Collisions for Round-Reduced LAKE [⋆]

Florian Mendel and Martin Schläffer

Graz University of Technology,
Institute for Applied Information Processing and Communications,
Inffeldgasse 16a, A-8010 Graz, Austria
{florian.mendel,martin.schlaeffer}@iaik.tugraz.at

Abstract. LAKE is a family of cryptographic hash functions presented at FSE 2008. It is an iterated hash function and defines two main instances with a 256 bit and 512 bit hash value. In this paper, we present the first security analysis of LAKE. We show how collision attacks, exploiting the non-bijectiveness of the internal compression function of LAKE, can be mounted on reduced variants of LAKE. We show an efficient attack on the 256 bit hash function LAKE-256 reduced to 3 rounds and present an actual colliding message pair. Furthermore, we present a theoretical attack on LAKE-256 reduced to 4 rounds with a complexity of 2^{109}. By using more sophisticated message modification techniques we expect that the attack can be extended to 5 rounds. However, for the moment our approach does not appear to be applicable to the full LAKE-256 hash function (with all 8 rounds).

Keywords: cryptanalysis, hash functions, collision attack

1 Introduction

A cryptographic hash function H maps a message M of arbitrary length to a fixed-length hash value h. A cryptographic hash function has to fulfill the following security requirements:

- *Collision resistance:* it is infeasible to find two messages M and M^*, with $M^* \neq M$, such that $H(M) = H(M^*)$.
- *Second preimage resistance:* for a given message M, it is infeasible to find a second message $M^* \neq M$ such that $H(M) = H(M^*)$.
- *Preimage resistance:* for a given hash value h, it is infeasible to find a message M such that $H(M) = h$.

The resistance of a hash function to collision and (second) preimage attacks depends on the length n of the hash value. Based on the birthday paradox the generic complexity for a collision attack is about $2^{n/2}$ hash computations, where n is the size of the hash value. For a preimage attack and a second preimage

[⋆] The work in this paper has been supported in part by European Commission under contract IST-2002-507932 (ECRYPT) and through the Austrian Science Fund (FWF) under grant number P19863.

© Springer-Verlag Berlin Heidelberg 2008

attack the generic complexity is about 2^n hash computations. If collisions and (second) preimages can be found with a complexity less than $2^{n/2}$ and 2^n the hash function is considered to be broken.

Recent cryptanalytic results focus on the collision resistance of hash functions. Collision attacks have been shown for many commonly used hash functions, like MD5 [13] and SHA-1 [4,12]. In the upcoming NIST competition [9] to find an alternative hash function to SHA-2, many new hash function designs will be proposed. Therefore, the cryptanalysis of new and alternative hash function designs like LAKE is of great interest. In this article, we will present a security analysis with respect to collision resistance for the hash function LAKE, proposed at FSE 2008 [2]. We are not aware of any published security analysis of this hash function until now.

The hash function LAKE is a new iterated hash function based on the HAIFA framework [3]. It is a software-oriented design and uses an internal wide-pipe strategy [7,8]. The two proposed variants of LAKE compute a 256-bit and 512-bit hash value and use an 8- and 10-round compression function, respectively. In our analysis we focus on the 256-bit variant LAKE-256 but the same attack applies to LAKE-512 as well. In the following we omit the bit size in the name if we refer to LAKE-256. We show collisions for round-reduced variants of LAKE where we exploit a structural weakness in the internal compression functions. We construct collisions in the used Boolean functions which are then extended to an attack on round-reduced variants of LAKE.

The remainder of this article is structured as follows. In the next section, we give a short description of the hash function LAKE with a focus on the relevant parts for our attacks. In Section 3, we explain the basic attack strategy and show a collision for a simplified variant of the full hash function. The results of the collision attacks on round-reduced variants are presented in Section 4. Finally, we conclude this paper with a short recommendation on how the LAKE design could be improved to withstand our attack.

2 Description of LAKE

The LAKE hash function is an iterated hash function based on the HAIFA framework [3]. It takes a salt and the message as its input. The message is padded by a specific padding rule and the initial chaining variable H_0 is computed form the initial value (IV) and parameterized by the (variable) output bit length d of the hash function. The LAKE family defines two main instances LAKE-256 and LAKE-512 which differ only in their used bit sizes, constants and rotation values. While our attack is not limited to LAKE-256 we focus on this instance of the LAKE family for the remainder of this paper.

The compression function of LAKE computes the next chaining variable H_t from the previous H_{t-1}, the current message block M_t the salt S and the current block index t. It consists of three parts which are shown in Figure 1. The function saltstate mixes the global chaining variable H_t with the salt S, and the block index t using 8 calls to the function g. The output of saltstate is written into the internal chaining variable $L^{(r-1)}$ which is twice as large as H_{t-1}. The function processmessage is the main part of the LAKE compression function and takes the current message block M_t and the current internal chaining variable $L^{(r-1)}$ as its input. The message block is first expanded by the message permutation $\sigma_r(i)$ and then incorporated into the internal chaining variables within r rounds. Every round of processmessage uses 16 calls to two nonlinear internal compression functions f and g. The feedforward function compresses the previous global chaining variable H_{t-1}, the salt S, the block index t, and the last internal chaining variable $L^{(r)}$ by 8 calls of the function f and produces the next chaining variable H_t.

