Study of near collisions in reduced versions of BLAKE, Grøstl and Keccak

Soham Sadhu

August 9, 2014

Abstract

- Cryptographic hash functions are used to create and verify digital signatures.
- U.S government standardizes the hash function to be used in all non military government agency through Federal Information and Processing Standards (FIPS) publications for Secure Hashing Algorithm (SHA).
- In October 2012, Keccak was chosen as the winner of SHA-3 competition amongst 64 candidates, including the finalists BLAKE and Grøstl.
- We study near collisions in reduced versions of BLAKE, Grøstl and Keccak; using hill climbing, random selection, simulated annealing and tabu search.

Table of contents I

Table of contents II

Table of contents III

Hash function

A *hash family* is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, satisfying the following conditions.[1]

- $\mathcal{X}$ is a set of possible messages
- $\mathcal{Y}$ is a finite set of hash function output
- $\mathcal{K}$, the *keyspace*, is a finite set of possible keys
- For each $K \in \mathcal{K}$, there is a hash function $h_k \in \mathcal{H}$. Each $h_k : \mathcal{X} \to \mathcal{Y}$

[1] Douglas R. Stinson. Cryptography Theory and Practice, chapter 4. Cryptographic Hash Functions. Chapman & Hall/CRC, Boca Raton, FL 33487-2742, USA, third edition, 2006.

Property of Hash function[2]

1. **Preimage resistance**
   Given: A hash function $h : \mathcal{X} \to \mathcal{Y}$ and an element $y \in \mathcal{Y}$.
   Find: $x \in \mathcal{X}$ such that $h(x) = y$.
2. **Second preimage**
   Given: A hash function $h : \mathcal{X} \to \mathcal{Y}$ and an element $x \in \mathcal{X}$.
   Find: $x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x) = h(x')$.
3. **Collision resistance**
   Given: A hash function $h : \mathcal{X} \to \mathcal{Y}$
   Find: $x, x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x') = h(x)$.

[2] Douglas R. Stinson. Cryptography Theory and Practice, chapter 4. Cryptographic Hash Functions. Chapman & Hall/CRC, Boca Raton, FL 33487-2742, USA, third edition, 2006.

Application of hash functions

1. **Digital forensics:** take a hash value of evidence, to later prove that it has not been tampered. [3]
2. **Password stored:** is salted and hashed, before inserting to database.
3. **File integrity:** take hash value of files between time intervals, to make sure; they have not been tampered.
4. **Pseudo random:** generator, based on a seed value.

[3] Richard P. Salgado. Fourth Amendment Search And The Power Of The Hash, volume 119 of 6, pages 38 46. Harvard Law Review Forum, 2006.

SHA-0

- SHA-0 proposed by NSA in 1993, later standardised by NIST.
- In 1995 Florent Chabaud and Antoine Joux, found collisions in SHA-0 with complexity of 2^{61}.
- In 2004, Eli Biham and Chen found near collisions for SHA-0, about 142 out of 160 bits to be equal.
- Full collisions were also found, when the number of rounds for the algorithm were reduced from 80 to 62.

SHA-1

- In 1995, SHA-0 replaced by SHA-1, designed by NSA[4]. SHA-1 had block size of 512 bits, size of 160 bits; and additional circular shift operation, to rectify weakness from SHA-0.
- In 2005, team from Shandong University found collisions on full version of SHA-1 requiring 2^{69} operations[5].

[4] James Joshi. Network Security: Know It All: Know It All. Newnes Know It All. Elsevier Science, 2008.

[5] Bruce Schneier. Sha-1 broken. http://www.schneier.com/blog/archives/2005/02/sha1_broken.html, February 2005.

SHA-2

- SHA-2 was designed by NSA, and released in 2001 by NIST. Family of functions of SHA-224, SHA-256, SHA-384, SHA-512.
- Computational operations for finding collisions in SHA-256 for 23-step was found to be around $2^{11.5}$, and for 24 step was $2^{28.5}$ respectively.
- Computational operations for finding collisions in SHA-512 for 23-step was found to be around $2^{16.5}$, and for 24 step was $2^{32.5}$ respectively[6].

[6] Somitra Kumar Sanadhya and Palash Sarkar. New collision attacks against up to 24- step sha-2. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, INDOCRYPT, volume 5365 of Lecture Notes in Computer Science, pages 91103. Springer, 2008.

SHA-3

- NIST announced competition for choosing SHA-3 on November, 2007. Entries accepted till October, 2008.
- 51 candidates from 64 submissions, were accepted for first round on December 9, 2008.
- Out of 5 finalists, on October 2, 2012; Keccak was announced winner amongst other four finalist, which included BLAKE and Grøstl.
- Keccak was chosen for large security margin, flexibility, and efficient hardware implementation.

FIPS PUB 202[7]

SHA3-224(M) = KECCAK[448](M || 01, 224)
SHA3-256(M) = KECCAK[512](M || 01, 256)
SHA3-384(M) = KECCAK[768](M || 01, 384)
SHA3-512(M) = KECCAK[1024](M || 01, 512)
SHAKE128(M, d) = KECCAK[256](M || 1111, d)
SHAKE256(M, d) = KECCAK[512](M || 1111, d)

[7] Information Technology Laboratory, National Institute of Standards and Technology. DRAFT FIPS PUB 202, SHA-3 Standard: Permutation Based Hash and Extendable-Output Functions. May 2014.

