# Practical pseudo-collisions for hash functions ARIRANG-224/384

Guo, Jian; Matusiewicz, Krystian; Knudsen, Lars R.; Ling, San; Wang, Huaxiong

2009

Guo, J., Matusiewicz, K., Knudsen, L.R., Ling, S., & Wang, H. (2009). Practical pseudo-collisions for hash functions ARIRANG-224/384. Lecture notes in computer science, 5867, 141-156.

https://hdl.handle.net/10356/94159

https://doi.org/10.1007/978-3-642-05445-7_9

# Practical pseudo-collisions for hash functions ARIRANG-224/384

Jian Guo[1⋆], Krystian Matusiewicz[2], Lars R. Knudsen[2], San Ling[1], and Huaxiong Wang[1]

[1] Division of Mathematical Sciences,
School of Physical and Mathematical Sciences,
Nanyang Technological University, Singapore
{guojian,lingsan,hxwang}@ntu.edu.sg
[2] Department of Mathematics,
Technical University of Denmark, Denmark
{K.Matusiewicz,Lars.R.Knudsen}@mat.dtu.dk

**Abstract.** In this paper we analyse the security of the SHA-3 candidate ARIRANG. We show that bitwise complementation of whole registers turns out to be very useful for constructing high-probability differential characteristics in the function. We use this approach to find near-collisions with Hamming weight 32 for the full compression function as well as collisions for the compression function of ARIRANG reduced to 26 rounds, both with complexity close to $2^0$ and memory requirements of only a few words. We use near collisions for the compression function to construct pseudo-collisions for the complete hash functions ARIRANG-224 and ARIRANG-384 with complexity $2^{23}$ and close to $2^0$, respectively. We implemented the attacks and provide examples of appropriate pairs of $H, M$ values. We also provide possible configurations which may give collisions for step-reduced and full ARIRANG.

**Keywords.** practical, pseudo-collision, ARIRANG, hash function

## 1 Introduction

ARIRANG [1] is one of the first-round candidates in the SHA-3 competition organized by NIST. It is an iterated hash function that uses a variant of the Merkle-Damgård mode augmented by a block counter. The compression function is a dedicated design that iterates a step transformation that can be seen as a target-heavy unbalanced Feistel network [9]. Its construction seems to be influenced by an earlier design called FORK-256 [4] with the important difference of using a bijective function based on a layer of S-boxes and an MDS mapping as the source of non-linearity. This prevents attacks similar to the ones developed for FORK-256 [7, 6, 2] from working on ARIRANG. A single sequence of 40 steps rather than four parallel branches makes it immune to meet-in-the-middle attacks [8].

⋆ The paper was partly done during the author's visit to Technical University of Denmark and was partly supported by a DCAMM grant there.

**Related Work** To the best of our knowledge, the only published previous work on ARIRANG is a step-reduced preimage attack by Hong *et al* [3]. Based on the meet-in-the-middle preimage attack framework developed by Sasaki *et al*, Hong *et al* were able to find [3-33] step-reduced pseudo-preimages with complexity $2^{241}$ and $2^{481}$ for ARIRANG-256 and ARIRANG-512, respectively.

**Our contributions** In this paper we report results of our security assessment of ARIRANG. The initial observation that motivated our analysis was the fact that differences created by complementing (flipping) all bits in a register propagate quite nicely through the function due to a particular interaction of the layer of S-boxes and an MDS mapping. We were able to exploit this fact to derive a range of attacks on the compression function and extend some of them to attacks on the complete hash function.

After a short description of ARIRANG given in section 2 we explain in details our ideas of managing all-ones differences in section 3 and show how to find conforming messages in section 4. After that, we describe two attacks on ARIRANG. In section 5 we show how to find collisions for 26 out of 40 steps of the compression function with complexity close to the cost of computing a single hash value of ARIRANG. Next, we show in Section 6 that by injecting all-ones difference in one of the chaining values we can easily (with complexity close to one evaluation) obtain 32-bit (resp. 64-bit) near collisions for the full compression function of ARIRANG-256 (resp. ARIRANG-512). We use the freedom of selecting in which chaining register we want to have differences to convert those near-collisions for the compression function to pseudo-collisions for the full hash functions ARIRANG-224 and ARIRANG-384 which we can obtain with complexity $2^{23}$ and close to $2^0$ respectively. Finally, we discuss some open problems and conclude in Section 8. Our results are summarized in Table 1.

**Table 1.** Summary of the results of this paper.

| **Compression function** | | |
|---|---|---|
| Result | Complexity | Example |
| 32-bit near-collision for full ARIRANG-256 compress | 1 | Y |
| 64-bit near-collision for full ARIRANG-512 compress | 1 | Y |
| 26-step collision for ARIRANG-256/512 | 1 | Y |
| **Hash function** | | |
| Result | Complexity | Example |
| pseudo-collision for full ARIRANG-224/384 hash | $2^{23}$ / 1 | Y |

## 2 Brief description of ARIRANG

We start with providing a minimal description of ARIRANG necessary to understand our attacks. More details can be found in the original submission document.

**Compression function** The fundamental building block of the hash function ARIRANG-256 (ARIRANG-512) is the compression function that takes 256-bit (512-bit) chaining value and 512-bit (1024-bit) message block and outputs a new 256-bit (512-bit) chaining value. The function, depicted in Fig. 1, consists of two main parts: the message expansion process and the iteration of the step transformation.

