Cryptanalysis of the LAKE Hash Family

Alex Biryukov[1], Praveen Gauravaram[3], Jian Guo[2], Dmitry Khovratovich[1], San Ling[2], Krystian Matusiewicz[3], Ivica Nikolić[1], Josef Pieprzyk[4], and Huaxiong Wang[2]

[1] University of Luxembourg, Luxembourg
{alex.biryukov,dmitry.khovratovich,ivica.nikolic}@uni.lu
[2] School of Physical and Mathematical Sciences,
Nanyang Technological University, Singapore
{guojian,lingsan,hxwang}@ntu.edu.sg
[3] Department of Mathematics,
Technical University of Denmark, Denmark
{P.Gauravaram,K.Matusiewicz}@mat.dtu.dk
[4] Centre for Advanced Computing - Algorithms and Cryptography,
Macquarie University, Australia
josef@ics.mq.edu.au

Abstract. We analyse the security of the cryptographic hash function LAKE-256 proposed at FSE 2008 by Aumasson, Meier and Phan. By exploiting non-injectivity of some of the building primitives of LAKE, we show three different collision and near-collision attacks on the compression function. The first attack uses differences in the chaining values and the block counter and finds collisions with complexity 2^{33}. The second attack utilizes differences in the chaining values and salt and yields collisions with complexity 2^{42}. The final attack uses differences only in the chaining values to yield near-collisions with complexity 2^{99}. All our attacks are independent of the number of rounds in the compression function. We illustrate the first two attacks by showing examples of collisions and near-collisions.

1 Introduction

The recent cryptanalytical results on the cryptographic hash functions following the attacks on MD5 and SHA-1 by Wang et al. [17,16,15] have seriously undermined the confidence in many currently deployed hash functions. Around the same time, new generic attacks such as multicollision attack [7], long message second preimage attack [9] and herding attack [8], exposed some undesirable properties and weaknesses in the Merkle-Damgård (MD) construction [12,5]. These developments have renewed the interest in the design of hash functions. Subsequent announcement by NIST of the SHA-3 hash function competition, aiming at augmenting the FIPS 180-2 [13] standard with a new cryptographic hash function, has further stimulated the interest in the design and analysis of hash functions.

The hash function family LAKE [1], presented at FSE 2008, is one of the new designs. It follows the design principles of the HAIFA framework [2,3] – a strengthened alternative to the MD construction.

As the additional inputs to the compression function, LAKE uses a random value (also called salt) and an index value, which counts the number of bits/blocks in the input message processed so far.

The first analysis of LAKE, presented by Mendel and Schläffer [11], has shown collisions for 4 out of 8 rounds. The complexity of their attack is 2^{109}. The main observation used in the attack is the non-injectivity of one of the internal functions. This property allows to introduce difference in the message words, which is canceled immediately, when the difference goes through the non-injective function.

Our contributions. Our attacks focus on finding collisions for the compression function of LAKE. Let $f(H, M, S, t)$ be a compression function of a HAIFA hash function using chaining values H, message block M, salt S and the block index t. We present the following three types of collision attacks. The first attack uses differences in the chaining values H and block index t, so we are looking for collisions of form $f(H, M, S, t) = f(H', M, S, t')$. We call it a (H, t)-type attack. The complexity of this attack is 2^{33} compression calls. The second attack deals with the differences injected in the chaining values and salt S, we call it a (H, S)-attack. We present how to find near-collisions of the compression function with the complexity 2^{30} of compression calls and extend it to full collisions with the complexity 2^{42}. The final attack, called a H-type attack, uses only differences in the chaining values and finds near-collisions for the compression function with the complexity 2^{99}. The success of our collision attacks relies on solving the systems of equations that originate from the differential conditions imposed by the attacks. We present some efficient methods to solve these systems of equations.

Our attacks demonstrate that increasing the number of rounds of LAKE does not increase its security as they all aim at canceling the differences within the first ProcessMessage function of the compression function.

2 Description of LAKE

In this section, we provide a brief description of the LAKE compression function, skipping details that are not relevant to our attacks. See [1] for details.

Basic functions – LAKE uses two functions f and g defined as follows

$$\begin{aligned} f(a,b,c,d) &= (a + (b \vee C_0)) + ((c + (a \wedge C_1)) \ggg 7) + \\ &\qquad ((b + (c \oplus d)) \ggg 13) \ , \\ g(a,b,c,d) &= ((a + b) \ggg 1) \oplus (c + d) \ , \end{aligned}$$

where each variable is a 32-bit word and C_0, C_1 are constants.

The compression function of LAKE has three components: SaltState, ProcessMessage and FeedForward. The functionality of these components are

```
Input: $H = H_0\|\dots\|H_7$, $S = S_0\|\dots\|S_3$, $t = t_0\|t_1$
Output: $F = F_0\|\dots\|F_{15}$
for $i = 0,\dots,7$ do
    $F_i = H_i$;
end
$F_8 = g(H_0, S_0 \oplus t_0, C_8, 0)$;
$F_9 = g(H_1, S_1 \oplus t_1, C_9, 0)$;
for $i = 10,\dots,15$ do
    $F_i = g(H_i, S_i, C_i, 0)$;
end
```

Algorithm 1. LAKE's SaltState

```
Input: $F = F_0\|\dots\|F_{15}$, $M = M_0\|\dots\|M_{15}$, $\sigma$
Output: $W = W_0\|\dots\|W_{15}$
for $i = 0,\dots,15$ do
    $L_i = f(L_{i-1}, F_i, M_{\sigma(i)}, C_i)$;
end
$W_0 = g(L_{15}, L_0, F_0, L_1)$;
$L_0 = W_0$;
for $i = 1,\dots,15$ do
    $W_i = g(W_{i-1}, L_i, F_i, L_{i+1})$;
end
```

Algorithm 2. LAKE's ProcessMessage

```
Input: $W = W_0\|\dots\|W_{15}$, $H = H_0\|\dots\|H_7$, $S = S_0\|\dots\|S_3$, $t = t_0\|t_1$
Output: $H = H_0\|\dots\|H_7$
$H_0 = f(W_0, W_8, S_0 \oplus t_0, H_0)$;
$H_1 = f(W_1, W_9, S_1 \oplus t_1, H_1)$;
for $i = 2,\dots,7$ do
    $H_i = f(W_i, W_{i+8}, S_i, H_i)$;
end
```

Algorithm 3. LAKE's FeedForward

```
Input: $H = H_0\|\dots\|H_7$, $M = M_0\|\dots\|M_{15}$, $S = S_0\|\dots\|S_3$, $t = t_0\|t_1$
Output: $H = H_0\|\dots\|H_7$
$F = \mathsf{SaltState}(H, S, t)$;
for $i = 0,\dots,r-1$ do
    $F = \mathsf{ProcessMessage}(F, M, \sigma_i)$;
end
$H = \mathsf{FeedForward}(F, H, S, t)$;
```

Algorithm 4. LAKE's CompressionFunction

described in Algorithms 1, 2 and 3, respectively. The whole compression function of LAKE is presented as Algorithm 4. Our attacks do not depend on the constants C_i for $i = 0,\dots,15$ and hence we do not provide their actual values here.

3 Properties and Observations

We first present some properties of the f function used in our analysis.

Observation 1. *Function $f(x,y,z,t)$ is non-injective with respect to the first three arguments x,y,z.*

For example, for x there exist two different values x and x' such that $f(x,y,z,t)$ $= f(x',y,z,t)$ for some y,z,t. The same property holds for y and z. This observation was mentioned by Lucks at FSE'08. Mendel and Schläffer independently found and used this property to successfully attack four out of eight rounds of LAKE-256. Non-injectivity of the function f can be used to cancel a difference in one of the first three arguments of f, when the rest of the arguments are fixed.

The following observation of the rotation on the modular addition allows us to simplify the analysis of f.

Lemma 1 ([6]). $(a+b) \ggg k = (a \ggg k) + (b \ggg k) + \alpha - \beta \cdot 2^{n-k}$*, where* $\alpha = \mathbf{1}[a^R_k + b^R_k \geq 2^k]$ *and* $\beta = \mathbf{1}[a^L_k + b^L_k + \alpha \geq 2^{n-k}]$.

Using Lemma (1), the function f can be written as

$$\begin{aligned} f(a,b,c,d) = a + b \vee C_0 + (c \ggg 7) + ((a \wedge C_1) \ggg 7) + (b \ggg 13) \\ + ((c \oplus d) \ggg 13) + \alpha_1 + \alpha_2 - \beta_1 \cdot 2^{25} - \beta_2 \cdot 2^{19}, \quad (1) \end{aligned}$$

where

$$\begin{aligned} \alpha_1 &= \mathbf{1}[c^L_7 + (a \wedge C_1)^L_7 \geq 2^7], & \beta_1 &= \mathbf{1}[c^R_7 + (a \wedge C_1)^R_7 + \alpha_1 \geq 2^{25}], \\ \alpha_2 &= \mathbf{1}[b^L_{13} + (c \oplus d)^L_{13} \geq 2^{13}], & \beta_2 &= \mathbf{1}[b^R_{13} + (c \oplus d)^R_{13} + \alpha_2 \geq 2^{19}]. \end{aligned}$$

Note that α_2 and β_2 are independent of a. Consider now the difference of the outputs of f induced by the difference in the variable a, i.e.

$$\begin{aligned} \Delta f &= f(a',b,c,d) - f(a,b,c,d) \\ &= [a' + (a' \wedge C_1) + \alpha'_1 - \beta'_1 \cdot 2^{25}] - [a + (a \wedge C_1) + \alpha_1 - \beta_1 \cdot 2^{25}] \\ &= a' + ((a' \wedge C_1) \ggg 7) - [a + ((a \wedge C_1) \ggg 7)] + (\alpha'_1 - \alpha_1) - (\beta'_1 - \beta_1) \cdot 2^{25} \\ &= f_a(a') - f_a(a) + (\alpha'_1 - \alpha_1) - (\beta'_1 - \beta_1) \cdot 2^{25}, \end{aligned}$$

where $f_a(a) \overset{\text{def}}{=} a + ((a \wedge C_1) \ggg 7)$.

