Practical Near-Collisions for Reduced Round Blake, Fugue, Hamsi and JH

Meltem Sönmez Turan[1], Erdener Uyan[2]

[1] Computer Security Division, National Institute of Standards and Technology, USA
[2] Institute of Applied Mathematics, Middle East Technical University, Turkey
meltem.turan@nist.gov, uerdener@metu.edu.tr

Abstract. A hash function is near-collision resistant, if it is hard to find two messages with hash values that differ in only a small number of bits. In this study, we use hill climbing methods to evaluate the near-collision resistance of some of the round SHA-3 candidates. We practically obtained *(i)* 184/256-bit near-collision for the 2-round compression function of Blake-32; *(ii)* 192/256-bit near-collision for the 2-round compression function of Hamsi-256; *(iii)* 820/1024-bit near-collisions for 10-round compression function of JH. We also observed practical collisions and near-collisions for reduced versions of F-256 function used in Fugue.
Keywords: Hash functions, Near-collisions, SHA-3 Competition.

1 Introduction

Hill climbing methods are simple heuristic algorithms that aim to provide "good" solutions to "hard" optimization problems in short running times. These algorithms start with a random candidate and iteratively improve the candidate by making small changes, then terminate after converging to a local optimum. They are successful for problems for which the value of the problem at a specific point gives some information about "close" points. For the well known traveling salesman problem, these methods get within approximately 10-15% of optimal solution in relatively short time [1].

There are many hard search problems in the field of cryptography, such as finding the secret key in symmetric cryptosystems or building efficient components with good cryptographic properties. However, the success of the simple optimization techniques have been very limited in most of these problems.

One of the reasons of the failure is that most of the cryptographic problems (e.g. searching for the secret key) have no "good" solutions except for the optimal solution. Another reason is due to the discontinuity of the most cryptographic functions, i.e. small changes in the input usually result in random looking changes in the outputs. Clark in his PhD thesis [2] claims that these techniques might give significant and surprising results if used in the right way. Searching for cryptographically strong Boolean functions is one of the cryptographic problems that benefit from these methods [3–5]. After the announcement of the SHA-3 hash competition by National Institute of Standards and Technology (NIST) [6], the submitted hash functions have been a prolific source of new cryptographic problems.

Security of a cryptographic hash function is evaluated based on its resistance to preimage, second preimage and collision attacks. Moreover, a secure hash function is expected to be indifferentiable from a random oracle and resist other attacks such as partial preimage and near-collision.

Truncating some of the output hash bits might be necessary for compatibility of systems or desired for the efficiency purposes. In such cases, near-collision results have significant importance, since the output differences may diminish after a truncation operation and collisions may be obtained.

Hill climbing methods seems to be more promising for searching near-collisions compared to other type of attacks, since the problem has many local optimal points. In this study, we analyze the compression functions of some of the second round SHA-3 candidates, namely Blake [7], Fugue [8], Hamsi [9] and JH [10] using a simple hill climbing method. We observed that for some of the reduced versions, the method produced better results compared to the generic random search. We practically present near-collision examples for reduced compression functions of Blake-32, Fugue-256, Hamsi-256 and JH-256 that were obtained in short times.

Organization of the paper is as follows. In Section 2, generic methods to find near collisions are discussed. Then, in Section 3, the proposed hill climbing method is described. Section 4, the results we obtained for reduced versions of Blake, Fugue, Hamsi and JH are presented. Finally, the results are summarized in Section 5.

2 Near-Collisions

A hash function is near-collision resistant, if it is "hard" to find two messages with hash values that differ in only a small number of bits.

Let h be a compression function that takes a m-bit message block and an n-bit chaining value CV as inputs and generates the n-bit next chaining value as output. An ϵ/n-bit near-collision on h is obtained whenever two message blocks M_1 and M_2 satisfying

$$HW(h(M_1, CV) \oplus h(M_2, CV)) = n - \epsilon \tag{1}$$

are found, where HW represents the Hamming weight. Clearly, $\epsilon = n$ corresponds to a collision on the compression function. A generic method to find near-collisions for a compression function is to generate input message and output chaining value pairs (M_i, C_i) and compare C_i's to find the closest pair. Since the pairs are needed to be stored, the method is not memory-efficient.