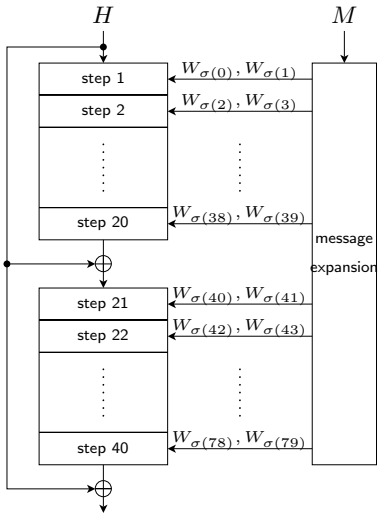


**Fig. 1.** Compression function of ARIRANG.

The message expansion function takes as input 16 words of the message $M_0, \ldots, M_{15}$ and produces 80 expanded message words in two stages. First, 32 words $W_i$ are generated according to the procedure described in Alg. 1, where $K_i$ are word constants and $r_i$ are fixed rotation amounts. Our attacks do not depend on their actual values. Next, these 32 words are used 80 times, two in each step transformation, in the order defined by the function $\sigma$ described in Table 2.

The iterative part uses the step transformation to update the state of 8 chaining registers, $a$, $b$, $\ldots$, $h$. First, the input chaining values $H[0], \ldots, H[7]$ are loaded into chaining registers $a$, $\ldots$, $h$. Then, the step transformation is applied 20 times. After 20 steps, the initial chaining

**Algorithm 1** Generation of expanded message words in ARIRANG.

**for** $i = 0, \ldots, 15$ **do**
  $W_i \leftarrow M_i$
**end for**
$W_{16} \leftarrow (W_9 \oplus W_{11} \oplus W_{13} \oplus W_{15} \oplus K_0) \lll r_0$
$W_{17} \leftarrow (W_8 \oplus W_{10} \oplus W_{12} \oplus W_{14} \oplus K_1) \lll r_1$
$W_{18} \leftarrow (W_1 \oplus W_3 \oplus W_5 \oplus W_7 \oplus K_2) \lll r_2$
$W_{19} \leftarrow (W_0 \oplus W_2 \oplus W_4 \oplus W_6 \oplus K_3) \lll r_3$

$W_{20} \leftarrow (W_{14} \oplus W_4 \oplus W_{10} \oplus W_0 \oplus K_4) \lll r_0$
$W_{21} \leftarrow (W_{11} \oplus W_1 \oplus W_7 \oplus W_{13} \oplus K_5) \lll r_1$
$W_{22} \leftarrow (W_6 \oplus W_{12} \oplus W_2 \oplus W_8 \oplus K_6) \lll r_2$
$W_{23} \leftarrow (W_3 \oplus W_9 \oplus W_{15} \oplus W_5 \oplus K_7) \lll r_3$

$W_{24} \leftarrow (W_{13} \oplus W_{15} \oplus W_1 \oplus W_3 \oplus K_8) \lll r_0$
$W_{25} \leftarrow (W_4 \oplus W_6 \oplus W_8 \oplus W_{10} \oplus K_9) \lll r_1$
$W_{26} \leftarrow (W_5 \oplus W_7 \oplus W_9 \oplus W_{11} \oplus K_{10}) \lll r_2$
$W_{27} \leftarrow (W_{12} \oplus W_{14} \oplus W_0 \oplus W_2 \oplus K_{11}) \lll r_3$

$W_{28} \leftarrow (W_{10} \oplus W_0 \oplus W_6 \oplus W_{12} \oplus K_{12}) \lll r_0$
$W_{29} \leftarrow (W_{15} \oplus W_5 \oplus W_{11} \oplus W_1 \oplus K_{13}) \lll r_1$
$W_{30} \leftarrow (W_2 \oplus W_8 \oplus W_{14} \oplus W_4 \oplus K_{14}) \lll r_2$
$W_{31} \leftarrow (W_7 \oplus W_{13} \oplus W_3 \oplus W_9 \oplus K_{15}) \lll r_3$

**Table 2.** Ordering $\sigma$ of expanded message words $W_i$ used in step transformations.

| $i$ | $\sigma(i)$ | $i$ | $\sigma(i)$ | $i$ | $\sigma(i)$ | $i$ | $\sigma(i)$ |
|---|---|---|---|---|---|---|---|
| 0, 1 | 16, 17 | 20, 21 | 20,21 | 40, 41 | 24, 25 | 60, 61 | 28, 29 |
| 2, 3 | 0, 1 | 22, 23 | 3, 6 | 42, 43 | 12, 5 | 62, 63 | 7, 2 |
| 4, 5 | 2, 3 | 24, 25 | 9,12 | 44, 45 | 14, 7 | 64, 65 | 13, 8 |
| 6, 7 | 4, 5 | 26, 27 | 15, 2 | 46, 47 | 0, 9 | 66, 67 | 3, 14 |
| 8, 9 | 6, 7 | 28, 29 | 5, 8 | 48, 49 | 2, 11 | 68, 69 | 9, 4 |
| 10, 11 | 18, 19 | 30, 31 | 22,23 | 50, 51 | 26, 27 | 70, 71 | 30, 31 |
| 12, 13 | 8, 9 | 32, 33 | 11,14 | 52, 53 | 4, 13 | 72, 73 | 15, 10 |
| 14, 15 | 10, 11 | 34, 35 | 1, 4 | 54, 55 | 6, 15 | 74, 75 | 5, 0 |
| 16, 17 | 12, 13 | 36, 37 | 7,10 | 56, 57 | 8, 1 | 76, 77 | 11, 6 |
| 18, 19 | 14, 15 | 38, 39 | 13, 0 | 58, 59 | 10, 3 | 78, 79 | 1, 12 |

values are XOR-ed to the current chaining values and the computation is carried on for another 20 steps. At the end, the usual feed-forward is applied by XOR-ing initial chaining values to the output of the iteration.