A detailed analysis (cf. Lemma 5) shows that given random a, a' and c, $P(\alpha_1 = \alpha'_1, \beta_1 = \beta'_1) = \frac{4}{9}$, so with the probability $\frac{4}{9}$, a collision for f_a is also a collision of f when the input difference is in a only. Let us call this a *carry effect*. However, if we have control over the variable c, we can adjust the values of $\alpha_1, \alpha'_1, \beta_1, \beta'_1$ and always satisfy this condition. From here we can see that $(a+b) \ggg k$ is not a good mixing function when modular differences are concerned.

This reasoning can be repeated for differences in the variable b and similarly for differences in a pair of the variables c, d. It is easy to see that also for those cases, with a high probability, collisions in f happen when the following functions collide

$$f_b(b) \overset{\text{def}}{=} b \vee C_0 + (b \ggg 13) \ ,$$
$$f_{cd}(c,d) \overset{\text{def}}{=} (c \ggg 7) + ((c \oplus d) \ggg 13) \ .$$

So, when we follow differences in one or two variables only, we can consider those variables without the side effects from other variables. We summarize the above observations below.

Observation 2. *Collisions or output differences of f for input differences in one variable can be made independent from the values of other variables.*

We denote the set of solutions for f_a and f_b with respect to input pairs and modular differences as

$$S_{fa} \overset{\text{def}}{=} \{(x,x')|f_a(x) = f_a(x')\} \ , \qquad S^A_{fa} \overset{\text{def}}{=} \{x - x'|f_a(x) = f_a(x')\} \ ,$$
$$S_{fb} \overset{\text{def}}{=} \{(x,x')|f_b(x) = f_b(x')\} \ , \qquad S^A_{fb} \overset{\text{def}}{=} \{x - x'|f_b(x) = f_b(x')\} \ .$$

Choose the odd elements from S^A_{fb} and define them to be $S^A_{fb_{odd}}$. Note that we can easily precompute all the above solution sets using 2^{32} evaluations of the appropriate functions and 2^{32} words of memory (or some more computations with proportionally less memory).

4 (H,t)-Type Attack

First, let us try to attack only the middle part of the compression function, i.e. ProcessMessage function. It consists of 8 rounds (10 rounds for LAKE-512). In every round, first all of the 16 internal variables are updated by the function f, and then all of them are updated by the function g.

Our differential trail is as follows:

1. Introduce a carefully chosen difference in F_0.
2. After the first application of the function f from all L_i, only L_0 has a non-zero difference.
3. After the first application of the function g none of W_i have any difference.

Let us show that this differential is possible. First let us prove that Step 2 is achievable. Considering that $L_i = f(L_{i-1}, F_i, M_{\sigma(i)}, C_i)$, we get that in L_i a difference can be introduced only through L_{i-1} and F_i (message words do not have differences, C_i are simply constants). Note that in the first round $\sigma(i)$ is defined as the identity permutation hence we can write M_i instead of $M_{\sigma(i)}$.

For ΔL_0 we require a non-zero difference

$$\Delta L_0 = f(F_{15}, F_0^{'}, M_0, C_0) - f(F_{15}, F_0, M_0, C_0) \neq 0. \qquad (2)$$

For ΔL_1 we require the zero difference

$$\Delta L_1 = f(L_0^{'}, F_1, M_1, C_1) - f(L_0, F_1, M_1, C_1) = 0. \tag{3}$$

From Observation 1, it follows that it is possible to get zero for ΔL_1. For all the other $\Delta L_i, i = 2..15$ we require the zero difference. This is trivially fulfilled because there are no inputs with difference. Now, let us consider Step 3. Note that $W_i = g(W_{i-1}, L_i, F_i, L_{i+1})$, so we can introduce a difference in W_i by any of W_{i-1}, L_i, F_i and L_{i+1}.

For ΔW_0, we require the zero difference, so we get

$$\Delta W_0 = g(L_{15}, L_0^{'}, F_0^{'}, L_1) - g(L_{15}, L_0, F_0, L_1) = 0. \tag{4}$$

Note that there are differences in two variables, L_0 and F_0, hence the above equation can be solved. For the indexes $i = 1, \ldots, 14$, we obtain

$$\Delta W_i = g(W_{i-1}, L_i, F_i, L_{i+1}) - g(W_{i-1}, L_i, F_i, L_{i+1}) = 0. \tag{5}$$

All the above equations hold as there are no differences in any of the arguments.

For W_{15}, we have

$$\Delta W_{15} = g(W_{14}, L_{15}, F_{15}, W_0) - g(W_{14}, L_{15}, F_{15}, W_0) = 0.$$

Notice that the last argument is not L_0 but rather W_0 because there are no temporal variables that store the previous values of L_i (see ProcessMessage). This non-symmetry in the ProcessMessage, which updates L registers stops the flow of the difference from L_0 to W_{15}.

So, after only one round, we can obtain an internal state with all-zero differences in the variables. Then the following rounds can not introduce any difference because there are no differences in the internal state variables or in the message words. So, if we are able to solve the equations that we have got then *the attack is applicable to any number of rounds, i.e. increasing the number of rounds in the ProcessMessage function does not improve the security of LAKE.*

Let us take a closer look at our equations. Equation (2) can be written as

$$\begin{aligned}\Delta L_0 &= f(F_{15}, F_0^{'}, M_0, C_0) - f(F_{15}, F_0, M_0, C_0) = \\ &= (F_0^{'} \vee C_0) - (F_0 \vee C_0) + [F_0^{'} + (M_0 \oplus C_0)] \ggg 13 - [F_0 + (M_0 \oplus C_0)] \ggg 13.\end{aligned}$$

Hereafter we will use that $(A + B) \ggg r = (A \ggg r) + (B \ggg r)$ with the probability $\frac{1}{4}$ (see [6]). The same holds when rotation to the left is used. Therefore, the above equation can be rewritten as

$$\Delta L_0 = (F_0^{'} \vee C_0) - (F_0 \vee C_0) + F_0^{'} \ggg 13 - F_0 \ggg 13. \tag{6}$$

Equation (3) can be written as

$$\begin{aligned}\Delta L_1 =& f(L_0^{'}, F_1, M_1, C_1) - f(L_0, F_1, M_1, C_1) = \\ &= L_0^{'} - L_0 + [M_1 + (L_0^{'} \wedge C_1)] \ggg 7 - [M_1 + (L_0 \wedge C_1)] \ggg 7 = \\ &= L_0^{'} - L_0 + (L_0^{'} \wedge C_1) \ggg 7 - (L_0 \wedge C_1) \ggg 7 = 0.\end{aligned}$$

Equation (4) can be written as

$$\begin{aligned}\Delta W_0 =& g(L_{15}, L_0', F_0', L_1) - g(L_{15}, L_0, F_0, L_1) = \\ =& [(L_{15} + L_0') \ggg 1] \oplus (F_0' + L_1) - [(L_{15} + L_0) \ggg 1] \oplus (F_0 + L_1) = 0.\end{aligned}$$

Let us try to extend the collision attack on the ProcessMessage function to the full compression function. First, let us deal with the initialization (function SaltState).

From the initialization of LAKE, it can be seen that the variables H_0 through H_7 are copied into F_0 through F_7. The variable F_8 depends on H_0 and t_0. Similarly, F_9 depends on H_1 and t_1. The rest of the variables do not depend on either t_0 or t_1. Since we need a difference in F_0 (for the previous attack on ProcessMessage function), we will introduce difference in H_0. Further, we can follow our previous attack on the ProcessMessage block and get collisions after the ProcessMessage function. The only difficulty is how to deal with F_8 since it does depend on H_0, which now has a non-zero difference. As a way out, we use the block index t_0. By introducing a difference in t_0 we can cancel the difference from H_0 in F_8. So we get the following equation

$$\begin{aligned}\Delta F_8 &= g(H_0', S_0 \oplus t_0', C_0, 0) - g(H_0, S_0 \oplus t_0, C_0, 0) = \\ &= ((H_0' + (S_0 \oplus t_0')) \ggg 1 \oplus C_0) - ((H_0 + (S_0 \oplus t_0)) \ggg 1 \oplus C_0) = 0.\end{aligned}$$

Let $\tilde{t_0'} = t_0' \oplus S_0$ and $\tilde{t_0} = t_0 \oplus S_0$. Then, the above equation gets the following form

$$\Delta F_8 = H_0' - H_0 + \tilde{t_0'} - \tilde{t_0} = 0.$$

Now, let us deal with the last building block of the compression function, the FeedForward function. Note that we have differences in H_0 and t_0 only. If we take a glance at the FeedForward procedure, we can see that H_0 and t_0 can be found in the same equation, and only there, which defines the new value for H_0. Since we require the zero difference in all of the output variables, we get the following equation

$$\begin{aligned}\Delta H_0 &= f(F_0, F_8, H_0', S_0 \oplus t_0') - f(F_0, F_8, H_0, S_0 \oplus t_0) = \\ &= \tilde{t_0'} \ggg 7 - \tilde{t_0} \ggg 7 + (\tilde{t_0'} \oplus H_0') \ggg 13 - (\tilde{t_0} \oplus H_0) \ggg 13 = 0.\end{aligned}$$

This concludes our attack. We have shown that if we introduce a difference in the chaining value H_0 and the block index t_0 only, it is possible to reduce the problem of finding collisions for the compression function of LAKE to the problem of solving a system of equations.