Another approach to find near-collisions is to randomly try input chaining values CV that minimize

$$HW(h(M_1, CV) \oplus h(M_2, CV)) \tag{2}$$

for a given M_1 and M_2 pair. Using this method, finding an ϵ/n-bit near-collision requires approximately $2^n/\binom{n}{\epsilon}$ many evaluations of the compression function with almost no memory requirement, only the best chaining value found so far is stored. Table 1 shows the expected complexity to obtain ϵ/n-bit near-collisions for a compression function with 256, 512 or 1024-bit output.

Table 1. Approximate complexity to obtain ϵ/n-bit near-collisions.

ϵ/n	Complexity ($\approx$)
128/256, 256/512, 512/1024	2^4
151/256, 287/512, 553/1024	2^{10}
166/256, 308/512, 585/1024	2^{20}
176/256, 323/512, 606/1024	2^{30}
184/256, 335/512, 623/1024	2^{40}
191/256, 345/512, 638/1024	2^{50}
197/256, 354/512, 651/1024	2^{60}

3 Hill Climbing Method

If the compression function h has strong diffusion properties, for a randomly chosen message M and input chaining value CV, the Hamming weight of

$$h(M, CV) \oplus h(M, CV \oplus \delta) \tag{3}$$

is approximately $\frac{n}{2}$, where δ is an n-bit vector with small Hamming weight. However if the diffusion of δ is not satisfied, $h(M, CV)$ and $h(M, CV \oplus \delta)$ might be correlated, i.e., the value of $h(M, CV)$ might provide some exploitable information about the value of $h(M, CV \oplus \delta)$. In such cases, the hill climbing algorithms to find near-collisions may work better than the generic approaches.

The aim of our hill climbing method is to minimize the function

$$f_{M_1,M_2}(x) = HW(h(M_1, x) \oplus h(M_2, x)) \tag{4}$$

where $x \in \{0,1\}^n$, for given message blocks M_1 and M_2. Let CV be a randomly chosen chaining value. We define the set of k-bit neighbors of CV as

$$S^k_{CV} = \{x \in \{0,1\}^n | HW(CV \oplus x) \leq k\}. \tag{5}$$

Clearly, the size of S^k_{CV} is equal to $\sum_{i=0}^{k} \binom{n}{i}$.

For message blocks M_1 and M_2, a chaining value CV is defined to be *k-opt*, if

$$f_{M_1,M_2}(CV) = \min_{x \in S^k_{CV}} f_{M_1,M_2}(x). \tag{6}$$

The hill climbing method presented in this section works as follows. Given a pair of message blocks M_1 and M_2, we randomly select a candidate chaining value CV and calculate $f_{M_1,M_2}(CV)$. Then, we search the set S^k_{CV} to find a better chaining value. If found, our candidate is updated. Then, a new search is started in the k-bit neighbor of the new candidate. The algorithm terminates whenever a *k-opt* chaining value is obtained. The pseudo-code of the method is presented in Algorithm 3.1.

Algorithm 3.1: HILLCLIMBING(M_1, M_2, k)

```
Randomly select CV;
f_best = f_{M1,M2}(CV);
while (CV is not k-opt)
    CV = x such that x ∈ S^k_CV with f(x) < f_best;
    f_best = f_{M1,M2}(CV);
return (CV, f_best)
```

Given current CV, the next candidate can be selected in two ways. In the first way, the first chaining value that has lower f value is chosen and this approach is known as the *greedy gradient ascent*. In the second way, the best chaining value in S^k_{CV} is chosen and this approach is known as the *steepest ascent*. After making preliminary experiments, we observe that the greedy approach results in better near-collisions in shorter times.

4 Experimental Results

Searching S^k_{CV}s with larger k (> 3) values might result in better near-collisions, but the method is no longer efficient. Moreover, when k is large, it is harder to find correlated $h(M, CV)$ and $h(M, CV \oplus \delta)$, where weight of δ is k. For our experiments, we use k values less than or equal to 2.

We repeat our experiments approximately 2^{25} times and consider our method successful, whenever we obtained an ϵ/n-near collision with $\epsilon \geq 184$ for $n = 256$, $\epsilon \geq 335$ for $n = 512$ and $\epsilon \geq 623$ for $n = 1024$. These bounds are achievable by generic random search with complexity of 2^{40} as given in Table 1.