The step transformation updates chaining registers using two expanded message words $W_{\sigma(2t)}$, $W_{\sigma(2t+1)}$ as follows

$$\begin{aligned} T_1 &\leftarrow \mathbf{G}^{(256)}(a_t \oplus W_{\sigma(2t)}), & T_2 &\leftarrow \mathbf{G}^{(256)}(e_t \oplus W_{\sigma(2t+1)}),\\ b_{t+1} &\leftarrow a_t \oplus W_{\sigma(2t)}, & f_{t+1} &\leftarrow e_t \oplus W_{\sigma(2t+1)},\\ c_{t+1} &\leftarrow b_t \oplus T_1, & g_{t+1} &\leftarrow f_t \oplus T_2,\\ d_{t+1} &\leftarrow c_t \oplus (T_1 \lll 13), & h_{t+1} &\leftarrow g_t \oplus (T_2 \lll 29),\\ e_{t+1} &\leftarrow d_t \oplus (T_1 \lll 23), & a_{t+1} &\leftarrow h_t \oplus (T_2 \lll 7). \end{aligned}$$

This transformation is illustrated in Fig. 2. In ARIRANG-256, it uses a function $\mathbf{G}^{(256)}$ which splits 32-bit input value into 4 bytes, transforms them using AES S-Box and feeds the result to the AES MDS transformation, as presented in Fig. 3. ARIRANG uses the same finite field as AES, defined by the polynomial $x^8 + x^4 + x^3 + x + 1$. MDS mapping for 256 bit variant is defined as

$$MDS_{4\times 4} = \begin{bmatrix} z & z+1 & 1 & 1 \\ 1 & z & z+1 & 1 \\ 1 & 1 & z & z+1 \\ z+1 & 1 & 1 & z \end{bmatrix}.$$

In ARIRANG-512, an analogous function $\mathbf{G}^{(512)}$ is defined using a layer of 8 S-boxes and an appropriate $8 \times 8$ MDS matrix.

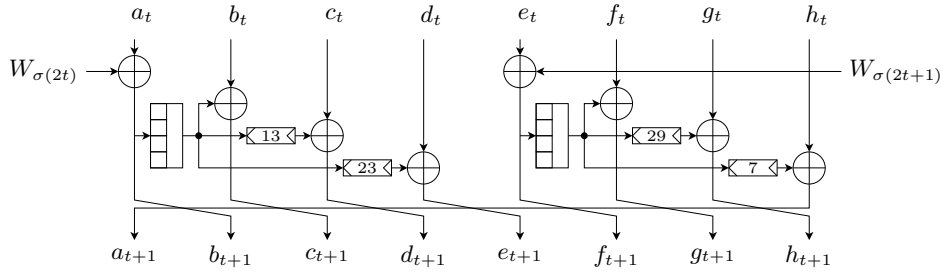


**Fig. 2.** Step transformation of ARIRANG updates the state of eight chaining registers.

**Hash function** The hash function ARIRANG is an iterative construction closely following the original Merkle-Damgård mode. The message is first

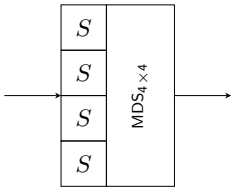


**Fig. 3.** Function $\mathbf{G}^{(256)}$ of ARIRANG-256 uses four AES S-Boxes followed by AES MDS mapping.

padded by a single '1' bit followed by an appropriate number of zero bits and a 64-bit field containing the length of the original message. After padding and appending block length field, the message is divided into 512-bit blocks and the compression function is applied to process each of the blocks one by one. The construction has one additional variable compared to the plain Merkle-Damgård mode. A new variable that stores the current message block index is introduced and its value is XOR-ed into chainings before each application of the compression function. However, this does not affect our attacks.

## 3 All-one differences

From the description of ARIRANG-256, it is clear that it uses only three essential building blocks: XORs, bit rotations and the function $\mathbf{G}^{(256)}$, which is the only part non-linear over $\mathbb{F}_2$.

Let us focus on the function $\mathbf{G}^{(256)}$ first. First, note that for the AES S-Box input difference of `0xff` maps to output difference `0xff` with probability $2^{-7}$, the two values $x$ for which $S(x) \oplus S(x \oplus \texttt{0xff}) = \texttt{0xff}$ are `0x7e`, `0x81`.

The second observation is that for the 256-bit MDS mapping all the vectors of the form $(a, a, a, a)$ are fixed points since $a{\cdot}z{+}a(z{+}1){+}a{+}a = a$.

This means all-one difference will map to all-one difference through $MDS_{4\times4}$. In turns, there are 16 32-bit values $x$ such that

$$\mathbf{G}^{(256)}(x) \oplus \mathbf{G}^{(256)}(x \oplus \texttt{0xffffffff}) = \texttt{0xffffffff}$$

and the probability of such a differential is $2^{-28}$.

This means we can consider a differential that uses only all-one differences in active registers. The big advantage of such differences is that they are rotation invariant, so we can easily model differentials like that by replacing all the rotations and function $\mathbf{G}^{(256)}$ with identity.

MDS mapping for ARIRANG-512 is different and all-ones is not its fixed-point, but after combining S-box layer with MDS, we get the dif-

ferential of the same type with probability $2^{-56}$, so the same principle applies to the larger variant as well.

To minimize the complexity of the attack, we need to use as few active $\mathbf{G}^{(256)}$-functions as possible in the part of the function where we cannot control input values to them. Since there are only $2^{16}$ possible combinations of all-one differences in message words and $2^{24}$ combinations including chaining registers $H[0], \ldots, H[7]$, it is easy to enumerate them all using a computer search.

We note that all-one differences trick is also used in [5].

## 4 Message Adjustments

The method used to find messages that make the differences in the actual function to follow the differential can be called a message adjustment strategy.

We have full control over the message words $W_0, \ldots, W_{15}$. Through combinations of the message words, we can still control some of the messages $W_i$ for $16 \leq i \leq 31$. We can modify the messages used in the first 4 steps freely, yet leaving the output chaining values of 4-th step unchanged by modifying the corresponding input chaining values $H[0], \ldots, H[7]$.