4.1 Solving Equation Systems

To find a collision for the full compression function of LAKE, we have to solve the equations that were mentioned in the previous sections. As a result, we get the following system equations (note that $H_0 = F_0$)

$$L_0^{'} - L_0 + (L_0^{'} \wedge C_1) \ggg 7 - (L_0 \wedge C_1) \ggg 7 = 0; \tag{7}$$

$$L_0^{'} - L_0 = (H_0^{'} \vee C_0) - (H_0 \vee C_0) + H_0^{'} \ggg 13 - H_0 \ggg 13; \tag{8}$$

$$[(L_{15} + L_0^{'}) \ggg 1] \oplus (H_0^{'} + L_1) - [(L_{15} + L_0) \ggg 1] \oplus (H_0 + L_1) = 0; \tag{9}$$

$$H_0^{'} - H_0 + \tilde{t_0^{'}} - \tilde{t_0} = 0; \tag{10}$$

$$\tilde{t_0^{'}} \ggg 7 - \tilde{t_0} \ggg 7 + (\tilde{t_0^{'}} \oplus H_0^{'}) \ggg 13 - (\tilde{t_0} \oplus H_0) \ggg 13 = 0. \tag{11}$$

Let us analyze Equation (7). By fixing $L_0^{'} - L_0 = R$ and rotating to the left by 7 bits, this equation can be rewritten as

$$(X + A) \wedge C = X \wedge C + B, \tag{12}$$

where $X = L_0, A = R, B = (-R) \lll 7, C = C_1$. Now, let us analyze Equation (8). Again, let us fix $L_0^{'} - L_0 = R$ and $H_0^{'} - H_0 = D$. Then Equation(8) gets the following form

$$(X + A) \vee C = X \vee C + B, \tag{13}$$

where $X = H_0, A = D, B = R - (D \ggg 13), C = C_0$. In Equation (9), if we regroup the components, we obtain

$$[(L_{15} + L_0^{'}) \oplus (L_{15} + L_0)] \ggg 1 = (H_0^{'} + L_1) \oplus (H_0 + L_1).$$

Then, the above equation is of the following form

$$((X + A) \oplus X) \ggg 1 = (Y + B) \oplus Y, \tag{14}$$

where $X = L_{15} + L_0, A = L_0^{'} - L_0, Y = L_1 + H_0, B = H_0^{'} - H_0$.

Now, let us analyze Equations (10) and (11). Let us fix $H_0^{'} - H_0 = D$. Note that from Equation (10), we have $\tilde{t_0^{'}} - \tilde{t_0} = -D$. If we rotate everything by 13 bits to the left in Equation (11), we get

$$(-D) \lll 6 + (\tilde{t_0^{'}} \oplus H_0^{'}) - (\tilde{t_0} \oplus H_0) = 0; \tag{15}$$

$$\tilde{t_0} = [(\tilde{t_0^{'}} \oplus H_0^{'}) - D \lll 6] \oplus H_0. \tag{16}$$

If we substitute $\tilde{t_0}$ in Equation (10) by the above expression, then we have

$$D + \tilde{t_0^{'}} - [(\tilde{t_0^{'}} \oplus H_0^{'}) - D \lll 6] \oplus H_0 = 0; \tag{17}$$

$$\tilde{t_0^{'}} = [(\tilde{t_0^{'}} \oplus H_0^{'}) - D \lll 6] \oplus H_0 - D. \tag{18}$$

If we XOR the value of $H_0^{'}$ to the both sides, we get

$$\tilde{t_0^{'}} \oplus H_0^{'} = ([(\tilde{t_0^{'}} \oplus H_0^{'}) - D \lll 6] \oplus H_0 - D) \oplus H_0^{'}. \tag{19}$$

Let us denote $\tilde{t_0'} \oplus H_0' = X$. Then we can write

$$X = [(X - D \lll 6) \oplus H_0 - D] \oplus H_0'; \tag{20}$$

$$X \oplus H_0' = (X - D \lll 6) \oplus H_0 - D. \tag{21}$$

Finally, we get an equation of the following form

$$(X \oplus K_1) + A = (X + B) \oplus K_2, \tag{22}$$

where $K_1 = H_0', A = R, B = -R \lll 6, K_2 = H_0$.

Lemma 2. *There exist efficient algorithms **Al1**,**Al2**,**Al3**,**Al4** for finding solutions for equations of type (12),(13),(14),(22).*

The description of these algorithms can be found in Appendix B.

Now, we can present our algorithm for finding solutions for the system of equations. With **Al1** we find a difference R (and values for L_0, L_0') such that Equation (7) holds. Actually, for the same difference R many distinct solutions (L_0, L_0') exist (experiments show that when Equation (7) is solvable, then there are around 2^5 solutions). Next, we pass as an input to **Al2** the difference R and we find a difference D (and values for H_0, H_0') such that Equation (8) holds. Again for a fixed R and D, many pairs (H_0, H_0') exist. We verified experimentally that for a random R and a "good" D, there are around 2^{10} solutions. Using Algorithm **Al3**, we check if we can find solutions for Equation (9), i.e. we try to find L_1 and L_{15}. Note that the input of **Al3** is the previously found sequence (L_0, L_0', H_0, H_0'). If **Al3** can not find a solution, then we get another pair (H_0, H_0') (or generate first a new difference D and then generate another 2^{10} pairs (H_0, H_0')). If **Al3** finds a solution to (9), then we use Algorithm **Al4** and try to find solutions for Equations (10) and (11), where the input to **Al4** is already found as the pair (H_0, H_0'). If **Al4** can not find a solution, then we can take a different pair (H_0, H_0') (or generate first a new difference D and then generate (H_0, H_0')) and then apply first **Al3** and then **Al4**.

4.2 Complexity of the Attack

Let us try to find the complexity of the algorithm. Note that when analyzing the initial equations, we have used the assumption that $(A + B) \ggg r = (A \ggg r) + (B \ggg r)$, which holds with the probability $\frac{1}{4}$(see [6]). In total, we used this assumption 5 times. In the equation for ΔF_0, we can control the exact value of M_1, so in total, we have used the assumption 4 times. Therefore, the probability that a solution of the system is a solution for the initial equations is 2^{-8}. This means that we have to generate 2^8 solutions for the system. Let us find the cost for a single solution.

The average complexity for both **Al1** and **Al2** is 2^1 steps. We confirmed experimentally that, for a random difference R, there exists a solution for Equation (7) with the probability 2^{-27}. So this takes $2^{27} \cdot 2^1 = 2^{28}$ steps using **Al1** and it finds 2^5 solutions for Equation (7). Similarly, for a random difference D, there is a solution for Equation (8) with the probability 2^{-27}. Therefore, this consumes $2^{27} \cdot 2^1 = 2^{28}$ steps and finds 2^{10} pairs (H_0, H_0) for Equation (8). The probability that a pair is a good pair for Equation (9) is 2^{-1} and that it is a good pair for Equations (10) and (11) is 2^{-12} (as explained in Appendix B). Thus, we need $2^1 \cdot 2^{12} = 2^{13}$ pairs, which we can be generated in $2^{28} \cdot 2^3 = 2^{31}$ steps. Since we need 2^8 solutions, the total complexity is 2^{39}. Note that this complexity estimate (a step) is measured by the number of calls to the algorithms that solve our specific equations. If we assume that a call to the algorithms is four times less efficient than the call to the functions f or g (which on average seems to be true), and consider the fact that the compression function makes a total of around 2^8 calls to the functions f or g, then we get that the total complexity of the collision search is around 2^{33} compression function calls.

Note that when a solution for the system exists, then this still does not mean that we have a collision. This is partially because we cannot control some of the values directly. Indeed, we can control directly only $H_0, H_0^{'}, t_0, t_0^{'}$. The rest of the variables, i.e. $L_0, L_0^{'}, L_1, L_{15}$, we can control through the message words M_i or with the input variables H_i, where $i > 0$. Since we pass these values as arguments for the non-injective function f, we may experience situation when we cannot get the exact value that we need. Yet, with an overwhelming probability, we can find the exact values. Let us suppose that we have a solution $(H_0, H_0^{'}, L_0, L_0^{'}, L_1, L_{15}, t_0, t_0^{'})$ for the system of equations. First, we find a message word M_0 such that $f(F_{15}, H_0, M_0, C_0) = L_0$. Notice that F_{15} can be previously fixed by choosing some value for H_7. Then, $f(F_{15}, H_0^{'}, M_0, C_0) = L_0^{'}$. We choose M_1 such that $[M_1 + (L_0^{'} \wedge C_1)] \ggg 7 - [M_1 + (L_0 \wedge C_1)] \ggg 7 = (L_0^{'} \wedge C_1) \ggg 7 - (L_0 \wedge C_1) \ggg 7$. This way the probability that the previous identity holds becomes 1. Then we find H_1 such that $f(L_0, H_1, M_1, C_1) = L_1$. At last, we find M_{15} such that $f(L_{14}, F_{15}, M_{15}, C_{15}) = L_{15}$. If such M_{15} does not exist, then we can change the value of L_{14} by changing M_{14} and then try to find M_{15}.

5 (H, S)-Type Attack

The starting idea for this attack is to inject differences in the input chaining variable H and the salt S and then cancel them within the first iteration of ProcessMessage. Consequently, no difference appears throughout the compression function until the FeedForward step. If the differences in the chaining and salt variables are selected properly, we can hope they cancel each other, so we get no difference at the output of the compression function.

5.1 Finding High-Level Differentials

To find a suitable differential for the attack, an approach similar to the one employed to analyse FORK-256 [10, Section 6] can be used. We model each of the registers a, b, c, d, as a single binary value δa, δb, δc, δd that denotes whether there is a difference in the register or not. Moreover, we assume that we are able to make any two differences cancel each other to obtain a model that can be expressed in terms of arithmetics over $\mathbb{F}_2$. We model the differential behavior of function g simply as $\delta g(\delta a, \delta b, \delta c, \delta d) = \delta a \oplus \delta b \oplus \delta c \oplus \delta d$, where $\delta a, \delta b, \delta c, \delta d \in \mathbb{F}_2$, as this description seems to be functionally closest to the original. For example, it is impossible to get collisions for g when only one variable has differences and such a model ensures that we always have two differences to cancel each other if we need no output difference of g. When deciding how to model $f(a, b, c, d)$, we have more options. First, note that when looking for collisions, there are no differences in message words and the last parameter of f is a constant, so we need to deal with differences in only two input variables a and b. Since we can find collisions for f when differences are only in a single variable (either a or b), we can model f not only as $\delta f(\delta a, \delta b) = \delta a \oplus \delta b$ but more generally as $\delta f(\delta a, \delta b) = \gamma_0(\delta a) \oplus \gamma_1(\delta b)$, where $\gamma_0, \gamma_1 \in \mathbb{F}_2$ are fixed parameters. Let us call the pair (γ_0, γ_1) a γ-configuration of δf and denote it by $\delta f_{[\gamma_0, \gamma_1]}$, As an example, $\delta f_{[1,0]}$ corresponds to $\delta f(\delta a, \delta b) = \delta a$, which means that whenever a difference appears in register b, we need to use the properties of f to find collisions in the coordinate b. For functions f appearing in FeedForward, we use the model $\delta f = \delta a \oplus \delta b \oplus \delta c \oplus \delta d$.

