Analysis and design of lightweight symmetric cryptographic algorithms

Jean-Philippe Aumasson

PhD defense

0. Introduction
1. Cryptanalysis
   - hash functions **MD5** and **HAVAL**
   - eSTREAM SW cowinner **Salsa20**
   - DTV standard **MULTI2**
   - SHA-3 candidate **MD6**
   - eSTREAM HW cowinners **Grain** and **Trivium**
2. Design of the SHA-3 candidate **BLAKE**
   - rationale and description
   - SW and HW performance
   - status in the SHA-3 competition
3. Conclusion

Symmetric crypto algorithms

Block ciphers: (key, plaintext) $\mapsto$ ciphertext

Stream ciphers: (key, nonce) $\mapsto$ keystream

Hash functions: ([key,] message) $\mapsto$ digest

MACs: (key, message) $\mapsto$ tag

Timeline (block stream hash)

1975: DES published (IBM/NSA)
1991: MD5 published (Rivest)
1991: DES broken (Biham-Shamir)
1993: SHA-0 published (NIST/NSA)
1994: RC4 published (Rivest)
1995: SHA-1 supersedes SHA-0
1998: SHA-0 broken (Chabaud-Joux)
2000: Rijndael chosen as the AES (Daemen-Rijmen)
2001: SHA-2 published (NIST/NSA)
2003: NESSIE recommends AES, Camellia, SHACAL
2003: NESSIE recommends Whirlpool, SHA-2
2003: NESSIE recommends none of the proposed designs
2004: MD5 broken (Wang et al.)
2005: SHA-1 broken (Wang et al.)
2008: eSTREAM cowinners announced
2008: eSTREAM cowinner F-FCSR broken
2008: RC4 broken (Maximov-Khovratovich)
2009: AES-192 and AES-256 broken (Biryukov-Khovratovich-Nikolic)
2012: selection of SHA-3

Compared lifetimes against shortcut attacks

DES: 1975-1991 (**16** years)
RC4: 1994-2008 (**14** years)
MD5: 1991-2004 (**13** years)
AES: 1998-2009 (**11** years)
SHA-1: 1995-2005 (**10** years)
SHA-0: 1993-1998 (**5** years)

$\Rightarrow$ **mean** of 11.5 years

- SHA-2 broken in 2014? realistic...
- SHA-3 broken in 2024?

Practical attacks only for MD5 and SHA-0 (and DES)

Compared lifetimes against shortcut attacks

DES: 1975-1991 (**16** years)
RC4: 1994-2008 (**14** years)
MD5: 1991-2004 (**13** years)
AES: 1998-2009 (**11** years)
SHA-1: 1995-2005 (**10** years)
SHA-0: 1993-1998 (**5** years)

$\Rightarrow$ **mean** of 11.5 years

- SHA-2 broken in 2014? realistic...
- SHA-3 broken in 2024?

Practical attacks only for MD5 and SHA-0 (and DES)

Need better understanding of algorithms' security, and better designs

What this thesis is about

Design of symmetric crypto algorithms

- simplicity (specs, understanding, implementation)
- security (simulate structureless transform)
- performance (implementability, tradeoff SW/HW, etc.)

Analysis of third-party designs

- find a structure in algorithms
- exploit it for shortcut attacks
- implement practical attacks

Motivations

Symmetric crypto used everywhere

- **contexts**: access control, Taser guns, DNS servers, toll systems, anti-counterfeiting, etc.
- **protocols**: encryption, integrity check, identification, etc.

Plus

- active research scene
- many under-understood problems
- diversity of algorithms / techniques
- thrill of attacking real-world systems

Preimage attacks on reduced MD5 and HAVAL

work with Florian Mendel

SAC ’08

MD5 cryptanalysis history

1992: specs published (Rivest)

1996: pseudo-collisions (den Boer-Bosselaers, Dobbertin)

"we feel that it is only prudent (. . .) to expect that collisions for the entire hash function might soon be found" (Robshaw, '96)

2004: collisions (Wang et al.)

"a collision can be found in mere seconds" (Stevens, '06)

2008: preimages for reduced versions

2009: preimages for full MD5 (Sasaki-Aoki)

Our new techniques for preimage search

Neutral words

- make computation independent of certain words
- done by exploiting structural properties of the algo

Local collisions

- split computation into two parts
- connect the two parts on n bits in $O(2^{n/2})$
- need freedom degrees from the neutral words

Techniques generalized by Sasaki-Aoki to attack full MD5

47 steps: word 2 input twice

1	$f(\ldots, 0)$	**17**	$g(\ldots, 1)$	**33**	$h(\ldots, 5)$
2	$f(\ldots, 1)$	**18**	$g(\ldots, 6)$	**34**	$h(\ldots, 8)$
3	$f(\ldots, 2)$	**19**	$g(\ldots, 11)$	**35**	$h(\ldots, 11)$
4	$f(\ldots, 3)$	**20**	$g(\ldots, 0)$	**36**	$h(\ldots, 14)$
5	$f(\ldots, 4)$	**21**	$g(\ldots, 5)$	**37**	$h(\ldots, 1)$
6	$f(\ldots, 5)$	**22**	$g(\ldots, 10)$	**38**	$h(\ldots, 4)$
7	$f(\ldots, 6)$	**23**	$g(\ldots, 15)$	**39**	$h(\ldots, 7)$
8	$f(\ldots, 7)$	**24**	$g(\ldots, 4)$	**40**	$h(\ldots, 10)$
9	$f(\ldots, 8)$	**25**	$g(\ldots, 9)$	**41**	$h(\ldots, 13)$
10	$f(\ldots, 9)$	**26**	$g(\ldots, 14)$	**42**	$h(\ldots, 0)$
11	$f(\ldots, 10)$	**27**	$g(\ldots, 3)$	**43**	$h(\ldots, 3)$
12	$f(\ldots, 11)$	**28**	$g(\ldots, 8)$	**44**	$h(\ldots, 6)$
13	$f(\ldots, 12)$	**29**	$g(\ldots, 13)$	**45**	$h(\ldots, 9)$
14	$f(\ldots, 13)$	**30**	$g(\ldots, 2)$	**46**	$h(\ldots, 12)$
15	$f(\ldots, 14)$	**31**	$g(\ldots, 7)$	**47**	$h(\ldots, 15)$
16	$f(\ldots, 15)$	**32**	$g(\ldots, 12)$		