Keccak sponge construction

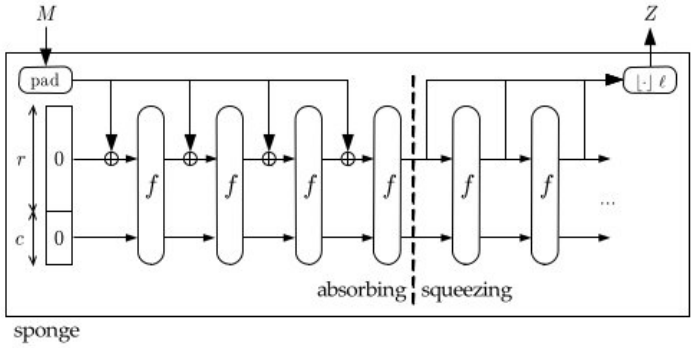


Figure: Sponge construction $Z = Sponge[f, pad, r](M, l)$[8]

[8] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions. http://sponge.noekeon.org/CSF-0.1.pdf,

Keccak state

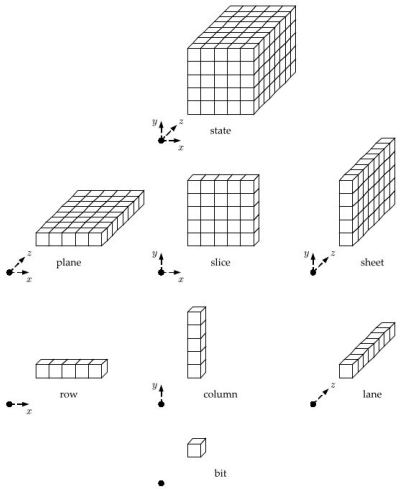


Figure: Keccak state terminology $Z = Sponge[f, pad, r](M, l)$[9]

Padding and permutations

- The message is padded with 10*1 to make it multiple of the block length.
- $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$
- The permutation round R is repeated for 12 + 2l times, where l is the lane length. The default capacity size is 576.

Contents of Keccak permutation round

$$\theta : a[x][y][z] \leftarrow a[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x+1][y'][z-1]$$

$$\rho : a[x][y][z] \leftarrow a[x][y][z-(t+1)(t+2)/2],$$

$$t \text{ satisfying } 0 \leq t < 24 \text{ and } \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \text{ in } G$$

$$\text{or } t = -1 \text{ if } x = y = 0,$$

$$\pi : a[x][y] \leftarrow a[x'][y'], \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix},$$

$$\chi : a[x] \leftarrow a[x] + (a[x+1] + 1)\, a[x+2],$$

$$\iota : a \leftarrow a + RC[i_r].$$

ι step

- The round constants are given by
  $$RC[i_r][0][0][2^j - 1] = rc[j + 7i_r] \text{ for all } 0 \leq j \leq l,$$
- $rc[t] = (x^t \ mod \ x^8 + x^6 + x^5 + x^4 + 1)$ mod x in GF(2)[x]

Properties of BLAKE hash function

Algorithm	Word	Message	Block	Digest	Salt
BLAKE-224	32	$< 2^{64}$	512	224	128
BLAKE-256	32	$< 2^{64}$	512	256	128
BLAKE-384	64	$< 2^{128}$	1024	384	256
BLAKE-512	64	$< 2^{128}$	1024	512	256

Table: Specification of available input, output, block and salt sizes for various BLAKE hash functions, size in bits. [11]

[11] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. Blake. http://www.131002.net/blake/blake.pdf, April 2012.

BLAKE construction

BLAKE is built on HAIFA (HAsh Iterative FrAmework) structure [12]which is an improved version of Merkle-Damgård function.

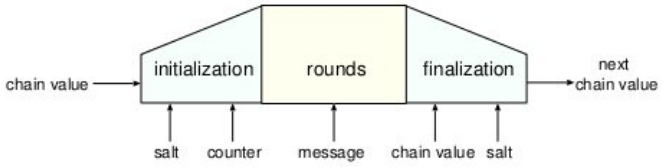


Figure: Local wide construction of BLAKE's compression function[13]

[12] Eli Biham and Orr Dunkelman. A framework for iterative hash functions - haifa. Cryptology ePrint Archive, Report 2007/278, 2007.

[13] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. Blake. http://www.131002.net/blake/blake.pdf, April 2012.

Padding rule

- For variant producing digest size 224, 256 input message is padded with '1' followed by '0' bits, so that length is 447 modulo 512. Followed by bit '1', and 64 bit unsigned big endian representation of block length.
- For variant producing digest size 384, 512 input message is padded by bit '1', followed by '0' bits till length is 895 modulo 1024. Followed by bit '1', and 128 bit unsigned big endian representation of block length in bits.

Compression algorithm

Algorithm 1 BLAKE Compression procedure[14]

1: $h^0 \leftarrow IV$
2: **for** $i = 0, \ldots, N-1$ **do**
3: $\quad h^{i+1} \leftarrow compress(h^i, m^i, s, l^i)$
4: **end for**
5: **return** h^N

[14] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. Blake. http://www.131002.net/blake/blake.pdf, April 2012.

