Demise of MD5 and SHA-1
Emerging New Hash

SHA-3

part 1

Stanisław Radziszowski
Computer Science
Rochester Institute of Technology
Rochester, New York, USA

March 2011

Abstract

A hash function $H : \{0,1\}^* \to \{0,1\}^m$ produces an m-bit digest of an arbitrary message, file, or even an entire file system. Typically, one wants hash functions to be easy to compute, but also infeasible to invert or to find collisions (pairs of inputs which hash to the same value). Hash functions are fundamental cryptographic primitives, and they are used extensively in authentication, preserving data integrity, digital signatures, and many other security applications.

Since 2007, the National Institute of Standards and Technology (NIST) is running a competition to design a new hash function to be used instead of a very popular but already broken MD5 (Message Digest) and the most used but much troubled SHA-1 (Secure Hash Algorithm). Out of 64 designs submitted in October 2008, now, in the final round there are 5 hash function candidates remaining. The new emerging standard, to be dubbed SHA-3, will be chosen in late 2012 from the current set of 5 finalists: BLAKE, Grøstl, JH, Keccak and Skein.

This talk will contain the background of hashing, the competition, rounds completed so far, an overview of the finalists and a prediction by the speaker who will be the winner.

.

Hash - simple, powerful idea

anything

(email, program, document, movie, file system ...)

$$x = y$$

$$\Updownarrow^{*}$$

$$H(x) = H(y)$$

256 bits

(32 bytes, like this "napisze do ciebie z dalekiej pod" ... no more)

Hashes in Practice

Applications of (cryptographic) hashes

- hash then sign
- time-stamping
- data authentication
- checksumming
- PGP email
- shadow passwords
- networking: SSL, SSH, VPN
- signatures: DSA, DSS (FIPS 186)
- MACs, HMAC (FIPS 198)
- PRF, PRNG, diffusers
- stream ciphers

The Problem

Design a (cryptographic) *hash* function $H : \{0,1\}^* \rightarrow \{0,1\}^m$ such that:

- H is *preimage resistant*, i.e. given z, it is infeasible to find any x such that $H(x) = z$
- H is *collision resistant*, i.e. it is infeasible to find any pair x and y such that $H(x) = H(y)$
- H is *resistant* to *second preimage-*, *zero preimage-* ($H^{-1}(0^m)$), *length extension-*, and other attacks.
- H is fast to compute, uses little memory
- H can operate in the streaming mode

Very LARGE bound on input length can be given, pick m as small as possible but still guaranteeing resistance properties

Merkle-Damgård iterated hash

$x \in \{0,1\}^*$ - input message
$M(x) = m_1 m_2 \cdots m_t$ - formatted input
m_t - padded, includes as tail $|x|$ in binary

H_i - chaining variables
g - postprocessing function
$compress$ - a "kind" of OWF

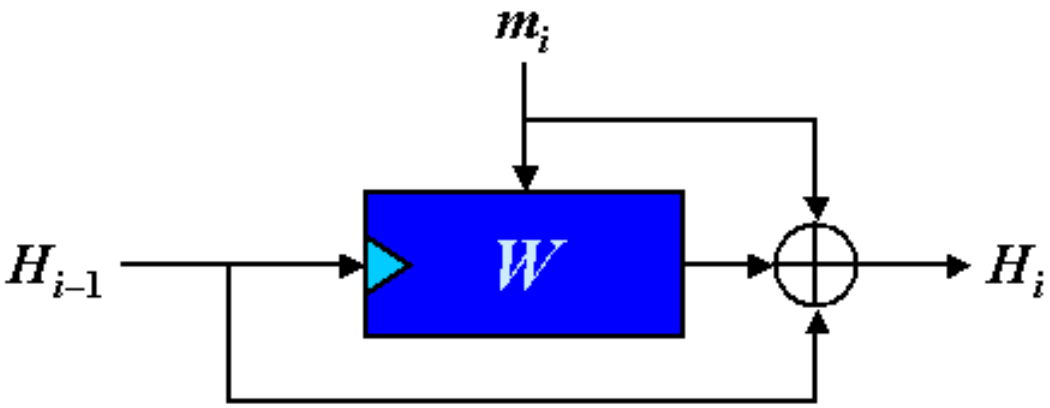
$H(x)$
    $H_0 = IV$;
    for $i = 1, 2, \ldots, t$ do
      $H_i = compress(H_{i-1} \| m_i)$;
    return $H(x) = g(H_t)$;

Merkle-Damgård iterated hash

Compression in r simpler rounds

Each m_i is "unfolded" into message schedule m_{ij}, $1 \leq j \leq r$

Compression from block cipher



Miyaguchi-Preneel compression, Whirlpool [14, 18]