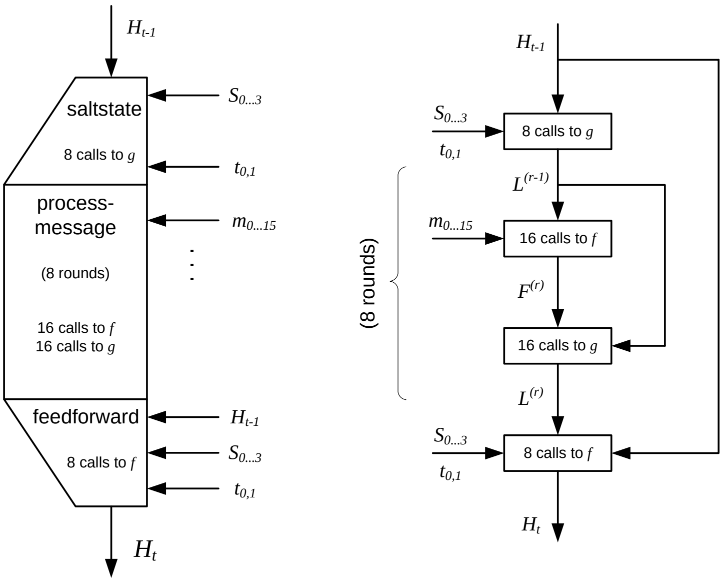

Fig. 1. The compression function of LAKE-256 consists of the three main parts saltstate, processmessage and feedforward which call two nonlinear internal compression functions f and g.

In the case of LAKE-256, the compression function uses $r = 8$ rounds and the message permutation of Table 1. The nonlinear internal compression functions f and g are defined by

$$f(a, b, c, d) = ((a + (b \vee C_0)) + (c + (a \wedge C_1)) \ggg 7) + ((b + (c \oplus d)) \ggg 13)$$
$$g(a, b, c, d) = ((a + b) \ggg 1) \oplus (c + d).$$

Depending on whether they are used in saltstate, processmessage or feedforward, these functions are parameterized by some constants $C_0, \ldots, C_{15}$, which are extracted from π:

Table 1. The index $k = \sigma_r(i)$ of the message permutation of LAKE-256 for the rounds R1-R8 of **processmessage**.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R2	1	6	11	0	5	10	15	4	9	14	3	8	13	2	7	12
R3	5	8	11	14	1	4	7	10	13	0	3	6	9	12	15	2
R4	0	7	14	5	12	3	10	1	8	15	6	13	4	11	2	9
R5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R6	1	6	11	0	5	10	15	4	9	14	3	8	13	2	7	12
R7	5	8	11	14	1	4	7	10	13	0	3	6	9	12	15	2
R8	0	7	14	5	12	3	10	1	8	15	6	13	4	11	2	9

$$\begin{array}{llll}
C_0 = \texttt{452821E6} & C_4 = \texttt{C0AC29B7} & C_8 = \texttt{9216D5D9} & C_{12} = \texttt{2FFD72DB} \\
C_1 = \texttt{38D01377} & C_5 = \texttt{C97C50DD} & C_9 = \texttt{8979FB1B} & C_{13} = \texttt{D01ADFB7} \\
C_2 = \texttt{BE5466CF} & C_6 = \texttt{3F84D5B5} & C_{10} = \texttt{D1310BA6} & C_{14} = \texttt{B8E1AFED} \\
C_3 = \texttt{34E90C6C} & C_7 = \texttt{B5470917} & C_{11} = \texttt{98DFB5AC} & C_{15} = \texttt{6A267E96}
\end{array}$$

In case of processmessage, the inputs of f are the previous internal chaining variables $L^{(r-1)}$, the current internal chaining variables $F^{(r)}$, the constants C_i, and the expanded message words m_k with $k = \sigma_r(i)$. The function g takes as input the current internal chaining variables $F^{(r)}$, the previous internal chaining variables $L^{(r-1)}$ using feed-forward and the new internal chaining variables $L^{(r)}$:

$$F_i^{(r)} = f(a,b,c,d) = f(F_{i-1}^{(r)}, L_i^{(r-1)}, m_k, C_i)$$
$$L_i^{(r)} = g(a,b,c,d) = g(L_{i-1}^{(r)}, F_i^{(r)}, L_i^{(r-1)}, F_{i+1}^{(r)})$$

Note that $F^{(r)}$ gets initialized by $L^{(r-1)}$ and $L^{(r)}$ gets initialized by $F^{(r)}$. We get for the sequence of chaining variables H_t and internal chaining variables $L^{(r)}$ and $F^{(r)}$:

$$H_{t-1} \rightarrow salt \rightarrow \underbrace{L^{(r-1)} \rightarrow f \rightarrow F^{(r)} \rightarrow g \rightarrow L^{(r)}}_{8\,rounds} \rightarrow feedforward \rightarrow H_t$$

3 Basic Attack Strategy

The basic observation for the attack on the compression function of LAKE is that the internal compression function f of processmessage is not bijective (not injective) regarding the chaining variables and message words. This means, that at least two message words exist, which result in the same output of f for fixed internal chaining variables. In fact, it is possible to find many different message words m_k and m_k^* which result in the same output of f. Using these inner collisions of the internal compression function f we can construct collisions for round-reduced versions of LAKE. Note that the same idea applies to both variants, LAKE-256 and LAKE-512 because the two variants differ only in the used word size, constants and rotation values.

3.1 Collisions for 1 Round of LAKE

In every round, each message word m_k is used only once by one of the 16 calls to the f function. Hence, we can construct a collision for one round of LAKE using a single inner collision in f (this has been independently observed by Stefan Lucks). By performing a collision attack on the 32-bit output of f we have been able to efficiently find many message pairs m_k and m_k^* for many internal chaining values $F_{i-1}^{(r)}$, $L_i^{(r-1)}$ and all constants C_i such that the output of f collides:

$$f(F_{i-1}^{(r)}, L_i^{(r-1)}, m_k, C_i) = f(F_{i-1}^{(r)}, L_i^{(r-1)}, m_k^*, C_i)$$

Note that the authors of LAKE have proposed to analyze a reduced variant of the hash function which uses the same constant in every round [1]. In this case we can simply use the same inner collision in f for every round of LAKE. Table 2 shows a collision for 8 rounds of LAKE using the same constant C_0 in each round which can be computed instantly on a standard PC.