With these assumptions, it is easy to see that such a model of the whole compression function is linear over $\mathbb{F}_2$ and finding the set of input differences (in chaining variables $H_0, \dots, H_7$ and salt registers $S_0, \dots, S_3$) is just a matter of finding the kernel of a linear map. Since we want to find only simple differentials, we are interested in those that use as few registers as possible. To find them, we can think of all possible states of the linear model as a set of codewords of a linear code over $\mathbb{F}_2$. That way, finding differentials affecting only few registers corresponds to finding low-weight codewords. So instead of an enumeration of all 2^{12} possible states of of $H_0, \dots, H_7, S_0, \dots, S_3$ for each γ-configuration of f functions, this can be done more efficiently by using tools like MAGMA [4].

We implemented this method in MAGMA and performed such a search for all possible γ-configurations of the 16 functions f appearing in the first ProcessMessage. We used the following search criteria: (a) as few active f functions as possible; (b) as few active g functions as possible; (c) non-zero differences appear only in the first few steps using function g as it is harder to adjust the values for later steps due to lack of variables we control; (d) we prefer γ-configurations $[1, 0]$ and $[0, 1]$ over $[1, 1]$ because it seems easier to deal with differences in one register than in two registers simultaneously.

The optimal differential for this set of criteria contains differences in registers $H_0, H_1, H_4, H_5, S_0, S_1$ with the following γ-configurations of the first seven f functions in ProcessMessage: $[0, 1]$, $[1, 1]$, $[0, 1]$, $[\cdot, \cdot]$, $[0, 1]$, $[1, 1]$, $[0, 1]$ (Note a simpler configuration (H_0, H_4, S_0) is not possible here). Unfortunately, the system of constraints resulting from that differential has no solutions, so we introduced a small modification of it, adding differences in registers H_2, H_6, S_2, ref. Figure 1. After introducing these additional differences, we gain more freedom at the expense of dealing with more active functions and we can find solutions for

```
SaltState
input: $H_0, \ldots, H_7,\ S_0, \ldots, S_3,\ t_0, t_1$

$\Delta F_0 \leftarrow \Delta H_0$
$\Delta F_1 \leftarrow \Delta H_1$
$\Delta F_2 \leftarrow \Delta H_2$
$F_3 \leftarrow H_3$
$\Delta F_4 \leftarrow \Delta H_4$
$\Delta F_5 \leftarrow \Delta H_5$
$\Delta F_6 \leftarrow \Delta H_6$
$F_7 \leftarrow H_7$
$F_8 \leftarrow g(\Delta H_0, \Delta S_0 \oplus t_0, C_8, 0)$ {s1}
$F_9 \leftarrow g(\Delta H_1, \Delta S_1 \oplus t_1, C_9, 0)$ {s2}
$F_{10} \leftarrow g(\Delta H_2, \Delta S_2, C_{10}, 0)$ {s3}
$F_{11} \leftarrow g(H_3, S_3, C_{11}, 0)$
$F_{12} \leftarrow g(\Delta H_4, \Delta S_0, C_{12}, 0)$ {s4}
$F_{13} \leftarrow g(\Delta H_5, \Delta S_1, C_{13}, 0)$ {s5}
$F_{14} \leftarrow g(\Delta H_6, \Delta S_2, C_{14}, 0)$ {s6}
$F_{15} \leftarrow g(H_7, S_3, C_{15}, 0)$
output: $F_0, \ldots, F_{15}$
```

```
FeedForward
input: $R_0, \ldots, R_{15},\ H_0, \ldots, H_7,$
       $S_0, \ldots, S_3,\ t_0, t_1$
$H_0 \leftarrow f(R_0, R_8, \Delta S_0 \oplus t_0, \Delta H_0)$ {f1}

$H_1 \leftarrow f(R_1, R_9, \Delta S_1 \oplus t_1, \Delta H_1)$ {f2}

$H_2 \leftarrow f(R_2, R_{10}, \Delta S_2, \Delta H_2)$ {f3}
$H_3 \leftarrow f(R_3, R_{11}, S_3, H_3)$
$H_4 \leftarrow f(R_4, R_{12}, \Delta S_0, \Delta H_4)$ {f4}
$H_5 \leftarrow f(R_5, R_{13}, \Delta S_1, \Delta H_5)$ {f5}
$H_6 \leftarrow f(R_6, R_{14}, \Delta S_2, \Delta H_6)$ {f6}
$H_7 \leftarrow f(R_7, R_{15}, S_3, H_7)$
output: $H_0, \ldots, H_7$
```

```
ProcessMessage
input: $F_0, \ldots, F_{15},\ M_0, \ldots, M_{15},\ \sigma$
$L_0 \leftarrow f(F_{15}, \Delta F_0, M_{\sigma(0)}, C_0)$ {p1}
$\Delta L_1 \leftarrow f(L_0, \Delta F_1, M_{\sigma(1)}, C_1)$ {p2}
$\Delta L_2 \leftarrow f(\Delta L_1, \Delta F_2, M_{\sigma(2)}, C_2)$ {p3}
$L_3 \leftarrow f(\Delta L_2, F_3, M_{\sigma(3)}, C_3)$ {p4}
$L_4 \leftarrow f(L_3, \Delta F_4, M_{\sigma(4)}, C_4)$ {p5}
$\Delta L_5 \leftarrow f(L_4, \Delta F_5, M_{\sigma(5)}, C_5)$ {p6}
$\Delta L_6 \leftarrow f(\Delta L_5, \Delta F_6, M_{\sigma(6)}, C_6)$ {p7}
$L_7 \leftarrow f(\Delta L_6, F_7, M_{\sigma(7)}, C_7)$ {p8}
$L_8 \leftarrow f(L_7, F_8, M_{\sigma(8)}, C_8)$
$\vdots$
$L_{15} \leftarrow f(L_{14}, F_{15}, M_{\sigma(15)}, C_{15})$

$W_0 \leftarrow g(L_{15}, L_0, \Delta F_0, \Delta L_1)$ {p9}
$W_1 \leftarrow g(W_0, \Delta L_1, \Delta F_1, \Delta L_2)$ {p10}
$W_2 \leftarrow g(W_1, \Delta L_2, \Delta F_2, L_3)$ {p11}
$W_3 \leftarrow g(W_2, L_3, F_3, L_4)$
$W_4 \leftarrow g(W_3, L_4, \Delta F_4, \Delta L_5)$ {p12}
$W_5 \leftarrow g(W_4, \Delta L_5, \Delta F_5, \Delta L_6)$ {p13}
$W_6 \leftarrow g(W_5, \Delta L_6, \Delta F_6, L_7)$ {p14}
$W_7 \leftarrow g(W_6, L_7, F_7, L_8)$
$\vdots$
$W_{15} \leftarrow g(W_{14}, L_{15}, F_{15}, W_0)$
output: $W_0, \ldots, W_{15}$
```

Fig. 1. High-level differential used to look for (H, S)-type collisions

the system of constraints. The labels for all constraints are defined by Figure 1, we will refer to them throughout the text.

The process of finding the actual pair of inputs following the differential can be split into two phases. The first one is to solve the constraints from ProcessMessage to get the required Fs (same as Hs used in SaltState). Then, in the second phase, we look at the SaltState to find appropriate salts to have constraints in FeedForward satisfied. We can do this because the output from ProcessMessage has only a small effect on the solutions for FeedForward.

5.2 Solving the ProcessMessage

An important feature of our differentials in ProcessMessage is that it can be separated into two disjoint groups, i.e. $(F_0, F_1, F_2, L_1, L_2)$ and $(F_4, F_5, F_6, L_5, L_6)$. Differentials for these two groups have exactly the same structure. Thanks to that, if we can find values for the differences in the first group, we can reuse them for the second group by making corresponding registers in the second group equal to the ones from the first group. Following Observation 2 we can safely say that the second group also follows the differential path with a high probability. Algorithm 5 gives the details of solving the constrains in the first group of ProcessMessage.

```
 1: Randomly pick $(L_2, L'_2) \in S_{fa}$
 2: repeat
 3:     Randomly pick $F_1$, compute $F'_1 = -1 - \Delta L_2 - F_1$
 4: until $f_b(F_1) - f_b(F'_1) \in S^A_{fb_{odd}}$
 5: repeat
 6:     Randomly pick $L_1, F_2$
 7:     Compute $L'_1 = f_b(F'_1) - f_b(F_1) + L_1$
 8:     Compute $F'_2$ so that $f_b(F'_2) = \Delta L_2 + f_a(L_1) - f_a(L'_1) + f_b(F_2)$
 9: until $p11$ is fulfilled
10: Pick $(F_0, F'_0) \in S_{fb}$ so that $\Delta F_0 + \Delta L_1 = 0$
```

Algorithm 5. Find solutions for the first group of differences of ProcessMessage

Correctness. We show that after the execution of Algorithm 5, it indeed finds values conforming to the differential. In other words, we show that constraints $p1 - p4$ and $p9 - p11$ hold. Referring to Algorithm 5:

Line 1: (L_2, L'_2) is chosen in such a way that $p4$ is satisfied.
Line 3: F'_1 is computed in such a way that $(F_1 + L_2) \oplus (F'_1 + L'_2) = -1$
Line 4: $\Delta L_1 = \Delta f_b(F_1)$ is odd together with $(F_1 + L_2) \oplus (F'_1 + L'_2) = -1$. This implies that $p10$ could hold, which will be discussed later in Lemma 3. The fact that $\Delta L_1 \in S^A_{fb_{odd}}$ makes it possible that $p1$ and $p9$ hold.
Line 7: L'_1 is computed in such a way that $p2$ holds.
Line 8: F'_2 is computed in such a way that $p3$ holds.
Line 9: after exiting the loop $p11$ holds.
Line 10: (F_0, F'_0) is chosen in such a way that $p1, p9$ hold.