4.1 Blake-32

Blake [7] is based on the HAIFA iteration mode with a compression function that uses a modified version of the stream cipher ChaCha. The compression function of Blake-32 inputs 256-bit CV, 512-bit message block, 128-bit salt and 64-bit counter and outputs 256-bit CV. The function is composed of 10 rounds and in each round, the nonlinear function G that operates on four words is applied to columns and diagonals of the state.

In our experiments, 1-bit difference to the input message blocks are given and the counter and the salt are fixed to zero. For 1-round compression function of Blake-32, we easily obtained 252/256-bit near-collisions. These near-collisions are obtained whenever we give a 1-bit difference to the $9th$, $11th$, $13th$ or $15th$ word of the message blocks. Then, we consider 1.5-round compression function in which the half round corresponds to the applications of G to the columns of the state. The best result we obtained for 1.5-round and 2-round Blake is 209/256-bit and 184/256-bit near collisions, respectively (See Table 2). For larger rounds, the hill climbing method did not provide significantly better results compared to the generic random search.

Table 2. Example Near Collisions for the Compression Function of Blake

1-Round Compression Function: 252/256-bit Near Collision	
M_1	8f4a6174 719e5909 41112fdc e5fa805a 1bdea684 b491ec4a 4deb8a83 5f31cf20 6a111277 4b6ff9f9 3f210a47 67388c82 a54cbe2a 3ac0d8e6 8042a2a5 c0549b9e
M_2	8f4a6174 719e5909 41112fdc e5fa805a 1bdea684 b491ec4a 4deb8a83 5f31cf20 6a111277 4b6ff9f9 3f210a47 67388c82 a54cbe2a 3ac0d8e6 8042a2a5 c0549b1e
CV	c34a1a90 c6955a4e c0c7e9ab cbf5b76c fbab3691 3368498b a8801cd7 20267316
$h(M_1, CV) \oplus h(M_2, CV)$	00000000 80000000 00000000 00000080 01000000 00000000 80000000 00000000
1.5-Round Compression Function: 209/256-bit Near Collision	
M_1	4ffcdfb9 5429ec40 18f9d1d6 c2b5b039 09c31d11 18d1bc19 532edb9c 58e3664a f757e1bf 6b0acf84 6d01bd05 0ec90891 a439a1bf c8de2b0e be5a524a ae843e5a
M_2	4ffcdfb9 5429ec40 18f9d1d6 c2b5b039 09c31d11 18d1bc19 532edb9c 58e3664a f757e1bf 6b0acf84 6d01bd05 0ec90891 a439a1bf c8de2b0e be5a524a ae843eda
CV	67134117 63e4044d 1a0bbd2b b99824e3 cb638884 8b8d284f 13977bba ad75b3a0
$h(M_1, CV) \oplus h(M_2, CV)$	00006020 80080801 88008008 80808898 412300a1 03003810 99100081 b1008118
2-Round Compression Function: 184/256-bit Near Collision	
M_1	3bd4eee9 035c9cd7 d35de9f7 cd3ab897 6f4fc516 e117aa80 ff72acc8 05c22424 87aa2e99 cec2210d 2fd0974b 652e8e26 37acc0e7 5a7a7157 c5bb6f9b 7853cda1
M_2	3bd4eee9 035c9cd7 d35de9f7 cd3ab897 6f4fc516 e117aa80 ff72acc8 05c22424 87aa2e99 cec2210d 2fd0974b 652e8e26 37acc0e7 5a7a7157 c5b96f9b 7853cda1
CV	c25dd2cd 2030a7b6 0fc043e8 5a0b5096 f084c81f 1f90d7d6 af48e019 34cd3554
$h(M_1, CV) \oplus h(M_2, CV)$	01c40003 180ac188 20818018 31442186 13309080 0858600b 143a4041 7f3144d0

The results presented in this paper are obtained by giving input difference to only the message bits. Giving additional differences to input chaining value, salt and counter as in [11] increases the flexibility of the attacker. Another flexibility for the attackers is to start the attack on a middle round, instead of the first round of the compression function as in [11, 12]. To compare the available results, we run our algorithm for 4-round compression function for a couple of days. Comparison of near-collision attacks on Blake-32 is given in Table 3.