Table 2. A colliding message pair for LAKE using the same constant C_0 in each round.

H_0	243F6A88	85A308D3	13198A2E	03707344	A4093822	299F31D0	082EFA98	EC4E6C89
M_0	7901FB66	7120239A	75018D7B	38EFC240	04BA14F4	54B5A198	60842D9A	05CE0AF7
	1A31E11B	40B1C10C	55F91C02	559DF366	74D6D973	455E48F2	31072B72	4DB56283
M_0^*	7D11BC59	7120239A	75018D7B	38EFC240	04BA14F4	54B5A198	60842D9A	05CE0AF7
	1A31E11B	40B1C10C	55F91C02	559DF366	74D6D973	455E48F2	31072B72	4DB56283
ΔM_0	0410473F	00000000	00000000	00000000	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
H_1	289B5613	0295350F	CA661380	699C892A	80CC3678	91B6F85B	FD0332EB	D89C925A
H_1^*	289B5613	0295350F	CA661380	699C892A	80CC3678	91B6F85B	FD0332EB	D89C925A

3.2 Collisions for more than 1 Round of LAKE

The original LAKE specification defines different constants for each round and we cannot use the same inner collision for every round anymore. However, the idea of constructing collisions in f can still be extended to attack more rounds of LAKE. Then, the same message pair m_k and m_k^* has to result in an inner collision of f for each of the attacked rounds. Due to the message expansion, the message word m_k is used in a different call of f in each round. However, in each call i, the f function differs only in the used constant C_i. For instance, if we want to construct a collision for the first two rounds of LAKE, we need to find a message pair m_k and m_k^* such that we have a collision in f in both rounds.

Assume we are using message word m_0. In the first round, m_0 is used in call $i = 0$ of the function f and in the second round, m_0 is used in call $i = 3$ of f (see Table 1). Hence, we need to find a message pair m_0 and m_0^*, which results in an inner collision of f and applies to both constants C_0 and C_3 simultaneously. One method to find such a pair is to search for each constant separately and check for matching message pairs. This method might work for two constants but is insufficient for more constants. In the following we show how this can be done more efficiently.

3.3 Inner Collisions in f using different Constants

A better method is to analyze the differential behaviour of the f function and choose message *differences* Δm_k, which are independent of the used constants. To find a message differences which results in a collision and thus, in a zero difference of the f function, we simplify the f function to:

$$f(a, b, m_k, C_i) = c_1 + ((m_k + c_2) \ggg 7) + ((c_3 + (m_k \oplus C_i)) \ggg 13) \qquad (1)$$

where the values c_1, c_2 and c_3 depend on the internal chaining variables $L_i^{(r-1)}$ and $F_{i-1}^{(r)}$. Because the majority of the remaining operations are modular additions and rotations we use signed bit differences in our attack. Note that more advanced techniques like generalized characteristics as used in the most recent attacks on SHA-1 are not needed in this case [5]. Signed bit differences have been introduced by Wang *et al.* in the analysis of the MD4-family of hash functions [11]. Using these differences, the carry expansions of the modular additions in Equation (1) can be controlled by imposing conditions on the absolute values (c_1, c_2 and c_3) and rotated without imposing further conditions. In the xor-addition $\Delta m_k \oplus C_i$ the sign of the signed bit difference Δm_k is flipped at each position where the constant C_i is one and does not change where C_i is zero. For a detailed description of signed bit differences, we refer to [6].

Before constructing a zero output difference of the f function, we define the differential representation of f regarding the message difference Δm_k by

$$\Delta f = (\underbrace{\Delta m_k}_{\Delta x} \ggg 7) + (\underbrace{(\Delta m_k \oplus C_i)}_{\Delta y} \ggg 13) = 0 \qquad (2)$$

where the differences Δx and Δy need to cancel each other after the rotations. For a collision over more than one round of LAKE, we need to fulfill equation 2 for different constants C_i but with the same message difference Δm_k. Therefore, we allow a signed bit difference in the message only at positions, where the values of the used constants are equal. In this case the difference Δy is independent of the used C_i. We define the equal positions of all used constants $C_{i_1}, C_{i_2}, \ldots$ by:

$$C_{eq}^{(p)} = \begin{cases} 1 & \text{if } C_{i_1}^{(p)} = C_{i_2}^{(p)} = \ldots \\ 0 & \text{otherwise} \end{cases} \qquad (3)$$

where $C^{(p)}$ denotes the bit position p of the value C. Note that the difference Δx is independent of each round. To get a zero difference of f for all rounds, the differences Δy has to be the same for each round and every used constant.

The more rounds we attack, the more constants C_i are used and the less is the Hamming weight of the equal positions C_{eq} of these constants. Since at each position we can choose between a negative, a positive or no difference, the number of the allowed signed message differences is $3^{hw(C_{eq})}$. If less differences are allowed in Δm_k the probability of a collision decreases. However, the search space gets reduced as well and we can check more (or even all) signed message differences. We have implemented a search tool similar as in [10], which uses carry expansions for the differences $\Delta x = \Delta m_k$ and $\Delta y = \Delta m_k \oplus C_i$. After the rotations we check whether the resulting differences cancel each other.