For example, changing $W_2$ and $H[6]$ by the same amount ($\oplus$ both with a same value) will keep the output of step 3 stable. Beyond step 4, if we change the value of $W_6$ in step 5, we still make the output of step 5 stable by changing the $H[4]$ by a same amount. However this change will be propagated by the right $G$ function in step 1, we can fix this by changing the $H[5], H[6]$ and $H[7]$ by proper values, respectively. This method applies to $W_7$ in step 5 similarly. In step 6, if $W_{19}$ is changed, we can still keep the output after step 6 stable. We achieve this by $\oplus$ with $H[7]$ by the same amount of the change. Note that this difference will be propagated through the left $G$ function in step 2 (Note we can only do this when the left $G$ in step 2 is not active). We can fix this by $\oplus$ with $H[0], H[1], H[2]$ by proper values, respectively. Then the change in $H[0]$ will be propagated through the G function in step 1. We then fix this by $\oplus$ with $H[0], H[1], H[2]$ by proper values. Similar method applies to $W_{18}$ in step 6.

## 5 Collisions for reduced round compression function

A search for collision configuration that minimizes the overall number of active $\mathbf{G}^{(256)}$ functions shows that the best strategy is to flip all message words. Then throughout the whole compression function only 16 out of 80 $\mathbf{G}^{(256)}$ are active. When we restrict the attention to steps 20-40 (the part

which almost certainly is beyond any message-modification techniques) we can find a configuration with only 5 active $\mathbf{G}^{(256)}$ and in fact only 3 in steps 22-40. Details of minimal paths are summarized in Table 3. The second characteristic with probability $2^{-140}$ in steps 21-40 shows that the claim made in [1, section 6.2, page 37] that "*there is no collision producing characteristics which has a probability higher than* $2^{-256}$ *in the last two rounds*" is based on assumptions that do not hold in practice.

**Table 3.** Results of search for collision characteristics in ARIRANG-256

| type | minimize | min. value | diffs in message words |
|---|---|---|---|
| collisions | total active $G$ | 16 | 0,...,15 (all) |
| collisions | active $G$ rounds 20-40 | 5 | 2,3,7,8,9,13 |

Even though using all-one differences does not seem to allow for finding good collision differentials for the full compression function, one can use them to mount an attack on its reduced-round variants. In the rest of this section we illustrate it with a method that instantly finds collisions for 26 steps of ARIRANG-256.

### 5.1 Finding Step Reduced Collision Differential

To find the optimal path for reduced-round attack, we searched the all-one differentials using the following criteria.

1. We count the number of active $G$ from step 11, as we have a complete control over the first 10 steps,
2. there are only differences in message words, not in chaining values,
3. the differential should give round reduced collision,
4. the differential should have minimum number of active $G$,
5. preferably, the active $G$-s should appear as early as possible.

The search result[3] shows a differential with differences in message words $M_4$, $M_6$, $M_8$, $M_{10}$ and the corresponding active $G$ is shown in Table 4, steps after 16 are not shown because there is no active $G$ between step 16 and step 26 and we do not consider steps after step 26.

### 5.2 Finding Step Reduced Collisions

To find the example of the 26-step reduced collision, we need to deal with all those active $G$ so that the input to the active $G$ are one of those

[3] Active G may not be paired with active messages, as the differences in message may be canceled by differences from preceding steps

**Table 4.** 26-step reduced collision characteristics in ARIRANG

| Step | W (left) | Active G (left) | W (right) | Active G (right) |
|---|---|---|---|---|
| 1 | $W_{16}$ | | $W_{17}$ | |
| 2 | $W_0$ | | $W_1$ | |
| 3 | $W_2$ | | $W_3$ | |
| 4 | $W_4$ | ✓ | $W_5$ | |
| 5 | $W_6$ | ✓ | $W_7$ | ✓ |
| 6 | $W_{18}$ | ✓ | $W_{19}$ | |
| 7 | $W_8$ | | $W_9$ | ✓ |
| 8 | $W_{10}$ | ✓ | $W_{11}$ | ✓ |
| 9 | $W_{12}$ | ✓ | $W_{13}$ | ✓ |
| 10 | $W_{14}$ | ✓ | $W_{15}$ | ✓ |
| 11 | $W_{20}$ | | $W_{21}$ | ✓ |
| 12 | $W_3$ | | $W_6$ | |
| 13 | $W_9$ | ✓ | $W_{12}$ | |
| 14 | $W_{15}$ | | $W_2$ | |
| 15 | $W_5$ | ✓ | $W_8$ | |

all-one difference pairs. As our algorithm runs in a deterministic way, we actually force the input to a chosen pair $(\gamma, \bar{\gamma}) = (\texttt{81818181, 7E7E7E7E})$. In the first 10 steps, whenever there is an active $G$, we can fix the input by modifying the immediate message word. After step 10, we follow the algorithm below:

1. For active $G$ in step 11, we change $W_{21}$ to the proper value by modifying $W_1$ and $W_3$ by the same amount so that $W_{18}$ does not change, we compensate the change of $W_1$ and $W_3$ using the method in section 4.
2. For active $G$ in step 13, we modify the message word $W_6$, which is used one step before. We modify $W_2$ also by a same amount so that $W_{19}$ is constant, and then compensate the changes.
3. For active $G$ in step 15, we modify $W_5$ directly. We compensate the change of $W_5$ and $W_{18}$.

As we can see the algorithm is deterministic, so the complexity is 1 with no memory requirements. An example of the chaining values and a pair of messages obtained using this procedure is shown in Table 5.

