Collisions on SHA-0 in one hour

IPA Cryptographic Workshop 2007 - Tokyo, Japan

Thomas Peyrin

joint work with **Stéphane Manuel** (INRIA)

AIST

Orange Labs

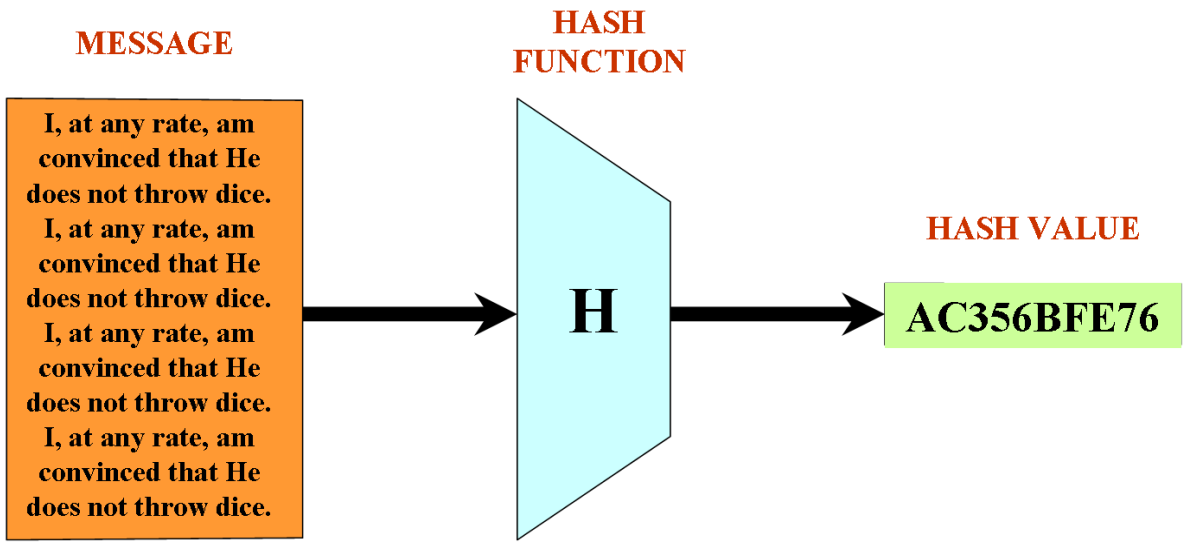
University of Versailles

December 13, 2007

Outline

Outline

What is a hash function ?



- H maps an **arbitrary length input** (the message M) to a **fixed length output** (typically $n = 128$, $n = 160$ or $n = 256$).
- H must be collision ($2^{n/2}$ function calls), 2nd-preimage (2^n function calls) and preimage resistant (2^n function calls).