Note that two signed bit differences in the MSB always cancel each other in the addition and are thus considered to be equal. Therefore, we can allow additional message differences at the MSB of each modular addition. A flip of the message difference in the MSB because of xoring it with different constants C_i results in the same difference. Since we can omit the sign of the regarding MSB in each of the 3 modular additions, we allow additional message differences at position 32, 13 and 6. A difference at position 13 in $\Delta m_k \oplus C_i$ gets rotated to the MSB in Δy and a difference at position 6 in $\Delta m_k \oplus C_i$ gets rotated to the same position as the MSB of Δm_k in Δx. By including these three cases, the search space can be increased and even includes all inner collisions of f.

4 Results of the Collision Attack

To attack more than one round of LAKE we have implemented a tool which checks for collisions in f depending on the used constants C_i. We first compute C_{eq} and determine all possible message differences Δm_k. Then, we use signed carry expansions of the message difference in Δx and Δy and check whether the differences cancel each other after the rotation. Table 3 shows which constant C_i is used for each message word m_k in each round. With our tool we are able to check all possible message differences if more than three different constants are used. In this case, the Hamming weight of C_{eq} and the search space is low enough to try all possible expanded differences. For all cases where only two constants are involved, we have limited the search to high probability differentials (with a short carry expansion) and can therefore find collisions with a high probability as well.

Table 3. For each message word m_k different constants C_i are used in every round due to the message permutation. The constants for R5-R8 are the same as for R1-R4.

	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}
R1	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}
R2	C_3	C_0	C_{13}	C_{10}	C_7	C_4	C_1	C_{14}	C_{11}	C_8	C_5	C_2	C_{15}	C_{12}	C_9	C_6
R3	C_9	C_4	C_{15}	C_{10}	C_5	C_0	C_{11}	C_6	C_1	C_{12}	C_7	C_2	C_{13}	C_8	C_3	C_{14}
R4	C_0	C_7	C_{14}	C_5	C_{12}	C_3	C_{10}	C_1	C_8	C_{15}	C_6	C_{13}	C_4	C_{11}	C_2	C_9

4.1 2 Rounds

For an attack on two rounds of LAKE, we need a collision in f with two different values of C_i. When attacking the first two rounds of LAKE we can choose one of the first two constants of Table 3. We have found the best result for the message word m_3. In the first round this message word is used in call 3 to f and thus, it is xored with the constant C_3. In the second round, m_3 is used in call 10 and xored with the constant C_{10}. Hence, we need to fulfill the following differential equations for f simultaneously:

$$\Delta f_3 = (\Delta m_3 \ggg 7) + ((\Delta m_3 \oplus C_3) \ggg 13) = 0 \tag{4}$$

$$\Delta f_{10} = (\Delta m_3 \ggg 7) + ((\Delta m_3 \oplus C_{10}) \ggg 13) = 0 \tag{5}$$

We allow signed differences in Δm_3 at all positions, where the constants C_3 and C_{10} are equal:

$$\begin{aligned} C_3 &= \texttt{34E90C6C} \\ C_{10} &= \texttt{D1310BA6} \\ C_{eq} &= \texttt{1A27F835} \\ \Delta m_3 &= \texttt{9A27F835} \end{aligned}$$

The number of the equal positions in C_3 and C_{10} is 16 and by including the three MSBs we get a maximum Hamming weight for the allowed message differences of $HW(\Delta m_3) = 17$.

Using our tool we have found the following four message differences, where each of them results in a zero difference of the f function. Note that each inverted message difference results in a collision as well.

$$\begin{aligned} \Delta m_3 &= \texttt{8207E820} \qquad & \Delta m_3 &= [\pm 32, -26, 19, 18, 17, 16, 15, 14, 12, 6] \\ \Delta m_3 &= \texttt{8207E821} \qquad & \Delta m_3 &= [\pm 32, -26, 19, 18, 17, 16, 15, 14, 12, 6, 1] \\ \Delta m_3 &= \texttt{8207F820} \qquad & \Delta m_3 &= [\pm 32, -26, 19, 18, 17, 16, 15, 14, 13, -12, 6] \\ \Delta m_3 &= \texttt{8207F821} \qquad & \Delta m_3 &= [\pm 32, -26, 19, 18, 17, 16, 15, 14, 13, -12, 6, 1] \end{aligned}$$

For these message difference we get many expanded differences Δx and Δy which cancel each other. For example, if we consider the message difference $\Delta m_3 = \texttt{8207E820}$, the signed differences Δx and Δy with the best probabilities are:

$$\begin{aligned} \Delta x &= [-32, 26, -20, 13, 12, -8, 7, 6] \\ \Delta y &= [-32, 26, -20, 18, 14, 12, 6] \end{aligned}$$

where the difference Δx occurs with probability 2^{-8} and Δy with probability 2^{-7}. After rotating these difference by 7 and by 13 we get the following two differences, which cancel each other in the third modular addition:

$$\begin{aligned} \Delta x \ggg 7 &= [32, 31, -25, 19, -13, 6, 5, -1] \\ \Delta y \ggg 13 &= [31, 25, -19, 13, -7, 5, 1] \end{aligned}$$

Therefore, we get an inner collision in f for both rounds with a probability of 2^{-15} each. Usually the expanded differences with the highest probabilities determine the complexity of the attack. However, if many expanded differences cancel each other, the actual complexity is determined by the sum of all probabilities. For the message difference $\Delta m_3 =$ 8207E820 we have found 2600 expanded signed differences Δx and 5486 expanded signed differences for Δy. By adding all possible combined probabilities of Δx and Δy we get an overall probability of $2^{-4.38}$ instead of 2^{-15}.

4.2 3 Rounds

The previous collision in f over two rounds can be easily extended to a collision over 3 rounds. To extend the attack we use a weakness in the message permutation. The message word m_3 is used in call 3 of the first round and in call 10 of the second and third round. Thus, the constant C_{10} is used twice and we can use the same collision for f as in the attack on two rounds. Note that we could do the same for message word m_{11} which uses the constant C_2 twice.

