Cryptanalysis of Reduced Variants of the FORK-256 Hash Function*

Florian Mendel[1,**], Joseph Lano[2], and Bart Preneel[2]

[1] Graz University of Technology
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, A–8010 Graz, Austria
Florian.Mendel@iaik.tugraz.at
[2] Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC,
Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium
{Joseph.Lano,Bart.Preneel}@esat.kuleuven.be

Abstract. FORK-256 is a hash function presented at FSE 2006. Whereas SHA-like designs process messages in one stream, FORK-256 uses four parallel streams for hashing. In this article, we present the first cryptanalytic results on this design strategy. First, we study a linearized variant of FORK-256, and show several unusual properties of this linearized variant. We also explain why the linearized model can not be used to mount attacks similar to the recent attacks by Wang *et al.* on SHA-like hash functions. Second, we show how collision attacks, exploiting the non-bijectiveness of the nonlinear functions of FORK-256, can be mounted on reduced variants of FORK-256. We show an efficient attack on FORK-256 reduced to 2 streams and present actual colliding pairs. We expect that our attack can also be extended to FORK-256 reduced to 3 streams. For the moment our approach does not appear to be applicable to the full FORK-256 hash function.

1 Introduction

Recent results in cryptanalysis of hash functions [6,5] show weaknesses in many commonly used hash functions, such as SHA-1 and MD5. Therefore, the cryptanalysis of alternative hash functions is of great interest. In this article, we will study the hash function FORK-256. It was proposed by Hong *et al.* at FSE 2006 [2]. FORK-256 was designed to be resistant against known-attack strategies including the attack by Wang *et al.* used to break SHA-1 [5].

In this article, we present the first cryptanalytic results on FORK-256 and stream-reduced variants. On the one hand we explain why the linearized model can not be used to mount attacks similar to the attack of Wang *et al.* on SHA-1. All the characteristics we found in the linearized variant of the hash function have a low probability to hold in the original FORK-256 hash function. On the

* This previous work was in part supported by grant No.2005-S-062(2005) from the KISA(Korea Information Security Agency).
** This author is supported by the Austrian Science Fund (FWF), project P18138.

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 85–100, 2007.
http://dx.doi.org/10.1007/11967668_6
© Springer-Verlag Berlin Heidelberg 2007

other hand, we show several unusual properties in the linearized variant of the hash function, which are not common in the linearized variants of other hash functions, as for instance the SHA-family [3].

Furthermore, we show how collision attacks, exploiting the non-bijectiveness of the nonlinear functions of FORK-256, can be mounted on reduced variants of FORK-256. We show an efficient attack on FORK-256 reduced to 2 streams and present a colliding message pair. We expect that the attack can be extended to FORK-256 reduced to 3 streams.

The remainder of this article is structured as follows. A description of the hash function is given in Section 2. In Section 3, we show that the linearized variant of the FORK-256 has several unusual properties. Differential attacks on FORK-256 using the linearized variant for finding a suitable characteristic are studied in Section 4. In Section 5, we give a truncated differential which can be used to break stream-reduced variants of FORK-256. A sample colliding message pair for FORK-256 reduced to two streams is given in this section as well. Conclusions are presented in Section 6.

2 Description of the Hash Function FORK-256

The FORK-256 hash function was proposed by Hong *et al.* in [2]. It is an iterative hash function that processes 512-bit input message blocks and produces a 256-bit hash value. Unlike other commonly used hash functions, such as the SHA-family, it consists of 4 parallel streams which we denote B_1, B_2, B_3 and B_4. In each stream the state variables are updated according to the expanded message words and combined with the chaining variables after the last step, depicted in Fig. 1. In the following, we briefly describe the FORK-256 hash function. It basically consists of two parts: the message expansion and the state update transformation. A detailed description of the hash function is given in [2].

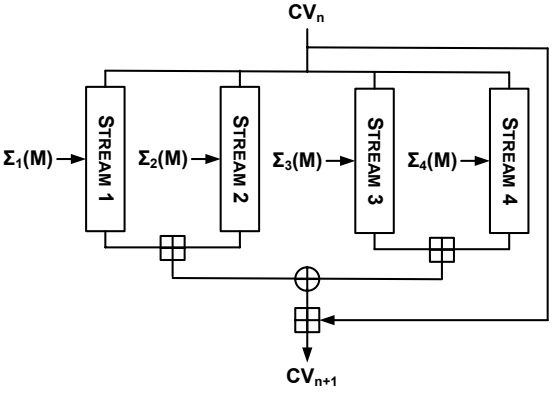

Fig. 1. Structure of FORK-256.

2.1 Message Expansion

The message expansion of FORK-256 is a permutation of the 16 message words m_i in each stream, where different permutations are used. The ordering of the message words for each stream is given by the permutations Σ_1, Σ_2, Σ_3 and Σ_4, where $\Sigma_j(M) = w_j = (m_{\sigma_j(0)}, \ldots, m_{\sigma_j(15)})$.