Differences propagation, general case

Pick random A_0, B_0, C_0, D_0 and M

1	$f(A_0, B_0, C_0, D_0, 0)$
2	$f(A_1, B_1, C_1, D_1, 1)$
3	$f(A_2, B_2, C_2, D_2, 2)$

Modify C_0 to $C_0^\star$

X	**1**	$f(A_0, B_0, C_0^\star, D_0, 0)$
X	**2**	$f(A_1, B_1, C_1, C_0^\star, 1)$
X	**3**	$f(C_0^\star, B_2, C_2, D_2, 2)$

$\Rightarrow$ all first steps affected (X=state modified)

Difference 1: in C_0 with chosen IV

Pick random A_0, C_0, D_0 and M **and set** $\mathbf{B_0 = 0}$

1	$f(A_0, B_0, C_0, D_0, 0)$
2	$f(A_1, B_1, C_1, D_1, 1)$
3	$f(A_2, B_2, C_2, D_2, 2)$

Modify C_0 to $C_0^\star$

$\surd$	**1**	$f(A_0, 0, C_0^\star, D_0, 0)$
$\surd$	**2**	$f(A_1, B_1, 0, C_0^\star, 1)$
X	**3**	$f(C_0^\star, B_2, C_2, 0, 2)$

$\Rightarrow$ only step 3 affected

Difference 2: in M_2

Pick random A_0, B_0, C_0, D_0 and M

1	$f(A_0, B_0, C_0, D_0, 0)$
2	$f(A_1, B_1, C_1, D_1, 1)$
3	$f(A_2, B_2, C_2, D_2, 2)$

Modify M_2

$\surd$	**1**	$f(A_0, B_0, C_0, D_0, 0)$
$\surd$	**2**	$f(A_1, B_1, C_1, C_0, 1)$
X	**3**	$f(A_2, B_2, C_2, D_2, 2)$

$\Rightarrow$ only step 3 affected

Combine differences 1 and 2

Pick random A_0, C_0, D_0 and M **and set** $\mathbf{B_0 = 0}$

	1	$f(A_0, B_0, C_0, D_0, 0)$
	2	$f(A_1, B_1, C_1, D_1, 1)$
	3	$f(A_2, B_2, C_2, D_2, 2)$

Modify C_0 to $C_0^\star$ **and** M_2

$\surd$	**1**	$f(A_0, 0, C_0^\star, D_0, 0)$
$\surd$	**2**	$f(A_1, B_1, 0, C_0^\star, 1)$
$\surd$	**3**	$f(C_0^\star, B_2, C_2, 0, 2)$

$\Rightarrow$ nothing changes! (diff. in M_2 cancels that in $C_0^\star$)

Application to 47-step MD5

1	$f(\ldots, 0)$	**17**	$g(\ldots, 1)$	**33**	$h(\ldots, 5)$
2	$f(\ldots, 1)$	**18**	$g(\ldots, 6)$	**34**	$h(\ldots, 8)$
3	$f(\ldots, 2)$	**19**	$g(\ldots, 11)$	**35**	$h(\ldots, 11)$
4	$f(\ldots, 3)$	**20**	$g(\ldots, 0)$	**36**	$h(\ldots, 14)$
5	$f(\ldots, 4)$	**21**	$g(\ldots, 5)$	**37**	$h(\ldots, 1)$
6	$f(\ldots, 5)$	**22**	$g(\ldots, 10)$	**38**	$h(\ldots, 4)$
7	$f(\ldots, 6)$	**23**	$g(\ldots, 15)$	**39**	$h(\ldots, 7)$
8	$f(\ldots, 7)$	**24**	$g(\ldots, 4)$	**40**	$h(\ldots, 10)$
9	$f(\ldots, 8)$	**25**	$g(\ldots, 9)$	**41**	$h(\ldots, 13)$
10	$f(\ldots, 9)$	**26**	$g(\ldots, 14)$	**42**	$h(\ldots, 0)$
11	$f(\ldots, 10)$	**27**	$g(\ldots, 3)$	**43**	$h(\ldots, 3)$
12	$f(\ldots, 11)$	**28**	$g(\ldots, 8)$	**44**	$h(\ldots, 6)$
13	$f(\ldots, 12)$	**29**	$g(\ldots, 13)$	**45**	$h(\ldots, 9)$
14	$f(\ldots, 13)$	**30**	$g(\ldots, 2)$	**46**	$h(\ldots, 12)$
15	$f(\ldots, 14)$	**31**	$g(\ldots, 7)$	**47**	$h(\ldots, 15)$
16	$f(\ldots, 15)$	**32**	$g(\ldots, 12)$		

Our results

Preimage search algorithms for the compression
\+ Tree-based algorithm to find a preimage for the hash
= Preimage search algorithms for the hash functions

MD5 45-step compression: 2^{100} compressions

MD5 47-step: 2^{102} compressions, memory 2^{39} bytes

HAVAL 3-pass: 2^{233} compressions, memory 2^{71} bytes

First step to attack full MD5, 5-pass HAVAL, SHA-2, etc.