There are 12 other ways to safely embed cipher into chaining, including designs by Matyas-Meyer-Oseas and Davies-Meyer

Compression types

Chaining/collection type

- narrow pipe (MD) -
  "small" state, possibly good enough
- wide pipe (MD) -
  large internal state, prevents
  multicollision attacks, fixed points
- sponge (MD) - large internal state
  permuted after each absorption
- HAIFA (MD) - wide pipe with salt
  and bit count so far injected into each
  compression, prevents multicollision
  attacks, fixed points, herding
- hash tree collection (not MD) -
  permits natural parallelization

Brief History (SHA-family biased)

- 1990 - **MD4**, Rivest, $m = 128$
- 1992 - **MD5**, Rivest, modified MD4
- 1993 - **SHA-0**, NIST, MD-like design
- 1995 - **SHA-1**, FIPS-180-1 $m = 160$
- 2002 - **SHA-2** family, NIST, FIPS-180-2 for $m = 256$, 384 and 512 bit digests [10]
- 2004-2006 Wang, Yu, Yin, et al. [22-24] collision attacks on MD5 and SHA-1
- 2007 - NIST calls for new designs [11]
- 2012 - **SHA-3** recommended to for use

All above hashes (so far) follow Merkle-Damgård template.

Hashes in Practice

The two most used hash functions

both of Merkle-Damgård type

- **MD5**, Rivest 1992

  128 bit hash, 512 bit blocks
  iterating 64-round compression c_{MD5}

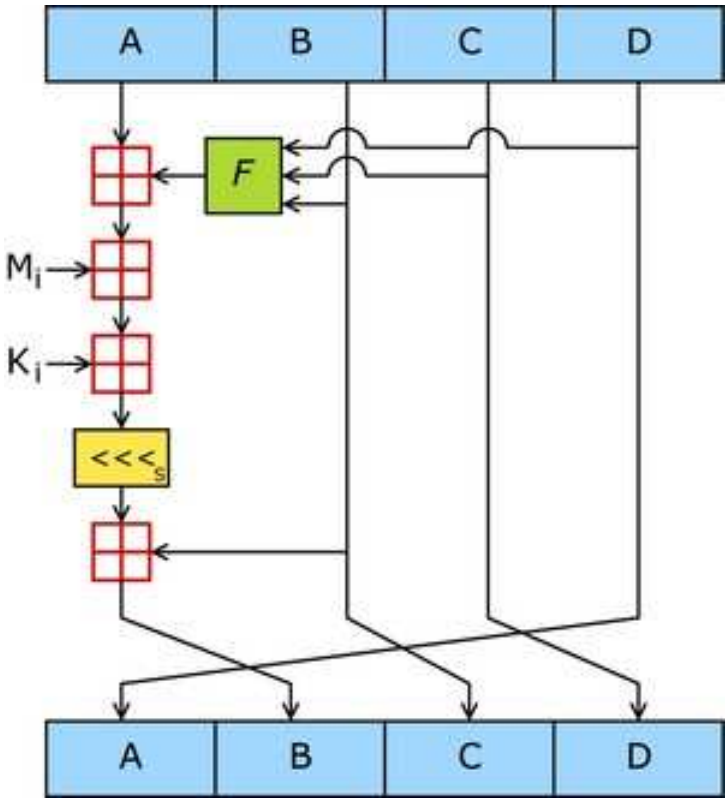
$$c_{MD5} : \{0,1\}^{640} \to \{0,1\}^{128}$$

- **SHA-1**, NSA/NIST 1995,
  created mainly for use in DSA

  160 bit hash, 512 bit blocks
  iterating 80-round compression c_{SHA-1}

$$c_{SHA-1} : \{0,1\}^{672} \to \{0,1\}^{160}$$

MD5



MD5 round structure [12]
(Wikipedia)

each unit is a 32-bit word

MD5 needs MD4

9.49 Algorithm MD4

INPUT: bitstring x of arbitrary bitlength $b \geq 0$. (For notation see Table 9.7.)
OUTPUT: 128-bit hash-code of x. (See Table 9.6 for test vectors.)