Table 1. Ordering of the message words.

step k	0		1		2		3		4		5		6		7	
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_1(i)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\sigma_2(i)$	14	15	11	9	8	10	3	4	2	13	0	5	6	7	12	1
$\sigma_3(i)$	7	6	10	14	13	2	9	12	11	4	15	8	5	0	1	3
$\sigma_4(i)$	5	12	1	8	15	0	13	11	3	10	9	2	7	14	4	6

2.2 State Update Transformation

The state update transformation starts from a (fixed) initial value IV of eight 32-bit registers and updates them in 4 parallel streams of 8 steps. In each step 2 message words are used to update the eight state variables. Fig. 2 shows one step of the state update transformation of FORK-256.

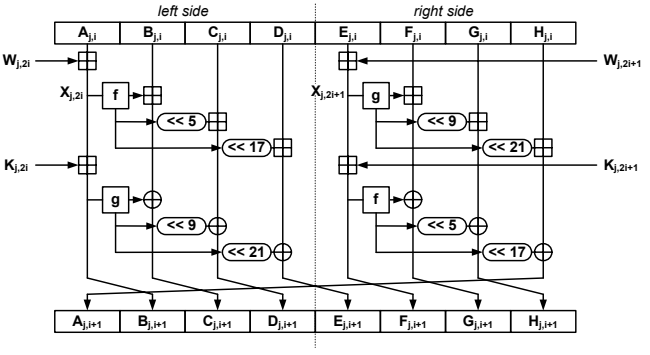

Fig. 2. Step i in stream B_j of FORK-256.

The non-linear functions f and g used in each step are defined as follows.

$$f(x) = x + (x \lll 7 \oplus x \lll 22)$$
$$g(x) = x \oplus (x \lll 13 + x \lll 27)$$

Table 2. Ordering of constants.

step i	$K_{1,2i}$	$K_{1,2i+1}$	$K_{2,2i}$	$K_{2,2i+1}$	$K_{3,2i}$	$K_{3,2i+1}$	$K_{4,2i}$	$K_{4,2i+1}$
0	δ_0	δ_1	δ_{15}	δ_{14}	δ_1	δ_0	δ_{14}	δ_{15}
1	δ_2	δ_3	δ_{13}	δ_{12}	δ_3	δ_2	δ_{12}	δ_{13}
2	δ_4	δ_5	δ_{11}	δ_{10}	δ_5	δ_4	δ_{10}	δ_{11}
3	δ_6	δ_7	δ_9	δ_8	δ_7	δ_6	δ_8	δ_9
4	δ_8	δ_9	δ_7	δ_6	δ_9	δ_8	δ_6	δ_7
5	δ_{10}	δ_{11}	δ_5	δ_4	δ_{11}	δ_{10}	δ_4	δ_5
6	δ_{12}	δ_{13}	δ_3	δ_2	δ_{13}	δ_{12}	δ_2	δ_3
7	δ_{14}	δ_{15}	δ_1	δ_0	δ_{15}	δ_{14}	δ_0	δ_1

Two step constants $K_{j,2i}$ and $K_{j,2i+1}$ are added in step i; the constants are different for each step of the stream. The order of the constants is different in each stream. The ordering of the constants for each stream is given in Table 2. For the actual values of the constants δ_0 to δ_{15} we refer to [2].

After the last step of the state update transformation, the chaining variables and the output values of the last step of the four streams are combined, resulting in the final value of one iteration (feed forward). The feed forward is a word-wise modular addition of the IV and the output of the state update transformation. The result is the final hash value or the initial value for the next message block.

3 L-FORK-256: A Linearized Variant of FORK-256

In this section, we analyze the linearized variant of FORK-256. We show that the linearized variant L-FORK-256 has several properties that are not common in the linearized variants of other hash functions. However, so far we do not see how these properties can be used in an attack on the original FORK-256 hash function. L-FORK-256 is constructed by replacing all modular additions in the hash function by XOR operations.

$$LH(CV_n, M_n) = CV_n \oplus \bigoplus_{j=1}^{4} B_j(CV_n, \Sigma_j(M_n)) \tag{1}$$

The 4 streams B_j can be described as follows.

$$B_j(CV_n, \Sigma_j(M_n)) = CV_n\mathrm{A} \oplus \Sigma_j(M_n)\mathrm{B} \oplus \mathrm{c}_j \tag{2}$$

$$= CV_n\mathrm{A} \oplus M_n\mathrm{S}_j\mathrm{B} \oplus \mathrm{c}_j \tag{3}$$

with S_j some permutation matrices, and A, B matrices that describe the action of L-FORK-256 on the chaining value input, respectively the message input. The

matrices A are of little importance, since:

$$LH(CV_n, M_n) = CV_n \oplus \bigoplus_{j=1}^{4} CV_n\mathrm{A} \oplus M_n\mathrm{S}_j\mathrm{B} \oplus \mathrm{c}_j \tag{4}$$

$$= CV_n \oplus M_n(\mathrm{S}_1 \oplus \mathrm{S}_2 \oplus \mathrm{S}_3 \oplus \mathrm{S}_4)\mathrm{B} \oplus (\mathrm{c}_1 \oplus \mathrm{c}_2 \oplus \mathrm{c}_3 \oplus \mathrm{c}_4) \tag{5}$$

$$= CV_n \oplus M_n\mathrm{SB} \oplus \mathrm{c}, \tag{6}$$

with $\mathrm{S} = \mathrm{S}_1 \oplus \mathrm{S}_2 \oplus \mathrm{S}_3 \oplus \mathrm{S}_4$ and $\mathrm{c} = \mathrm{c}_1 \oplus \mathrm{c}_2 \oplus \mathrm{c}_3 \oplus \mathrm{c}_4$.