Probability and Complexity Analysis. Let us consider the probability for exiting the loops in Algorithm 5. We require $f_a(F_1) - f_a(F'_1) \in S^A_{fb_{odd}}$ and the constraint $p11$ to hold. The size of the set $S^A_{fb_{odd}}$ is around 2^{11}. By assuming that $f_a(F_1) - f_a(F'_1)$ is random, the probability to have it in $S^A_{fb_{odd}}$ is 2^{-21}. This needs to be done only once. Now we show that the constraint $p11$ is satisfied with the probability 2^{-24}. We have sufficiently many choices, i.e. 2^{64}, for (L_1, F_2) to have

$p11$ satisfied. The constraint $p11$ requires that $[(W_1 + L_2) \ggg 1] \oplus (F_2 + L_3) = [(W_1 + L_2']) \ggg 1] \oplus (F_2' + L_3)$, which is equivalent to $[(W_1 + L_2) \oplus (W_1 + L_2')] \ggg 1 = (F_2 + L_3) \oplus (F_2' + L_3)$, where $W_1, L_2, L_2', F_2, F_2'$ are given from previous steps. We have choices for L_3 by choosing an appropriate $M_{\sigma(3)}$. The problem could be rephrased as follows: *given random A and D, what is the probability to have at least one x such that $x \oplus (x + D) = A$?*

To answer this question, let us note first that $x \oplus y = (1, \ldots, 1)$ iff $x + y = -1$. This is clear as $y = \overline{x}$ and always $(x \oplus \overline{x}) + 1 = 0$. Now we can show the following result.

Lemma 3. *For any odd integer d, there exist exactly two x such that $x \oplus (x+d) = (1, \ldots, 1)$. They are given by $x = (-1-d)/2$ and $x = (-1-d)/2 + 2^{n-1}$.*

Proof. $x \oplus (x + d) = -1$ implies that $x + x + d = -1 + k2^n$ for an integer k, so $x = \frac{-1-d+k2^n}{2}$. Only when d is odd, $x = \frac{-1-d}{2} + k2^{n-1}$ an integer and a solution exists. As we are working in modulo 2^n, $k = 0, 1$ are the only solutions. $\square$

Following the lemma, given an odd ΔL_1 and $(F_1 + L_2) \oplus (F_1' + L_2') = -1$, we can always find two W_0 such that $(W_0 + L_1) \oplus (W_0 + L_1') = -1$, then $p10$ follows. Such W_0 could be found by choosing an appropriate L_{15}, which could be adjusted by choosing $M_{\sigma(15)}$ (if such $M_{\sigma(15)}$ does not exist, although the chance is low, we can adjust L_{14} by choosing $M_{\sigma(14)}$).

Coming back to the original question, consider A as "0"s and blocks of "1"s. Following the lemma above, for $A_i = 0$, we need $D_i = 0$ (except "0" as MSB followed by a "1"); for a block of "1"s, say $A_k = A_{k+1} = \cdots = A_{k+l} = 1$, the condition that needs to be imposed on D is $D_k = 1$. By counting the number of "0"s and the number of blocks of "1"s, we can get number of conditions needed. For an n-bit A, the number is $\frac{3n}{4}$ on average (cf. Appendix Lemma 4).

For LAKE-256, it is 24, so the probability for $p11$ to hold is 2^{-24}. We will need to find an appropriate L_3 so that $p11$ holds. Note that we have control over L_3 by choosing the appropriate $M_{\sigma(3)}$. For each differential path found, we need to find message words fulfilling the path. The probability to find a correct message is $1 - \frac{1}{e}$ for the first path by assuming f_c is random (because for a random function from n bits to n bits, the probability that a point from the range has a preimage is $1 - \frac{1}{e}$), and $\frac{4}{9}$ for the second path because of the carry effect. For example, given L_0, F_{15}, F_0, C_0, the probability to have $M_{\sigma(0)}$ so that $L_0 = f(F_{15}, F_0, M_{\sigma(0)}, C_0)$ is $1 - \frac{1}{e}$. The same $M_{\sigma(0)}$ satisfies $L_0' = f(F_{15}', F_0', M_{\sigma(0)}, C_0)$ (note for this case $F_{15}' = F_{15}$ and $L_0 = L_0'$) with the probability $\frac{4}{9}$. So for each message word, the probability for it to fulfill the differential path is 2^{-2}. We have such restrictions on $M_{\sigma(0)} - M_{\sigma(2)}, M_{\sigma(4)} - M_{\sigma(6)}$ (we don't have such restriction on $M_{\sigma(3)}$ and $M_{\sigma(7)}$ because we still have control over F_3 and F_7), so overall complexity for solving ProcessMessage is $5 \cdot 2^{36}$ in terms of calls to f_a or f_b. The compression function of LAKE-256 calls functions f and g 136 times each and f_a, f_b contain less than half of the operations used in f. So the complexity for this part of the attack is 2^{30} in terms of the number of calls to the compression function.

Solving the second group of ProcessMessage. After we are done with the first group, we can have the second group of differential path for free by assigning $F_{i+4} = F_i$, $F'_{i+4} = F'_i$ for $i = 0, 1, 2$ and $L_{i+4} = L_i, L'_{i+4} = L'_i$ for $i = 1, 2$. In this way, we can have the constrains $p5 - p8$ and $p12$ automatically satisfied. Similarly, for the constraints $p13$ and $p14$, we will need appropriate W_4 and L_7. We have control over W_4 by choosing F_3 and L_4 (note we need to keep L_3 stable to have $p11$ satisfied, this can be achieved by choosing appropriate $M_{\sigma(3)}$). We also have control over L_7 by choosing $M_{\sigma(7)}$. That way we can force the difference to vanish within the first ProcessMessage. Table 2 in Appendix shows an example of a set of solutions.

5.3 Near Collisions

In this section we explain how to get a near collision directly from collisions of ProcessMessage. Refer to SaltState and FeedForward in Fig. 1. Note that the function $g(a, b, c, d)$ with differences at positions (a, b) means $\Delta a + \Delta b = 0$, then constraints $(s1 - s6)$ in SaltState can be simplified to

$$s1 : \Delta H_0 + \Delta S_0 = 0; \tag{23}$$
$$s2 : \Delta H_1 + \Delta S_1 = 0; \tag{24}$$
$$s3 : \Delta H_2 + \Delta S_2 = 0. \tag{25}$$

Note that $H_{i+4} = H_i, H'_{i+4} = H'_i$ for $i = 0, 1, 2$ as required by ProcessMessage, Let $t_0 = t_1 = 0$, then conditions $s4 - s6$ follow $s1 - s3$. Conditions in FeedForward could be simplified to

$$f1 : f_{cd}(S_0, H_0) = f_{cd}(S'_0, H'_0), \tag{26}$$
$$f2 : f_{cd}(S_1, H_1) = f_{cd}(S'_1, H'_1), \tag{27}$$
$$f3 : f_{cd}(S_2, H_2) = f_{cd}(S'_2, H'_2) \tag{28}$$

and $f4 - f6$ follow $f1 - f3$. This set of constraints can be grouped into three independent sets (si, fi) for $i = 0, 1, 2$ each one of the same type, i.e. $\Delta H + \Delta S = 0$ and $f_{cd}(S, H) = f_{cd}(S', H')$.

To find near collisions, we proceed as follows. First we choose those S_i with $S'_i = S_i - \Delta H_i$ so that the Hamming weight of $f_{cd}(S'_i, H'_i) - f_{cd}(S_i, H_i)$ is small for $i = 0, 1, 2$. Thanks to that, only small differences are expected in the final output of the compression function, due to the fact that inputs from a, b of the function f have only carry effect to the final difference of f when inputs differ in c, d only. We choose values of S_i without going through the compression function, so the number of rounds of the compression function does not affect our algorithm. Further, the complexity for finding values of S_i is much smaller than that of ProcessMessage, so it does not increase the 2^{30} complexity. Experiments show that, based on the collision in ProcessMessage, we can have near collisions with a very little additional effort. Table 3 in Appendix shows a sample result with 16-bit of differences out of 256 bits of the output.

5.4 Extending the Attack to Full Collisions

It is clear that finding full collisions is equivalent to solving Equations (26)-(28). The complexity to solve a single equation is around 2^{12} (as done for solving Equations (10) and (11)). Looking at Algorithm 5, $(s1, f1)$ can be checked when F_1 and F'_1 are chosen, so it does not affect the overall complexity. The pair $(s0, f0)$ can be checked immediately after (L_1, L'_1) is given as show in Line 7 of Algorithm 5. Similarly, $(s2, f2)$ can be checked after (F_2, F'_2) is chosen in Line 8. So the overall complexity for our algorithm to get a collision for the full compression function is 2^{54}.

5.5 Reducing the Complexity

In this subsection, we show a better way (rather than randomly) to choose (L_2, L'_2) so that the probability for the constraint $p11$ to hold increases, which reduces the complexity for collision finding to 2^{42}.

