Extending FORK-256 Attack to the Full Hash Function

Scott Contini Krystian Matusiewicz Josef Pieprzyk

Centre for Advanced Computing Algorithms and Cryptography,
Department of Computing, Macquarie University

ICICS 2007, 12 December 2007

Outline

Introduction

- FORK-256 is a dedicated cryptographic hash function designed by Hong et al. and presented during second NIST hash workshop and FSE 2006.
- Heuristic design, but with some unorthodox design choices.
- Meant as a possible replacement for SHA-256 (compatible interface, better speed).

History of cryptanalysis of FORK-256

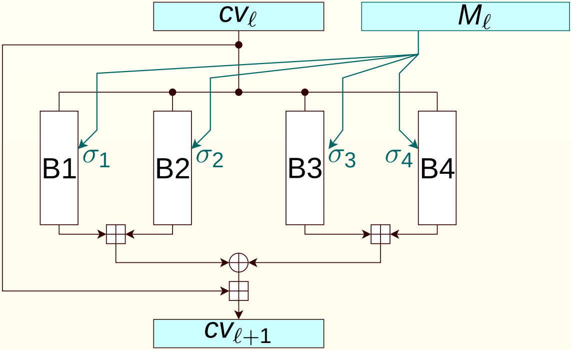
Received considerable cryptanalytic attention since it was proposed in 2006.

- Matusiewicz, Contini, Pieprzyk – IACR ePrint 2006/317
  cryptanalysis of reduced variants
- Mendel, Lano, Preneel – CT-RSA 2007
  cryptanalysis of reduced variants
- Matusiewicz, Peyrin, Billet, Contini, Pieprzyk – FSE 2007
  cryptanalysis of the full compression function

Our current contribution: Extending the attack to the full hash function (actually, with any predefined IV).

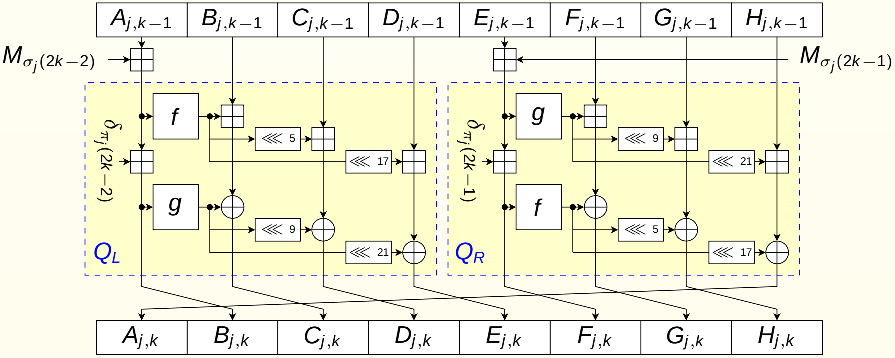
Outline

FORK-256



- 256 bits of chaining variable *cv*
- 512 bits of message *M*
- each branch uses a different permutation $(\sigma_1, \sigma_2, \sigma_3, \sigma_4)$ of message words $M_0, \ldots, M_{15}$
- each branch B1, B2, B3, B4 consists of **8 steps**

Structure of FORK-256 : step transformation



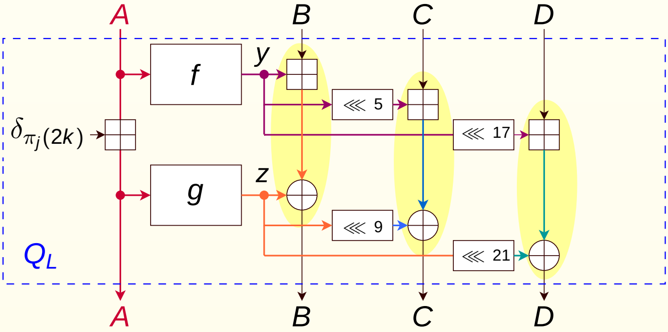
- there are 8 steps in each branch
- step transformation – composition of 3 simple operations
  - addition of two different message words
  - two parallel **Q-structures**
  - rotation of registers

Functions f and g

$$f(x) = x + \left(x^{\lll 7} \oplus x^{\lll 22}\right)$$
$$g(x) = x \oplus \left(x^{\lll 13} + x^{\lll 27}\right)$$