3.1 Action on special message words

The four streams of FORK-256 are quite similar. Only the ordering of the constants and the message words is different in each stream.

Observation 1 *If we have a message M with repeating message words $M = m_0, m_1, \ldots, m_{15}$ with $m_i = m_j \, \forall i, j$, then $\Sigma_1(M) = \Sigma_2(M) = \Sigma_3(M) = \Sigma_4(M)$.*

Observation 2 *In L-FORK-256, for inputs that are repeating messages, we have that $B_1 = B_2 = B_3 = B_4$.*

Consequently, the description (2) of a stream can be reduced to:

$$B_j(CV_n, M_n) = CV_n\mathrm{A} \oplus M_n\mathrm{B} \oplus \mathrm{c}_j \tag{7}$$

and the description(6) of L-FORK-256 becomes

$$LH(CV_n, M_n) = CV_n \oplus \mathrm{c}, \tag{8}$$

which is independent of M_n.

Observe that having a repeating message as input is a *sufficient* condition for this effect, but it is not a necessary condition. It can be verified that (8) holds whenever the input message satisfies the following 12 conditions:

$$\begin{aligned}
&m_6 = m_7 = m_9 \\
&m_3 = m_5 = m_{12} = m_{13} \\
&m_1 = m_2 = m_3 \oplus m_6 \oplus m_{15} \\
&m_0 = m_8 = m_3 \oplus m_6 \oplus m_{14} \\
&m_{10} = m_{11} = m_3 \oplus m_{14} \oplus m_{15} \\
&m_4 = m_6 \oplus m_{14} \oplus m_{15} \ .
\end{aligned} \tag{9}$$

3.2 Fixed-points

In L-FORK-256 we can easily construct a fix-point for any value of the chaining variables CV_n.

Theorem 1. *A two-block message can be used to construct a two-step fixed point in L-FORK-256.*

Proof: By combining two repeated messages, we can construct a fixed point for L-FORK-256. Let $M_1 = m\| \cdots \|m$ and $M_2 = \overline{m}\| \cdots \|\overline{m}$, then

$$\begin{aligned} CV_n &= LH(CV_{n-1}, M_2) \\ &= CV_{n-1} \oplus \mathrm{c} \\ &= LH(CV_{n-2}, M_1) \oplus \mathrm{c} \\ &= (CV_{n-2} \oplus \mathrm{c}) \oplus \mathrm{c} \\ &= CV_{n-2} \ . \end{aligned}$$

□

Theorem 2. *Two fixed-points and an arbitrary message block M_3 can be used to produce a collision in L-FORK-256.*

Proof: Let M_1, M_2 be repeating messages and let M_3 be an arbitrary message. Define $y = \text{L-FORK-256}(IV, M_3)$. Then the hash values of $M = M_3|M_2|M_1$ and $M^* = M_2|M_1|M_3$ are given by:

$$\begin{aligned} CV_2 &= LH(IV, M_1) = \mathrm{c} \oplus IV \\ CV_3 &= LH(CV_2, M_2) = \mathrm{c} \oplus \mathrm{c} \oplus IV = IV \\ CV_4 &= LH(CV_3, M_3) = LH(IV, M_3) = y \end{aligned}$$

$$\begin{aligned} CV_2^* &= LH(IV, M_3) = y \\ CV_3^* &= LH(CV_2^*, M_1) = \mathrm{c} \oplus y \\ CV_4^* &= LH(CV_3^*, M_2) = \mathrm{c} \oplus \mathrm{c} \oplus y = y \end{aligned}$$

□

3.3 Output dependencies

In L-FORK-256 we found several output dependencies. However, statistical tests show that these are not present in the original hash function.

Observation 3 *Three linear dependencies exist between the 256 output bits of L-FORK-256. These 3 dependencies are the following:*

$$\sum_{i=0}^{127} w_{2i+1} = 0, \quad \sum_{i=1}^{128} w_{2i} = 0, \quad \sum_{i=33}^{160} w_i = 0 \ . \tag{10}$$

From (10) it follows that the parity of the output of L-FORK-256 is constant.

4 Differential analysis

In this section, we analyze the security of FORK-256 against differential attacks. We study the impact of the type of attack that was used by Wang *et al.* to break SHA-1 [5]. The attack can be summarized as follows. First, find a collision producing characteristic with high probability in the linearized variant of the hash function. Second, use random trials to find a message pair that follows the linear characteristic.

