Efficient FPGA implementations of high-dimensional cube testers on the stream cipher Grain-128

Jean-Philippe Aumasson Itai Dinur *Luca Henzen*
Willi Meier Adi Shamir

SHARCS ’09

Agenda

Grain-128

Cube testers

Software precomputations

FPGA implementation

Results and extrapolation

Conclusions

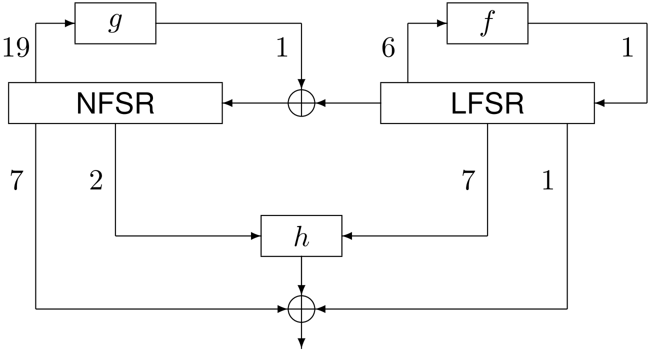
Grain-128

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

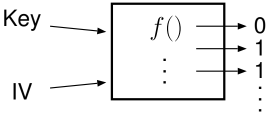


$\deg f = 1$, $\deg g = 2$, $\deg h = 3$

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

Comparison…

Cube attacks…

1. find 128 cubes whose order-n derivative has degree 1
2. recontruct their derivatives via black-box linearity tests
3. evaluate derivatives and solve linear system to recover the key

Cube testers…

- distinguishers rather than key-recovery
- need less precomputation than cube attacks
- don't require derivatives of degree-1, but with any unexpected and testable property

How to determine variable bits?

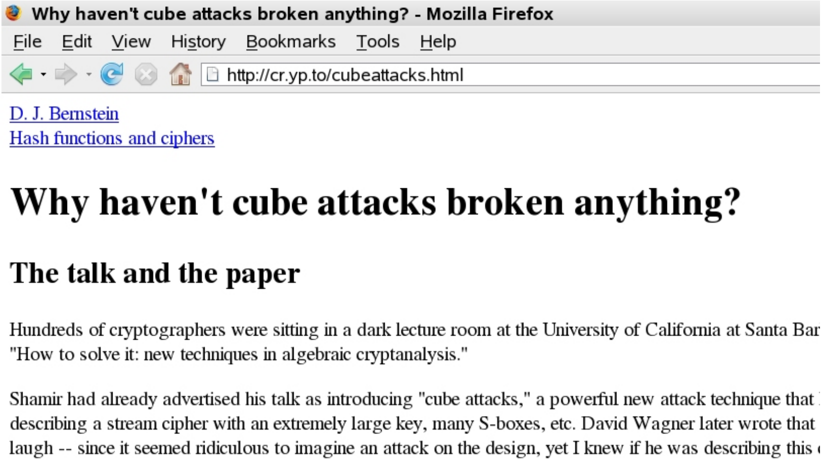
Complexity bottleneck, and main distinction with previous high-order differential attacks

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


Why haven't cube attacks broken anything? - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://cr.yp.to/cubeattacks.html

D. J. Bernstein
Hash functions and ciphers

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

What about cube testers?

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 $x_0x_1x_2 + x_1x_2x_3x_4x_5$

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 $x_0x_1x_2 + x_1x_2x_3x_4x_5$

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 $x_0x_1x_2 + x_1x_2x_3x_4x_5$

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 ≈ 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

Software precomputation

Bitsliced implementation

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

Evolutionary algorithm

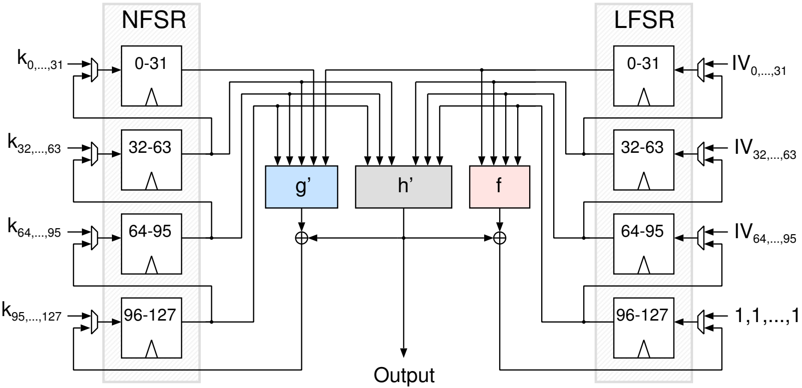
- generic discrete optimization tool
- search variables that maximize the number of rounds attackable
- huge search space, e.g. $\binom{96}{32} \geq 2^{84}$
- quickly converges into local optima