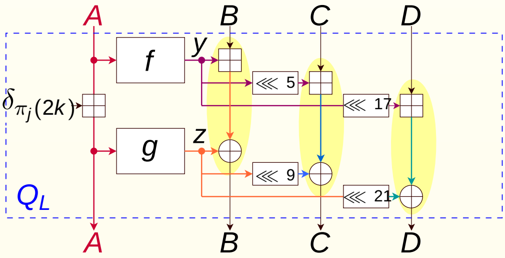
Outline

Micro-collisions in the step transformation



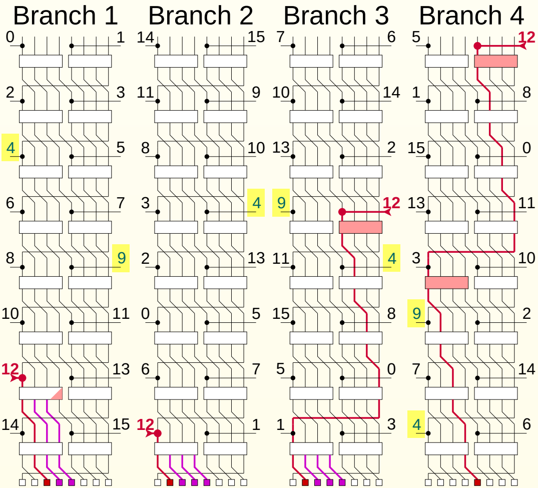
Micro-collision: a difference in register A does not propagate to the selected register (B,C,D).
If it does not propagate to more than one other register we have *simultaneous micro-collisions*.

Micro-collisions



- Let us fix a modular difference d. Having a value a of register A and $a' = a + d$, we can efficiently determine sets $\mathcal{B}_a$, $\mathcal{C}_a$, $\mathcal{D}_a$ of values of B, C, D such that simultaneous micro-collisions appear in all three lines.
- If sets $\mathcal{B}_a$, $\mathcal{C}_a$, $\mathcal{D}_a$ are non-empty, we call such a an allowable value (meaning we can achieve micro-collisions for that value of a)

Using micro-collisions in a differential path



We need microcollisions in only three and 1/3 Q-structures.

Only four output registers are influenced by the differential.

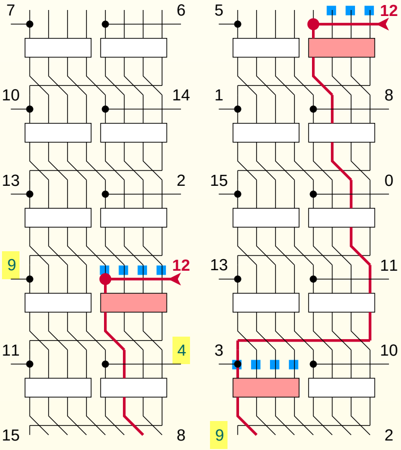
Using a difference with only 13 MSB set we reduce this to 108 bits.

$d =$ `0xdd080000` or $d =$ `0x22f80000`

Collisions: the principle of the attack

- Get three micro-collisions in branches 3 and 4.
  This leaves two message words M_4 and M_9 free, the rest is fixed
- Using different values of M_4 and M_9 compute branch 1 and hope that there is a single micro-collision in strand D in step 7.
- If a micro-collision there is found, compute the rest of the function and check the output difference.
  Note that the output differences have weights always ≤ 108

Obtaining micro-collisions in branches 3 and 4



To deal with branches 3 and 4 we have to:

1) Set values of

$$F_0^{(4)}, G_0^{(4)}, H_0^{(4)}.$$

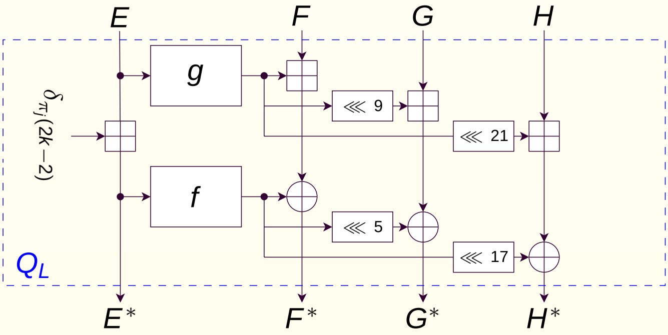
2) Set values of

$$A_4^{(4)}, B_4^{(4)}, C_4^{(4)}, D_4^{(4)}$$

3) Set values of

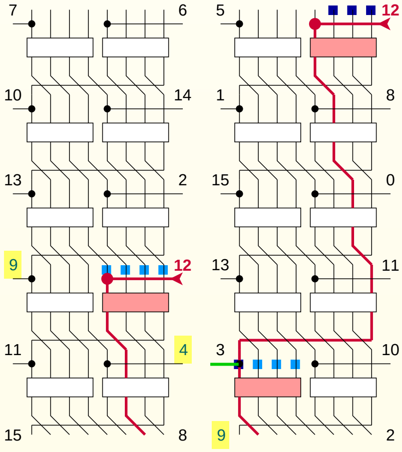
$$E_3^{(3)}, F_3^{(3)}, G_3^{(3)}, H_3^{(3)}$$

A property of Q-structures

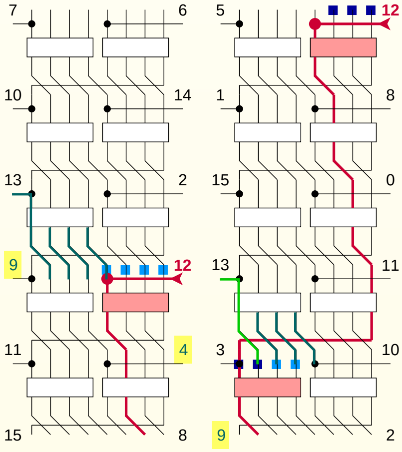


- We can set E^* to any value by adjusting the value of E
- We can set F^* to any value by adjusting the value of F (true for G, H too).

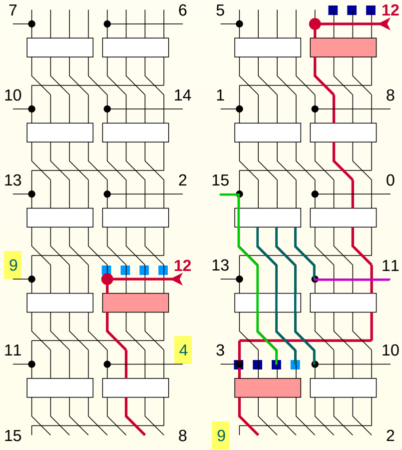
Adjusting the values: branch 4 before step 5



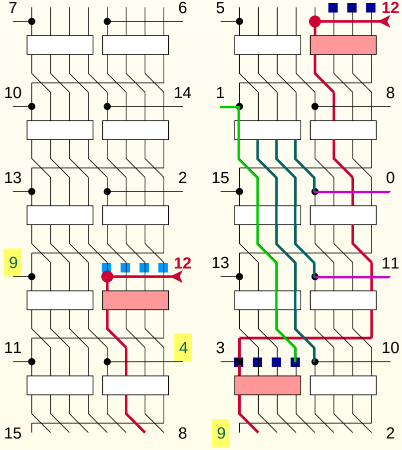
Adjusting the values: branch 4 before step 5



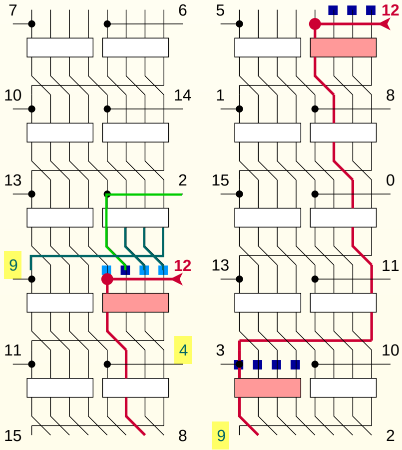
Adjusting the values: branch 4 before step 5



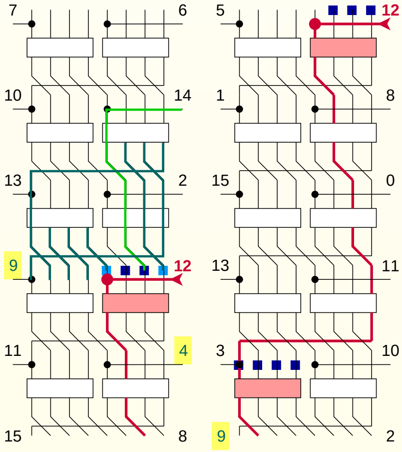
Adjusting the values: branch 4 before step 5