## 6 Pseudo-collisions for ARIRANG-224 and ARIRANG-384

If we relax the condition of no difference at the output of the compression function we can find much better differentials. A near-collision attack

**Table 5.** 26-step reduced collision for ARIRANG-256 with differences in $M$ only.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| input H | C0E5A81E | 952A32CB | 730C4EB7 | 78730E23 | 757D7CAC | 00000000 | D69B0F52 | D69B0F52 |
| M | D69B0F52 | 78730E23 | D69B0F52 | 730C4EB7 | E3E3E3E3 | 952A32CB | 1A1A1A1A | 49494949 |
| | 00000000 | 02020202 | D3DCBDB8 | D9BDE3CB | 562D250E | 9B9F0611 | 662E4BD8 | E75B0B2F |
| M' | D69B0F52 | 78730E23 | D69B0F52 | 730C4EB7 | 1C1C1C1C | 952A32CB | E5E5E5E5 | 49494949 |
| | FFFFFFFF | 02020202 | 2C234247 | D9BDE3CB | 562D250E | 9B9F0611 | 662E4BD8 | E75B0B2F |
| step 26 | B4931778 | F1615E8C | 0E3756B9 | 93ED3536 | 4EBCBBFE | 86C9ADD8 | 34334617 | 340155F6 |

for the complete compression function makes use of the three particular features of the compression function of ARIRANG. The first one is the existence of all-ones differentials. The second element that enables our attack is the fact that in the first steps we can manipulate chaining values and message words to adjust input values of $G$-functions, similarly to the message modification strategy. Finally, we exploit the double-feed-forward feature of the compression function (cf. Fig. 1) to restrict the differences to only first half of the steps.

Once we have such near-collisions for the compression function, we can use them to construct pseudo-collisions for the complete hash function ARIRANG-224 and ARIRANG-384. This is possible thanks to the details of message padding and the way the final digest is produced. Because the final hash value is just a truncated chaining value, we can introduce the chaining differences in the register which is going to be truncated when producing the digest. Also, the padding and appending the length information does not use a separate message block but rather a few last words of a block. This means we need to deal with only one message block with the last three words determined by the padding scheme and the message length.

We will talk about ARIRANG-224, however our attack is not specific to it, so it also works for ARIRANG-384.

### 6.1 Finding Near Collision Differential

Based on the same idea and model as used for searching the collision, we did the search for finding near collisions and we observed an interesting phenomenon. With input differences in a single chaining variable, we could get differentials that go through the first twenty steps and collapse back to the same register at step 20. Then after the middle feed-forward, there is no difference in chaining registers and nothing happens until the final feed-forward. Only then the initial difference is injected again and results in an output difference restricted to only one register, 32 bits in case of ARIRANG-256. Actually all configurations with differences in chaining

variables behaves similarly, we can treat them as combinations of single difference.

With difference in H[7], we find it is easy to find the appropriate chaining values and messages. And advantage of this differential is, H[7] of the final output is discarded for ARIRANG-224 and ARIRANG-384, hence instead of near collision, it gives collisions. The differential with corresponding active G is listed in Table 6 and the detailed picture of it can be found in Fig 4. There is no active G after step 18, and there is no difference in the output before the final feed-forward. Steps after 18 are not listed in Table 6.

**Table 6.** Active $G$ functions in $H[7]$ near collision characteristics for ARIRANG.

| Step | W (left) | Active G (left) | W (right) | Active G (right) |
|---|---|---|---|---|
| 1 | $W_{16}$ | | $W_{17}$ | |
| 2 | $W_0$ | ✓ | $W_1$ | |
| 3 | $W_2$ | | $W_3$ | ✓ |
| 4 | $W_4$ | ✓ | $W_5$ | ✓ |
| 5 | $W_6$ | | $W_7$ | |
| 6 | $W_{18}$ | | $W_{19}$ | |
| 7 | $W_8$ | | $W_9$ | ✓ |
| 8 | $W_{10}$ | | $W_{11}$ | ✓ |
| 9 | $W_{12}$ | | $W_{13}$ | |
| 10 | $W_{14}$ | | $W_{15}$ | |
| 11 | $W_{20}$ | | $W_{21}$ | |
| 12 | $W_3$ | ✓ | $W_6$ | |
| 13 | $W_9$ | | $W_{12}$ | ✓ |
| 14 | $W_{15}$ | ✓ | $W_2$ | ✓ |
| 15 | $W_5$ | | $W_8$ | |
| 16 | $W_{22}$ | | $W_{23}$ | |
| 17 | $W_{11}$ | | $W_{14}$ | ✓ |
| 18 | $W_1$ | | $W_4$ | ✓ |

### 6.2 Finding Chaining Values and Messages

The algorithm used to solve the near collision starts with setting all messages and chaining values to be a random value, here we make use of 0. To get pseudo-collisions for the complete hash function, we need to consider the message padding and the encoding of the block length. In ARIRANG, the message padding is performed by appending '1' followed by as many zeros as necessary and the message length is encoded in the last two words. To accommodate for this, we use 13 word long message

which we can manipulate freely and fix $M_{13} = 10\cdots0_2$ and $M_{14}, M_{15}$ to contain encoded length (which is $13 \cdot 32$ for ARIRANG-224 and $13 \cdot 64$ for ARIRANG-384). Thanks to that, the input to the compression function is consistent with the definition of the hash function and we still have a complete control over 13 message words $M_0, \ldots, M_{12}$. Now we can focus on finding a message pair that follows the differential in the compression function and we proceed as follows.