Applications

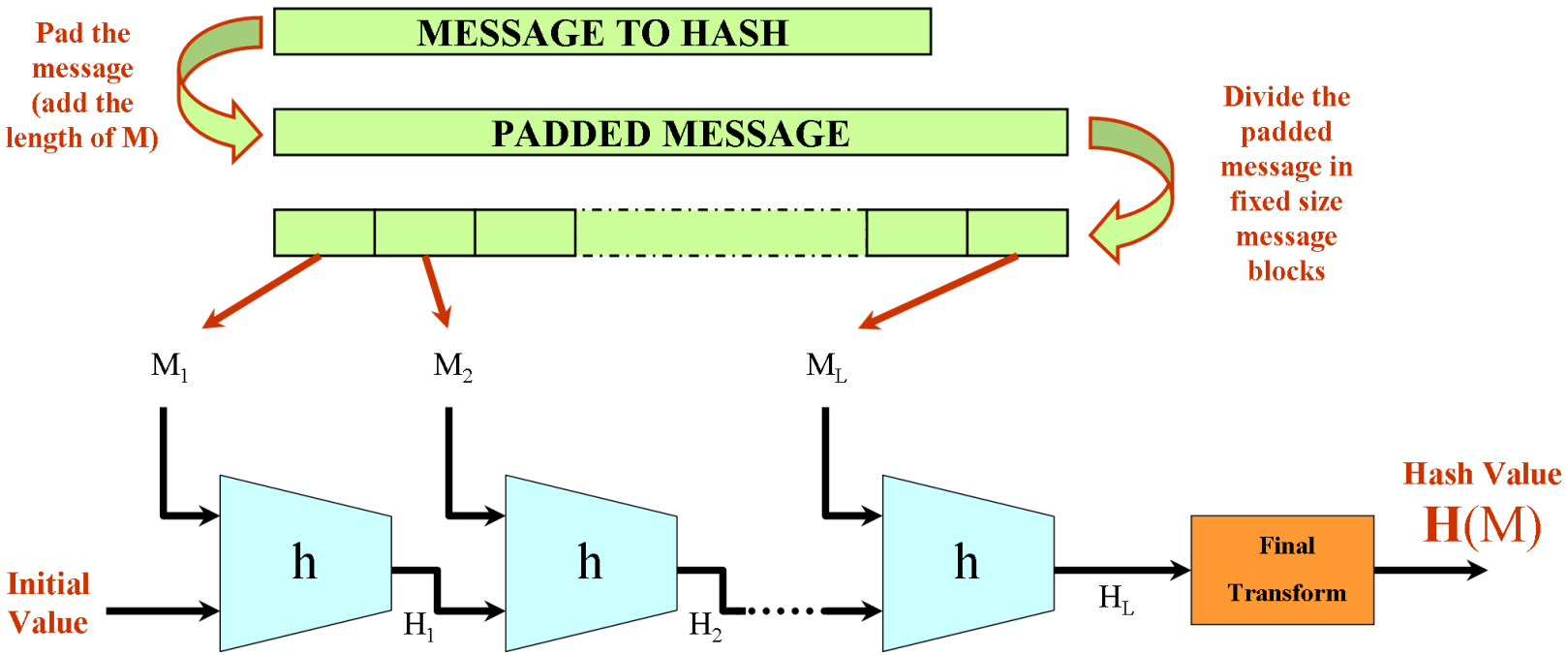
Hash functions are useful tools for many applications:

- digital signatures (hash-and-sign, ...): improves performance and security for signatures.
- used to build MACs (HMAC is used in SSL/TLS, IPSec, ...).
- password protection.
- confirmation of knowledge/commitment.
- pseudo-random string generation/key derivation.

How to build a hash function (usually) ?

compression function + **domain extension algorithm**.

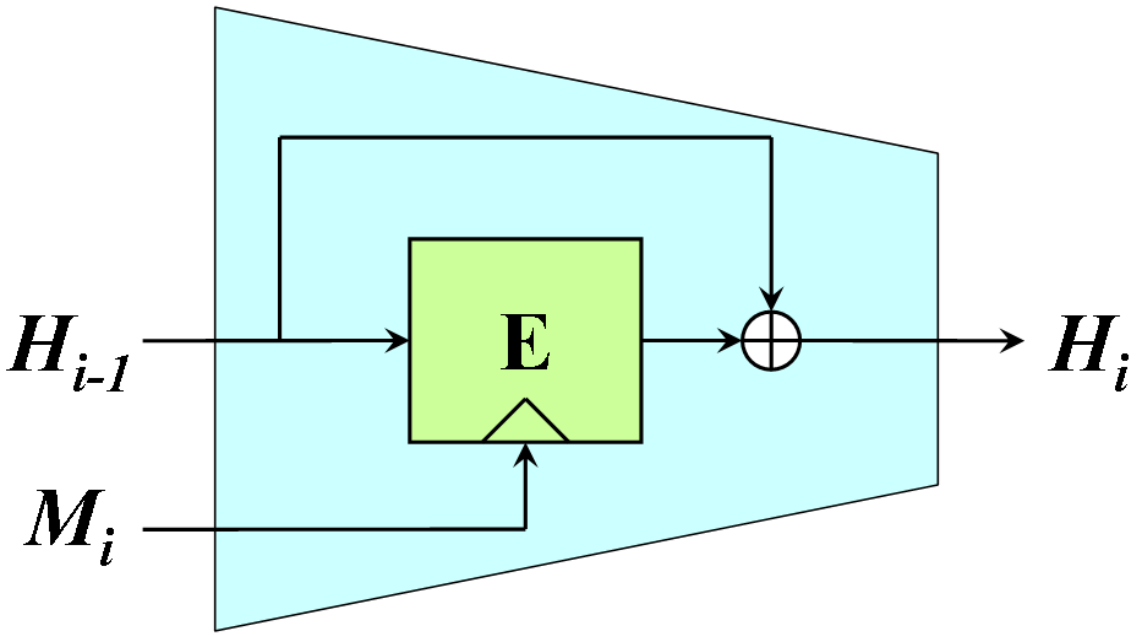
The Merkle-Damgård algorithm



How to build a hash function (usually) ?