A Colliding Message for 3-round LAKE. By using the message difference $\Delta m_3 =$ 8207E820 we can construct a collision for LAKE reduced to three rounds with a complexity of about $2^{3 \cdot 4.38} \approx 2^{13.2}$ round evaluations (less than 1 second on a standard PC), since we can get a collision for each round with a probability of $2^{-4.38}$. The colliding message pair is given in Table 4. Note that h_0 is the initial value and h_1 is the final hash value.

Table 4. A colliding message pair for LAKE reduced to 3 rounds.

H_0	243F6A88	85A308D3	13198A2E	03707344	A4093822	299F31D0	082EFA98	EC4E6C89
M_0	2ED54018	259E7BED	6A7D12A0	12780007	57979D36	619A5DE1	2F1FA8A0	09D72979
	3428C041	1439951D	63537711	144840C4	7C75D35E	70C613E9	23DCA632	52DB6AB9
M_0^*	2ED54018	259E7BED	6A7D12A0	907FE827	57979D36	619A5DE1	2F1FA8A0	09D72979
	3428C041	1439951D	63537711	144840C4	7C75D35E	70C613E9	23DCA632	52DB6AB9
ΔM_0	00000000	00000000	00000000	8207E820	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
H_1	0969AF41	101EA7CE	CBF3F2FE	E47832EB	60FFD511	DA156A75	150B3A20	F003BA7E
H_1^*	0969AF41	101EA7CE	CBF3F2FE	E47832EB	60FFD511	DA156A75	150B3A20	F003BA7E

4.3 More than 3 rounds

To attack more than 3 rounds we have first tried to construct a collision which uses only 3 different constants. This could be done for the message words m_0, m_3, m_8 and m_{11} (see Table 3). However, even by checking all possible message differences and carry expansions, we did not find a collision in these message words. Anyway, by trying all message words which use four different constants, we have found solutions for m_4 and m_7. The involved constants are C_4, C_7, C_5, C_{12} for m_4 and C_7, C_{14}, C_6, C_1 for m_7.

For the 4 round collision we have only found a characteristic with low probability. The possible message differences are $\Delta m_7 = \Delta m_4 = \pm 1$. Thus, we allow a difference only in the LSB of the message word. Note that the LSB of the involved constants is 1 and the xor operation flips the message difference. Therefore, the differences in Δx and Δy have a opposite sign and cancel each other if the following conditions are fulfilled ($i = 4, 7, 5, 12$):

$$F_{i-1}^{(r)} \wedge C_1 = 0 \tag{6}$$

$$L_i^{(r-1)} + C_i = \texttt{FFFFFFFF} \tag{7}$$

Under these conditions, the differences do not get changed by the rotations and we can get an inner collision in f for every round of LAKE.

Let us consider the case $\Delta m_4 = -1$ with $m_4 = 0$. By fulfilling the previous conditions the resulting values before the rotation are always either $\texttt{00000000}$ or $\texttt{FFFFFFFF}$. These values do not get changed by the rotation and we get for $m_4 = 0$:

$$\underbrace{(0 + F_{i-1}^{(r)} \wedge C_1)}_{F_{i-1}^{(r)} \wedge C_1 = 0} \ggg 7 + \underbrace{(L_i^{(r-1)} + (0 \oplus C_i))}_{L_i^{(r-1)} + C_i = \texttt{FFFFFFFF}} \ggg 13 = \texttt{FFFFFFFF} \tag{8}$$

and for $m_4^* = m_4 - 1 = 0 - 1$:

$$(\underbrace{0 - 1 + F_{i-1}^{(r)} \wedge C_1}_{(F_{i-1}^{(r)} \wedge C_1) - 1 = \texttt{FFFFFFFF}}) \ggg 7 + \underbrace{(L_i^{(r-1)} + ((0-1) \oplus C_i)}_{L_i^{(r-1)} + 1 + C_i = 0}) \ggg 13 = \texttt{FFFFFFFF} \tag{9}$$

The two equations (6) and (7) hold in each round with a probability of $2^{-32-15} = 2^{-47}$, since the Hamming weight of C_1 is 15. Hence, we can get a collision for LAKE reduced to r rounds with a probability of $2^{-r \cdot 47}$ and for $r = 4$ rounds we get a probability of 2^{-188}.

Note that the difference $\Delta m_k = \pm 1$ works for any message word and any number of rounds, as long as the LSB of all involved constants is 1. However, due to the low probability we have only attacked 4 rounds of LAKE using message modification. By more sophisticated message modification techniques, we expect that an attack up to 5 rounds of the LAKE compression function is possible.

4.4 A Collision Attack for 4 Rounds of LAKE

The attack complexity of 2^{188} for 4 rounds of LAKE can be improved by using message modification techniques introduced by Wang *et al.* in the analysis of MD5 and SHA-1 [13,12]. In general, the idea of message modification is to use the degrees of freedom in the message to fulfill conditions on the state variables. This improves the probability of the attack and in the following we will show how message modification can be done for the first 2 rounds of LAKE.

Message Modification. In the first round we use basic message modification which simply adjusts the message words such that the conditions in the internal chaining variables are fulfilled. To fulfill the conditions on $F_3^{(1)} \wedge C_1 = 0$ we adjust $F_3^{(1)}$ by modifying m_3. Because of the right rotation, we can start by modifying bit 7 of the message and proceed up to bit 25 without getting any conflict due to carries. The remaining 6 bits are fulfilled by brute force which results in a complexity of 2^6. Since all other modifications change message words after call 3 of f, we perform this modification only once at the beginning. Therefore, this modification does not increase the overall complexity. Next we need to fulfill the conditions of $L_4^{(0)} + C_4 = \texttt{FFFFFFFF}$. Note that $L_4^{(0)}$ depends on the IV. By using an arbitrary first message block we can construct the needed value of $L_4^{(0)}$. This has a complexity of 2^{32} but needs do be done only once as well.