Cryptanalysis of MULTI2

work with Jorge Nakahara and Pouyan Sepehrdad

FSE ’09

MULTI2

- block cipher designed by Hitachi in **1988**
- cipher of the Japanese digital-TV/radio standard ISDB (2000), previously standardized by ARIB
- used for conditional access and copy control
- encrypts stream packets in CBC or OFB mode

B-CASカード（表面）

B-CASカード（裏面）

MULTI2 specs

64-bit blocks

64-bit data key and 256-bit system key “expanded” to a single 256-bit encryption key

Feistel structure with functions like

$$\begin{aligned}(x, k_i) \quad \mapsto \quad & ((((x + k_i) \lll 1) + x + k_i - 1) \lll 4) \oplus \\ & (((x + k_i) \lll 1) + x + k_i - 1)\end{aligned}$$

32 rounds in ISDB

Security analysis

Previous attack on 12 rounds (Matsui-Yamagishi, 1994)

Our results:

- recover all $(64 + 256)$ keys bits in 2^{191} (any #rounds)
- linear cryptanalysis on 20 rounds in 2^{93}
- **related-key slide attack** in 2^{136} (any #rounds)

MULTI2 broken (security $\ll$ #key bits)

Safe in practice for ISDB (64-bit security, complicated mode of operation)

First international publication on MULTI2

Cryptanalysis of reduced Salsa20

work with Shahram Khazaei and Simon Fischer

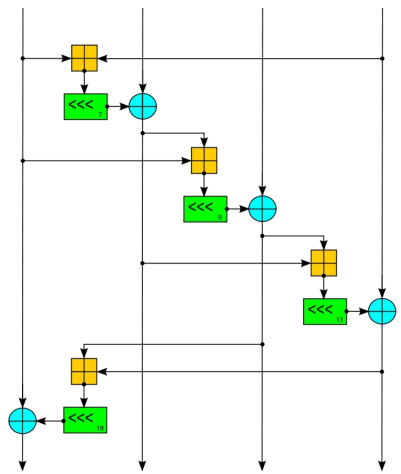
FSE ’08

The Salsa20 stream cipher

Cowinner of the eSTREAM competition

Faster than AES-CTR (ex: 4 vs. 12 cpb on a Core 2)

Add-xor-rotate core function



Default cipher in the **NaCl** C library

Salsa20 algorithm

"Salsa20 is, at first glance, a traditional stream cipher; at second glance, a hash function in counter mode."
(Bernstein)

16×32-bit state initialized with constants, key, IV, counter:

c_0	k_0	k_1	k_2
k_3	c_1	v_0	v_1
t_0	t_1	c_2	k_4
k_5	k_6	k_7	c_3

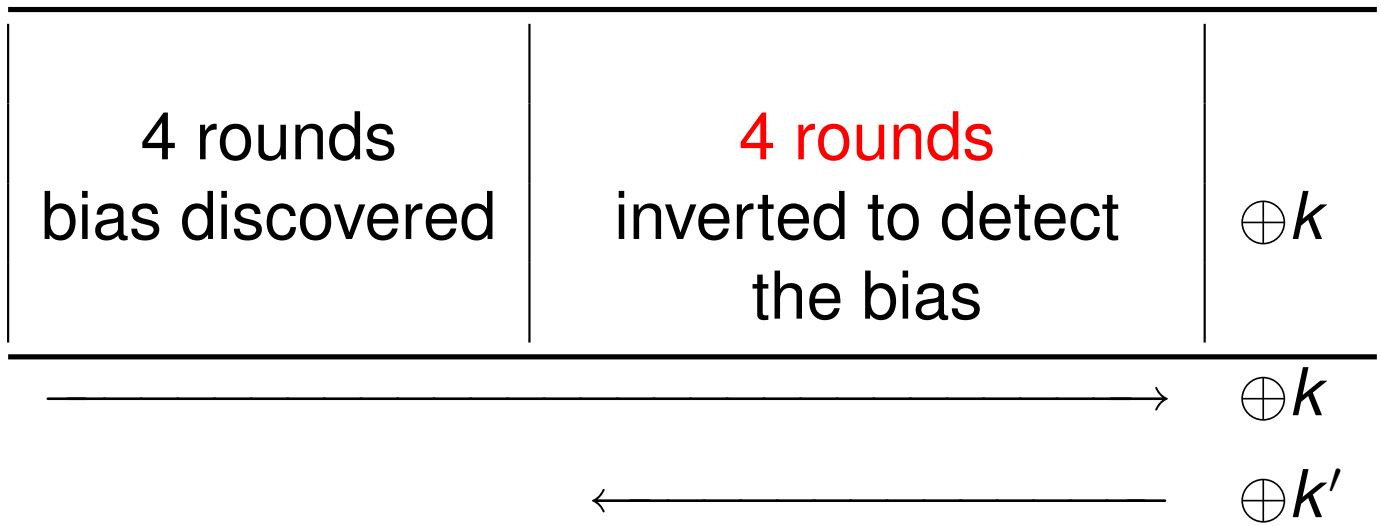
Transform columns, diagonals, columns, etc.

After 20 rounds, xor with initial state

Use counter n to encrypt (xor) data block nb. n

Attack strategy

Structure $\pi(k) \oplus k$, with k a 256-bit key



Invert 4 rounds to observe biased bit, but need k...