Note the constraint $p11$ is as follows. Given W_1, L_2, L'_2, what is the probability to have L_3 and (F_2, F'_2) so that $((W_1 + L_2) \oplus (W_1 + L'_2)) \ggg 1 = (F_2+L_3)\oplus(F'_2+L_3)$. We calculate the probability by counting the number of 0s and block of 1s in $((W_1 + L_2)\oplus(W_1 + L'_2)) \ggg 1$ (let's denote it as $\alpha = \#(((W_1+L_2)\oplus(W_1+L'_2)) \ggg 1))$. Now we show that the number α can be reduced within the first loop of the algorithm, i.e. given only (L_2, L'_2) and (F_1, F'_1), we are able to get α and hence, by repeating the loop sufficiently many times, we can reduce α to a number smaller than 24 (we don't fix it here, but will give it later).

Note that to find α, we still need W_1 besides (L_2, L'_2). Now we show W_1 can be computed from (L_2, L'_2) and (F_1, F'_1) only. $W_1 \overset{\text{def}}{=} ((W_0 + L_1) \ggg 1) \oplus (F_1 + L_2)$, where we restrict $(W_0 + L_1) \oplus (W_0 + L'_1) = -1$. Denote $S = (W_0 + L_1)$, then the equation can be derived to $S \oplus (S + \Delta L_1) = -1$, where $\Delta L_1 \overset{\text{def}}{=} f_b(F'_1) - f_b(F_1)$.

So let's make 2^y more effort in the first loop so that α is reduced by y. The probability for the first loop to exit becomes 2^{-33-y} and for the second loop, the probability becomes 2^{-60+y}. Choosing the optimal value $y = 13$ (y must be an integer), the probabilities are 2^{-46} and 2^{-47}, respectively. Hence this gives final complexity 2^{42} for collision searching.

6 $(\boldsymbol{H})$-Type Attack

Let us introduce difference only in the chaining value H_0. Hence, this difference after the SaltState procedure, will produce differences in F_0 and F_8. In the first application of the ProcessMessage procedure the following differential is used:

1. Let F_0 has some specially chosen difference. Also, F_8 has some difference that depends on the difference in F_0.
2. After the first application of the function f only $L_0, L_1, \ldots, L_8$ have non-zero differences
3. After the first application of the function g all W_i have zero differences

Again, we should prove that this differential is possible. Basically, we should check only for the updates with non-zero input differences and zero output difference (other updates hold trivially). Hence, we should prove that we can get the zero difference in L_9 and $W_i, i = 0, \dots, 8$. Since f is non-injective, it is possible to get the zero difference in L_9. For $W_0, \dots, W_8$ is also possible to get zero differences because their updating functions g always have at least two arguments with differences. Therefore, this differential is valid.

Now, let us write the system of equations that we require. Note that $L_i - L_i^{'} = \delta_i, i = 0, \dots, 8$. The system is as follows

$$f\ (F_{15}, L_0, M_0, C_0) = L_0, f(F_{15}, L_0', M_0, C_0) = L_0', \tag{29}$$
$$f\ (L_0, F_1, M_1, C_1) = L_1, f(L_0', F_1, M_1, C_1) = L_1', \tag{30}$$
$$f\ (L_{i-1}, F_i, M_i, C_i) = L_i, f(L_{i-1}', F_i, M_i, C_i) = L_i', i = 2, \dots, 6, \tag{31}$$
$$f\ (L_7, L_8, M_8, C_9) = L_8, f(L_7, L_8', M_8, C_9) = L_8', \tag{32}$$
$$f\ (L_8, F_9, M_9, C_9) = L_9, f(L_8', F_9, M_9, C_9) = L_9, \tag{33}$$
$$g\ (L_{15}, L_0, F_0, L_1) = W_0, g(L_{15}, L_0', F_0', L_1') = W_0, \tag{34}$$
$$g\ (W_{i-1}, L_i, F_i, L_{i+1}) = W_i, g(W_{i-1}, L_i', F_i, L_{i+1}') = W_i, i = 1, \dots, 7, \tag{35}$$
$$g\ (W_7, L_8, L_8, L_9) = g(W_7, L_8^{'}, L_8^{'}, L_9). \tag{36}$$

Let us focus on Equation (35). It can be rewritten as

$$(W_{i-1} + L_i) \ggg 1 \oplus (F_i + L_{i+1}) = (W_{i-1} + L_i^{'}) \ggg 1 \oplus (F_i + L_{i+1}^{'}) \quad (= W_i).$$

Similarly as in the previous attacks, we get the following equation

$$((X + A) \oplus X) \ggg 1 = (Y + B) \oplus Y, \tag{37}$$

where $X = W_{i-1} + L_i^{'}, A = L_i - L_i^{'}, Y = F_i + L_{i+1}^{'}, B = L_{i+1} - L_{i+1}^{'}$. In **Al3** of Appendix B, we have explained how to split this equation into two equations, $((X + A) \oplus X) = -1$,$(Y + B) \oplus Y$=-1, and solve them separately. The solution $X = \overline{A \gg 1}, Y = \overline{B \gg 1}$ exists when LSB of A and B are 1. Hence, for W_{i-1} and F_i we get

$$W_{i-1} = \overline{(L_i - L_i^{'}) \gg 1} - L_i^{'} = \overline{\delta_i \gg 1} - L_i^{'}, \tag{38}$$
$$F_i = \overline{(L_{i+1} - L_{i+1}^{'}) \gg 1} - L_{i+1}^{'} = \overline{\delta_{i+1} \gg 1} - L_{i+1}^{'}. \tag{39}$$

If we put these values in the equation for W_i we obtain

$$W_i = (W_{i-1} + L_i^{'}) \ggg 1 \oplus (F_i + L_{i+1}^{'}) = \overline{\delta_i \gg 1} \ggg 1 \oplus \overline{\delta_{i+1} \gg 1}. \tag{40}$$

This means that we can split equations of the type (35) into two equations and solve them separately. Also, from (38) and (39) we get that $W_i = F_i$.

Now let us explain how to get two pairs that satisfy the whole differential. First, by choosing randomly $L_0, L_0^{'}, F_{15}, M_0, F_1$, and M_1, we produce a solution for Equations (29),(30), (34) and (35). Actually, we need to satisfy only Equation (35), i.e. $W_0 = \overline{(L_1 - L_1^{'}) \gg 1} - L_1^{'} = \overline{\delta_1 \gg 1} - L_1^{'}$, because the values of

$L_0^j, L_1^j, j = 1, 2$ can be any, and finding a solution for (34) is trivial. Then, by taking some M_2 and F_2 we produces $L_2^j = f(L_1^j, F_2, M_2, C_2), j = 1, 2$. Having the values of δ_1 and δ_2, we can find the new value of F_1

$$F_1 = W_1 = \overline{\delta_1 \gg 1} \ggg 1 \oplus \overline{\delta_2 \gg 1}.$$

Since we have changed the value of F_1, then the values of L_1 and $L_1^{'}$ might change. Therefore, we find another value of M_1 such that the old values of $L_1, L_1^{'}$ stay the same. Note, that is is not always possible. Yet, with the probability 2^{-2} this value can be found. As a result, we have fixed the values of M_1, F_1, L_2, and $L_2^{'}$. Using the same technique, we can fix the values of $M_2, \ldots, M_6, F_2, F_6, L_3^j, L_7^j, j = 1, 2$ such that (35) would hold for $i = 2, \ldots, 6$. In short, the following is done. Let the values of W_{i-1}, M_i, F_i, L_i, and $L_i^{'}$ be fixed. First we generate any L_{i+1} and $L_{i+1}^{'}$. Then we find the value of F_i from (39). Then, we change the value of M_i. This way, the values of $L_i, L_i^{'}$ stay the same, but now $W_{i+1}, L_i^j, M_i, F_i, L_{i+1}^j, j = 1, 2$ satisfy (35).

Now let us fix the right $L_8, L_8^{'}$ such that

$$f(L_8, F_9, M_9, C_9) = f(L_8^{'}, F_9, M_9, C_9). \tag{41}$$

We try different M_8, S_0 (notice that the values of $F_8, F_8^{'}$ depend on $F_0, F_0^{'}$, and S_0), and create different pairs $(L_8, L_8^{'})$. If this pair satisfies (41) and (38) then we change M_7 and F_7 as described previously. Finally, we change M_9 and F_9 so that (36) will hold. First, we find the good value of L_9 from the equation $L_9 = \overline{\Delta_2 \gg 1} - L_8^{'}$ and than change M_9 and F_9 to achieve this value. As a result, we have fixed all the values such that all equations hold.

After the ProcessMessage procedure, there are no differences in any of the state variables. The FeedForward procedure, which produces the new chaining value, depends on the initial chaining value, the internal state variables, the salt, and the block index. Since there is a difference only in the initial chaining value (only in H_0), it means that there has to be a difference in the new chaining variable H_0 (and only there). If we repeat the attack on ProcessMessage with different input difference Δ_1, we can produce a near collision with a low Hamming difference. If, in the truncated digest LAKE-224, the first 32 bits were truncated instead of the last 32 bits, we could find a real collisions for the compression function of LAKE-224.

Now, let us estimate the complexity of our attack. For finding good random $L_0, L_0^{'}, F_{15}, M_0, F_1$, and M_1 that satisfies the first set of equations we have to try 2^{32} different values. For successfully fixing the correct $F_i, M_i, i = 1, \ldots, 7$, we have to start with $(2^2)^7 = 2^{14}$ different δ_1. For finding a good pair $(L_8, L_8^{'})$ that satisfies (41) and (38) we have to try $2^{27} \cdot 2^{32} = 2^{59}$ different M_8, S_8. Hence, the total attack complexity is around 2^{105} computations. If we apply the same reasoning for computing the complexity in the number of compression function calls as it was done in the two previous attacks, we will get that the near collision algorithm requires around 2^{99} calls to the compression function of LAKE-256.

7 Conclusions

We presented three different collision attacks on the compression function of LAKE-256. All of them make use of some weaknesses of the functions used to build the compression function. The first two of them facilitate the additional variables of salt and block counter required by the HAIFA compression functions. Due to a weak mixing of those variables, we were able to better control diffusion of differences.

All our attacks cancel the injected differences within the first ProcessMessage and later only in the final FeedForward again and therefore are independent of the number of rounds.