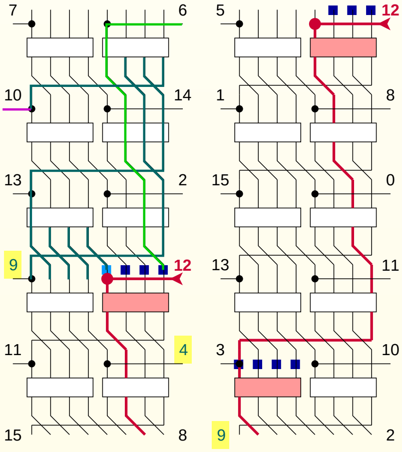
Adjusting the values: branch 3 before step 4



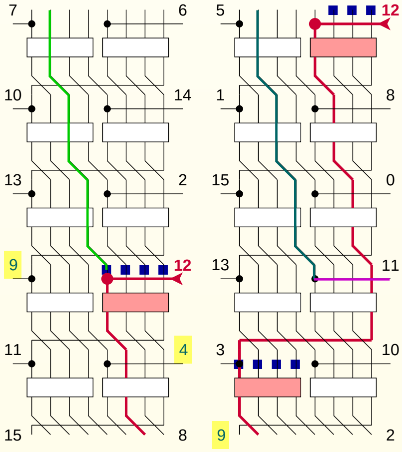
Adjusting the values: branch 3 before step 4



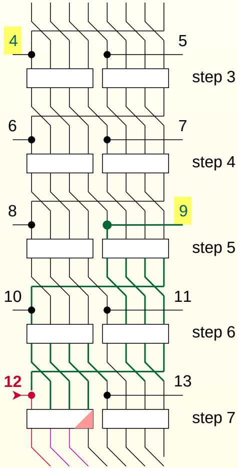
Adjusting the values: branch 3 before step 4



Adjusting the values: branch 3 before step 4



Passing branch 1



We use free words M_4 and M_9 to search for the values that yield a single micro-collision in step 7.

Outline

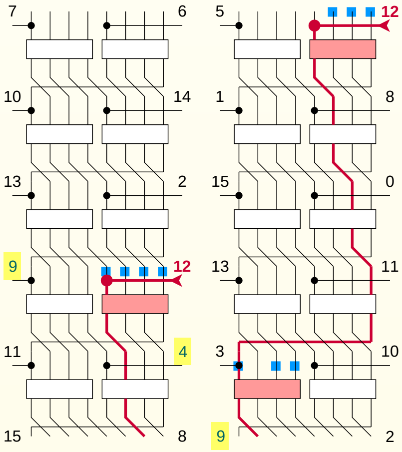
Why we are not happy?

Problem: During the attack, we have to adjust the value of B_0.

- This value depends on other values of message words so we cannot know it in advance or precompute it.
- We cannot use this method to obtain (near)collisions for the full hash function that needs a predefined IV.

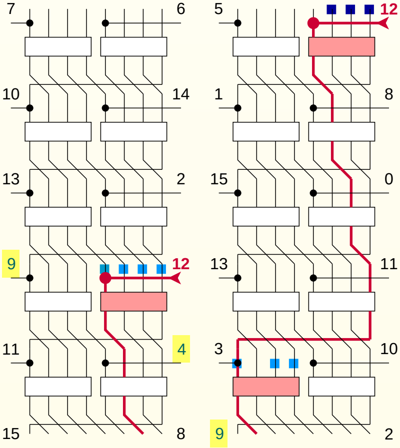
- We want to extend the attack to the full hash function.
- We want to avoid the need for modification of B_0.