Initialization of the state

$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix} \leftarrow \begin{pmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ s_0 \oplus c_0 & s_1 \oplus c_1 & s_2 \oplus c_2 & s_3 \oplus c_3 \\ t_0 \oplus c_4 & t_0 \oplus c_5 & t_1 \oplus c_6 & t_1 \oplus c_7 \end{pmatrix}$$

After initialization the matrix is operated for 14 or 16 rounds depending on version, on the following groups represented as $G_i(a, b, c, d)$

$$\begin{array}{lll} G_0(v_0, v_8, v_{12}) & G_1(v_1, v_5, v_9, v_{13}) & G_2(v_2, v_6, v_{10}, v_{14}) \\ G_3(v_3, v_7, v_{11}, v_{15}) & G_4(v_0, v_5, v_{10}, v_{15}) & G_5(v_1, v_6, v_{11}, v_{12}) \\ G_6(v_2, v_7, v_8, v_{13}) & G_7(v_3, v_4, v_9, v_{14}) & \end{array}$$

BLAKE permutation operation for 224 and 256 variant

$a \leftarrow a + b + (m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)})$
$d \leftarrow (d \oplus a) \ggg 16$
$c \leftarrow c + d$
$b \leftarrow (b \oplus c) \ggg 12$
$a \leftarrow a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)})$
$d \leftarrow (d \oplus a) \ggg 8$
$c \leftarrow c + d$
$b \leftarrow (b \oplus c) \ggg 7$

$+$ addition in modulo 2^{32}
$\ggg k$ rotate to right by k bits
$\oplus$ bitwise XOR
r permutation round
i is from G_i

Permutation round selection, σ function[15]

σ_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_1	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
σ_2	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
σ_3	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
σ_4	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
σ_5	2	12	6	10	0	11	8	3	4	13	7	5	15	14	1	9
σ_6	12	5	1	15	14	13	4	10	0	7	6	3	9	2	8	11
σ_7	13	11	7	14	12	1	3	9	5	0	15	4	8	6	2	10
σ_8	6	15	14	9	11	3	0	8	12	2	13	7	1	4	10	5
σ_9	10	2	8	4	7	6	1	5	15	11	9	14	3	12	13	0

[15] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. Blake. http://www.131002.net/blake/blake.pdf, April 2012.

BLAKE permutation operation for 384 and 512 variant

$$
\begin{aligned}
a &\leftarrow a + b + (m_{\sigma_r(2i)} \oplus c_{\sigma_r(2i+1)}) \\
d &\leftarrow (d \oplus a) \ggg 32 \\
c &\leftarrow c + d \\
b &\leftarrow (b \oplus c) \ggg 25 \\
a &\leftarrow a + b + (m_{\sigma_r(2i+1)} \oplus c_{\sigma_r(2i)}) \\
d &\leftarrow (d \oplus a) \ggg 16 \\
c &\leftarrow c + d \\
b &\leftarrow (b \oplus c) \ggg 11
\end{aligned}
$$

$+$ addition in modulo 2^{64}

Finalization

$$
\begin{aligned}
h'_0 &\leftarrow h_0 \oplus s_0 \oplus v_0 \oplus v_8 \\
h'_1 &\leftarrow h_1 \oplus s_1 \oplus v_1 \oplus v_9 \\
h'_2 &\leftarrow h_2 \oplus s_2 \oplus v_2 \oplus v_{10} \\
h'_3 &\leftarrow h_3 \oplus s_3 \oplus v_3 \oplus v_{11} \\
h'_4 &\leftarrow h_4 \oplus s_0 \oplus v_4 \oplus v_{12} \\
h'_5 &\leftarrow h_5 \oplus s_1 \oplus v_5 \oplus v_{13} \\
h'_6 &\leftarrow h_6 \oplus s_2 \oplus v_6 \oplus v_{14} \\
h'_7 &\leftarrow h_7 \oplus s_3 \oplus v_7 \oplus v_{15}
\end{aligned}
$$

Padding

- The message is split into blocks of 512 bits for variants of digest size upto 256 bits; and into 1024 bit block for variant of digest size above 256 bits.
- Bit '1' is appended, then $w = -N - 65 \ mod \ l$, 0 bits are appended; followed by 64 bit representation of $(N + w + 65)/l$.
- Due to encoding of message in padded block, the maximum size of message for short variants is $2^{73} - 577$ bits, and that for variants above 256 is $2^{74} - 1089$bits.

Initial values and rounds [16]

Permutations	Digest size	Recommended value of r
P_{512} and Q_{512}	8 - 256	10
P_{1024} and Q_{1024}	264 - 512	14

Table: Recommended number of rounds

n	iv_n
224	00 ... 00 00 e0
256	00 ... 00 01 00
384	00 ... 00 01 80
512	00 ... 00 02 00

Table: Initial values for Grøstl-n function.

[16] Søren Steffen Thomsen, Martin Schläffer, Christian Rechberger, Florian

Hashing the message

After padding, the message is broken to blocks, and processed sequentially. An initial $h_0 = \text{iv}$ is defined.

$$h_i \leftarrow f(h_{i-1}, m_i) \text{ for } i = 1, \ldots, t.$$

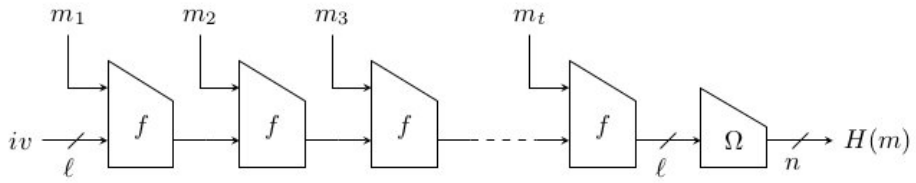


Figure: Grøstl hash function [17]