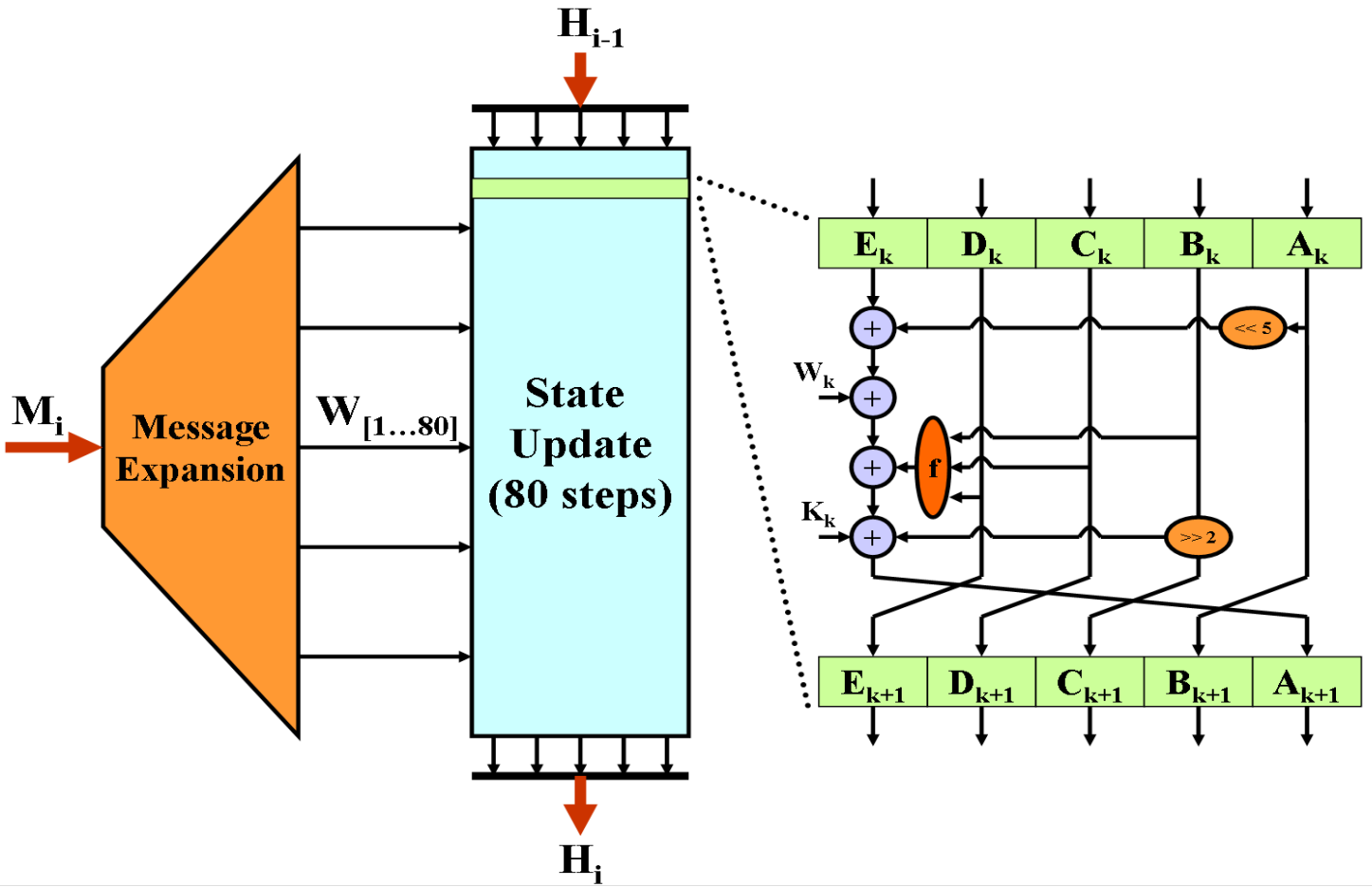
compression function + domain extension algorithm.