1. Steps 1-9, whenever there is an active $G$, we force the input to the $G$ to $\gamma$ ($(\gamma, \bar{\gamma})$ is one of good input pairs to $\mathbf{G}^{(256)}$) by modifying the immediate $W$ values.
2. Step 12, we modify $W_3$. Note that $W_3$ is also used in step 3 and 6 ($W_{18}$), we can compensate this change using the method described before.
3. Step 13, we modify $W_{20}$ through $W_0$, we also modify $W_2$ so that $W_{19}$ keeps stable. We compensate the change of $W_0$ and $W_2$ again using the described method.
4. Step 14, left active $G$ can be dealt with using $W_6$ and $W_2$.
5. Step 15, right active $G$ can be choosing a random $W_9$, we compensate the change of $W_9$ used in step 7 by modifying $H[6]$. However the input to the left $G$ in step 3 changes, we compensate this using $W_{19}$ in step 6, $H[0]$ and $H[1]$ in step 1. Again input to left G in step 1 changes as $H[0]$ changes, we compensate as done for change of $W_7$. Note $W_{19}$ can only be changed indirectly, here we use $W_2$ and then compensate using $H[6]$. We repeat this step until we find the right active G in step 14 is good. Note we can do the compensation work only after a good value is found.
6. step 17, we modify $W_5$ which is used in step 15. Then we compensate the change of $W_5$ and $W_{18}$
7. Step 18, the active $G$ is dealt with by using $W_4$ and $W_0$.

The only active $G$ left is the one in step 15. We leave this to a chance by looping over different $W_9$. This requires $2^{28}$ tries, which is equivalent to around $2^{23}$ ($2^{51}$ for ARIRANG-384) calls to the compression function as we only need to compute two $G$ functions in the loop and there are 80 such computations in the compression function. Examples shown in Table 7 can be found in few seconds on a standard computer, and the the algorithm has no memory requirements apart from a few words used for intermediate variables.

### 6.3 Collisions for ARIRANG-384

We can find collisions for ARIRANG-384 the same way as done for ARIRANG-224. However, the corresponding complexity of $2^{51}$ is too high for

**Table 7.** Collision Example for ARIRANG-224.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| input H | 969F43DE | 781BBD62 | E6E7CEC7 | 075AF1AC | EE30CDD2 | 670D94E4 | 7AD337C6 | 60026A7A |
| input H' | 969F43DE | 781BBD62 | E6E7CEC7 | 075AF1AC | EE30CDD2 | 670D94E4 | 7AD337C6 | 9FFD9585 |
| M | 43F40822 | 00000000 | 22EE1F96 | 30B48FFB | AD6E028F | 958F43D5 | 5819FFF7 | 00000000 |
| | 00000000 | 34B65233 | 00000000 | C16DE896 | 00000000 | 80000000 | 00000000 | 000001A0 |
| output H | CBF6A53B | 0D7EB2CB | ACFD326A | 2BA6E962 | 4C2087AA | 2ABD938A | 221AED0E | |
| output H' | CBF6A53B | 0D7EB2CB | ACFD326A | 2BA6E962 | 4C2087AA | 2ABD938A | 221AED0E | |
| H $\oplus$ H' | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | |

a standard computer to handle. To get over this difficulty, we can use the fact that the final transform for ARIRANG-384 is done by discarding the last two chaining values, i.e. $H[6]$ and $H[7]$. So besides $H[7]$-differential, we can also consider $H[6]$-differential and $H[6-7]$-differential (Indeed this also gives near collisions with outputs differ in $H[6]$ and $H[7]$). Thanks to a different positions of active G-functions, it turns out that the $H[6]$-differential can be solved with complexity 1. Table 8 lists the active G for this differential. Note that this differential works for all instances of ARIRANG. So this also gives another solution for finding 224/256 near collision for ARIRANG-256 with complexity 1.

**Table 8.** Active $G$ functions in $H[6]$ near collision characteristics for ARIRANG.

| Step | W (left) | Active G (left) | W (right) | Active G (right) |
|---|---|---|---|---|
| 1 | $W_{16}$ | | $W_{17}$ | |
| 2 | $W_0$ | | $W_1$ | |
| 3 | $W_2$ | ✓ | $W_3$ | |
| 4 | $W_4$ | | $W_5$ | ✓ |
| 5 | $W_6$ | ✓ | $W_7$ | ✓ |
| 6 | $W_{18}$ | | $W_{19}$ | |
| 7 | $W_8$ | | $W_9$ | |
| 8 | $W_{10}$ | | $W_{11}$ | ✓ |
| 9 | $W_{12}$ | | $W_{13}$ | ✓ |
| 10 | $W_{14}$ | | $W_{15}$ | |
| 11 | $W_{20}$ | | $W_{21}$ | |
| 12 | $W_3$ | | $W_6$ | |
| 13 | $W_9$ | ✓ | $W_{12}$ | |
| 14 | $W_{15}$ | | $W_2$ | ✓ |
| 15 | $W_5$ | ✓ | $W_8$ | ✓ |
| 16 | $W_{22}$ | | $W_{23}$ | |
| 17 | $W_{11}$ | | $W_{14}$ | |
| 18 | $W_1$ | | $W_4$ | ✓ |
| 19 | $W_7$ | | $W_{10}$ | ✓ |

Referring to table 8, we can solve this differential (finding chaining values and messages) using the following procedure:

1. Step 1-9 can be handled as usual.
2. Step 13, we modify $W_6$ in step 12. We compensate the change of $W_6$ and $W_{19}$
3. Step 14, we modify $W_2$ directly and then compensate the change of $W_2$ and $W_{19}$
4. Step 15, for the left active G, we modify $W_5$ and compensate; for the right active G, we modify $W_8$. Note that the change of $W_8$ can be compensated similarly as done for $W_{19}$.
5. Step 18, we modify $W_4$ and $W_0$ simultaneously.
6. Step 19, we modify $W_1$ as used in step 18 and $W_7$ simultaneously.

As shown above, every step in the algorithm is deterministic, hence it gives complexity close to 1. Experiments also support the result, collisions can be found in terms of $\mu$s. An example of collision for ARIRANG-384 is shown in Table 9, note it is also 448/512 near collision for ARIRANG-512.