220 correct key bits sufficient of observe bias (key k')

First recover 220 key bits, then bruteforce the rest

Our results

Key-recovery on **7 rounds** in 2^{151}

Key-recovery on **8 rounds** in 2^{251}

- best results so far on Salsa20
- also works on the variant ChaCha (up to 7 rounds)
- eSTREAM selected Salsa20 with 12 rounds
- technique reused to attack Skein (Asiacrypt '09)

Cube testers and applications

work with Itai Dinur, Luca Henzen, and Adi Shamir

FSE ’09

Origins

Related to previous statistical tests on ANF's by Filiol, Saarinen, O'Neil, Englund-Johansson-Turan, and to high-order differential techniques

Directly inspired by **cube attacks** (Dinur-Shamir, '08)

Initially developed to attack (reduced) MD6

			Session 9	*Invited Talk*
			Chair	*Matt Robshaw*
11:10	-	12:10	***How to Solve it: New Techniques in Algebraic Cryptanalysis*** Adi Shamir	

			Session 11	*Invited Talk*
			Chair	*Hovav Shacham*
11:10	-	12:10	***The MD6 hash function*** Ronald Rivest	

(CRYPTO '08)

In a nutshell...

Any mapping $f : \{0,1\}^n \to \{0,1\}^m$ admits an ANF wrt GF(2) composed of m equations in n variables

Formal computation of order-$(n-k)$ derivative of f:

1. fix k input bits
2. varying $(n-k)$ bits over $2^{(n-k)}$ values, compute

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

3. obtain m polynomials in $(n-k)$ variables of degree $\leq (n-k)$

If f has degree $(n-k+1)$, then we obtain **linear** polynomials
$\Rightarrow$ can recover the k fixed bits in $O(k^3)$! (cube attacks)

In a nutshell. . .

For key-recovery, need

- linear derivatives
- reconstruct ANF of derivative with black-box queries and linearity tests (precomputed)

Cube testers give distinguishers, and

- apply algebraic property tester in a black-box way
- do not need knowledge of derivative's ANF
- test any structure of high-order derivatives (linearity, density, low-degree, etc.)
- need no expensive precomputation

$\Rightarrow$ at least as powerful as cube attacks

How to determine set of cube variables?

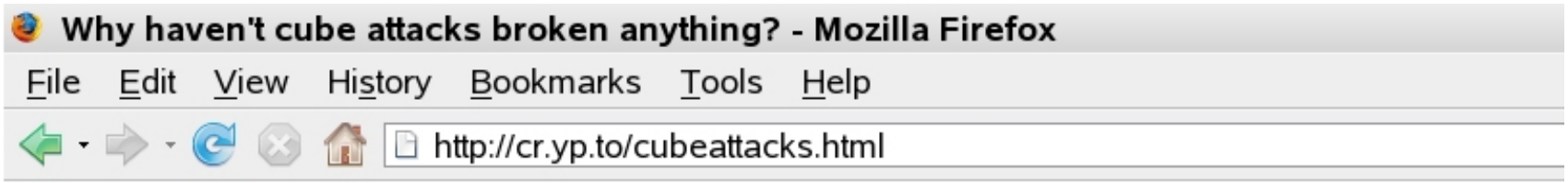
Complexity bottleneck, and main distinction to previous high-order techniques

Analytically: determine sets of variables by analyzing the algorithm

Ex: Trivium

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

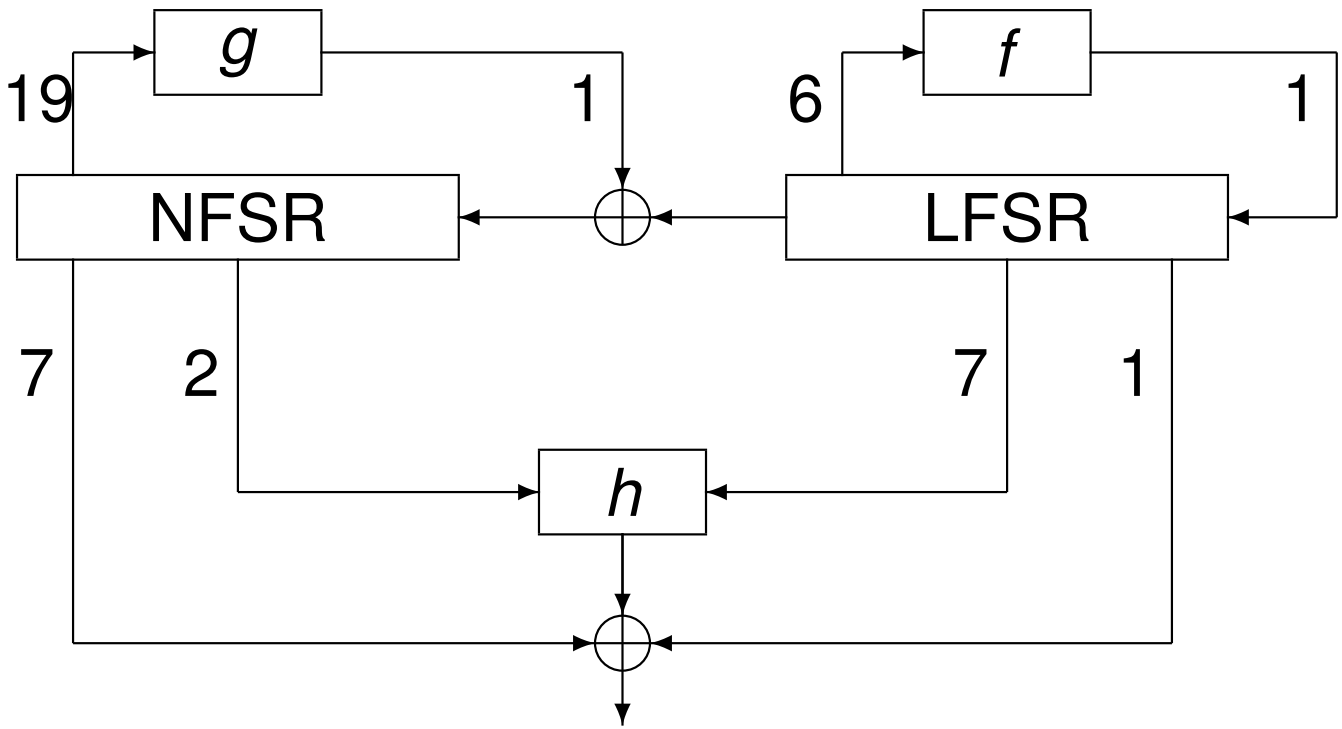
Empirically: explore the search space to find good sets of variables with discrete optimization tools



What about cube testers?