[17] Søren Steffen Thomsen, Martin Schläffer, Christian Rechberger, Florian

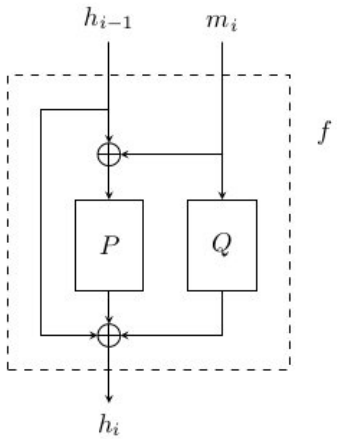
Input mapping

Mapping of the input bytes to the state in following order.

$$\begin{bmatrix} 00 & 08 & 10 & 18 & 20 & 28 & 30 & 38 \\ 01 & 09 & 11 & 19 & 21 & 29 & 31 & 39 \\ 02 & 0a & 12 & 1a & 22 & 2a & 32 & 3a \\ 03 & 0b & 13 & 1b & 23 & 2b & 33 & 3b \\ 04 & 0c & 14 & 1c & 24 & 2c & 34 & 3c \\ 05 & 0d & 15 & 1d & 25 & 2d & 35 & 3d \\ 06 & 0e & 16 & 1e & 26 & 2e & 36 & 3e \\ 07 & 0f & 17 & 1f & 27 & 2f & 37 & 3f \end{bmatrix}$$

Permutation f function

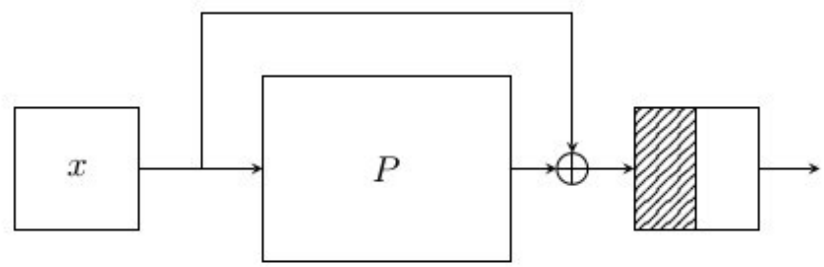
$$f(h, m) = P(h \oplus m) \oplus Q(m) \oplus h.$$



Omega truncate function

The Ω function consists of a $trunc_n(x)$ that outputs only the trailing n bits of input x.

$$\Omega(x) = trunc_n(P(x) \oplus x).$$



Contents of P and Q function

The P and Q functions are represented by a round, with slight variation in variables they operate.

$$R = MixBytes \cdot ShiftBytes \cdot SubBytes \cdot AddRoundConstant$$

Add Round Constant

$A \leftarrow A \oplus C[i]$ where A is state matrix and C is constant matrix.

$$P_{1024} : C[i] = \begin{bmatrix} 00 \oplus i & 10 \oplus i & 20 \oplus i \ldots f0 \oplus i \\ 00 & 00 & 00 \ldots 00 \\ \vdots & \vdots & \vdots \ldots \vdots \end{bmatrix}$$

and

$$Q_{1024} : C[i] = \begin{bmatrix} ff & ff & ff \ldots ff \\ \vdots & \vdots & ff \ldots ff \\ ff \oplus i & ef \oplus i & df \oplus i \ldots 0f \oplus i \end{bmatrix}$$

Substitute byte

$a_{i,j} \leftarrow S(a_{i,j}), 0 \leq i < 8, 0 \leq j < v.$

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Shift byte

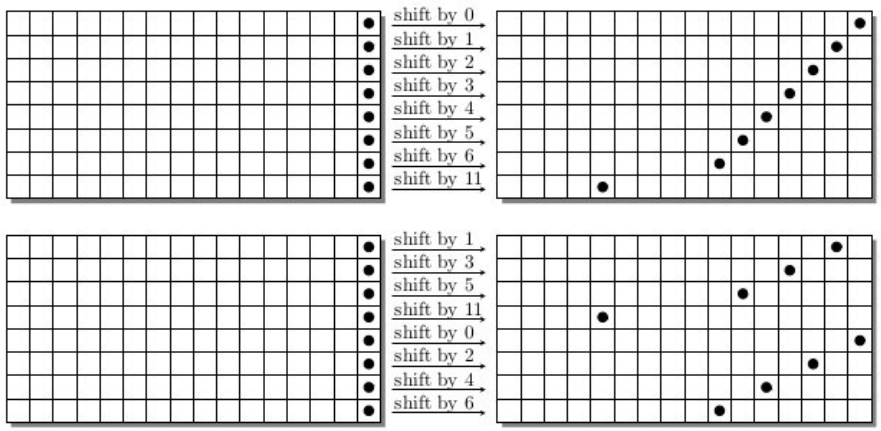


Figure: ShiftBytes transformation of permutation P_{1024}(top) and

Mix byte

$A \leftarrow B \times A$

B is a finite field over $\mathbb{F}_{256}$, defined over $\mathbb{F}_2$ by polynomial $x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1$.

$$B = \begin{bmatrix} 02 & 02 & 03 & 04 & 05 & 03 & 05 & 07 \\ 07 & 02 & 02 & 03 & 04 & 05 & 03 & 05 \\ 05 & 07 & 02 & 02 & 03 & 04 & 05 & 03 \\ 03 & 05 & 07 & 02 & 02 & 03 & 04 & 05 \\ 05 & 03 & 05 & 07 & 02 & 02 & 03 & 04 \\ 04 & 05 & 03 & 05 & 07 & 02 & 02 & 03 \\ 03 & 04 & 05 & 03 & 05 & 07 & 02 & 02 \\ 02 & 03 & 04 & 05 & 03 & 05 & 07 & 02 \end{bmatrix}$$

Collisions

1. Collision in hash function, is when we have two different messages hashing to the same value, using same initial value.
2. Full collision is when all the bits agree in the hash output for two different messages.
3. Semi-free start collision is where you change the initial value and obtain collisions for two different message.