For the second round of LAKE we need to use advanced message modification techniques (solving a system of equations). The equation $F_6^{(2)} \wedge C_1 = 0$ of the second round can be fulfilled with a probability of 2^{-15} without message modification and the equation $L_7^{(1)} + C_7 = \texttt{FFFFFFFF}$ is fulfilled with a probability of 2^{-32}. Note that $L_7^{(1)}$ depends on the output of the first round but can be changed by advanced message modification. This means that we can change $F_8^{(1)}$ by modifying message word m_8 and correct the changes by adjusting message word m_{15}.

The Collision Search for 4 Rounds of LAKE. The search for a collision of LAKE reduced to 4 rounds can be summarized by the following steps. The sequence of internal chaining variables and calls to f and g are illustrated in the appendix to comprehend the message modification steps:

1. We fulfill the 32 conditions on $L_4^{(0)}$ by choosing an arbitrary first message block M_0. This has a complexity of 2^{32} evaluations of the compression function and needs to be done only once at the beginning of the search.
2. Next we choose random message words $m_0, \ldots, m_3$ to compute the internal chaining variables $F_0^{(1)}, \ldots, F_3^{(1)}$.
3. The 15 conditions on $F_3^{(1)}$ can be fulfilled by adjusting m_3 using basic message modification. This step has a complexity of about 2^6 calls to $F_3^{(1)} = f(F_2^{(1)}, L_3^{(0)}, m_3, C_3)$. Since we do not change $m_0, \ldots, m_3$ later on, this step needs to be done only once as well.
4. The remaining message words $m_4, \ldots, m_{15}$ are chosen at random to compute the internal chaining variables $F_4^{(1)}, \ldots, F_{15}^{(1)}$ and $L_0^{(1)}, \ldots, L_7^{(1)}$ to check the conditions on $L_7^{(1)}$.
5. To fulfill the conditions on $L_7^{(1)}$ we compute the required value of $F_8^{(1)}$ by simply inverting the function $L_7^{(1)} = g(L_6^{(1)}, F_7^{(1)}, L_7^{(0)}, F_8^{(1)})$ and get for $F_8^{(1)} = (L_7^{(1)} \oplus ((L_6^{(1)} + F_7^{(1)}) \ggg 1)) - L_7^{(0)}$.

6. We can generate this required value of $F_8^{(1)}$ by modifying m_8 in $F_8^{(1)} = f(F_7^{(1)}, L_8^{(0)}, m_8, C_8)$ using basic message modification with a complexity of about 2^6 calls to f.
7. The modification of m_8 and $F_8^{(1)}$ leads to new values in the internal chaining variables starting from $F_9^{(1)}$. Note that $L_7^{(1)} = g(L_6^{(1)}, F_7^{(1)}, L_7^{(0)}, F_8^{(1)})$ depends only on $L_6^{(1)}$ and values prior to $F_8^{(1)}$. To guarantee that $L_7^{(1)}$ does not get changed again, it is sufficient to require that $F_{15}^{(1)}$ does not change.
8. We can ensure this by adjusting the message word m_{15} such that $F_{15}^{(1)}$ has the same value as prior to the modification of m_8. Then, the values $L_0^{(1)}, \ldots, L_7^{(1)}$ do not change and the conditions on $L_7^{(1)}$ stay fulfilled. This modification of m_{15} has again a complexity of about 2^6 calls to f.
9. The conditions on $F_6^{(2)}$ and on the internal chaining variable of round 3 and 4 can be fulfill by randomly choosing message words $m_9, \ldots, m_{14}$. We ensure the conditions on $L_7^{(1)}$ by modifying m_{15} again. Note that we have enough degrees of freedom in these 6 message words to fulfill these remaining $15 + 47 + 47 = 109$ conditions by brute-force.

These message modification techniques improve the attack complexity significantly. By performing the collision search as described above we can construct collisions for LAKE reduced to 4 rounds with an overall complexity of about 2^{109} compression function evaluations. Note that the complexity can actually be smaller if early stopping techniques are used. By applying more advanced message modification techniques we expect to be able to break up to 5 rounds of LAKE.

5 Conclusion

In this paper we have presented the first cryptanalytic results on the hash function family LAKE. We have shown how collision attacks, exploiting inner collisions in the nonlinear functions of LAKE, can be mounted on reduced variants of the hash function. We have presented an efficient attack on LAKE reduced to 3 (out of 8) rounds. Moreover, we have shown a theoretical attack on LAKE reduced to 4 rounds with a complexity of 2^{109}. We expect that our attack can also be extended to LAKE reduced to 5 rounds by using more sophisticated message modification techniques. Note that the same strategy can be used to attack LAKE-512 as well. For the moment our approach does not appear to be applicable to the full hash function.

However, this does not prove that the hash function is secure. Further analysis is required to get a good view on the security margins of LAKE. In our analysis we have shown that the security of LAKE strongly depends on the choice of the constants. Due to a weak combination of constants, attacks on round-reduced versions of LAKE are possible. Further, we note that the non-bijectiveness regarding the chaining variables can be used to cancel differences in the internal chaining variables as well. To prevent our attack we suggest to design internal compression functions which are bijective and thus, invertible regarding the message words and each chaining variable. Further, the security of these functions should not depend on the (good) choice of the used constants.