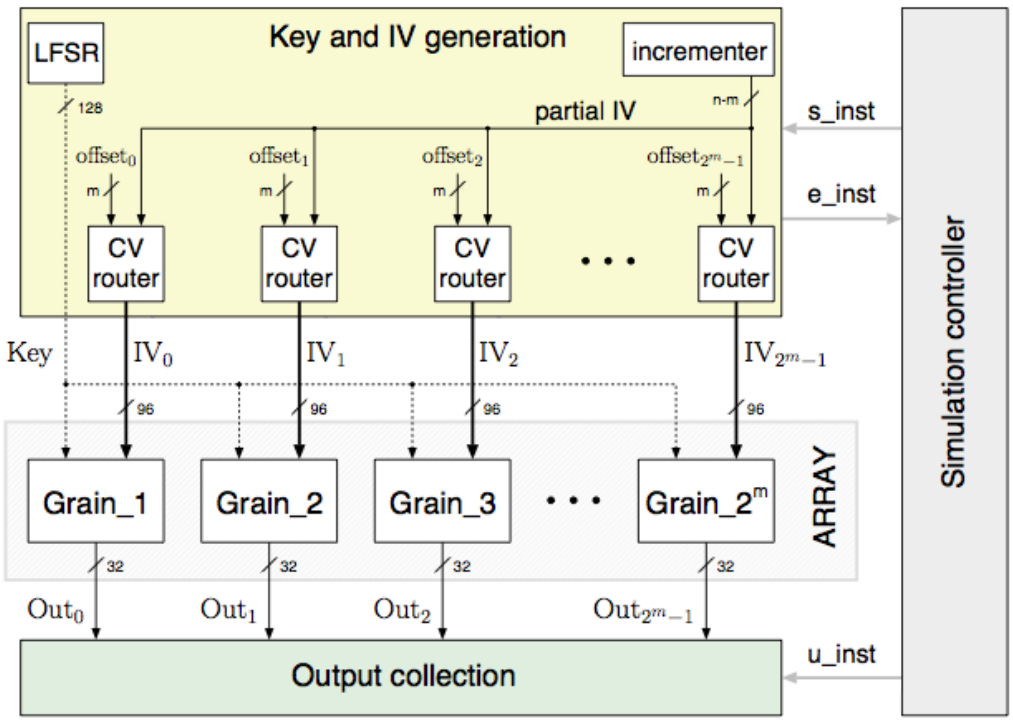
Going against the Grain (-128)

- 128-bit version of the eSTREAM cowinner Grain-v1
- designed by Hell, Johansson, Maximov, Meier
- DPA and related-key attacks
- previous attack on version with 192 rounds (of 256)



Arsenal 1/3: hardware “cracking machine”

- Xilinx Virtex-5 FPGA, VHDL programs
- 256 instances of 32×Grain-128 in parallel
- efficient implementation of cube testers
- attacks involving more than 2^{54} clocks in ≈1 day