4.1 Finding a characteristic

Finding a collision in the linearized variant of FORK-256 is not difficult since it depends only on the differences in the message words. Two messages M and $M^* = M \oplus \Delta$ collide if and only if:

$$h_1^* \oplus h_1 = (\mathrm{M} \oplus \Delta)\mathrm{SB} \oplus IV \oplus \mathrm{c} \oplus (\mathrm{MSB} \oplus IV \oplus \mathrm{c}) = \Delta\mathrm{SB} = 0 \qquad (11)$$

The matrices S and B are described in Section 3. A collision-producing difference can be found by solving the set of linear equations given in (11).

Furthermore, the following theorem shows that every near-collision [1] can be turned into a collision with only a minor increase in complexity.

Theorem 3. *For L-FORK-256, every two-block message difference of the form* (Δ, Δ) *produces a two-block collision.*

Proof:

$$\begin{aligned}
\text{L-FORK-256}((\mathrm{M}_1 \oplus \Delta)\|(\mathrm{M}_2 \oplus \Delta)) &= LH(LH(IV, \mathrm{M}_1 \oplus \Delta), \mathrm{M}_2 \oplus \Delta) \\
&= LH(IV \oplus (\mathrm{M}_1 \oplus \Delta)\mathrm{SB} \oplus \mathrm{c}, \mathrm{M}_2 \oplus \Delta) \\
&= (IV \oplus (\mathrm{M}_1 \oplus \Delta)\mathrm{SB} \oplus \mathrm{c}) \oplus (\mathrm{M}_2 \oplus \Delta)\mathrm{SB} \oplus \mathrm{c} \\
&= (IV \oplus \mathrm{M}_1\mathrm{SB} \oplus \mathrm{c}) \oplus \mathrm{M}_2\mathrm{SB} \oplus \mathrm{c} \\
&= \text{L-FORK-256}(\mathrm{M}_1\|\mathrm{M}_2)
\end{aligned}$$

□

4.2 Minimizing the number of conditions

It is difficult to bound the number of conditions that have to be fulfilled in order to guarantee that the message follows the characteristic in the original FORK-256 hash function. A commonly used approach is to use the Hamming weight of the expanded message to approximate the attack complexity. This approximation is useful for SHA-1, but does not hold in the case of FORK-256. A property of L-FORK-256 is that collision producing differences with very low weight in the message, can easily result in very high weights in the internal states of the four separate streams. Hence, to get a more accurate approximation of the final attack complexity the weight of the internal state variables has to be considered as well.

Table 3. Smallest Hamming weight found for FORK-256 and reduced variants.

stream	collision	near-collision
one stream	52	35
1 & 2	384	135
3 & 4	384	288
full hash	1280	704

We used algorithms from coding theory to find characteristics with low Hamming weight. Even if the algorithms are probabilistic they are expected to do a good job as they did in the case of SHA-1 [4]. In Table 3, the smallest found weights for FORK-256 and reduced variants are shown.

Converting the Hamming weights to numbers of conditions is complicated by the following issues.

1. One equation may cover several conditions imposed on bits in identical positions of several registers.
2. One equation may cover conditions imposed on bits in neighboring positions of several registers.
3. Conditions imposed on bits in the MSB position of a 32-bit word may be fulfilled automatically, due to carry overflow effects.
4. Some conditions might be reworked to linear conditions involving only message bits. Such conditions are easy to fulfill and don't contribute to the probability of the characteristic.

A rough estimation of the work factor can be made by taking the Hamming weight of the internal state variables and the weight of the expanded message. For FORK-256 with all four streams, the estimate probability for a random message having the chosen differences to follow the linear characteristic and to collide is 2^{-1280}. The probability for a near-collision is 2^{-704}. These probabilities are too small to pose a realistic threat to the hash function. Note that the smallest found Hamming weight for one stream is equal to the local collision given in [2].

4.3 A differential characteristic for 4 steps with probability 1

For four (out of eight) steps of FORK-256 there exists a characteristic with probability 1. If we choose the same difference δ in all message words $m_i^* = m_i \oplus \delta$, for $i = 0, \ldots, 15$ and differences in all chaining variables $A'_{j,k} = \cdots = H'_{j,k} = \delta$ for $j = 1, \ldots, 4$ for a $k < 5$ then we have after 4 steps:

$$A'_{j,k+4} = B'_{j,k+4} = \cdots = H'_{j,k+4} = 0 \quad \text{for } j = 1, \ldots, 4$$

This characteristic holds with probability 1 for $\delta =$ `80000000`. For all the other cases the probability of the characteristic is approximately $2^{-\mathrm{HW}(\delta)*12*4}$. It is difficult to use this characteristic to break FORK-256. To construct a collision we would need a characteristic (not necessary linear) for the first 4 steps in each stream that produces the needed differences in all the chaining variables.