Near collisions found with hill climbing

1. A ϵ/n bit near collision for hash function for two messages M_1 and M_2, where $M_1 \neq M_2$ is defined as $HW(h(M_1, CV) \oplus h(M_2, CV)) = n - \epsilon$.
2. Near collisions in which more than 75% of the bits were same for two different messages, were found for reduced rounds of BLAKE-32, Hamsi-256 and JH.
3. Hill Climbing starts with a random candidate, and then choosing a random successor that has a better fit to the solution. Ideally $HW(h(M, CV) \oplus h(M, CV + \delta)) = n/2$ where δ is n-bit vector with small Hamming weight.

Table of effort to find near collisions with random search

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}

Table: Approximate complexity to find a ϵ/n-bit near collision by generic random search [24]

[24] Meltem Sönmez Turan and Erdener Uyan. Practical near-collisions for reduced round blake, fugue, hamsi and jh. Second SHA-3 conference, August 2010. http://csrc.nist.gov/groups/ST/hash/sha-

How hill climbing works

1. Hill climbing algorithm will be to minimize the function $f_{M_1,M_2}(x) = HW(h(M_1, x) \oplus h(M_2, x))$.
2. CV is chosen as any random chaining value. Then the set of k-bit neighbours for the CV are created $S^k_{CV} = \{x \in \{0,1\}^n \mid HW(CV \oplus x) \leq k\}$.
3. Hill climbing is used to obtain k-optimum condition from k-bit neighbours $f_{M_1,M_2}(CV) = \min_{x \in S^k_{CV}} f_{M_1,M_2}(x)$.

Hill Climbing algorithm

Algorithm 2 Hill Climbing algorithm (M_1, M_2, k)

1: Randomly select CV
2: $f_{best} = f_{M_1,M_2}(CV)$
3: **while** (CV is not k-opt) **do**
4: CV = x such that $x \in S^k_{CV}$ with $f(x) < f(best)$
5: $f_{best} = f_{M_1,M_2}(CV)$
6: **end while**
7: **return** (CV, f_{best})

Simulated annealing I

1: **function** SIMULATED-ANNEALING(M_1, M_2, CV, schedule)
2: current $\leftarrow$ CV
3: **for** t = 1 to ∞ **do**
4: T $\leftarrow$ schedule(t)
5: **if** T = 0 **then**
6: **return** current
7: **end if**
8: next $\leftarrow$ a randomly selected successor from set $S^k_{current}$
9: $\Delta E \leftarrow f_{M_1,M_2}(current) - f_{M_1,M_2}(next)$
10: **if** ΔE > 0 **then**
11: current $\leftarrow$ next
12: **else**
13: current $\leftarrow$ next, with probability $e^{\Delta E/T}$

Simulated annealing II

```
14:         end if
15:     end for
16: end function
```


Tabu search I

```
1: function TABU-SEARCH(TabuList_size, M_1, M_2, CV)
2:     S_best ← CV
3:     TabuList ← null
4:     while S_best not k-opt do
5:         CandidateList ← null
6:         S_neighbourhood ← S^k_{S_best}
7:         for S_candidate ∈ S_best_neighbourhood do
8:             if (¬ContainsAnyFeatures( S_candidate, TabuList )) then
9:                 CandidateList ← S_candidate
10:            end if
11:        end for
12:        S_candidate ← LocateBestCandidate( CandidateList )
```


Tabu search II

```
13:        if Cost( S_candidate ) ≤ Cost( S_best ) then
14:            S_best ← S_candidate
15:            TabuList ← featureDifferences(S_candidate, S_best)
16:            while TabuList > TabuList_size do
17:                DeleteFeature( TabuList )
18:            end while
19:        end if
20:    end while
21:    return S_best
22: end function
```


Random selection I

```
1: function RANDOM-SELECTION(M_1, M_2, CV,
   number_of_trials)
2:     current ← CV
3:     trial ← 0
4:     while trial < number_of_trials do
5:         next ← randomly selected candidate from S^k_current
6:         if f_{M_1,M_2}(next) − f_{M_1,M_2}(current) then
7:             current ← next
8:         end if
9:     end while
10:    return current
11: end function
```

Hypothesis

- Reduced state Keccak, has better resistance to near collisions than BLAKE and Grøstl, for search algorithms hill climbing, simulated annealing, tabu search and random selection.
- Simulated annealing and tabu search, are better at finding near collisions compared to hill climbing and random selection.
- State size has no effect on efficiency of Keccak permutation rounds.

Input

- The seed message is "The quick brown fox jumps over the lazy dog". 20 pairs are made from the string, for each of the three categories start, middle and end.
- The pair contains the original seed message and the seed message toggled for one bit. For each category 20 bits are toggled. The bits are toggled at the start, middle and end of the seed string.
- For input string 01100010 00011000, then in input file 1.txt; pair will be made with this seed string and **1**1100010 00011000, in file 2.txt the other string will be 0**0**100010 00011000.