**Table 9.** Pseudo-collision example for ARIRANG-384.

| | | | | |
|---|---|---|---|---|
| input H | BA36BCB93BFD8D20 | 6B951DB399EB2EDC | 1950E807876279AE | AF16B3C9901076DC |
| | 62372888DECEB1E5 | 939957A5F4B4EE05 | AA31DB9CB0EF684C | 49B72A01D8C86B6F |
| input H' | BA36BCB93BFD8D20 | 6B951DB399EB2EDC | 1950E807876279AE | AF16B3C9901076DC |
| | 62372888DECEB1E5 | 939957A5F4B4EE05 | 55CE24634F1097B3 | 49B72A01D8C86B6F |
| M | B5127D606F0860D8 | 3E2BD987F6626D29 | 4EF941810127832F | 0000000000000000 |
| | B5127D606F0860D8 | A8FF942B50A3F3F8 | A99E61F4B41D9347 | F6E3114F3EAAA5E1 |
| | AFE28E981D9AE700 | 0000000000000000 | C80D9570708720C3 | AD8760D00E4D14C8 |
| | 0000000000000000 | 8000000000000000 | 0000000000000000 | 0000000000000340 |
| output H | 5939B28C23F6435F | BFA7FC0F59F0BFF7 | FBF8D1923EED2060 | AE79BE18FC078E32 |
| | F4CE359791C979E7 | 543F7F214A45D0A9 | 193A61B727F9BC5A | 3E8CFA173B9D48B2 |
| output H' | 5939B28C23F6435F | BFA7FC0F59F0BFF7 | FBF8D1923EED2060 | AE79BE18FC078E32 |
| | F4CE359791C979E7 | 543F7F214A45D0A9 | E6C59E48D80643A5 | 3E8CFA173B9D48B2 |

### 6.4 Pseudo-Preimages

It is possible to further extend the pseudo-near-collision attack to pseudo-preimages of ARIRANG. Take the configuration $H = (0, 1, 0, 0, 0, 0, 0, 0)$ for example, we are able to solve it in time 1 and it gives a near collision with all-one difference in $H[1]$ of final output. Note that once one such near collision pair is found, we are able to find $2^{32}$ pairs by trying different values for $W_1$ ($W_7, W_0$, and $W_4$ are changed accordingly) and compensate at the beginning. To find exact values, we need to compute steps 18 – 40

only, so the complexity to find one pair is reduced to about $2^{-1}$. Given a target $t$, any match with $t$ or $t \oplus 0^{32}1^{32}0^{192}$ will give us a pseudo-preimage. So we are able to find a match by finding $2^{255}$ different values, and finding each value costs $2^{-1}$. The overall complexity for finding a pseudo-preimage is $2^{254}$ for ARIRANG-256. Similarly, we can find pseudo-preimage for ARIRANG-512 within $2^{510}$. However this does not give a preimage attack, as converting pseudo-preimage to preimage requires the complexity to be less than $2^{n-2}$ in general.

## 7 Possible Extensions

With the similar method above, we can see that it is reasonable to count the active G from step 21, as most of the time, we can handle the first 20 steps using the message adjustment with low complexity. We did the search and found two interesting configurations ($M = (0,1,1,0,0,1,0,1,0,0,1,0,0,1,0,1)$ and $M = (0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0)$, where $i$-th bit of the configuration indicates whether there is a difference in $M[i]$) which gives 29-step reduced and 34-step reduced collisions with 1 and 2 active Gs, respectively. These two configurations may give step-reduced collisions with complexity less than birthday bound. With configuration $M = (1,1,0,0,1,1,0,0,1,1,0,0,0,0,1,0), H = (1,0,0,0,0,1,1,1)$ we may find [2-37] step reduced pseudo-collision as there are only 4 active G after step 20 and the active G in step 21 seems easy to deal with. With configuration $M = (0,0,1,1,0,0,0,1,1,1,0,0,0,1,0,0)$, we may find semi-free-start collision for full ARIRANG as there are 5 active Gs after step 20 and seems those 3 active Gs in step 21 and 23 can be dealt with by modifying the chaining values.

Some investigation shows that similar idea of message adjustment can be used to find collisions based on semi-free-start collision. Note that when messages are modified, chaining values are modified in accordingly. We can do the reverse: modify the chaining values to those we required, and change the messages accordingly. However we need to be careful to ensure that active Gs are not affected.

## 8 Conclusions

We presented a range of attacks on ARIRANG. They all use the same type of differential based on flipping all bits in a register and the fact that all-one differences propagate with non-zero probability through the non-linear function $\mathbf{G}^{(256)}$ and are not affected by all the other building blocks of the function.

This approach allowed us to find collisions for step-reduced compression function and pseudo-collisions for the hash function. Even though

this method seems to be effective when looking for collisions for up to around 30 steps, we do not see a way to extend it to a collision attack on the full hash function at the moment.

A possible alternative approach would be to consider other types of differences. Note that we can get high-probability local collision patterns by having only one S-box active inside of $\mathbf{G}^{(256)}$ and canceling the (dense) output differences in later steps by appropriate differences in message words. With this approach we can have up to 18 S-boxes active in the part of the function beyond our message-modification control to beat the birthday bound. The main difficulty seems to find a superposition of such local patterns that agrees with the message expansion process.

One could also think about ways to "patch" the design to defend against our attacks. It seems that the double feed-forward is not a good idea as it enabled us to skip half of the steps of the function in our pseudo-collision attack. Moreover, it should not be possible to use all-one differences that easily. To this end, one could either break the symmetry of rotations somewhere (perhaps in the message expansion process as seen in SHA-256 that uses also shifts in addition to rotations) or modify the MDS mapping to make sure that none of the possible output differences of the layer of S-boxes obtained for all-one input difference maps to all-ones difference through the MDS. However, all those fixes are quite ad-hoc and address only one particular attack strategy exploited in this paper.

## Acknowledgements

The work in this paper was supported in part by the National Research Foundation of Singapore under Research Grant NRF-CRP2-2007-03 and the Singapore Ministry of Education under Research Grant T206B2204.

Krystian Matusiewicz was supported by grant 274-07-0246 from the Danish Research Council for Technology and Production Sciences.