It is possible to extend the result on the compression function to a a semi-free start near-collision attack on reduced round Blake-32, by choosing short messages such that the padding and the message fits one message block, i.e. the length of the padded message is 512-bits.

Table 3. Comparison of results on reduced-round compression function of Blake-32

Paper	Rounds	Complexity	Type	Difference
✓	1	2^1	252/256-bit near-collision	Message
✓	1.5	$< 2^{26}$	209/256-bit near-collision	Message
✓	2	$< 2^{26}$	184/256-bit near-collision	Message
[12]	4 (4-7)	2^{21}	152/256-bit near-collision	Message, CV
✓	4	$2^{37.39}$	182/256-bit near-collision	Message
[11]	4 (3-6)	2^{56}	232/256-bit near-collision	Message, CV, Salt, Counter

4.2 Fugue

Fugue, designed by Halevi et al. [8], is a sponge-like design inspired by Grindahl. Fugue is based on the F-256 function that uses a large internal state of thirty 32-bit words. F-256 operates 32-bit message blocks using a round transformation that consists of the following operations; *(i)* **TIX**(I) that loads the 32-bit message blocks to the state, *(ii)* **ROR3** that rotates the state by three columns, *(iii)* **CMIX** that mixes columns and *(iv)* **SMIX** that applies a nonlinear substitution to the first four columns of the state. The pseudocode of F-256 is given in Algorithm 4.1. The default value of (r, g_1, g_2) is $(2, 10, 13)$.

Algorithm 4.1: F-256$(M_1, \ldots, M_m, IV_0, \ldots, IV_7, r, g_1, g_2)$

for $i \leftarrow 0$ **to** 21
  $S_i = 0;$
for $i \leftarrow 22$ **to** 29
  $S_i = IV_{i-22};$
for $i \leftarrow 1$ **to** m
  $TIX(M_i);$
  for $j \leftarrow 1$ **to** r
    $ROR3; CMIX; SMIX;$
for $i \leftarrow 1$ **to** g_1
  $ROR3; CMIX; SMIX;$
for $i \leftarrow 1$ **to** g_2
  $S_4+=S_0;\ S_{15}+=S_0;\ ROR15;\ SMIX;$
  $S_4+=S_0;\ S_{16}+=S_0;\ ROR14;\ SMIX;$
return $(S_1, S_2, S_3, S_4, S_{15}, S_{16}, S_{17}, S_{18}.)$

In our experiments, we selected 32-bit random messages without considering the padding scheme. We made our experiments on 260 $(= 2 \times 10 \times 13)$ various versions of F-256 based on the selection of r, g_1 and g_2. For each version, we repeat the experiment 2^{10} times and the results better than 184/256-bit near-collisions are summarized in Table 4. Table 5 gives examples for three of these cases.

Table 4. Summary of best results for different reduced versions of F-256

(r, g_1, g_2)	Best Near-collision result
(1,1,1),(1,1,2),(1,2,1),(1,2,2),(1,2,3),(1,2,4),(1,2,5),(2,1,1),(2,1,2),(2,1,3),(2,1,4),(2,1,5)	Collision
(1,1,3),(1,2,6),(1,3,1),(1,3,2),(1,3,3),(1,3,4),(1,3,5),(1,3,6),(1,3,7),(1,3,8),(2,1,6),(2,2,1),(2,2,2),(2,2,3),(2,2,4),(2,2,5),(2,2,6),(2,2,7),(2,2,8)	$\geq$ 231/256-bit near-collision
(1,1,4),(1,1,5),(1,2,7),(1,2,8),(1,3,9),(1,3,10),(2,1,7),(2,1,8),(2,2,9),(2,2,10)	$\geq$ 184/256-bit near-collision