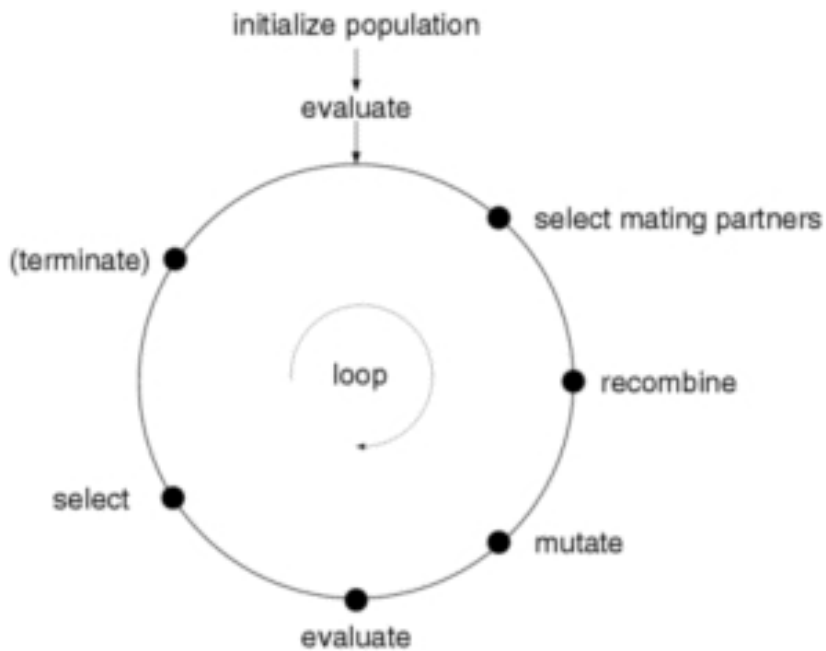
Arsenal 2/3: bitsliced C implementation

- run 64 instances of Grain-128 in parallel on a desktop
- used for low-complexity attacks on parameters optimization

```
u64 grain80_bitsliced64( u64 * key, u64 * iv, int rounds ) {

  u64 l[80+rounds], n[80+rounds], z=0;
  int i,j;

  /* initialize registers */
  for(i=0; i<64; i++){
    n[i]= key[i];
    l[i]= iv[i];
  }
  for(i=64; i<80; i++){
    n[i]= key[i];
    l[i]= 0xFFFFFFFFFFFFFFFFULL;
  }

  for(i=0; i<rounds; i++){
    /* clock */
    l[i+80] = l[i] ^ l[i+13] ^ l[i+23] ^ l[i+38] ^ l[i+51] ^ l[i+62];

    n[i+80] = l[i] ^ n[i] ^ n[i+9] ^ n[i+14] ^ n[i+21] ^ n[i+28] ^ n[i+33] ^ n[i+37] ^ n[i+37] ^ n[i+45] ^ n[i+52] ^ n[i+60] ^
      n[i+62] ^ ( n[i+63] & n[i+60] ) ^ ( n[i+37] & n[i+33] ) ^ ( n[i+15] & n[i+9] ) ^
      ( n[i+60] & n[i+52] & n[i+45] ) ^ (n[i+33] & n[i+28] & n[i+21] ) ^ ( n[i+63] & n[i+45] & n[i+28] & n[i+9] ) ^
      ( n[i+60] & n[i+52] & n[i+37] & n[i+33] ) ^ ( n[i+63] & n[i+60] & n[i+21] & n[i+15] ) ^
      ( n[i+63] & n[i+60] & n[i+52] & n[i+45] & n[i+37] ) ^ ( n[i+33] & n[i+28] & n[i+21] & n[i+15] & n[i+9] ) ^
      ( n[i+52] & n[i+45] & n[i+37] & n[i+33] & n[i+28] & n[i+21] );

    z = l[i+25] ^ n[i+63] ^ ( l[i+3] & l[i+64] ) ^ (l[i+46] & l[i+64]) ^ (l[i+64] & n[i+63]) ^ (l[i+3] & l[i+25] & l[i+46] ) ^
      (l[i+3] & l[i+46] & l[i+64] ) ^ (l[i+3] & l[i+46] & n[i+63] ) ^ (l[i+25] & l[i+46] & n[i+63] ) ^ (l[i+46] & l[i+64] & n[i+63] );
    z = n[i + 1] ^ n[i + 2] ^ n[i + 4] ^ n[i + 10] ^ n[i + 31] ^ n[i + 43] ^ n[i + 56] ^ z;

    l[i+80] ^= z; n[i+80] ^= z;
  }

  /* return 1 keystream bit */
  z = (n[i+12] & l[i+8]) ^ (l[i+13] & l[i+20]) ^ (n[i+95] & l[i+42]) ^ (l[i+60] & l[i+79]) ^ (n[i+12] & n[i+95] & l[i+95]);
  z = n[i + 2] ^ n[i + 15] ^ n[i + 36] ^ n[i + 45] ^ n[i + 64] ^ n[i + 73] ^ n[i + 89] ^ z ^ l[i + 93];

  return z;
}
```

Arsenal 3/3: evolutionary programming

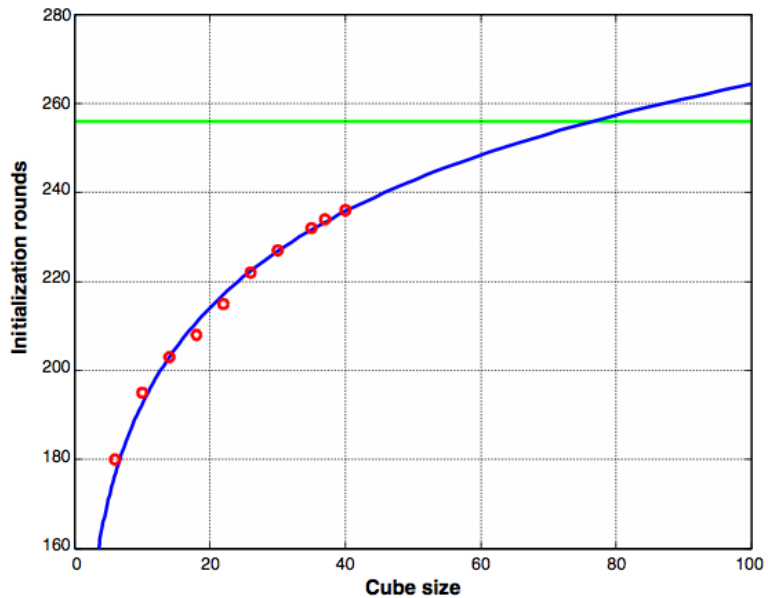
- population-based metaheuristic
- used to search for good variable sets
- C program combined with bitsliced code
- ad hoc optimizations (weak variables, etc.)



Our results

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

Extrapolation:



Suggests existence of distinguishers in 2^{77}

$\Rightarrow$ unlikely to guarantee 128-bit security

Other applications

Stream cipher **Trivium**

- eSTREAM HW cowinner by de Cannière and Preneel
- previous cube attacks on 67% of the cipher
- **cube testers** on 77% of the cipher

Hash function **MD6**

- submission to SHA-3 by Rivest et al.
- tree-hashing based on low-degree compression
- first (practical) attack on reduced MD6
- distinguisher improved by Khovratovich

Design of the SHA-3 candidate BLAKE

work with Luca Henzen and Raphael C.-W. Phan

Submission to the SHA-3 Competition

Mission statement

NIST called for submission of hash algorithms. . .