5 Truncated differential attack

The function f and g map a 32-bit input word to a 32-bit output. By design, these functions are not invertible (although their linear approximations are). This means we could try to construct collisions by using values $x \neq x^*$ which have the property that $f(x) = f(x^*)$ and $g(x+\delta_k) = g(x^*+\delta_k)$. For the analysis, it is most convenient to consider as difference operation the modular difference:

$$x' = x - x^* \bmod 2^{32} \ . \tag{12}$$

5.1 One Stream

We first consider one stream of the hash function. For the attack, we want to exploit a truncated characteristic of the following form:

$$\begin{aligned}
&(0,0,0,0,0,0,0,0) && && \text{(13a)}\\
&\quad\Downarrow && x \neq x^*, f(x) = f(x^*), g(x+\delta_k) = g(x^*+\delta_k) &\\
&(0,x',0,0,0,0,0,0) && && \text{(13b)}\\
&\quad\Downarrow && \oplus \text{ changes the difference} &\\
&(0,0,\alpha,0,0,0,0,0) && && \text{(13c)}\\
&\quad\Downarrow && \oplus \text{ changes the difference} &\\
&(0,0,0,\beta,0,0,0,0) && && \text{(13d)}\\
&\quad\Downarrow && \oplus \text{ changes the difference} &\\
&(0,0,0,0,\gamma,0,0,0) && && \text{(13e)}
\end{aligned}$$

The difference in the 5th register can be canceled by adding a message block with a suitable difference. Alternatively, the characteristic can be concatenated with a rotated version of itself.

The probability of the first step depends on the difference x' and on the value of the constant δ_k. There are many δ_k values for which the probability of the differential equals 0, but there is also a significant fraction for which one can find at least one x' such that the probability becomes greater than zero. We call a δ_k value *weak*, if there exists at least one difference x' for which the probability of the differential is greater than 0.

5.2 Weak Constants

By doing an exhaustive search we found 2 *weak* constants for the right side and 4 *weak* constants for the left side of one stream of FORK-256. The *weak* constants are shown in Table 4.

These weak constants can be used to break one stream of FORK-256. To break FORK-256 with more than one stream we would need more weak constants (see Section 5.4). Therefore, we have to extend the concept of weak constants as described in the next section.

Table 4. Weak constants in FORK-256.

side	constant	x	x^*
left side	δ_2	AEB691E5	06DEF69A
	δ_1	6FF2F3E9	4B4D2A05
	δ_3	67EAC4D8	27A61343
	δ_7	20D331A5	04549CDC
right side	δ_{10}	D73BC777	445C5563
	δ_{14}	EDFD4D5B	BE452586

5.3 Semi-Weak Constants: Extending the Idea of Weak Constants

Instead of searching for pairs x, x^* having zero differences at the output of f and g, we can extend the search to pairs x, x^* such that:

$$\begin{aligned} f(x) \oplus f(x^*) &= \Delta f \\ g(x + \delta_k) \oplus g(x^* + \delta_k) &= \Delta g \end{aligned} \tag{14}$$

and

$$\begin{aligned} \Delta f &= \Delta g \\ \Delta f \ll 5 &= \Delta g \ll 9 \\ \Delta f \ll 17 &= \Delta g \ll 21 \ , \end{aligned}$$

where the last condition is equivalent to $\Delta f = \Delta g \ll 4$. We found many pairs x, x^* and constants δ_k which fulfill (14). A subset of these are given in Table 7 and Table 8 in the appendix. We restrict the search to values Δf and Δg with low Hamming weight to keep the final attack complexity low. Note that additional conditions have to be fulfilled to guarantee that the differences at the output of f and g cancel out within one step. In detail, the probability that the differences cancel out is approximately $2^{-3 \cdot \mathrm{HW}(\Delta f)}$. Note, that for $\Delta f \neq 0$ the minimal Hamming weight is 8.

5.4 The Full Hash Function

If we consider the full hash, then we have to take into account two effects:

1. Due to the different permutations Σ_j, the message blocks enter in the different streams at different steps. This property complicates the attack.
2. Due to the final addition of the streams, we can convert near-collisions for each of the streams into collisions for the full hash. This property facilitates the attack.

If we consider a variant of FORK-256 where the output of the functions f and g are not considered, then there are no interactions between the different registers in the streams. For this variant, we can easily construct a collision. Note that for each $x_{j,i}$ which has a non-zero difference we need one weak constant to

guarantee that the original FORK-256 hash function behaves like the variant. To minimize the number of (needed) *weak* constant, we have to minimize the number of differences in $x_j = (x_{j,0}, \ldots, x_{j,15})$, for $j = 1, 2, 3, 4$. In Table 5, we list the best results we found for FORK-256 and stream-reduced variants. Since we would need 12 weak constants to break the original FORK-256 hash function, this attack strategy is not applicable to FORK-256. We expect a complexity of at least $2^{12 \cdot 3 \cdot \mathrm{HW}(\Delta f)} \geq 2^{288}$ applications of the compression function to find a collision in FORK-256. However, FORK-256 reduced to 2 streams can be broken easily with this method as shown in Section 5.5.