The Davies-Meyer construction



The SHA-0 hash function

SHA-0: built in 1993, 160-bit output.



The SHA-0 hash function

Message expansion:

$$W_i = \begin{cases} M_i, & \text{for } 0 \leq i \leq 15 \\ W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}, & \text{for } 16 \leq i \leq 79 \end{cases}$$

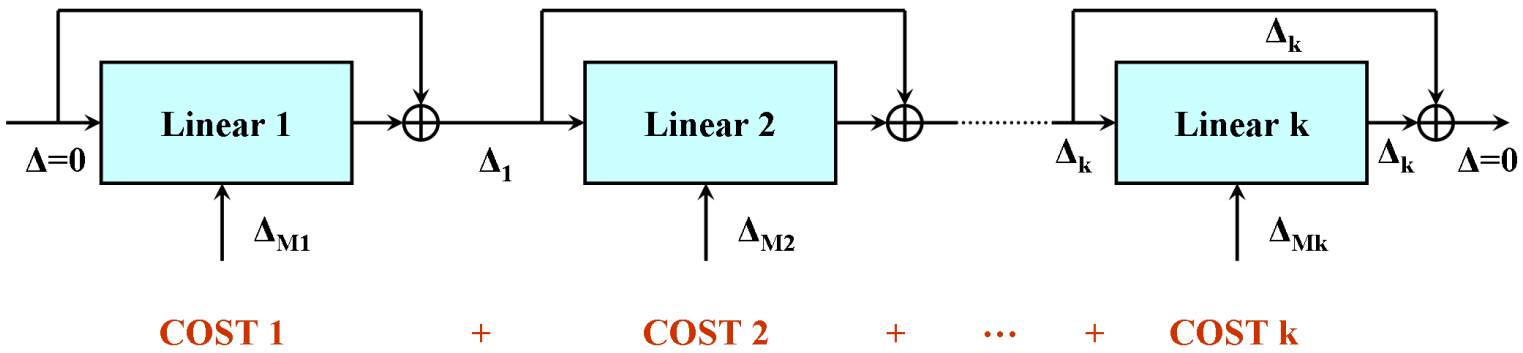
Boolean functions:

step i	$f_i(B, C, D)$
$1 \leq i \leq 20$	$f_{IF} = (B \wedge C) \oplus (\overline{B} \wedge D)$
$21 \leq i \leq 40$	$f_{XOR} = B \oplus C \oplus D$
$41 \leq i \leq 60$	$f_{MAJ} = (B \wedge C) \oplus (B \wedge D) \oplus (C \wedge D)$
$61 \leq i \leq 80$	$f_{XOR} = B \oplus C \oplus D$

Outline

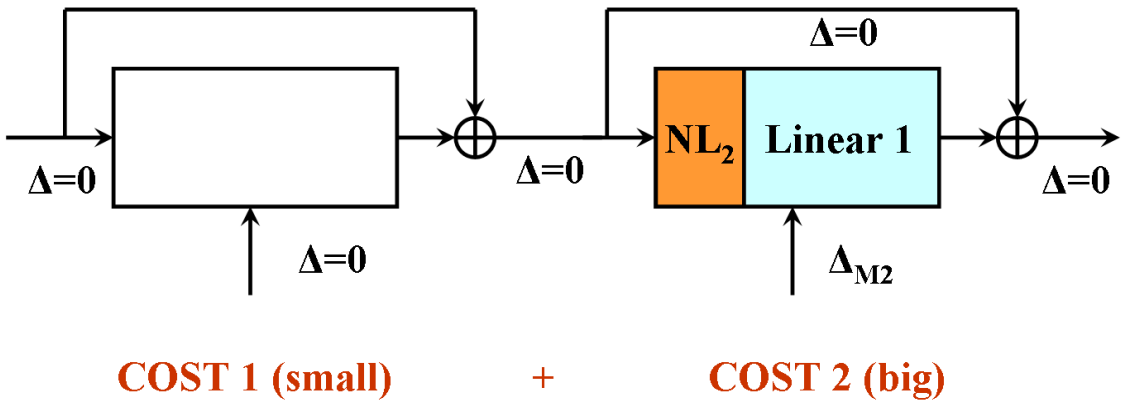
Chabaud and Joux (1998) - Biham *et al.*(2004)

- **local collision**: insert a perturbation and correct it! Then find perturbation and corrections vectors such that the overall difference mask satisfies the message expansion.
- **neutral bits**: speedup technique during collision search.
- **multi-block technique**: you can use several blocks to find a collision.