Better message adjustment strategy



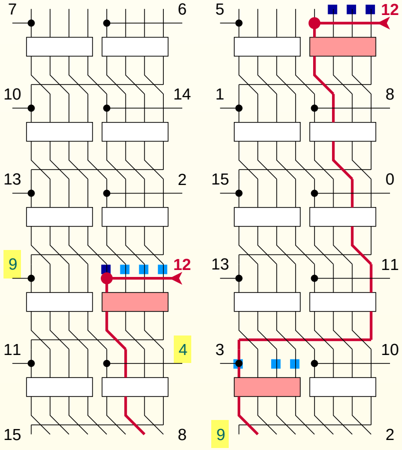
- Solve branch 4 step 1
- Deal with whole branch 3: use M_{13} to preserve the value of $E_3^{(3)}$
- Finish with branch 4
- In branch 4 leave a single micro-collision in $C_5^{(4)}$ to chance

New message adjustment: branch 4 step 1



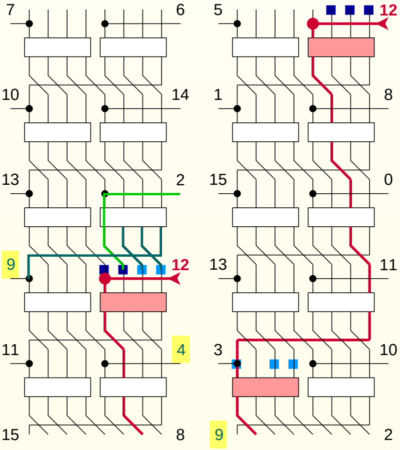
- Find a value x_1 that may lead to simultaneous microcollisions in Q_R in step 1 of branch 4
- Pick appropriate constants for F_0, G_0, H_0
- Set M_{12} to $E_0 - x_1$, $M'_{12} = M_{12} + d$

New strategy: branch 3 (1)

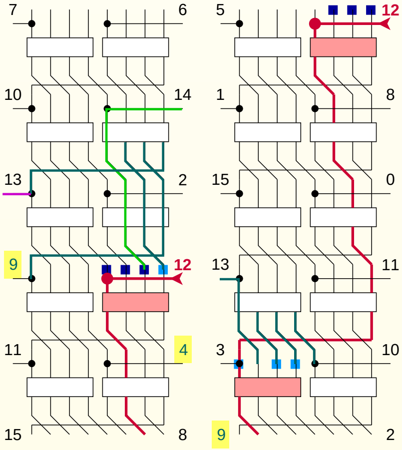


- Pick message words M_7, M_6, M_{10}, M_{14}, M_{13}, M_2 randomly and compute until step 4
- If $E_4^{(3)} + M_{12}$ does not allow for finding simultaneous micro-collisions, start over. [We need around 2^{23} trials]
- When it does, keep that value and later adjust values of $F_4^{(3)}$, $G_4^{(3)}$, $H_4^{(3)}$

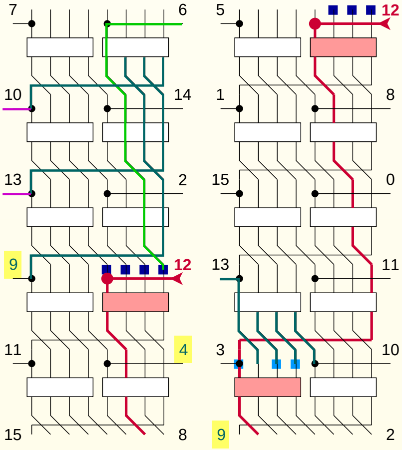
New strategy: branch 3, adjusting $F_3^{(3)}$



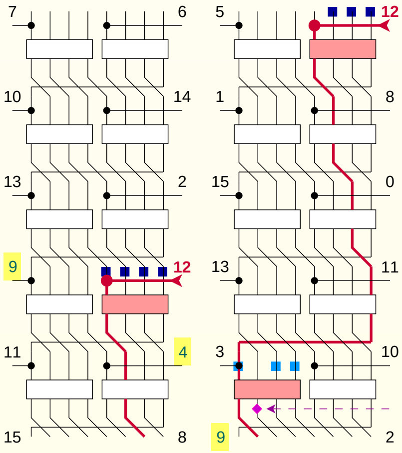
New strategy: branch 3, adjusting $G_3^{(3)}$