Table 5. Number of weak constants needed to produce a collision in FORK-256 and stream-reduced variants. Note a 'x' denotes a difference in the word $x_{j,i}$ in stream j.

differences	stream 1	stream 2	stream 3	stream 4
m_6, m_{12}	------x-----x--x	----------x-x-	-x-----xx-----x-	-x------x------x
m_2, m_{12}	--x--------xx---	--------x-----x-	-----x-x----x-x-	-x------x--x----
m_9, m_{13}	---------x---x--	---x-----xx-----	----x-x------x-x	------x---x----x
m_2, m_6	--x---x----x---x	--------x---x---	-x---x--x---x---	----------x---x
m_4, m_7	----x--x-----xx-	-------x-----xx-	x---------------	
m_0, m_{10}	x--------xx-----	-----x----x-x---		-----x---x--x---
m_3, m_8, m_9, m_{10}	---x----xx------		--x---x--------	---x----xx------
m_5, m_{10}		-----x-----xx---	--x--------xx---	x---------------
m_3, m_{12}		------x------xx	-------x-----xx	-x--------------
m_0, m_9	x---------------	---x-----------		
m_{13}	------------x--		----x--------x--	
m_5	-----x------x---		-----------x---	
m_6	------x--------x			---------------x
m_2	--x--------x----			----------x----
m_{12}		-------------x-	-------x------x-	
m_3		------x--------x	---------------x	
m_4		-------x------x-		-------------x-
m_9		---x------x-----		----------x-----
m_{14}, m_{15}			---x----------	----x----------

5.5 A collision for two streams of FORK-256

In this section, we present a collision for FORK-256 reduced to two streams using the attack strategy described before. In the following, we will describe how to construct a collision in FORK-256 reduced to stream 1 and stream 2. Note that the attack would work similar to break FORK-256 reduced to 2 other streams.

Considering the ordering of the message words in stream 1 and stream 2 (see Table 1), we see that the distance (in number of steps) between m_0 and m_9 is 4 in both streams. Hence, we need only two weak constants in the attack. The attack can be summarized as follows.

1. Choose $x_{1,0} =$ `7AB8131D` and $x^*_{1,0} =$ `728D1B48` and calculate $B_{1,1}$ and $B^*_{1,1}$.
2. Choose $x_{2,3} =$ `E2E5A2A9` and $x^*_{2,3} =$ `A6378BEC` and calculate $F_{2,2}$ and $F^*_{2,2}$.
3. Choose suitable values for $x_{1,2}$, $x_{1,4}$ and $x_{1,6}$ such that $E'_{1,4} = -x'_{2,3}$.

4. Choose suitable values for $x_{2,5}$, $x_{2,7}$ and $x_{2,9}$ such that $A'_{2,5} = -x'_{1,0}$.
5. We have 24 conditions on $B_{1,0}, C_{1,0}, D_{1,0}$ that have to be fulfilled to guarantee that the differences at the output of f and g cancel out in step 0 of stream 1. Therefore, we use an arbitrary first message block to get suitable values for $B_{1,0}, C_{1,0}, D_{1,0}$ that satisfy all necessary conditions. To find this first message block takes at most 2^{24} hash computations.
6. We also have 24 conditions on $F_{2,1}, G_{2,1}, H_{2,1}$ in stream 2 that have to be fulfilled to guarantee that the differences at the output of f and g cancel out in step 1 of stream 2. Therefore, we have to modify $x_{2,1}$ to satisfy these conditions. We can find a suitable value for $x_{2,1}$ in at most 2^{24} trials.
7. Calculate m_i for $i = 0, \ldots, 15$ from the x-values calculated in step 1-6.

$$
\begin{aligned}
m_0 &= w_{1,0} = x_{1,0} - A_{1,0} \\
m_{15} &= w_{2,1} = x_{2,1} - E_{2,0} \\
m_2 &= w_{1,2} = x_{1,2} - A_{1,1} \\
m_9 &= w_{2,3} = x_{2,3} - E_{2,1} \\
m_4 &= w_{1,4} = x_{1,4} - A_{1,2} \quad * \\
m_{10} &= w_{2,5} = x_{2,5} - E_{2,2} \\
m_6 &= w_{1,6} = x_{1,6} - A_{1,3} \\
m_4 &= w_{2,7} = x_{2,7} - E_{2,3} \quad * \\
m_{13} &= w_{2,9} = x_{2,9} - E_{2,4}
\end{aligned}
$$

Note that there are 2 conditions on m_4. To satisfy both conditions we calculate first $m_4 = w_{1,4} = x_{1,4} - A_{1,2}$ and then we use m_8 to modify $E_{2,3}$ such that $m_4 = w_{1,7} = x_{2,7} - E_{2,3}$ holds.