- *"implementable in a wide range of HW and SW platforms"*
- with standard collision/preimage resistance
- supporting HMAC and randomized hashing

Our goals:

- submit a bulletproof design
- break/attack competitors
- be chosen as finalist

Our submission BLAKE

Design principles:

- make it simple
- do not reinvent the wheel
- do not optimize for a specific platform
- build on previous knowledge and scrutiny

Design objectives: good tradeoffs

- SW/HW
- confidence/performance
- conservativeness/novelty

32- and a 64-bit versions, same interface as SHA-2

- “drop-in” replacement
- framework familiar to implementers

32- and a 64-bit versions, same interface as SHA-2

- “drop-in” replacement
- framework familiar to implementers

Iteration mode **HAIFA** (Biham-Dunkelman, ’06)

- extension of the Merkle-Damgård construction
- countermeasures against generic attacks
- HAIFA simplified for BLAKE (but security preserved)

32- and a 64-bit versions, same interface as SHA-2

- “drop-in” replacement
- framework familiar to implementers

Iteration mode **HAIFA** (Biham-Dunkelman, ’06)

- extension of the Merkle-Damgård construction
- countermeasures against generic attacks
- HAIFA simplified for BLAKE (but security preserved)

Core algorithm **ChaCha** (Bernstein, ’08)

- variant of Salsa20 analyzed within eSTREAM
- own attacks on 7 rounds (FSE ’08)
- parallelizable (both in SW and HW)
- compact and secure implementations

BLAKE's core function

Permutation of words (a, b, c, d) with key=message

BLAKE-32

$a \mathrel{+}= m_i \oplus const_i$
$a \mathrel{+}= b$
$d = (a \oplus d) \ggg 16$
$c \mathrel{+}= d$
$b = (b \oplus c) \ggg 12$
$a \mathrel{+}= m_j \oplus const_j$
$a \mathrel{+}= b$
$d = (a \oplus d) \ggg 8$
$c \mathrel{+}= d$
$b = (b \oplus c) \ggg 7$

BLAKE-64

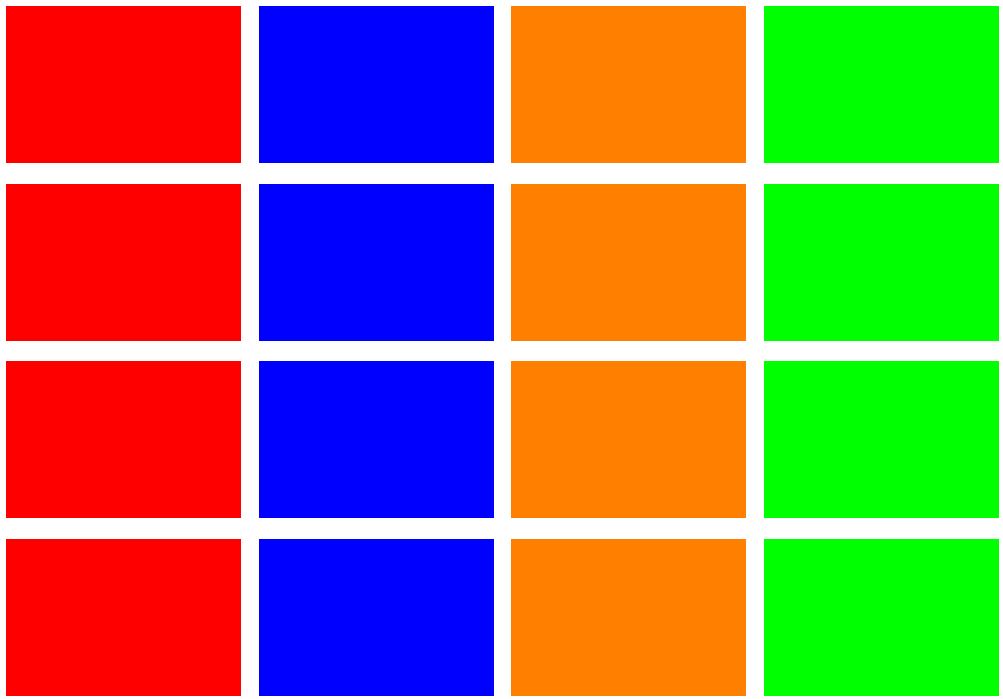
$a \mathrel{+}= m_i \oplus const_i$
$a \mathrel{+}= b$
$d = (a \oplus d) \ggg 32$
$c \mathrel{+}= d$
$b = (b \oplus c) \ggg 25$
$a \mathrel{+}= m_j \oplus const_j$
$a \mathrel{+}= b$
$d = (a \oplus d) \ggg 16$
$c \mathrel{+}= d$
$b = (b \oplus c) \ggg 11$

Compression function state

Initialized with salt, counter, chaining value

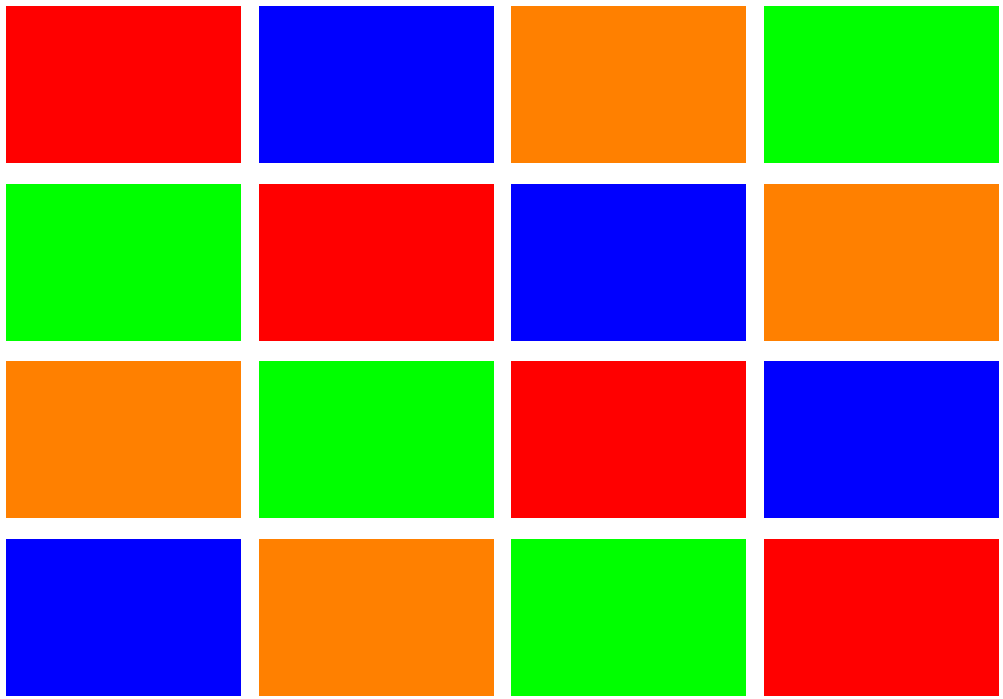
BLAKE round

Apply the core function to each column. . .