New strategy: branch 3, adjusting $H_3^{(3)}$

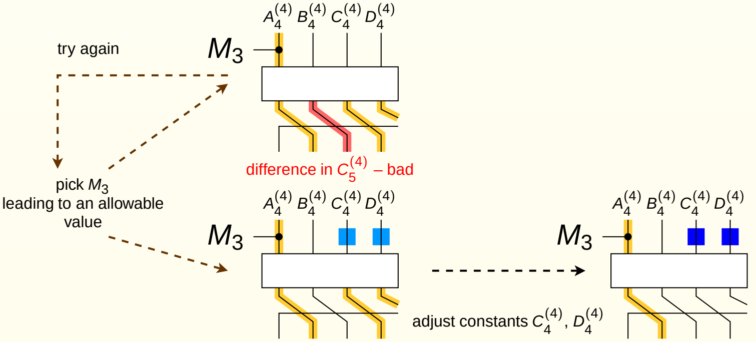


New strategy: branch 4



- Pick M_5, M_1, M_{15} randomly.
- Pick M_8, M_0, M_{11} that preserve the modular difference.
- Compute up to step 4.
- Keep picking allowable values of M_3 and testing if there is no difference in $C_5^{(4)}$
- Once we find a good value of M_3, we can adjust constants $C_4^{(4)}$ and $D_4^{(4)}$

Passing step 5 in branch 4



- We need around 2^{19} trials like that

New strategy: complexity of getting close hashes

- Work effort of passing branches 3 and 4 for using the difference $0x22f80000$:

$$< 2^{24} \cdot 2^{19} \cdot 2^{-3} = 2^{40}$$

  FORK evaluations.
- The second phase of the attack (branches 1 and 2) dominates with the complexity of 2^{58}.
- Conclusion: The new strategy for dealing with branches 3 and 4 does not affect the total complexity of the attack.

Fixing appropriate chaining values F_0, G_0, H_0

- We removed the need for the fourth initial chaining value to be used.
- The three values F_0, G_0, H_0 have to be set to one of possible constants required by simultaneous micro-collisions in step 1 of branch 4.
- If we require three particular values, we can achieve this in 2^{96} FORK evaluations by hashing a random prepended message block
- In fact we can do much better: any of the set of good constants for each register will do.

Estimating the probability of getting good constants

- Let $\mathcal{F}_a$, $\mathcal{G}_a$, $\mathcal{H}_a$ denote sets of constants that yield a micro-collision in line F, G, H for an allowable value a.
- Probability that a random IV value will match one of those values for each register F, G, H is

$$P = 1 - \prod_{a \in \mathcal{A}} \left(1 - \frac{|\mathcal{F}_a| \cdot |\mathcal{G}_a| \cdot |\mathcal{H}_a|}{2^{96}} \right)$$

- For original differences $d = \texttt{0xdd080000}$ and $d = \texttt{0x22f80000}$ it is equal to $P = 2^{-64.8}$,
- for other differences it may be much bigger, e.g. for $d = \texttt{0x3f6bf009}$ we have $P = 2^{-21.7}$.

Example of a near-collision

Example of a near-collision of weight 42 for the complete hash function FORK-256. The first block is used to obtain the desired values of chaining registers that enable the attack on the compression function.

M	2d4458a4	57976f57	3e44cfd9	1ab54cb2	7ec11870	173f6573	6141c261	7db20d3e
	2feeb74d	5fac87a6	61a73fa1	3454b23d	451d389b	78f061ec	7c32fb06	57ef1928
	79dcd071	39dc97f0	3a1bff42	031d364c	fef000e6	40873ef5	d0741256	649430cf
	97ef5538	3eab6a7e	b4f9cf72	9eba8257	4e84d457	5a6c49b6	ad1d9711	0f69afa2
M'	2d4458a4	57976f57	3e44cfd9	1ab54cb2	7ec11870	173f6573	6141c261	7db20d3e
	2feeb74d	5fac87a6	61a73fa1	3454b23d	451d389b	78f061ec	7c32fb06	57ef1928
	79dcd071	39dc97f0	3a1bff42	031d364c	fef000e6	40873ef5	d0741256	649430cf
	97ef5538	3eab6a7e	b4f9cf72	9eba8257	8df0c460	5a6c49b6	ad1d9711	0f69afa2
diff	00000000	83480012	32b4070c	681a1279	648600ad	00000000	00000000	00000000

Outline

Conclusions

Presented an improved attack on FORK-256.

- finds collisions for any value of IV
- breaks the full hash function
- practical for finding near-collisions

Thank you!