The SHA-3 first round candidate BLAKE, a successor of LAKE, uses a different ProcessMessage function. Hence, our attacks do not apply to BLAKE. We believe that the efficient methods to solve the systems of equations and to find high level differentials presented in this paper may be useful to analyse other dedicated designs based on modular additions, rotations and XORs and constitute a nice illustration of how very small structural weaknesses can lead to attacks on complete designs.

Acknowledgments

This work is supported in part by Singapore National Research Foundation under Research Grant NRF-CRP2-2007-03, the Singapore Ministry of Education under Research Grant T206B2204, and the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. Praveen Gauravaram is supported by the Danish Research Council for Technology and Production Sciences grant number 274-08-0052. Dmitry Khovratovich is supported by PRP "Security & Trust" grand of the University of Luxembourg. Krystian Matusiewicz is supported by the Danish Research Council for Technology and Production Sciences grant number 274-07-0246. Ivica Nikolić is supported by the Fonds National de la Recherche Luxembourg grant TR-PHD-BFR07-031. Josef Pieprzyk is supported by Australian Research Council grants DP0663452 and DP0987734. We thank Wei Lei for helpful discussions and anonymous reviewers of ASIACRYPT 2008 and FSE 2009 for their useful comments.

References

1. Aumasson, J.-P., Meier, W., Phan, R.: The hash function family LAKE. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 36–53. Springer, Heidelberg (2008)
2. Biham, E., Dunkelman, O.: A framework for iterative hash functions – HAIFA. IACR ePrint Archive, Report 2007/278 (2007), `http://eprint.iacr.org/2007/278`
3. Biham, E., Dunkelman, O., Bouillaguet, C., Fouque, P.-A.: Re-visiting HAIFA and why you should visit too. In: ECRYPT workshop Hash functions in cryptology: theory and practice (June 2008), `http://www.lorentzcenter.nl/lc/web/2008/309/presentations/Dunkelman.pdf` (accessed on 11/23/2008)

4. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system I: The user language. Journal of Symbolic Computation 24(3-4), 235–265 (1997), `http://magma.maths.usyd.edu.au/`
5. Damgård, I.B.: A design principle for hash functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, Heidelberg (1989)
6. Daum, M.: Cryptanalysis of Hash Functions of the MD4-Family. PhD thesis, Ruhr-Universität Bochum (May 2005)
7. Joux, A.: Multicollisions in iterated hash functions. Application to cascaded constructions. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 306–316. Springer, Heidelberg (2004)
8. Kelsey, J., Kohno, T.: Herding Hash Functions and the Nostradamus Attack. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 183–200. Springer, Heidelberg (2006)
9. Kelsey, J., Schneier, B.: Second preimages on n-bit hash functions for much less than 2^n work. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 474–490. Springer, Heidelberg (2005)
10. Matusiewicz, K., Peyrin, T., Billet, O., Contini, S., Pieprzyk, J.: Cryptanalysis of FORK-256. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 19–38. Springer, Heidelberg (2007)
11. Mendel, F., Schläffer, M.: Collisions for round-reduced LAKE. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 267–281. Springer, Heidelberg (2008)
12. Merkle, R.C.: One way hash functions and DES. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 428–446. Springer, Heidelberg (1989)
13. National Institute of Standards and Technology. Secure hash standard (SHS). FIPS 180-2 (August 2002)
14. Paul, S., Preneel, B.: Solving systems of differential equations of addition. In: Boyd, C., Nieto, J.M.G. (eds.) ACISP 2005. LNCS, vol. 3574, pp. 75–88. Springer, Heidelberg (2005)
15. Wang, X., Yin, Y.L., Yu, H.: Collision search attacks on SHA-1 (Feburary 13, 2005), `http://theory.csail.mit.edu/~yiqun/shanote.pdf`
16. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
17. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)

A Collision Examples

Table 1. (H, t)-colliding pair for the compression function of LAKE

h_0	63809228	6cc286da	00000000	00000000	00000000	00000000	00000000	00000540
h_0'	ba3f5d77	6cc286da	00000000	00000000	00000000	00000000	00000000	00000540
M	55e07658	00000009	00000000	00000000	00000000	00000000	00000000	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	00000002	5c41ab0e
F_0	0265e384	00000000						
F_1	aba71835	00000000						
S	00000000	00000000	00000000	00000000				
H	79725351	e61a903f	730aace9	756be78a	b679b09d	de58951b	f5162345	14113165

Table 2. Example of a pair of chaining values F, F' and a message block M that yield a collision in ProcessMessage

F	1E802CB8	799491C5	1FE58A14	07069BED	1E802CB8	799491C5	1FE58A14	74B26C5B
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
F'	C0030007	B767CE5E	30485AE7	07069BED	C0030007	B767CE5E	30485AE7	74B26C5B
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
M	683E64F1	9B0FC4D9	0E36999A	A9423F09	27C2895E	1B76972D	BEF24B1C	78F25F25
	00000000	00000000	00000000	00000000	00000000	00000000	657C34F5	3A992294
L	D0F3077A	31A06494	395A0001	10E105FC	82026885	31A06494	395A0001	10E105FC
	ECF7389A	2F4D466F	9FFC71E1	54BAFAE6	FCDDBCDB	E635FFB7	5D302719	CD102144
L'	D0F3077A	901D9145	95A99FDB	10E105FC	82026885	901D9145	95A99FDB	10E105FC
	ECF7389A	2F4D466F	9FFC71E1	54BAFAE6	FCDDBCDB	E635FFB7	5D302719	CD102144
$L^{\oplus}$	00000000	A1BDF5D1	ACF39FDA	00000000	00000000	A1BDF5D1	ACF39FDA	00000000
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
W	1F210513	1A8E2515	1932829B	1C00C039	1F210513	1A8E2515	1932829B	F4A060BE
	5F868AC3	D8959978	E8F3FF4A	E20AC1C3	8941C0F8	EA8BC74E	6ECDD677	82CFFECE
W'	1F210513	1A8E2515	1932829B	1C00C039	1F210513	1A8E2515	1932829B	F4A060BE
	5F868AC3	D8959978	E8F3FF4A	E20AC1C3	8941C0F8	EA8BC74E	6ECDD677	82CFFECE

Table 3. Example of a pair of chaining values F, F', salts S, S' and a message block M that yield near collision in CompressionFunction with 16 bits differences out of 256 bits output. Hs are final output.

F	7B2000C4	23E79FBD	73D102C3	88E0E02B	7B2000C4	23E79FBD	73D102C3	00000000
F'	801FF801	18C0005E	846FD480	88E0E02B	801FF801	18C0005E	846FD480	00000000
S	00010081	23043423	03C5B03E	D44CFD2C				
S'	FB010944	2E2BD382	F326DE81	D44CFD2C				
M	00000012	64B31375	CFA0A77E	8F7BE61F	1E30C9D3	6A9FB0DA	290E506E	3AAE159C
	00000000	00000000	00000000	00000000	00000000	00000000	00000000	1B89AA75
H	261B50AA	3873E2BE	BDD7EC4D	7CE4BFF8	007BB4D4	869473FF	833D9EFA	9DABEDDA
H'	361150AA	387BE23E	FDD6E84D	7CE4BFF8	1071B4D4	869C737F	C33C9AFA	9DABEDDA
$H^{\oplus}$	100A0000	00080080	40010400	00000000	100A0000	00080080	40010400	00000000

B Lemmas and Proofs

Lemma 4. *Given random x of length n, then the average number of "0"s and block of "1"s, excluding the case "0" as MSB followed by "1", is $\frac{3n}{4}$.*

Proof. Denote C_n as the sum of the counts for "0"s and blocks of "1"s for all x of length n, denote such x as x_n. Similarly we define P_n as the sum of the counts for all x of length n with MSB "0" (let's denote such x as x_n^0); and Q_n for the sum of the counts for all x of length n with MSB "1" (denote such x as x_n). It is clearly that

$$C_n = P_n + Q_n \tag{42}$$

Note that there are 2^{n-1} many x with length $n-1$, half of them with MSB "0", which contribute to P_{n-1} and the other half with MSB "1", which contribute to Q_{n-1}. Now we construct x_n of length n from x_{n-1} of length $n-1$ in the following way:

– Append "0" with each x_{n-1}, this "0" contribute to C_n once for each x_{n-1} and there are 2^{n-2} many such x_{n-1}.

– Append “1” with each x_{n-1}, this “1” does not contribute to C_n
– Append “0” with each x^0_{n-1}, this contributes 2^{n-2} to C_n
– Append “1” with each x^0_{n-1}, this contributes 2^{n-2} to C_n

So overall we have $C_n = P_{n-1} + P_{n-1} + 2^{n-2} + Q_{n-1} + 2^{n-2} + Q_{n-1} + 2^{n-2} = 3 \cdot 2^{n-2} + 2C_{n-1}$. Note $C_1 = 2$, solving the recursion, we get $C_n = \frac{3n+1}{4} \cdot 2^n$. Exclude the exceptional case, we have final result $\frac{3n}{4}$ on average.