BLAKE round

. . . then to each diagonal



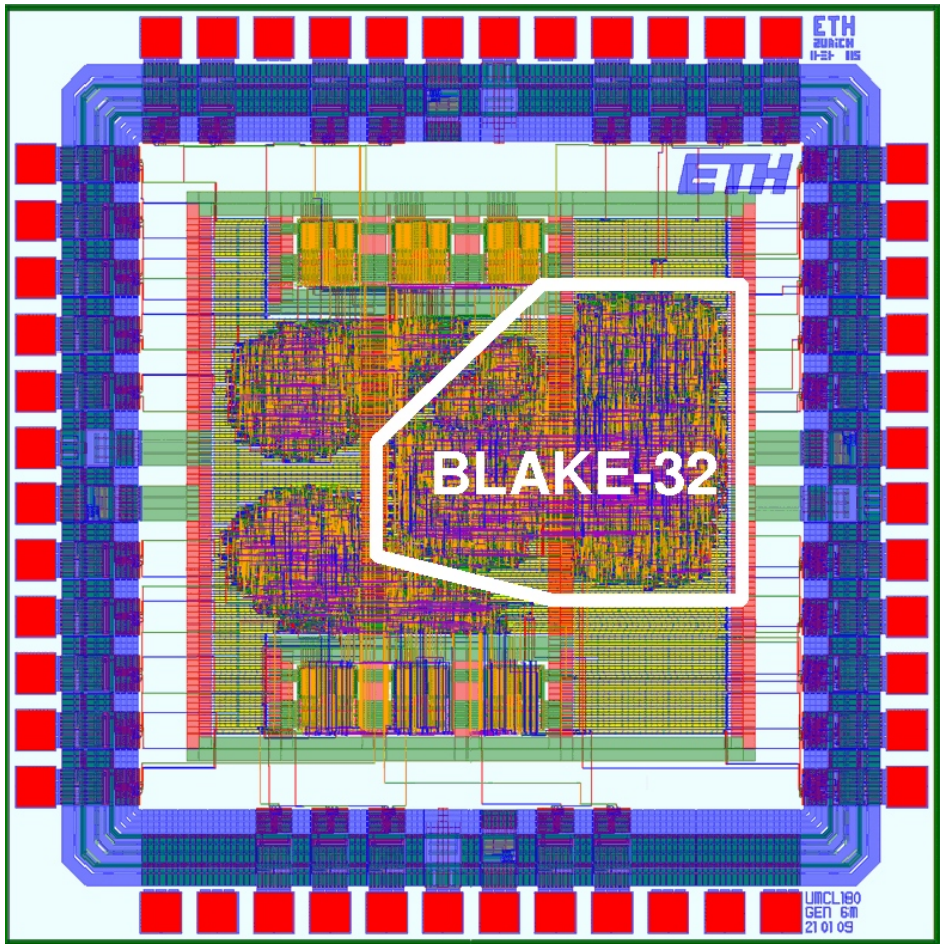
Software performance

- compact implementation ($<$190 C lines)
- fast in both 32- and 64-bit modes
- SIMD instructions help, but are not vital
- round 4$\times$parallelizable
- short critical portion
- one of the fastest and most flexible candidates

Cycles/byte for long messages			
quartile	**median**	**quartile**	**hash**
9.84	9.91	9.93	bmw256
9.92	9.95	10.18	blake32
10.25	10.25	10.25	shabal512
12.77	12.79	12.81	blake64
13.00	13.16	13.36	simd256
13.51	13.52	13.52	bmw512

Hardware performance

- implementation on various FPGA's and ASIC's
- 4 architectures for area/speed tradeoff
- in-silico implementation on 13.5 kGE



BLAKE in the SHA-3 competition

One of the 14 second round candidates

Third-party implementations on ASIC, 8-bit, etc.

Third-party cryptanalysis: best attack on 2.5 rounds

"BLAKE's performance is quite good. It has modest memory requirements, and appears to be suitable for a wide range of platforms" (NIST)

"The best results against BLAKE (. . .) appear to pose no threat to the design" (NIST)

Conclusion

Summary of contributions

First preimage attack for MD5 and new techniques that lead to attacks on the full version (SAC '08)

Cryptanalysis of the cipher of the ISDB DTV standard and theoretical break (FSE '09)

Best known attacks on the eSTREAM cowinner Salsa20 (FSE '08)

Attacks on reduced Trivium and MD6 (FSE '09)

Break of the state-of-the-art stream cipher Grain-128 (SHARCS '09)

Design of a second round SHA-3 candidate

Other contributions (not in the thesis)

Design of BLAKE's predecessor LAKE (FSE '08)

Analysis of iteration modes (Africacrypt '08, Indocrypt '08)

Break of SHA-3 submissions:

- Dynamic SHA2 (SAC '09)
- ESSENCE (submitted)
- MCSSHA (WEWoRC '09)
- Vortex (Africacrypt '09)

Analysis of SHA-3 second round candidates:

- CubeHash (ACISP '09)
- Skein (Asiacrypt '09)