With this method we can easily construct collisions in the FORK-256 variant. Once we have fixed the values of the chaining variables by using an arbitrary first message block and have determined $x_{1,0}$, $x_{1,2}$, $x_{1,4}$, $x_{1,6}$, $x_{2,1}$, $x_{2,3}$, $x_{2,5}$, $x_{2,7}$, and $x_{2,9}$ we can construct many collisions by solving the system of equations given in step (7) of the attack. We can construct about $2^{(16-9)\cdot 32}$ colliding message pairs once we have fixed all the x-values. This can be compared to having 224 *neutral*

Table 6. A colliding message pair for FORK-256 reduced to two streams.

h_0	06A09E667 0BB67AE85 03C6EF372 0A54FF53A 0510E527F 09B05688C 01F83D9AB 05BE0CD19
M_0	0F427DBAA 06FBF0CB7 0413F646C 0FCE4800E 0AF327AFD 05CB1B99A 00C879908 0FD5EA595 0BA603C95 06CF74DC6 0516E4AD5 01E43C9B5 03A112367 0258259E8 0FC3FA69D 0CD4F8D0C
h_1	06A09E667 0C1F86BBC 0D2856B94 052847CA9 0B8D977FE 0EE42EED7 0A309479B 05C5A4DA8
M_1	010AE2CB6 000000000 010ABB697 000000000 0197E717C 000000000 01FDE8BA2 000000000 0D4A419E3 0E3082DF1 0E7C9B7DB 000000000 000000000 0B93DF199 000000000 0314E6339
M_1^*	0088334E1 000000000 010ABB697 000000000 0197E717C 000000000 01FDE8BA2 000000000 0D4A419E3 0A65A1734 0E7C9B7DB 000000000 000000000 0B93DF199 000000000 0314E6339
M'_1	0082AF7D5 000000000 000000000 000000000 000000000 000000000 000000000 000000000 000000000 03CAE16BD 000000000 000000000 000000000 000000000 000000000 000000000
h_2	06A09E667 06D320398 00E1A7F40 0A359E80E 0E029DE72 019F5C484 032084418 0836E2FD8
h_2^*	06A09E667 06D320398 00E1A7F40 0A359E80E 0E029DE72 019F5C484 032084418 0836E2FD8

bits [1] in the message. In Table 6, we give a colliding message for FORK-256 reduced to the first 2 streams.

Complexity Analysis. In this section, we will give a detailed complexity analysis of the attack on FORK-256 reduced to 2 streams. The attack basically consists of 2 parts:

1. Find the values of Table 7 and Table 8 in the appendix
2. Find suitable x-values.

The first part of the attack has complexity of at most $2^{32} \cdot 4 = 2^{34}$ computations of f and g. This is equivalent to at most 2^{28} computation of the compression function of FORK-256. Note that the real complexity might be much lower, since g is only calculated if Δf is correct. While the first part of the attack is computational expensive, the second part of the attack has a comparable low complexity. For each difference in $x_{j,i}$ for $j = 1, 2, 3, 4$ and $i = 0, \ldots, 15$ we have to fulfill 24 conditions on the chaining variables and further find 3 suitable x-values to guarantee the that the difference cancel out after 4 steps. Therefore, we need 9 x-values in the attack on the first 2 streams. To find all these x-values and calculating the first block to fulfill all conditions on the chaining variables take us at most $7 \cdot 2^{32}$ calculations of f and g, and 2^{24} hash computations. Note that the probability for finding a suitable x-value is much higher than 2^{-32} in practice. Thus, the complexity will be much lower than 2^{29} for this part of the attack. Hence, the final attack complexity for both parts of the attack is about 2^{29} for the first collision. Many other collisions can be constructed with probability 1 afterward.

Improving/Extending the Attack. There are several ways to improve the attack. In the following we list some of them:

1. The attack can be modified to construct collisions in FORK-256 reduced to two other streams.
2. As shown by way of an example, the degrees of freedom (the number of neutral bits) we have in the attack on 2 streams is quite large. Thus, we can try to extend the attack to 3 streams of FORK-256.
3. Since the number of needed weak constants is too large for an attack on the original FORK-256 hash function, we could try to construct a near-collision in the hash function (only 6 weak constants needed).
4. Instead of searching for values (x, x^*) for which $\Delta f = \Delta g$, we can extend the search to values for which $\Delta f \neq \Delta g$, but the differences cancel out due to carries of the modular addition. Therefore, we have to find a good method to reduce the search space and the runtime for finding these values, respectively.

6 Conclusions

In this article we presented the first cryptanalytic results on the hash function FORK-256. We showed that the linearized variant of FORK-256 has several

unusual properties which do not exist in the linearized variants of other hash functions. We also explained why the linearized model can not be used to mount attacks similar to the recent attacks by Wang *et al.* on SHA-like hash functions.

Furthermore, we showed how collision attacks, exploiting the non-bijectiveness of the nonlinear functions of FORK-256, can be mounted on reduced variants of FORK-256. We presented an efficient attack on FORK-256 reduced to 2 streams. Moreover, we expect that our attack can also be extended to FORK-256 reduced to 3 streams. For the moment our approach does not appear to be applicable to the full FORK-256 hash function.

However, this does not prove that FORK-256 is secure. Further analysis is required to get a good view on the security margins of FORK-256.

Acknowledgements

The authors wish to thank Christophe De Cannière, Christian Rechberger, Norbert Pramstaller, Vincent Rijmen, and the anonymous referees for useful comments and discussions.