Input

1. For the middle section, the bits are toggled to either side of the middle bit, in the string. For example for seed string 01100010 00011000, the file 1.txt will have seed and string 0110001**1** 00011000, and file 2.txt will have modified string as 01100010 **1**0011000.
2. For the ending section the bit toggling starts from the least significant bits, so for seed string 01100010 00011000, file 1.txt has updated string 01100010 0001100**1**, and file 2.txt have updated string 01100010 0001101**0**.

Output structure

1. The output folder is arranged to have the results in the following way starting from the upper most folder Output/chaining_value/collision_search/digest_size/SHA3_finalist/numbe
2. The results for the input message pair of the respective file are inserted to corresponding output file. Output for input file start/1.txt will be found in experiment parameter defined path and then start/1.txt

Ouput file

1. The file contains 8 parameters, 3 each for either success or failure. Success is defined as finding near collision that is more than 65% of the bits similar. The parameters are number of trials in success/failure, total iterations, and average iterations.
2. The rest 2 are total iterations and average iterations. These iteration numbers are mostly useful for hill climbing, since iterations for other algorithms are fixed.

Number of trials and iterations

1. The number of trials was initially kept at 128, but then increased to 256 to get more accurate numbers. In each trial of experiment, the chaining value is randomly selected.
2. The iterations are the number of times that that algorithm has to loop through for its' operation. This is chosen, over time required since differences in algorithm implementation may skew the time taken for finding effectiveness of the search algorithm.

Chaining value and neighbourhood

1. The chaining value length can be varied from 32, 64, 128, 256, 512 and 1024. We chose to do experiment with 32 bit chaining value, and then 64 bit chaining value.
2. Other higher chaining value lengths were not tried, given the expensive time computation, and relatively less success, in finding collisions.
3. k for k bit neighbourhood value is limited to less than equal to 2. Numbers above that are not optimal.

Demo

Demo

Observations on implementation

- Use system or language representation of words like int and long, instead of array of bytes, made it easier.
- Java has signed representation of int and long. This should be considered when rotating blocks.
- In Keccak the bits are ordered in little endian format, which is differnt that other general implementation that take big endian.
- During bitwise operation of byte or short datatype, Java will upcast the data to int and then downcast result. This should be guarded against.

Iterations for BLAKE, chaining value 32 bits

Digest Size	Rounds			
	1	2	3	4
224	803	891	886	883
256	804	891	892	885
384	1137	898	899	902
512	1118	902	903	906

Table: Average iterations over all input cases for Hill Climbing for BLAKE for chaining value of bit length 32

Iterations for BLAKE, chaining value 64 bits

Digest Size	Rounds		
	1	2	3
224	4190	3465	3483
256	4264	3456	3431
384	3885	3515	3528
512	3984	3535	3559

Table: Average iterations over all input cases for Hill Climbing for BLAKE for chaining value of bit length 64

Iterations for Grøstl, chaining value 32 bits

Digest Size	Rounds			
	1	2	3	4
224	875	888	886	885
256	889	887	891	889
384	872	897	898	897
512	896	904	904	902

Table: Average iterations over all input cases for Hill Climbing for Grøstl for chaining value of bit length 32

Iterations for Grøstl, chaining value 64 bits

Digest Size	Rounds		
	1	2	3
224	3687	3468	3469
256	3714	3479	3473
384	3594	3522	3512
512	3581	3544	3559

Table: Average iterations over all input cases for Hill Climbing for Grøstl for chaining value of bit length 64

Iterations for Keccak, chaining value 32 bits

Digest Size	Rounds			
	1	2	3	4
224	532	880	883	888
256	532	888	889	895
384	533	892	899	904
512	533	900	905	902

Table: Average iterations over all input cases for Hill Climbing for Keccak for chaining value of bit length 32

Iterations for Keccak, chaining value 64 bits

Digest Size	Rounds		
	1	2	3
224	2118	3535	3457
256	2118	3557	3466
384	2134	3676	3521
512	2139	3764	3562

Table: Average iterations over all input cases for Hill Climbing for Keccak for chaining value of bit length 64

Iterations for other search algorithms

1. Number of iterations for random simulation, simulated annealing for 32 bit chaining value is fixed at 1024 iterations.
2. For 64 bit chaining value, iterations are set to 11 times the digest size. That is 2464, 2816, 4224, 5632 iterations for digest sizes 224, 256, 384 and 512 bits.
3. For tabu search for Grøstl digest length 224, and for rounds 1 and 2, averaged in iterations about 335254.443.