1. *Definition of constants.* Define four 32-bit initial chaining values (IVs):
   $h_1 = 0x67452301$, $h_2 = 0xefcdab89$, $h_3 = 0x98badcfe$, $h_4 = 0x10325476$.
   Define additive 32-bit constants:
   $y[j] = 0, 0 \leq j \leq 15$;
   $y[j] = 0x5a827999, 16 \leq j \leq 31$; (constant = square-root of 2)
   $y[j] = 0x6ed9eba1, 32 \leq j \leq 47$; (constant = square-root of 3)
   Define order for accessing source words (each list contains 0 through 15):
   $z[0..15] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$,
   $z[16..31] = [0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15]$,
   $z[32..47] = [0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]$.
   Finally define the number of bit positions for left shifts (rotates):
   $s[0..15] = [3, 7, 11, 19, 3, 7, 11, 19, 3, 7, 11, 19, 3, 7, 11, 19]$,
   $s[16..31] = [3, 5, 9, 13, 3, 5, 9, 13, 3, 5, 9, 13, 3, 5, 9, 13]$,
   $s[32..47] = [3, 9, 11, 15, 3, 9, 11, 15, 3, 9, 11, 15, 3, 9, 11, 15]$.
2. *Preprocessing.* Pad x such that its bitlength is a multiple of 512, as follows. Append a single 1-bit, then append $r - 1$ (≥ 0) 0-bits for the smallest r resulting in a bitlength 64 less than a multiple of 512. Finally append the 64-bit representation of $b \bmod 2^{64}$, as two 32-bit words with least significant word first. (Regarding converting between streams of bytes and 32-bit words, the convention is little-endian; see Note 9.48.) Let m be the number of 512-bit blocks in the resulting string ($b + r + 64 = 512m = 32 \cdot 16m$). The formatted input consists of $16m$ 32-bit words: $x_0 x_1 \ldots x_{16m-1}$. Initialize: $(H_1, H_2, H_3, H_4) \leftarrow (h_1, h_2, h_3, h_4)$.
3. *Processing.* For each i from 0 to $m - 1$, copy the i^{th} block of 16 32-bit words into temporary storage: $X[j] \leftarrow x_{16i+j}$, $0 \leq j \leq 15$, then process these as below in three 16-step rounds before updating the chaining variables:
   (*initialize working variables*) $(A, B, C, D) \leftarrow (H_1, H_2, H_3, H_4)$.
   (*Round 1*) For j from 0 to 15 do the following:
   $t \leftarrow (A + f(B, C, D) + X[z[j]] + y[j]), (A, B, C, D) \leftarrow (D, t \lll s[j], B, C)$.
   (*Round 2*) For j from 16 to 31 do the following:
   $t \leftarrow (A + g(B, C, D) + X[z[j]] + y[j]), (A, B, C, D) \leftarrow (D, t \lll s[j]), B, C)$.
   (*Round 3*) For j from 32 to 47 do the following:
   $t \leftarrow (A + h(B, C, D) + X[z[j]] + y[j]), (A, B, C, D) \leftarrow (D, t \lll s[j]), B, C)$.
   (*update chaining values*) $(H_1, H_2, H_3, H_4) \leftarrow (H_1 + A, H_2 + B, H_3 + C, H_4 + D)$.
4. *Completion.* The final hash-value is the concatenation: $H_1 || H_2 || H_3 || H_4$
   (with first and last bytes the low- and high-order bytes of H_1, H_4, respectively).

MD4 definition in CRC [13]
collisions found in 1995

MD5 as an edit to MD4

9.51 Algorithm MD5

INPUT: bitstring x of arbitrary bitlength $b \geq 0$. (For notation, see Table 9.7.)
OUTPUT: 128-bit hash-code of x. (See Table 9.6 for test vectors.)
MD5 is obtained from MD4 by making the following changes.

1. *Notation.* Replace the Round 2 function by: $g(u,v,w) \stackrel{\text{def}}{=} uw \vee v\overline{w}$.
   Define a Round 4 function: $k(u,v,w) \stackrel{\text{def}}{=} v \oplus (u \vee \overline{w})$.
2. *Definition of constants.* Redefine unique additive constants:
   $y[j]$ = first 32 bits of binary value $\text{abs}(\sin(j+1))$, $0 \leq j \leq 63$, where j is in radians and "abs" denotes absolute value. Redefine access order for words in Rounds 2 and 3, and define for Round 4:
   $z[16..31] = [1,6,11,0,5,10,15,4,9,14,3,8,13,2,7,12]$,
   $z[32..47] = [5,8,11,14,1,4,7,10,13,0,3,6,9,12,15,2]$.
   $z[48..63] = [0,7,14,5,12,3,10,1,8,15,6,13,4,11,2,9]$.
   Redefine number of bit positions for left shifts (rotates):
   $s[0..15] = [7,12,17,22,7,12,17,22,7,12,17,22,7,12,17,22]$,
   $s[16..31] = [5,9,14,20,5,9,14,20,5,9,14,20,5,9,14,20]$,
   $s[32..47] = [4,11,16,23,4,11,16,23,4,11,16,23,4,11,16,23]$,
   $s[48..63] = [6,10,15,21,6,10,15,21,6,10,15,21,6,10,15,21]$.
3. *Preprocessing.* As in MD4.
4. *Processing.* In each of Rounds 1, 2, and 3, replace "$B \leftarrow (t \hookleftarrow s[j])$" by "$B \leftarrow B + (t \hookleftarrow s[j])$". Also, immediately following Round 3 add:
   (*Round 4*) For j from 48 to 63 do the following:
   $t \leftarrow (A + k(B,C,D) + X[z[j]] + y[j]), (A,B,C,D) \leftarrow (D, B + (t \hookleftarrow s[j]), B, C)$.
5. *Completion.* As in MD4.