Table 5. Example near-collisions for Fugue

(r,g_1,g_2)=**(2,1,5):** Collision	
M_1	bce97e99
M_2	e60cdffb
CV	abc6c947 328bc6cd 24f38ca6 92ec7e0d 2bad9edc 1a87407e 263df40e 08e04f24
$h(M_1, CV) \oplus h(M_2, CV)$	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
(r,g_1,g_2)=**(2,2,8):** 233/256-bit near-collision	
M_1	689bbd81
M_2	8190b5d7
CV	6ed96b2e 2ae5c7ab 0d8d69cb c5e7b6a7 eec2db5a ac01de5f e9a8c177 9586f645
$h(M_1, CV) \oplus h(M_2, CV)$	00000000 00000000 a3483006 60810025 00000000 00000000 00000000 05800910
(r,g_1,g_2)=**(2,2,10):**184/256-bit Near collision	
M_1	dfabff02
M_2	190f9aae
CV	ebe94b66 2317fc47 2e6fdd25 639b599d ba370a60 bae80646 24704d9d c422d075
$h(M_1, CV) \oplus h(M_2, CV)$	1070011a 104182c0 f0513849 474e0448 b1645436 20240251 00000000 45088e35

4.3 Hamsi

Hamsi, designed by Küçük [9], is based on the concatenate-permute-truncate design strategy. The compression function of Hamsi-256 inputs a 32-bit message block and a 256-bit chaining value and outputs a 256-bit chaining value. The compression function acts on a state of 512 bits, which can be considered as a 4x4 matrix of 32-bit words.

First, 32-bit message block is expanded to 256 bits using a linear code (128,16,70) over $\mathbb{F}_4$. Then, the expanded message and the chaining value, each of being eight 32-bit words is loaded to the state of Hamsi-256. Then, the state is XORed with the predefined constants and a round counter and each of the 128 columns of the state goes through a 4x4 s-box. Finally, a linear transformation L, is applied to the four independent diagonals of the state. The compression function has 3 rounds, and a round transformation contains addition of constants, substitution and diffusion operations.

Nikolic [13] found 231/256-bit pseudo near-collisions for the compression function of Hamsi-256 for fixed message blocks. Wang et al. [14] improved the attack and practically showed 233/256-bit pseudo near-collisions for the compression function of Hamsi-256. In both attacks, the message block is fixed and the difference is given to the input chaining value. It should also be noted that the weight of the input differences on chaining values is smaller than the weight of the output difference that makes it harder to use the near-collisions to attack the hash function.

In our experiments, no input difference is given to the chaining values and two random 32-bits message blocks are chosen as input. Giving a small-weight differences to the message does not provide better results, since input differences are expanded by the linear code. The near-collision results obtained for 1 and 2 round compression function are provided in Table 6. For 3-round compression function, the hill climbing method did not provide results significantly better than the generic random search.

Table 6. Example Near-collisions for the Compression Function of Hamsi

1- Round Compression Function: 232/256-bit Near-collision	
M_1	`22e20185`
M_2	`dd1dfe7a`
CV	`f6bf6de4 13429c65 b149b61a af8ed58d` `e3068bc8 e0397375 22866132 a8c5d4d3`
$h(M_1, CV) \oplus h(M_2, CV)$	`00042000 80040000 28040100 10000000` `40080802 c8080000 00040000 0801004b`
2- Round Compression Function: 192/256-bit Near-collision	
M_1	`cf15a470`
M_2	`2287860c`
CV	`5b0ef41a f6933669 9d50a0b1 f3a0d239` `63d65d26 fdca6f81 1509bfea f6e73e66`
$h(M_1, CV) \oplus h(M_2, CV)$	`8810058e 00021462 c330a008 7224440b` `02008812 31040d80 8a9c0060 0c028448`

4.4 JH

JH, defined by Wu [10], is an iterated hash function with a compression function structure as seen in Figure 1. In the compression function of JH, the 1024-bit chaining value and the 512-bit message block are compressed into the 1024-bit chaining value. Initially, the lower half of the state is XORed with the input message block and then the bijection function E is applied. Then, the upper half of the state is XORed with the input message block (See Figure 1). The bijective function includes a grouping function, the round function (run 35 times), an additional substitution layer together with a de-grouping function. Basic building blocks of the compression function are two 4×4 s-boxes and $(4, 2, 3)$ Maximum Distance Separable (MDS) code over $GF(2^4)$.