Iterations for Keccak state reduced 200 bits, chaining value 32 bits

Digest Size	Rounds			
	3	4	5	6
224	888	886	887	886
256	890	887	892	886
384	901	899	897	898
512	904	903	899	903

Table: Average iterations over all input cases for Hill Climbing for Keccak state reduced to 200 bits for chaining value of bit length 32

Iterations for Keccak state reduced 400 bits, chaining value 32 bits

Digest Size	Rounds			
	3	4	5	6
224	886	888	887	889
256	892	886	891	893
384	893	899	898	899
512	906	902	905	904

Table: Average iterations over all input cases for Hill Climbing for Keccak state reduced to 400 bits for chaining value of bit length 32

Iterations for Keccak state reduced 800 bits, chaining value 32 bits

Digest Size	Rounds			
	3	4	5	6
224	883	886	888	882
256	891	890	891	887
384	896	901	898	899
512	901	905	901	905

Table: Average iterations over all input cases for Hill Climbing for Keccak state reduced to 800 bits for chaining value of bit length 32

Iterations for 75% bits matching collision

Rounds	1				2			
Digest size	224	256	384	512	224	256	384	512
BLAKE	803	814	1137	1121	887	889	898	908
Grøstl	874	885	877	897	887	890	902	905
Keccak	532	532	533	533	882	885	897	902
Keccak-200	994	977	959	957	899	902	903	897
Keccak-400	836	899	972	955	886	903	903	906
Keccak-800	568	574	574	908	894	920	947	918

Table: Average iterations over all input cases for Hill Climbing for variations of Keccak and other hashing algorithms. Chaining value is bit length 32, and the near collision is 75% bit match.

Near collisions BLAKE with 32 bit chaining value

Digest Size	Rounds			
	1	2	3	4
224	41/306	13/5	34/8	25/5
256	39/256	4/3	10/4	3/2
384	58/384	0/0	0/0	0/0
512	58/384	0/0	0/0	0/0

Table: Near collisions BLAKE with 32 bit chaining value

Near collisions BLAKE with 64 bit chaining value

Digest Size	Rounds		
	1	2	3
224	56/384	38/13	43/10
256	55/384	9/3	16/4
384	60/284	0/0	0/0
512	59/384	0/0	0/0

Table: Near collisions BLAKE with 64 bit chaining value

Near collisions Grøstl with 32 bit chaining value

Digest Size	Rounds			
	1	2	3	4
224	10/6	16/6	29/7	33/9
256	1/2	6/3	10/3	7/3
384	12/128	0/0	0/0	0/0
512	22/145	0/0	0/0	0/0

Table: Near collisions Grøstl with 32 bit chaining value

Near collisions Grøstl with 64 bit chaining value

Digest Size	Rounds		
	1	2	3
224	43/23	40/12	49/11
256	25/10	14/5	15/3
384	14/256	0/0	0/0
512	28/158	0/0	0/0

Table: Near collisions Grøstl with 64 bit chaining value

Near collisions Keccak with 32 bit chaining value

Digest Size	Rounds			
	1	2	3	4
224	60/384	60/384	40/11	33/8
256	60/384	60/384	11/4	8/3
384	60/384	60/384	1/1	0/0
512	60/384	60/384	0/0	0/0

Table: Near collisions Keccak with 32 bit chaining value

Near collisions Keccak with 64 bit chaining value

Digest Size	Rounds		
	1	2	3
224	60/384	60/384	55/21
256	60/384	60/384	20/7
384	60/384	60/384	1/1
512	60/384	60/384	0/0

Table: Near collisions Keccak with 64 bit chaining value

Collision for 75% bit match

Rounds	1					
Digest size	224	256	384	512	224	256
BLAKE	26/512	23/512	22/400	21/266	0/0	0/0
Grøstl	0/0	0/0	7/256	8/256	0/0	0/0
Keccak	60/768	60/768	60/768	60/768	60/768	60/768
Keccak-200	20/256	20/256	1/1	0/0	1/1	0/0
Keccak-400	20/256	20/256	20/256	4/2	9/123	7/99
Keccak-800	60/768	60/768	60/768	20/256	60/733	60/740

Table: Collisions for 75% bit matching, for 32 bit chaining value. Application of hill climbing search. Collision instances for start, middle and end are grouped together.

Near collision numbers for Keccak internal state of 200 bits

Size	Rounds			
	3	4	5	6
224	33/8	30/6	30/8	31/9
256	12/4	5/3	8/3	11/4
384	0/0	0/0	0/0	0/0
512	0/0	0/0	0/0	0/0

Table: Collisions for Keccak state reduced to 200 bits, with hill climbing for 32 bit chaining value.

Near collision numbers for Keccak internal state of 400 bits

Size	Rounds			
	3	4	5	6
224	19/7	33/9	27/8	30/8
256	6/3	7/3	3/3	6/3
384	0/0	0/0	0/0	0/0
512	0/0	0/0	0/0	0/0

Table: Collisions for Keccak state reduced to 400 bits, with hill climbing for 32 bit chaining value.

Near collision numbers for Keccak internal state of 800 bits

Size	Rounds			
	3	4	5	6
224	38/9	27/9	32/9	32/7
256	7/4	9/5	5/2	8/4
384	2/2	0/0	0/0	0/0
512	0/0	0/0	0/0	0/0

Table: Collisions for Keccak state reduced to 800 bits, with hill climbing for 32 bit chaining value.

