**256**

For larger cubes we shall need more computational power

Search for good cubes

Evolutionary algorithm: generic discrete optimization tool

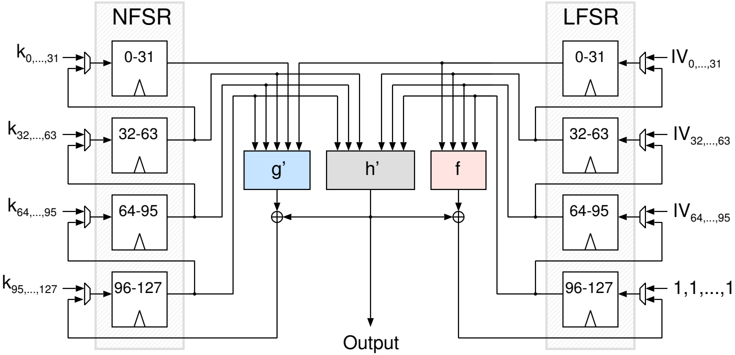
In a nutshell: population = subset of variables

1. initialize population pseudorandomly
2. reproduction (crossover + mutation)
3. selection of best fitting individuals
4. go to 2.

#generations (steps 2-4) before halting = parameter

Grain-128 in FPGA

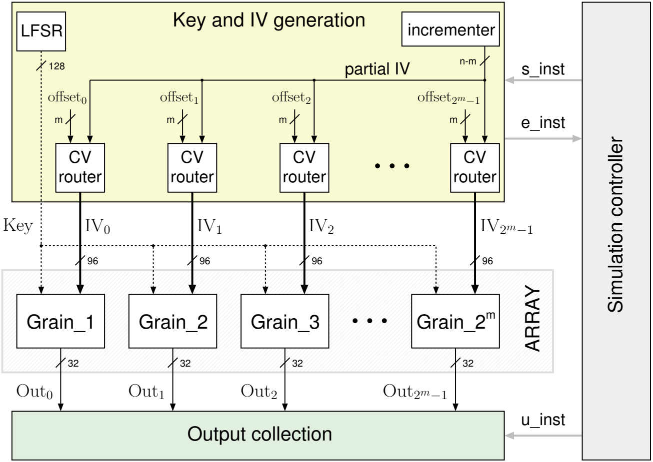
- 32× parallelization (32 cipher clocks/system clock)
- on **Xilinx Virtex-5 LX330**: 180 slices for 1 instance at 200 MHz
- 256 instances: 46080 slices, of available 51 840 slices available



Cube testers in FPGA

- exploit (almost) all the slices available
- 256 Grain-128 modules work on distinct IVs
- additional units to generate inputs and to store results
  - simulation controller
  - input generator
  - output collector
- evaluation of cubes for 32 consecutive rounds
- LSFR to generate keys efficiently

FPGA parallel cube tester core



Performance and results

- evaluation of $(n+8)$-dimensional cubes as fast as for n-dimensional cubes with a single instance
- approx. 10 seconds for a cube of degree 30 (64 runs)
- approx. 3 hours for a cube of degree 40 (64 runs)

Cube dimension	30	35	37	40	44	46	50
Nb. of queries	2^{22}	2^{27}	2^{29}	2^{32}	2^{36}	2^{38}	2^{42}
Time	0.17 sec	5.4 sec	21 sec	3 min	45 min	3 h	2 days

Performance and results

- evaluation of $(n + 8)$-dimensional cubes as fast as for n-dimensional cubes with a single instance
- approx. 10 seconds for a cube of degree 30 (64 runs)
- approx. 3 hours for a cube of degree 40 (64 runs)