Acknowledgments

We thank the authors of LAKE for sending us a preliminary version of their paper and for helpful discussions.

The information in this paper is provided as is, and no guarantee or warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. Jean-Philippe Aumasson. The Hash Function Family LAKE. FSE talk, 2008. `http://fse2008.epfl.ch/docs/slides/day_1_sess_3/aumasson%20lake_slides.pdf`.
2. Jean-Philippe Aumasson, Willi Meier, and Raphael C.-W. Phan. The Hash Function Family LAKE. In Kaisa Nyberg, editor, *FSE*, LNCS. Springer, 2008. To appear.
3. Eli Biham and Orr Dunkelman. A Framework for Iterative Hash Functions - HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007. `http://eprint.iacr.org`.
4. Christophe De Cannière, Florian Mendel, and Christian Rechberger. Collisions for 70-Step SHA-1: On the Full Cost of Collision Search. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *LNCS*, pages 56–73. Springer, 2007.
5. Christophe De Cannière and Christian Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006.
6. Magnus Daum. *Cryptanalysis of Hash Functions of the MD4-Family*. PhD thesis, Ruhr-Universität Bochum, May 2005.
7. Stefan Lucks. Design Principles for Iterated Hash Functions. Cryptology ePrint Archive, Report 2004/253, 2004. `http://eprint.iacr.org`.
8. Stefan Lucks. A Failure-Friendly Design Principle for Hash Functions. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *LNCS*, pages 474–494. Springer, 2005.
9. National Institute of Standards and Technology (NIST). Cryptographic Hash Project, 2007. `http://www.nist.gov/hash-competition`.
10. Martin Schläffer and Elisabeth Oswald. Searching for Differential Paths in MD4. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *LNCS*, pages 242–261. Springer, 2006.
11. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 1–18. Springer, 2005.
12. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
13. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.

A Advanced Message Modification

The step update functions for the first two rounds of LAKE. The internal chaining variables on which we impose conditions for the attack on 4 rounds of LAKE are underlined.

A.1 Round 1

$$F_0^{(1)} = f(L_{15}^{(0)}, L_0^{(0)}, m_0, C_0) \qquad L_0^{(1)} = g(F_{15}^{(1)}, F_0^{(1)}, L_0^{(0)}, F_1^{(1)}) \tag{10}$$
$$F_1^{(1)} = f(F_0^{(1)}, L_1^{(0)}, m_1, C_1) \qquad L_1^{(1)} = g(L_0^{(1)}, F_1^{(1)}, L_1^{(0)}, F_2^{(1)}) \tag{11}$$
$$F_2^{(1)} = f(F_1^{(1)}, L_2^{(0)}, m_2, C_2) \qquad L_2^{(1)} = g(L_1^{(1)}, F_2^{(1)}, L_2^{(0)}, \underline{F_3^{(1)}}) \tag{12}$$
$$\underline{F_3^{(1)}} = f(F_2^{(1)}, L_3^{(0)}, m_3, C_3) \qquad L_3^{(1)} = g(L_2^{(1)}, \underline{F_3^{(1)}}, L_3^{(0)}, F_4^{(1)}) \tag{13}$$
$$F_4^{(1)} = f(\underline{F_3^{(1)}}, \underline{L_4^{(0)}}, m_4, C_4) \qquad L_4^{(1)} = g(L_3^{(1)}, F_4^{(1)}, \underline{L_4^{(0)}}, F_5^{(1)}) \tag{14}$$
$$F_5^{(1)} = f(F_4^{(1)}, L_5^{(0)}, m_5, C_5) \qquad L_5^{(1)} = g(L_4^{(1)}, F_5^{(1)}, L_5^{(0)}, F_6^{(1)}) \tag{15}$$
$$F_6^{(1)} = f(F_5^{(1)}, L_6^{(0)}, m_6, C_6) \qquad L_6^{(1)} = g(L_5^{(1)}, F_6^{(1)}, L_6^{(0)}, F_7^{(1)}) \tag{16}$$
$$F_7^{(1)} = f(F_6^{(1)}, L_7^{(0)}, m_7, C_7) \qquad \underline{L_7^{(1)}} = g(L_6^{(1)}, F_7^{(1)}, L_7^{(0)}, F_8^{(1)}) \tag{17}$$
$$F_8^{(1)} = f(F_7^{(1)}, L_8^{(0)}, m_8, C_8) \qquad L_8^{(1)} = g(\underline{L_7^{(1)}}, F_8^{(1)}, L_8^{(0)}, F_9^{(1)}) \tag{18}$$
$$F_9^{(1)} = f(F_8^{(1)}, L_9^{(0)}, m_9, C_9) \qquad L_9^{(1)} = g(L_8^{(1)}, F_9^{(1)}, L_9^{(0)}, F_{10}^{(1)}) \tag{19}$$
$$F_{10}^{(1)} = f(F_9^{(1)}, L_{10}^{(0)}, m_{10}, C_{10}) \qquad L_{10}^{(1)} = g(L_9^{(1)}, F_{10}^{(1)}, L_{10}^{(0)}, F_{11}^{(1)}) \tag{20}$$
$$F_{11}^{(1)} = f(F_{10}^{(1)}, L_{11}^{(0)}, m_{11}, C_{11}) \qquad L_{11}^{(1)} = g(L_{10}^{(1)}, F_{11}^{(1)}, L_{11}^{(0)}, F_{12}^{(1)}) \tag{21}$$
$$F_{12}^{(1)} = f(F_{11}^{(1)}, L_{12}^{(0)}, m_{12}, C_{12}) \qquad L_{12}^{(1)} = g(L_{11}^{(1)}, F_{12}^{(1)}, L_{12}^{(0)}, F_{13}^{(1)}) \tag{22}$$
$$F_{13}^{(1)} = f(F_{12}^{(1)}, L_{13}^{(0)}, m_{13}, C_{13}) \qquad L_{13}^{(1)} = g(L_{12}^{(1)}, F_{13}^{(1)}, L_{13}^{(0)}, F_{14}^{(1)}) \tag{23}$$
$$F_{14}^{(1)} = f(F_{13}^{(1)}, L_{14}^{(0)}, m_{14}, C_{14}) \qquad L_{14}^{(1)} = g(L_{13}^{(1)}, F_{14}^{(1)}, L_{14}^{(0)}, F_{15}^{(1)}) \tag{24}$$
$$F_{15}^{(1)} = f(F_{14}^{(1)}, L_{15}^{(0)}, m_{15}, C_{15}) \qquad L_{15}^{(1)} = g(L_{14}^{(1)}, F_{15}^{(1)}, L_{15}^{(0)}, L_0^{(1)}) \tag{25}$$