References

1. Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *LNCS*, pages 290–305. Springer, 2004.
2. Deukjo Hong, Jaechul Sung, Seokhie Hong, Sangjin Lee, and Dukjae Moon. A New Dedicated 256-bit Hash Function: FORK-256. In Matt Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Proceedings*, volume 4047 of *LNCS*, pages 195–209. Springer, 2006.
3. National Institute of Standards and Technology (NIST). FIPS-180-2: Secure Hash Standard, August 2002. Available online at `http://www.itl.nist.gov/fipspubs/`.
4. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Exploiting Coding Theory for Collision Attacks on SHA-1. In Nigel P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, volume 3796 of *LNCS*, pages 78–95. Springer, 2005.
5. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
6. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.

Table 7. List of (semi) weak constants for the left side of the stream.

side	constants	$\Delta f = \Delta g$	x	x^*
left side	δ_0	11111111	7AB8131D	728D1B48
	δ_0	88888888	B2D7C3DA	3B10A457
	δ_1	11111111	DC7A5519	38EC29EF
	δ_1	11111111	E02717D6	9FED7307
	δ_1	44444444	C2886A61	545D72A2
	δ_1	88888888	8DC83B78	1C547838
	δ_2	22222222	650CA295	4605419A
	δ_2	44444444	BEF57D44	2CF60FEB
	δ_3	22222222	E3E8525C	4CA6452C
	δ_3	44444444	8EA41F57	642967E8
	δ_3	44444444	7B080CA7	304EB46C
	δ_4	22222222	64B41C80	4D83EDBA
	δ_5	44444444	F92C0421	46119614
	δ_5	88888888	130F16FD	113044A8
	δ_6	22222222	8DD2989F	3FC9AB68
	δ_8	11111111	FE26B64C	CA52EA30
	δ_8	11111111	AD85682B	609F1D2F
	δ_8	22222222	BAF886F0	4BAF0F68
	δ_8	44444444	B14939BC	0D0B62B8
	δ_8	44444444	CE341C7A	AC04B7A3
	δ_9	11111111	B16A5B43	97949A93
	δ_9	44444444	FEF6F543	4D044E8C
	δ_9	88888888	7AB68B12	68C08524
	δ_{10}	88888888	F542812D	71F08875
	δ_{11}	88888888	50411ED1	23B25243
	δ_{11}	44444444	F7DF0AAC	7C65633B
	δ_{11}	22222222	4AB0742B	17E1B95C
	δ_{12}	44444444	CE44B12D	8EDD5A2B
	δ_{12}	11111111	E940C5B8	C0304FF9
	δ_{12}	44444444	86B755E0	73EF1636
	δ_{12}	11111111	5566F6BE	3F3136F2
	δ_{13}	22222222	48A7C925	279A5753
	δ_{13}	44444444	AE36E874	12D10ADA
	δ_{13}	22222222	FACB2049	F947DBD2
	δ_{14}	22222222	E545F52D	46511638
	δ_{14}	88888888	678E6534	02271592
	δ_{14}	22222222	CD454CD7	3D6A82F0
	δ_{14}	88888888	F3508338	C32F4A66
	δ_{15}	22222222	68B8B75D	46A9FF78
	δ_{15}	44444444	F7C30C12	56C94895

Table 8. List of (semi) weak constants for the right side of the stream.

side	constants	$\Delta f = \Delta g$	x	x^*
	δ_0	11111111	87311631	CF174A81
	δ_0	88888888	9AE34AAD	E9BBB576
	δ_0	88888888	7078180F	CA9E34B0
	δ_1	22222222	8A7B922A	8515FD65
	δ_1	44444444	49E17C65	D2FAFF64
	δ_3	11111111	B93446E3	3AEE54AD
	δ_3	44444444	47233861	190D5338
	δ_4	22222222	5C40490B	4D886BE9
	δ_5	44444444	8673BC03	636F7E88
	δ_5	44444444	CC6F6AFE	AAF1DE10
	δ_6	22222222	249BD62F	717C851E
	δ_6	22222222	E5C43BC9	9C7E42D8
	δ_{10}	11111111	F0D362CD	E15DA3A4
	δ_{12}	11111111	E2E5A2A9	A6378BEC
right side	δ_{12}	22222222	02FCA84E	A822C4E6
	δ_{12}	22222222	F150D9B4	DA63A7EA
	δ_{12}	22222222	24193476	93C46D96
	δ_{13}	22222222	B43BA7D4	A491977E
	δ_{13}	22222222	DF3661E0	A6F79CF2
	δ_{13}	22222222	DF3661A0	A6F79CB2
	δ_{13}	11111111	28E0B213	C91908C7
	δ_{13}	44444444	13BB91E2	B7F968E6
	δ_{14}	88888888	99394D77	73F1C4C9
	δ_{14}	44444444	B5FAEFDB	6A6FE934
	δ_{14}	88888888	AC9747A5	77F40F98
	δ_{15}	11111111	1D405A4E	0BAE9B75
	δ_{15}	22222222	C0D7FE3A	53480ECC
	δ_{15}	88888888	AE4B89E3	6EDF99DA
	δ_{15}	88888888	AE0B89E3	6E9F99DA