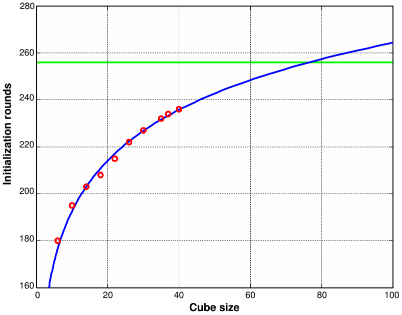
Cube dimension	30	35	37	40	44	46	50
Nb. of queries	2^{22}	2^{27}	2^{29}	2^{32}	2^{36}	2^{38}	2^{42}
Time	0.17 sec	5.4 sec	21 sec	3 min	45 min	3 h	2 days

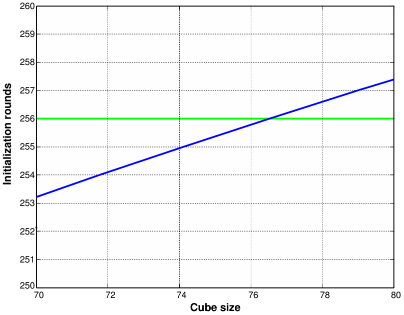
Found a **distinguisher on 237 rounds in** 2^{54} **clocks**

- #samples×#cipher clocks×#evaluations$= 64 \times 256 \times 2^{40} = 2^{54}$

Extrapolation

Logarithmic extrapolation with standard linear model





cubes of degree 77 conjectured sufficient for the **full Grain-128**

$\Rightarrow$ attack in 2^{83} initializations vs. 2^{128} ideally

Observations on Grain-v1

Differences:

- The size of the LFSR and the NFSR is 80-bit
- 80-bit keys, 64-bit IVs, and 160 initialization rounds
- Feedback polynomial of NFSR has degree six and is less sparse
- Filter function *h* is denser
- Algebraic degree and density converge faster towards ideal ones

Rounds	64	70	73	79	81
Cube dimension	6	10	14	20	24

Grain-v1 seems to resist cube testers and basic cube attack techniques

Conclusions

Cube attacks

- Generic algebraic cryptanalysis methods
- Differ from established algebraic attacks
- Cryptanalysis of simplified and full variants of well known stream ciphers, e.g., Trivium, Grain-128
- Seem applicable only for symmetric crypto systems with inherently low degree components

Cube testers

+

- more general than classical cube attacks
- no precomputation
- “polymorphic”
- first dedicated hardware for cube testers on Grain-128
- Grain-v1: much more resistent (higher degree of boolean function *g*)

–

- only gives distinguishers
- only finds feasible attacks
- relevant for a minority of functions (like cube attacks)

References

S. O'Neil: Algebraic structure defectoscopy. Cryptology ePrint Archive, Report 2007/378, 2007.

H. Englund, Th. Johansson, M. Sönmez Turan: A framework for chosen IV statistical analysis of stream ciphers, INDOCRYPT 2007, pp. 268-281.

S. Fischer, S. Khazaei, W. Meier: Chosen IV statistical analysis for key recovery attacks on stream ciphers. In AFRICACRYPT 2008, pp. 236-245.

M. Vielhaber. Breaking ONE.FIVIUM by AIDA an algebraic IV differential attack: Cryptology ePrint Archive, Report 2007/413, 2007.

S. Khazaei, W. Meier: New directions in cryptanalysis of self-synchronizing stream ciphers, INDOCRYPT 2008, pp. 15-26.

I. Dinur, A. Shamir: Cube Attacks on Tweakable Black Box Polynomials, EUROCRYPT 2009. Also on Cryptology ePrint Archive, Report 2008/385.

J.-P. Aumasson, I. Dinur, W. Meier, A. Shamir: Cube Testers and key Recovery Attacks On Reduced-Round MD6 and Trivium, FSE 2009, pp. 1-22.

J.-P. Aumasson, I. Dinur, L. Henzen, W. Meier, A. Shamir: Efficient FPGA Implementations of High-Dimensional Cube Testers on the Stream Cipher Grain-128, SHARCS 2009. Also on Cryptology ePrint Archive, Report 2009/218.

Open Problems

How to predict the asymptotic growth of degree of maxterm?

How to find the best cubes?