Near collisions BLAKE with 32 bit chaining value

Digest Size	Rounds			
	1	2	3	4
224	39/256	0/0	1/1	1/1
256	38/256	0/0	0/0	1/1
384	39/353	0/0	0/0	0/0
512	50/342	0/0	0/0	0/0

Table: Near collisions BLAKE with 32 bit chaining value

Near collisions Grøstl with 32 bit chaining value

Digest Size	Rounds			
	1	2	3	4
224	1/1	1/1	2/2	0/0
256	0/0	0/0	1/1	1/1
384	12/128	0/0	0/0	0/0
512	15/129	0/0	0/0	0/0

Table: Near collisions Grøstl with 32 bit chaining value

Near collisions Grøstl with 64 bit chaining value

Digest Size	Rounds		
	1	2	3
224	0/0	0/0	0/0
256	0/0	0/0	0/0
384	13/256	0/0	0/0
512	19/129	0/0	0/0

Table: Near collisions Grøstl with 64 bit chaining value

Near collisions Keccak with 32 bit chaining value

Digest Size	Rounds			
	1	2	3	4
224	60/384	60/384	0/0	0/0
256	60/384	60/384	0/0	1/1
384	60/384	60/384	0/0	0/0
512	60/384	60/384	0/0	0/0

Table: Near collisions Keccak with 32 bit chaining value

Near collisions BLAKE with 32 bit chaining value

Digest Size	Rounds			
	1	2	3	4
224	41/291	10/5	24/6	28/6
256	39/256	3/2	4/2	7/3
384	57/384	0/0	0/0	0/0
512	58/384	0/0	0/0	0/0

Table: Near collisions BLAKE with 32 bit chaining value

Near collisions Grøstl with 32 bit chaining value

Digest Size	Rounds			
	1	2	3	4
224	7/6	15/4	26/7	20/7
256	0/0	0/0	3/2	3/3
384	12/128	0/0	0/0	0/0
512	22/148	0/0	0/0	0/0

Table: Near collisions Grøstl with 32 bit chaining value

Near collisions Keccak with 32 bit chaining value

Digest Size	Rounds			
	1	2	3	4
224	60/384	60/384	30/11	24/5
256	60/384	60/384	13/4	8/3
384	60/384	60/384	0/0	0/0
512	60/384	60/384	0/0	0/0

Table: Near collisions Keccak with 32 bit chaining value

Validity of hypothesis

1. Two hypothesis are disproved. Keccak performs poorly for reduced rounds of 1, and 2. And equivalent performance to that of Grøstl and BLAKE, for rounds 3 and 4.
2. Simulated annealing does not perform as good as hill climbing for finding collisions in reduced rounds of the examined algorithm.
3. Random selection performance is almost comparable to simulated annealing.
4. Reducing the internal state of Keccak, did not affect its security margin to generic methods.

Collision resistance in reduced rounds

1. For round 1 in all SHA-3 examined algorithm, collisions were found in almost all instances and trials.
2. For round 2, this behaviour is continued in Keccak for all collision search algorithm except tabu search. Thus Keccak is weaker for 2 rounds compared to BLAKE and Grøstl.
3. For rounds 3 and 4, Keccak seems to have equivalent resistance to that of BLAKE and Grøstl.

Feasibility of collision search algorithm

1. Hill climbing, with greedy ascent seems to be the most feasible search algorithm for finding near collisions, while tabu search is least feasible.
2. Hill climbing consistently finds more collisions than any other algorithm, with less iterations.
3. Random selection is comparable with simulated annealing, in finding near collisions in 3 selected hashing algorithms.

State size reduction for Keccak

1. State size reduction does not negatively affect the security margin of Keccak.
2. This can be explained on the basis that lowering bit rate increases squeezing and absorbing phase, for a message to be hashed, thus subjecting it to more permutation rounds.
3. For low permutations rounds like 1, this increases margin of Keccak reduced rounds, but it gradually plateaus.

Effect of digest size

1. Collision resistance seems to be proportional to the digest size.
2. This could be due to exponential increase in the search space, it becomes harder to find better peaks for the search algorithm to move on.
3. A reason could be, that higher digest sizes have more internal state space, to execute the functions, thus making them more collision resistant.

Effect of number of rounds

1. The collision resistance increases as the number of rounds is increased.
2. No collisions are found in 256 trials, for digest sizes 384 and above; having processed more than 4 rounds, for any of the hashing algorithms.
3. It should be noted that for each algorithm permutation recommended permutation rounds differ. Grøstl has least number of permutation round, while Keccak has most of the permutation round.

Chaining value length

1. The effort to find collision, rises asymptotically for chaining value lengths.
2. Smaller chaining value lengths, like 32 bits are more effective in finding collision, than 64 bit chaining value.
3. Smaller chaining values, have smaller neighbourhoods to choose from thus increasing the feasibility.

Bit differences in particular position

1. On whole collision resistance is agnostic to bit difference in any specific position.
2. All the 3 algorithms inspected here with search algorithms, show little variation or bias, in finding collisions based on where the bit is updated.
3. The diffusion property seems to hold uniformly for all the 3 hashing algorithms.

Future work

1. The experiment can be repeated with Skein and JH, other SHA-3 finalist alongwith Keccak.
2. Other parameters could manipulated could be, decrease digest size but increase rounds.
3. Instead of benchmark at 65% bits match for near collision, try for more exacting match for collisions.
4. Try to see a relationship between chaining values, if possible.

Acknowledgements

- Thanks to Prof. Radziszowski for his patience and guidance.
- Thanks to redditors p1mrx and deejaydarvin, and crypto stack exchange user Richie Frame for explanation of bit/byte ordering in Keccak.
- Crypto stack exchange user fgrieu for help in mix byte implementation in Grøstl.
- Larry Bugbee for updating his BLAKE Python implementation, which I used as reference for my Java implementation.
- Geoffrey De Smet for tabu lists document.
- Ted Hopp, for pointing out correct way of bit wise XOR for long in Java.
- Alexandre Yamajako for pointing specification to find inverse degree of Keccak function.

Questions

Questions?