Lemma 5. *Given random $a, a', x \in \mathbb{Z}_{2^n}$ and $k \in [0, n)$, $\alpha \overset{\mathrm{def}}{=} \mathbf{1}[a^L_k + x^L_k \geq 2^k]$, $\alpha' \overset{\mathrm{def}}{=} \mathbf{1}[a'^L_k + x^L_k \geq 2^k]$, $\beta \overset{\mathrm{def}}{=} \mathbf{1}[a^R_k + x^R_k + \alpha \geq 2^{n-k}]$, $\beta' \overset{\mathrm{def}}{=} \mathbf{1}[a'^R_k + x^R_k + \alpha \geq 2^{n-k}]$ as defined in Lemma 1, then $P(\alpha = \alpha', \beta = \beta') = \frac{4}{9}$.*

Proof. Consider α and α' first, $P(\alpha = \alpha' = 1) = P(a^L_k + x^L_k \geq 2^k, a'^L_k + x^L_k \geq 2^k)$. This is equal to $P(x^L_k \geq (2^k - min\{a^L_k, a'^L_k\}))$ what in turns can be rewritten as $P(a^L_k \geq a'^L_k)P(x^L_k \geq 2^k - a'^L_k) + P(a'^L_k > a^L_k)P(x^L_k \geq 2^k - a^L_k) = \frac{1}{2} \cdot \frac{1}{3} + \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{3}$.
Similarly we can prove $P(\alpha = \alpha' = 0) = \frac{1}{3}$, so $P(\alpha = \alpha') = \frac{2}{3}$. Note the definitions of β and β' contain α and α', but $\alpha, \alpha' \in \{0, 1\}$, which is generally much smaller than 2^{n-k}, so the effect of α to β is negligible. We can roughly say $P(\beta = \beta') = \frac{2}{3}$. So $P(\alpha = \alpha', \beta = \beta') = P(\alpha = \alpha')P(\beta = \beta') = \frac{4}{9}$.

Lemma 6. *There exist an algorithm (**Al1**) for finding all the solutions for the equation of the form $(X \wedge C) + A = (X + B) \wedge C$. The complexity of **Al1** depends only on the constant C.*

Lemma 7. *There exist an algorithm (**Al2**) for finding all the solutions for the equation of the form $(X \vee C) + A = (X + B) \vee C$. The complexity of **Al2** depends only on the constant C.*

Proof. The proofs for the two facts are very similar with some minor changes, so we will prove only Lemma 6.
Let $X = x_{31} \ldots x_1 x_0, A = a_{31} \ldots a_1 a_0, B = b_{31} \ldots b_1 b_0, C = c_{31} \ldots c_1 c_0$. Then for each i we have:

$$(x_i \wedge c_i) \oplus a_i \oplus F_i = (x_i \oplus b_i \oplus r_i) \wedge c_i, \tag{43}$$

where $F_i = m(x_{i-1} \wedge c_{i-1}, a_{i-1}, F_{i-1})$ is the carry at the $(i-1)$th position of $(X \wedge C + A)$, $r_i = m(x_{i-1}, b_{i-1}, r_{i-1})$ is the carry at the $(i-1)$th position of $X + B$, and $m(x, y, z) = xy \oplus xz \oplus yz$.
Equation (43), simplifies to $a_i \oplus F_i = 0$ when $c_i = 0$ and when $c_i = 1$ we get $a_i \oplus F_i = b_i \oplus r_i$.
Let us assume that we have found the values for F_i and r_i for some i. We find the smallest $j > 0$ such that $c_{i+j} = 0$. Then from the fact that $a_i \oplus F_i = 0$ and the definition of F_i we get:

$$\begin{aligned} a_{i+j} =& F_{i+j} = m(x_{i+j-1}, a_{i+j-1}, F_{i+j-1}) = \\ =& m(x_{i+j-1}, a_{i+j-1}, m(x_{i+j-2}, a_{i+j-2}, F_{i+j-2})) = \ldots \\ =& m(x_{i+j-1}, a_{i+j-1}, m(x_{i+j-2}, a_{i+j-2}, m(\ldots, m(x_i, a_i, F_i)) \ldots)) \end{aligned}$$

In the above equation, only $x_i, x_{i+1}, \dots x_{i+j-1}$ are unknown. So we can try all the possibilities, which are 2^j, and find all the solutions. Let us denote by $\tilde{X}$ the set of all solutions.

Now, let us find the smallest $l > 0$ such that $c_{i+j+l} = 1$. Notice that we can easily find F_{i+j+1} if considering $c_{i+j+F_0} = 0$ for $F_0 \in (0, l)$ and using $a_i \oplus F_i = 0$:

$$\begin{aligned}
F_{i+j+1} =& m(0, a_{i+j}, F_{i+j}) = m(0, a_{i+j}, a_{i+j}) = a_{i+j} \\
F_{i+j+2} =& m(0, a_{i+j+1}, F_{i+j+1}) = m(0, F_{i+j+1}, F_{i+j+1}) = m(0, a_{i+j}, a_{i+j}) = a_{i+j} \\
& \dots \\
F_{i+j+l} =& m(0, a_{i+j+l-1}, F_{i+j+l-1}) = a_{i+j}
\end{aligned}$$

From the relationship $a_i \oplus F_i = b_i \oplus r_i$ and definition of r_i we get:

$$\begin{aligned}
a_{i+j+l} \oplus F_{i+j+l} \oplus b_{i+j+l} = r_{i+j+l} = m(x_{i+j+l-1}, b_{i+j+l-1}, r_{i+j+l-1}) = \\
= m(x_{i+j+l-1}, b_{i+j+l-1}, m(x_{i+j+l-2}, b_{i+j+l-2}, r_{i+j+l-2})) = \dots \\
= m(x_{i+j+l-1}, b_{i+j+l-1}, m(\dots, m(x_i, b_i, r_i) \dots))
\end{aligned}$$

In the above equation, only $x_i, x_{i+1}, \dots, x_{i+j+l-1}$ are unknown. So we check all the possibilities by taking $(x_i, x_{i+1}, \dots, x_{i+j-1})$ from the set $\tilde{X}$ and the rest of the variables take all the possible values. If the equation has a solution, then this means we have fixed another F_{i+j+l}, r_{i+j+l}, and we can continue searching using the same algorithm.

The complexity of the algorithms is 2^q, where q is size of the longest consecutive sequence of ones followed by consecutive zero sequence (in the case above $q = j + l$) in the constant C. Taking into consideration the value of the constant C_1 used in the compression function of LAKE-256, we get that complexity of our algorithm for this special case is 2^8. Yet, the average complexity can be decreased additionally if first the necessary conditions are checked. For example, if we have two consecutive zeros in the constant C_1 at positions i and $i+1$ then it has to hold $a_{i+1} = a_i$. If we check for all zeros, then only with probability of 2^{-10} a constant A can pass this sieve. Therefore, the math expectancy of the complexity for a random A is less than 2^1. Note that when $\vee$ function is used instead of $\wedge$, than 0 and 1 change place. Therefore, our algorithm has a complexity of 2^6 when C_0 is used as a constant. Yet, same as for $\wedge$, early break-up strategies significantly decrease these complexities for the case when solution does not exist. Again, the average complexity is less than 2^1. □

Lemma 8. *There exist an algorithm (**Al3**) for finding a solution for the following equation:* $((X + A) \oplus X) \gg 1 = (Y + B) \oplus Y$.

Proof. Instead of finding a solution w.r.t. X and Y we split the equation into a system

$$(X + A) \oplus X = -1, \quad (Y + B) \oplus Y = -1 \ . \tag{44}$$

We can do this because the value of -1 is invariant of any rotation. We may loose some solutions, but further we will prove that if such a solution exist then our algorithm will find it with probability 2^{-2}.

We will analyze only left equation of (44); the second one can be solved analogously. Let $X = x_{31} \dots x_0, A = a_{31} \dots a_0$. Then for ith bit we get: $(x_i \oplus a_i \oplus c_i) \oplus x_i = 1$, where c_i is the carry at $(i-1)$ position of $X + A$, i.e. $c_i = m(x_{i-1}, a_{i-1}, c_{i-1})$. Obviously, this equation can be rewritten as $a_i = c_i \oplus 1$. For the $(i+1)$th bit we get $a_{i+1} = c_{i+1} \oplus 1 = m(x_i, a_i, c_i) \oplus 1 = m(x_i, a_i, a_i \oplus 1) \oplus 1 = x_i a_i \oplus x_i(a_i \oplus 1) \oplus a_i(a_i \oplus 1) \oplus 1 = x_i \oplus 1$. So, we can easily find the value of x_i for each i. When $i = 31$, x_{31} can be arbitrary. For the case when $i = 0$, considering that $c_0 = 0$, from $a_i = c_i \oplus 1$ we get $a_0 = 1$. Therefore, if $a_0 = 1$ then (44) is solvable in constant time. The solutions are $X = \overline{A \gg 1} + i2^{32}, i = 0, 1$. Finally, for the whole system, we have that solution exist if $a_0 = b_0 = 1$, which means with probability 2^{-2}. □

Lemma 9. *There exists an algorithm (**Al4**) for finding all the solutions for equations of the type* $(X \oplus C) + A = (X + B) \oplus K$.

Proof. We base our algorithm fully on the results of [14]. There, Paul and Preneel show, in particular, how to solve equations of the form: $(x + y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma$. Let us XOR to the both sides of the initial equation the expression $A \oplus B \oplus C$ and denote $\tilde{K} = K \oplus A \oplus B \oplus C$. Then, the equation gets the following form: $((X \oplus C) + A) \oplus A \oplus B \oplus C = (X + B) \oplus \tilde{K}$. For the $(i+1)$th bit position, we have $\tilde{k}_{i+1} = s_{i+1} \oplus F_{i+1}$, where s_i is the carry at the ith position of $(X \oplus C) + A$, and F_i is the carry at ith position of $X + B$. From the definition of s_i we get $s_{i+1} = (x_i \oplus c_i)a_i \oplus (x_i \oplus c_i)s_i \oplus a_i s_i = (x_i \oplus c_i)a_i \oplus (x_i \oplus c_i \oplus a_i)s_i = (x_i \oplus c_i)a_i \oplus (x_i \oplus c_i \oplus a_i)(\tilde{k}_i \oplus F_i)$.

From the definition of F_i we get $F_{i+1} = x_i b_i \oplus x_i F_i \oplus b_i F_i$. This means that $\tilde{k}_{i+1}$ can be computed from x_i,a_i,b_i,c_i,F_i, and $\tilde{k}_i$. Further, we apply the algorithm demonstrated in [14]. The only difference is that for each bit position we have only two unknowns x_i and F_i, whereas in [14] have three unknowns. Yet, this difference is not crucial, and the algorithm can be applied.

Our experimental results (Monte-Carlo with 2^{32} trials), show that the probability that a solution exists, when A, B, C and K are randomly chosen is around 2^{-12}. □