Wang *et al.* (2005)

- **non-linear part**: modify (by hand!) the first steps of the differential path (where the attackers has control) so that everything behave not necessarily linearly. This allows to use less constraint perturbation vectors, thus better ones may be found.
- **message modification**: another speedup technique during collision search.



Naito *et al.* (2006)

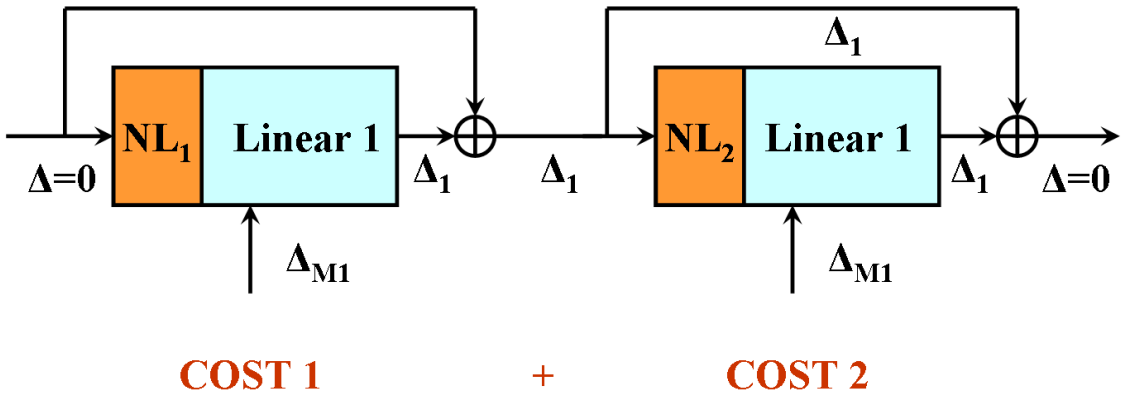
- **submarine modification**: another speedup technique during collision search.
- **complexity:**
  - 2^{36} function calls theoretically ...
  - ... but requires 100 hours on average with a good PC.
  - our estimation: $2^{40,3}$ function calls practically.

Complexity should be given in terms of function calls with an efficient implementation on the same computer (i.e. OpenSSL).

Outline

Possible improvements

- multi-block technique is an improvement (better vectors of perturbation), non-linear part is another improvement (speedup the collision search): why not use both ?
- multi-block technique + non-linear part is the core of the current attacks against SHA-1, why not SHA-0 ?
- because Wang *et al.*'s attack is not reusable (everything found by hand) !



Possible improvements

Hopefully we now have good techniques to avoid the problems.

- 1^{st} **problem:**
  - if we use a better (less constraint) perturbation vector, how to find new non-linear parts for it ?
  - use the **automated non-linear part generator** from De Cannière and Rechberger (2006).
- 2^{nd} **problem:**
  - we now have no more message modification known, how to speedup the collision search ?
  - use the **boomerang attacks** from Joux and Peyrin (2007) that can easily speedup the collision search.

A new perturbation vector

- **we can remove one old constraint on the perturbation vector:**
  - **no local collision starting after step 74.**
- then we look for the best vectors in terms of minimizing the number of remaining conditions between steps 16 and 80 (where the attacker has no more control).
- the adaptability of the perturbation vector with speedup techniques can also be taken in account, the starting step for counting conditions can also be increased depending on the speedup technique used.
- several good possible vectors found.

A new perturbation vector

Steps 1 to 20																				
vec.	1	1	1	1	0	1	0	1	1	0	1	1	1	0	0	0	1	0	0	0
♯ cond.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	0	2	1

Steps 21 to 40																				
vec.	1	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
♯ cond.	1	0	2	0	2	0	1	0	0	0	0	0	0	0	1	0	1	1	0	1

Steps 41 to 60																				
vec.	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	1
♯ cond.	1	1	1	0	1	1	1	0	1	0	1	1	1	1	0	1	2	1	2	2

Steps 61 to 80																				
vec.	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0
♯ cond.	1	1	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	1	0	0

The boomerang improvement

Boomerang attacks:

- a collision search speedup framework: the attacker build auxiliary differentials (on a small number of steps) that can be used as neutral bits or message modifications.
- we chose the neutral bits setting, much more easier to set and use.
- two types of auxiliary differentials used: a light but short one, and a heavy but long one.
- on average, we can set 5 auxiliary differentials (7 for the first block), and we thus expect an improvement of a factor 2^5 on the raw attack (without speedup)

First auxiliary differential

i	A_i	W_i
-1:	----------	
00:	----------	----------
01:	----------	----------
02:	----------	----------
03:	----------	----------
04:	----------	----------
05:	----------b----	----------
06:	----------b----	----------a--
07:	----------a--	----------$\overline{a}$-------
08:	----------0	----------
09:	----------1	----------
10:	----------	----------
11:	----------	----------$\overline{a}$
12:	----------	----------
13:	----------	----------
14:	----------	----------
15:	----------	----------

Second auxiliary differential

i	A_i	W_i
-1:	----------d----	
00:	----------d----	----------a--
01:	----------e-a--	----------$\bar{a}$-------
02:	----------e---1	----------b--
03:	----------b-0	----------$\bar{b}$------$\bar{a}$
04:	----------0	----------$\bar{a}$
05:	----------0	----------$\bar{a}$
06:	----------	----------$\bar{b}$
07:	----------	----------$\bar{b}$
08:	----------	----------
09:	----------f----	----------
10:	----------f----	----------c--
11:	----------c--	----------$\bar{c}$-------
12:	----------0	----------
13:	----------0	----------
14:	----------	----------$\bar{c}$
15:	----------	----------$\bar{c}$

A collision example

	1^{st} block		2^{nd} block	
	M_1	M'_1	M_2	M'_2
W_0	0x4643450b	0x46434549	0x9a74cf70	0x9a74cf32
W_1	0x41d35081	0x41d350c1	0x04f9957d	0x04f9953d
W_2	0xfe16dd9b	0xfe16dddb	0xee26223d	0xee26227d
W_3	0x3ba36244	0x3ba36204	0x9a06e4b5	0x9a06e4f5
W_4	0xe6424055	0x66424017	0xb8408af6	0x38408ab4
W_5	0x16ca44a0	0x96ca44a0	0xb8608612	0x38608612
W_6	0x20f62444	0xa0f62404	0x8b7e0fea	0x0b7e0faa
W_7	0x10f7465a	0x10f7465a	0xe17e363c	0xe17e363c
W_8	0x5a711887	0x5a7118c5	0xa2f1b8e5	0xa2f1b8a7
W_9	0x51479678	0xd147963a	0xca079936	0x4a079974
W_{10}	0x726a0718	0x726a0718	0x02f2a7cb	0x02f2a7cb
W_{11}	0x703f5bfb	0x703f5bb9	0xf724e838	0xf724e87a
W_{12}	0xb7d61841	0xb7d61801	0x37ffc03a	0x37ffc07a
W_{13}	0xa5280003	0xa5280041	0x53aa8c43	0x53aa8c01
W_{14}	0x6b08d26e	0x6b08d26c	0x90811819	0x9081181b
W_{15}	0x2e4df0d8	0xae4df0d8	0x312d423e	0xb12d423e

A_2	B_2	C_2	D_2	E_2
0x6f84b892	0x1f9f2aae	0x0dbab75c	0x0afe56f5	0xa7974c90

Complexity comparison

Team	Theoretical	Practical	Time on a PC
Chabaud and Joux (1998)	2^{61}		
Biham *et al.* (2004)	2^{51}	2^{51}	20 years
Wang *et al.* (2005)	2^{39}		
Naito *et al.* (2006)	2^{36}	$2^{40,3}$	100 hours
Our results	2^{33}	$2^{33,6}$	1 hour

Conclusion

There is room for improvements:

- improve the collision search speedup: find an optimized adhoc message modification process instead of the easy-to-use boomerang attacks (just as Wang *et al.* did).
- with the assumption of having roughly the same message modification possibilities than for the Wang *et al.*'s perturbation vector, one would have a theoretical complexity of 2^{31} function calls approximatively (a few minutes on a standard PC) ...
- ... but this is just a possible complexity, there is NO proof that everything would indeed behave like for the Wang *et al.*'s case.

Thank you!