In [15], Rijmen et al. found 1008/1024-bit semi-free-start near-collision for 19 rounds of JH for all hash sizes with $2^{156.77}$ compression function calls and $2^{143.70}$

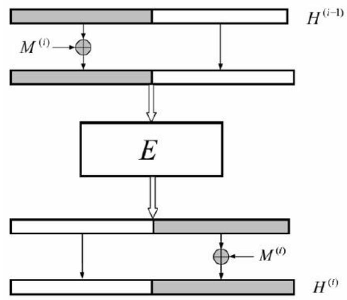

Fig. 1. Compression function of JH

byte memory complexity, and 768/1024-bit semi-free-start near-collision for 22 rounds with $2^{156.56}$ compression function calls and the same memory complexity, employing the rebound attack [16].

In our experiments, we choose two 512-bit random messages with 1-byte difference, and without considering the padding block, the attack is successful up to 10 rounds of the compression function of JH (out of 35) and the best results are summarized in Table 7.

Table 7. Near-collisions for the compression function of JH

Rounds	Near-collision	Complexity
1	1023/1024	$2^{20.31}$
2	1020/1024	$2^{18.57}$
3	1019/1024	$2^{19.20}$
4	1013/1024	$2^{19.80}$
5	1005/1024	$2^{25.01}$
6	991/1024	$2^{27.57}$
7	942/1024	$2^{20.71}$
8	907/1024	$2^{24.24}$
9	816/1024	$2^{19.77}$
10	820/1024	$2^{23.24}$

Table 8 provides example near-collisions for 9 and 10 round compression function of JH.

Table 8. Example near-collisions for 9-round and 10-round compression function of JH

9- Round Compression Function: 816/1024-bit Near Collision								
M_1	7b6d6a9e	464d09e1	86410000	35aeff35	db02a693	1da2914e	0e340511	4bb9b2df
	9847eb69	ab7422cd	efa4d5ed	eb7c248f	c09f84f4	8e71652f	c8af1bed	911a8de6
M_2	9b6d6a9e	464d09e1	86410000	35aeff35	db02a693	1da2914e	0e340511	4bb9b2df
	9847eb69	ab7422cd	efa4d5ed	eb7c248f	c09f84f4	8e71652f	c8af1bed	911a8de6
CV	64cdd586	e453fbab	60c0a125	a596b15e	22735167	8d69b439	b8039dd3	327bacbb
	55685b28	5a717a0b	e1cc05c8	fc607792	fc31f4cb	49ff1ca2	be3aba98	1618e6a3
	da5021d9	895c668b	ab40f1c5	6526e807	4074d5b1	e8141140	63bc2df1	8f738ba6
	5def4921	0385997f	da7b308d	30f64dd7	56a7301e	64bc927a	da94cded	3ede8236
$h(M_1, CV) \oplus h(M_2, CV)$	54504100	45114010	50045455	40400101	41444001	15450001	00554501	11041044
	44004114	10004501	10455441	04115401	40551514	14105014	01500441	01501004
	b0010405	04010514	44511000	54001541	05100545	04144510	10040144	00514404
	11445500	45005400	01000400	01100014	44040455	44440000	05000405	45441440
10- Round Compression Function: 820/1024-bit Near Collision								
M_1	2dcdeb76	ed262d2f	16c56a55	90cb76fa	59e71f06	765a5e59	6aa1ba10	24fe14b1
	aaa28629	918fea7f	da88deba	87110630	ca28d5ed	83465471	be02a361	2df6564f
M_2	2bcdeb76	ed262d2f	16c56a55	90cb76fa	59e71f06	765a5e59	6aa1ba10	24fe14b1
	aaa28629	918fea7f	da88deba	87110630	ca28d5ed	83465471	be02a361	2df6564f
CV	faa3c300	af6a90ae	b49356e2	6994afd8	ef1a1119	5a43864d	d2a9b5f1	bcc08129
	468a89c5	df2c42eb	8abe5884	f3688af1	98978ec7	b63c05a3	5af13a34	43c52bc2
	2313f9b7	e8013174	2a3389ff	439c0432	ad4ab2e8	23934359	33a12345	52a427f7
	bbae8074	2bf65083	ec04ee67	21e2e376	20760866	ad6f586e	97837de8	22c7c119
$h(M_1, CV) \oplus h(M_2, CV)$	d0848cda	80b0560d	00000000	00000000	00000000	00000000	24981865	56b25240
	4a83359e	400c1b6b	00000000	00000000	00000000	00000000	13709c6e	db64dc89
	06e12007	4490779e	00000000	00000000	00000000	00000000	e417dc75	f465014e
	44496142	3105c9a0	00000000	00000000	00000000	00000000	5404400f	a8013ca8

5 Conclusion

In this study, we consider simple hill-climbing methods to find near-collisions for the reduced round compression functions of some of the round two SHA-3 candidates. The hill-climbing methods produced better results compared to the generic random search, when the diffusion of chaining value bits is not fully satisfied.