The authors would like to thank Christian Rechberger, Praveen Gauravaram and the anonymous reviewers for the helpful comments and Wei Lei for his shell script.

## References


1. Chang, D., Hong, S., Kang, C., Kang, J., Kim, J., Lee, C., Lee, J., Lee, J., Lee, S., Lee, Y., Lim, J., Sung, J.: ARIRANG: SHA-3 Proposal. NIST SHA-3 candidate, http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/documents/ARIRANG.zip
2. Contini, S., Matusiewicz, K., Pieprzyk, J.: Extending FORK-256 attack to the full hash function. In: Qing, S., Imai, H., Wang, G. (eds.) ICICS 2007. LNCS, vol. 4861, pp. 296–305. Springer, Heidelberg (2007)
3. Hong, D., Kim, W.-H., Koo, B.: Preimage attack on arirang. Cryptology ePrint Archive, Report 2009/147 (2009), http://eprint.iacr.org/2009/147
4. Hong, D., Sung, J., Lee, S., Moon, D., Chee, S.: A new dedicated 256-bit hash function. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 195–209. Springer, Heidelberg (2006)
5. Indesteege, S., Mendel, F., Rechberger, C., Schläffer, M.: Practical Collisions for SHAMATA. In: Jacobson, M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 1–15. Springer, Heidelberg (2009)
6. 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)
7. Mendel, F., Lano, J., Preneel, B.: Cryptanalysis of reduced variants of the FORK-256 hash function. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 85–100. Springer, Heidelberg (2007)
8. Saarinen, M.-J.: A Meet-in-the-Middle collision attack against the new FORK-256. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 10–17. Springer, Heidelberg (2007)
9. Schneier, B., Kesley, J.: Unbalanced Feistel networks and block cipher design. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 121–144. Springer, Heidelberg (1996)

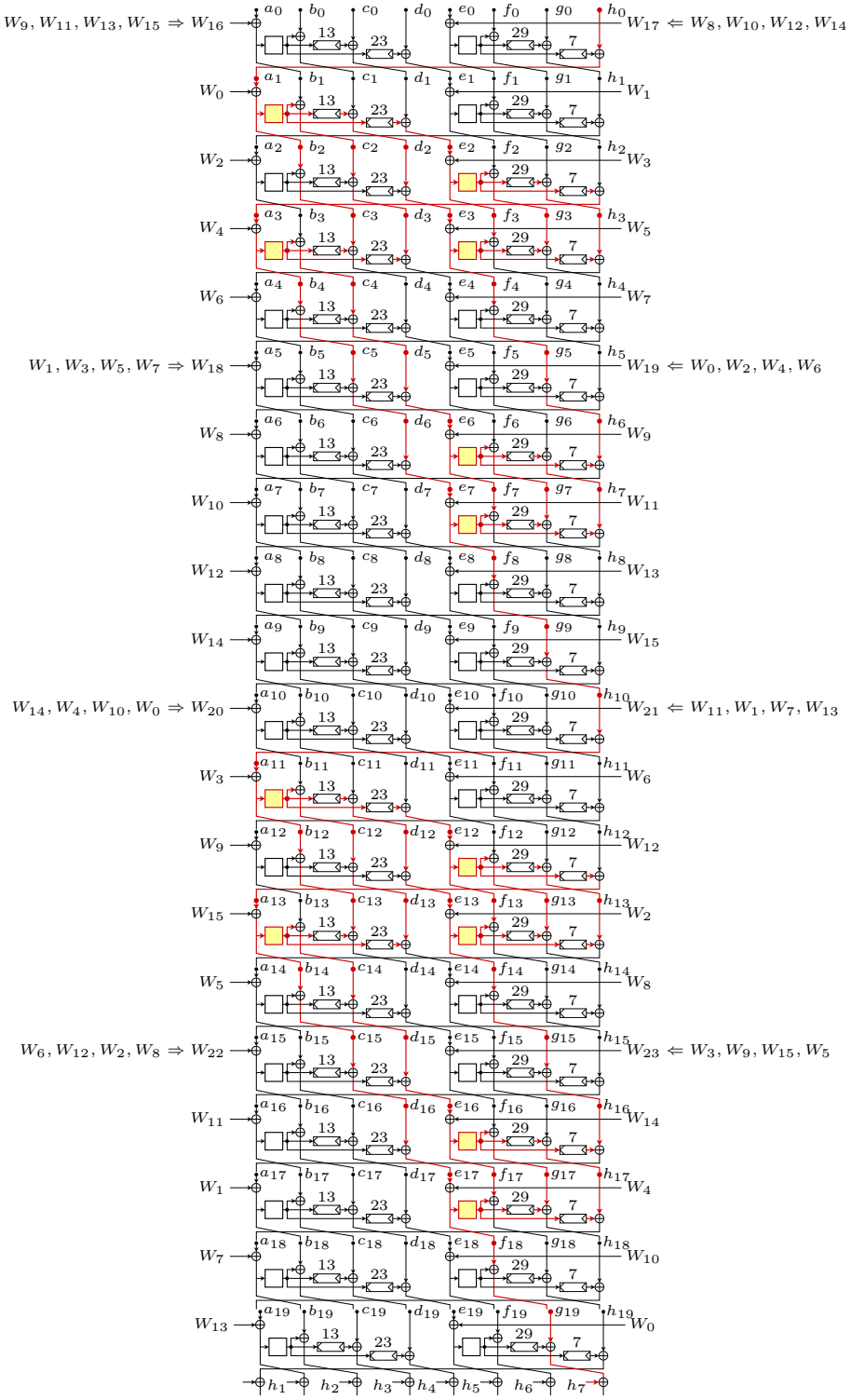


**Fig. 4.** Differential path in steps 1-20 used to find near-collisions in the compression function. There are no differences in steps 21-40.