A.2 Round 2

$$F_0^{(2)} = f(L_{15}^{(1)}, L_0^{(1)}, m_1, C_0) \qquad L_0^{(2)} = g(F_{15}^{(2)}, F_0^{(2)}, L_0^{(1)}, F_1^{(2)}) \tag{26}$$
$$F_1^{(2)} = f(F_0^{(2)}, L_1^{(1)}, m_6, C_1) \qquad L_1^{(2)} = g(L_0^{(2)}, F_1^{(2)}, L_1^{(1)}, F_2^{(2)}) \tag{27}$$
$$F_2^{(2)} = f(F_1^{(2)}, L_2^{(1)}, m_{11}, C_2) \qquad L_2^{(2)} = g(L_1^{(2)}, F_2^{(2)}, L_2^{(1)}, F_3^{(2)}) \tag{28}$$
$$F_3^{(2)} = f(F_2^{(2)}, L_3^{(1)}, m_0, C_3) \qquad L_3^{(2)} = g(L_2^{(2)}, F_3^{(2)}, L_3^{(1)}, F_4^{(2)}) \tag{29}$$
$$F_4^{(2)} = f(F_3^{(2)}, L_4^{(1)}, m_5, C_4) \qquad L_4^{(2)} = g(L_3^{(2)}, F_4^{(2)}, L_4^{(1)}, F_5^{(2)}) \tag{30}$$
$$F_5^{(2)} = f(F_4^{(2)}, L_5^{(1)}, m_{10}, C_5) \qquad \underline{L_5^{(2)}} = g(L_4^{(2)}, F_5^{(2)}, L_5^{(1)}, \underline{F_6^{(2)}}) \tag{31}$$

$$\underline{F_6^{(2)}} = f(F_5^{(2)}, L_6^{(1)}, m_{15}, C_6) \qquad L_6^{(2)} = g(\underline{L_5^{(2)}}, \underline{F_6^{(2)}}, L_6^{(1)}, F_7^{(2)}) \tag{32}$$

$$F_7^{(2)} = f(\underline{F_6^{(2)}}, \underline{L_7^{(1)}}, m_4, C_7) \qquad L_7^{(2)} = g(L_6^{(2)}, F_7^{(2)}, \underline{L_7^{(1)}}, F_8^{(2)}) \tag{33}$$

$$F_8^{(2)} = f(F_7^{(2)}, L_8^{(1)}, m_9, C_8) \qquad L_8^{(2)} = g(L_7^{(2)}, F_8^{(2)}, L_8^{(1)}, F_9^{(2)}) \tag{34}$$

$$F_9^{(2)} = f(F_8^{(2)}, L_9^{(1)}, m_{14}, C_9) \qquad L_9^{(2)} = g(L_8^{(2)}, F_9^{(2)}, L_9^{(1)}, F_{10}^{(2)}) \tag{35}$$

$$F_{10}^{(2)} = f(F_9^{(2)}, L_{10}^{(1)}, m_3, C_{10}) \qquad L_{10}^{(2)} = g(L_9^{(2)}, F_{10}^{(2)}, L_{10}^{(1)}, F_{11}^{(2)}) \tag{36}$$

$$F_{11}^{(2)} = f(F_{10}^{(2)}, L_{11}^{(1)}, m_8, C_{11}) \qquad L_{11}^{(2)} = g(L_{10}^{(2)}, F_{11}^{(2)}, L_{11}^{(1)}, F_{12}^{(2)}) \tag{37}$$

$$F_{12}^{(2)} = f(F_{11}^{(2)}, L_{12}^{(1)}, m_{13}, C_{12}) \qquad L_{12}^{(2)} = g(L_{11}^{(2)}, F_{12}^{(2)}, L_{12}^{(1)}, F_{13}^{(2)}) \tag{38}$$

$$F_{13}^{(2)} = f(F_{12}^{(2)}, L_{13}^{(1)}, m_2, C_{13}) \qquad L_{13}^{(2)} = g(L_{12}^{(2)}, F_{13}^{(2)}, L_{13}^{(1)}, F_{14}^{(2)}) \tag{39}$$

$$F_{14}^{(2)} = f(F_{13}^{(2)}, L_{14}^{(1)}, m_7, C_{14}) \qquad L_{14}^{(2)} = g(L_{13}^{(2)}, F_{14}^{(2)}, L_{14}^{(1)}, F_{15}^{(2)}) \tag{40}$$

$$F_{15}^{(2)} = f(F_{14}^{(2)}, L_{15}^{(1)}, m_{12}, C_{15}) \qquad L_{15}^{(2)} = g(L_{14}^{(2)}, F_{15}^{(2)}, L_{15}^{(1)}, L_0^{(2)}) \tag{41}$$