We run the algorithms approximately 2^{25} times and compared the best obtained near-collision to the one obtained with 2^{40} complexity with generic random search.

We practically obtained *(i)* 184/256-bit near-collision for the 2-round compression function of Blake-32; *(ii)* 192/256-bit near-collision for the 2-round compression function of Hamsi-256; *(iii)* 820/1024-bit near-collisions for 10-round compression function of JH. For Fugue, it is possible to define 260 different reduced versions by the selection of the parameter (r, g_1, g_2). We obtained collisions for 12 reduced cases near-collisions with distance less than 25 for 19 cases and near collisions with distance less than or equal to 72 for 10 cases.

The results obtained in this study do not affect the security of the hash functions against preimage, second preimage and collision attacks, but rather give a security margin of the compression functions against near-collision attacks. Since Fugue, Hamsi and JH process an additional message block including the padding, the results cannot be directly extended to the hash function. For 2-round Blake-32, by selecting message blocks that include the padding, the results can be extended to a semi-free start near-collision attack.

References

1. David S. Johnson and Lyle A. Mcgeoch. The Traveling Salesman Problem: A Case Study in Local Optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310, 1997.
2. John Andrew Clark. *Metaheuristic search as a cryptological tool.* PhD thesis, Department of Computer Science, University of York, 2001.
3. William Millan and Andrew Clark. Smart hill climbing finds better boolean functions. In *In Workshop on Selected Areas in Cryptology 1997, Workshop Record*, pages 50–63, 1997.
4. William Millan and Andrew Clark. Boolean function design using hill climbing methods. In *4th Australian Conference on Information Security and Privacy*, pages 1–11. Springer-Verlag, 1999.
5. Yuriy Izbenko, Vladislav Kovtun, and Alexandr Kuznetsov. The design of boolean functions by modified hill climbing method. *Information Technology: New Generations, Third International Conference on*, 0:356–361, 2009.
6. National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. *Federal Register*, 27(212):62212–62220, 2007. Available at: http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf.
7. Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. Sha-3 proposal BLAKE. Submission to NIST, 2008.

8. Shai Halevi, William E. Hall, and Charanjit S. Jutla. The Hash Function Fugue. Submission to NIST (updated), 2009.
9. Özgül Küçük. The Hash Function Hamsi. Submission to NIST, 2008.
10. Hongjun Wu. The Hash Function JH. Submission to NIST (updated), 2009.
11. Jean-Philippe Aumasson, Jian Guo, Simon Knellwolf, Krystian Matusiewicz, and Willi Meier. Differential and invertibility properties of blake. In Seokhie Hong and Tetsu Iwata, editors, *FSE*, volume 6147 of *Lecture Notes in Computer Science*, pages 318–332. Springer, 2010.
12. Bozhan Su, Wenling Wu, Shuang Wu, and Le Dong. Near-collisions on the reduced-round compression functions of skein and blake. Cryptology ePrint Archive, Report 2010/355, 2010.
13. Ivica Nikolic. Near Collisions for the Compression Function of Hamsi-256. CRYPTO rump session, 2009.
14. Meiqin Wang, Xiaoyun Wang, Keting Jia, and Wei Wang. New Pseudo-Near-Collision Attack on Reduced-Round of Hamsi-256. Cryptology ePrint Archive, Report 2009/484, 2009.
15. Vincent Rijmen, Deniz Toz, and Kerem Varici. Rebound Attack on Reduced-Round Versions of JH. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption, FSE 2010*, Lecture Notes in Computer Science, page 18, Seoul,Korea, 2010. Springer-Verlag.
16. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Orr Dunkelman, editor, *FSE*, volume 5665 of *LNCS*, pages 260–276. Springer, 2009.