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

To evaluate larger cubes we need more computational power

Grain-128 in FPGA

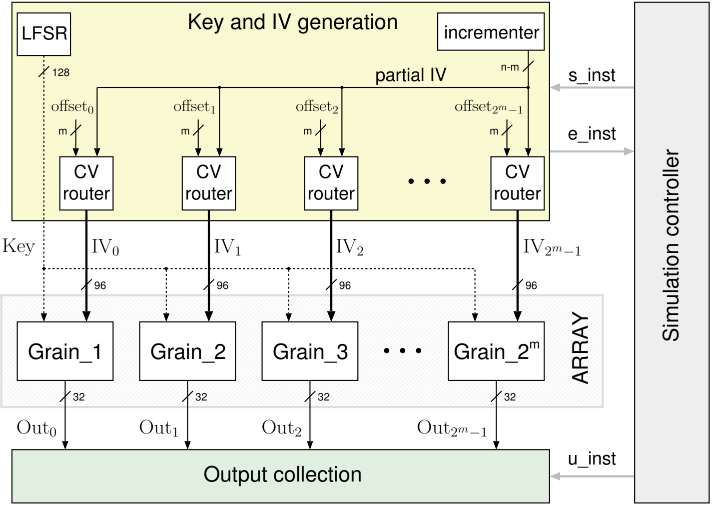
- 32$\times$ parallelization (32 cipher clocks/system clock)
- on **Xilinx Virtex-5 LX330**: 180 slices for 1 instance at 200 MHz
- 256 instances: 46080 slices, of available 51 840 slices available



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)

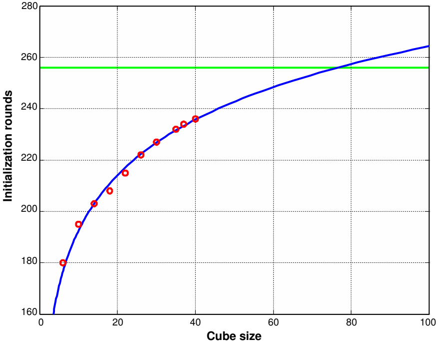
Cube dimension	30	35	37	40	44	46	50
Nb. of queries	2^{22}	2^{27}	2^{29}	2^{32}	2^{36}	2^{38}	2^{42}
Time	0.17 sec	5.4 sec	21 sec	3 min	45 min	3 h	2 days

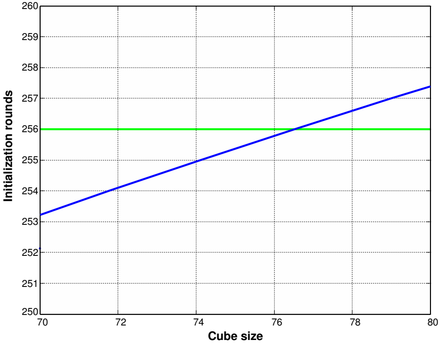
Found a **distinguisher on 237 rounds in 2^{54} clocks**

- #samples$\times$#cipher clocks$\times$#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

Conclusions

First dedicated hardware for cube attacks/testers

Cube attacks/testers seem to have eventually broken something

High variance of cubes' efficiency; preliminary discrete optimization step essential

Software experiments on Grain-v1: much more resistant (higher degree g)

The end

Thanks for your attention

Questions?

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

Key-recovery attacks

- Search for IV terms with linear superpoly in the key bits (or maxterms)
- Search for maxterms is difficult for reduced variants of Grain-128

- Key bits mix non-linearly together before mixing with the IV bits
- Output bits polynomials contain few IV terms whose superpoly is linear in the key bits
- Applying linearization techniques becomes a complicated task

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