MD5 as an edit of MD4 in CRC [13]

MD6 submitted to SHA-3 competition, but no longer a candidate.

SHA-1 in Standards

HMAC

Keyed message authentication codes

key K and pads have 512 bits each

ipad $=$ 3636 ... 36,
opad $=$ 5C5C ... 5C,

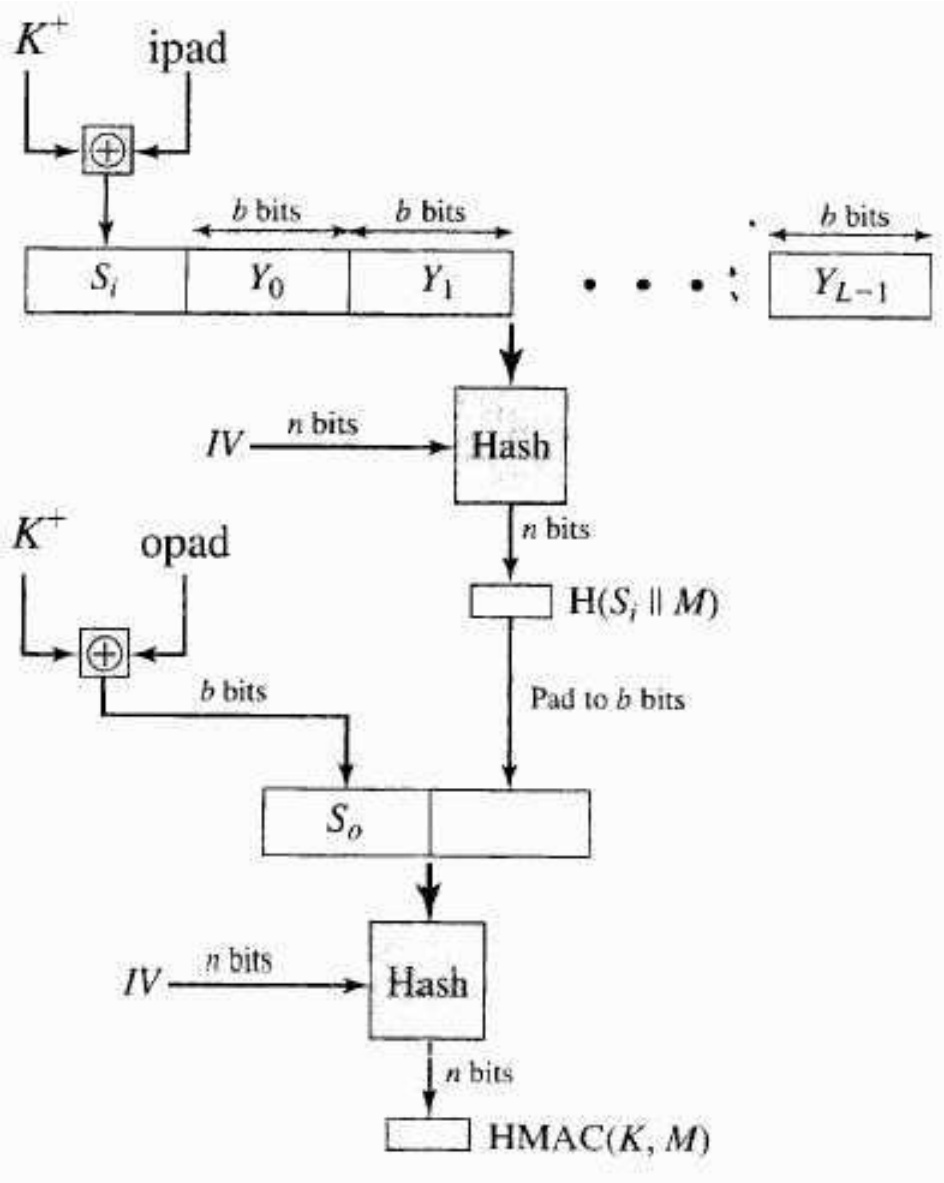
$\text{HMAC}_K(x) =$
$\quad \text{SHA-1}((K \oplus opad) \parallel \text{SHA-1}((K \oplus ipad) \parallel x))$.

DSA and ECDSA

Elliptic Curve
Digital Signature Algorithm

...
$s = k^{-1}(\text{SHA-1}(x) + mr) \pmod q$
...

HMAC

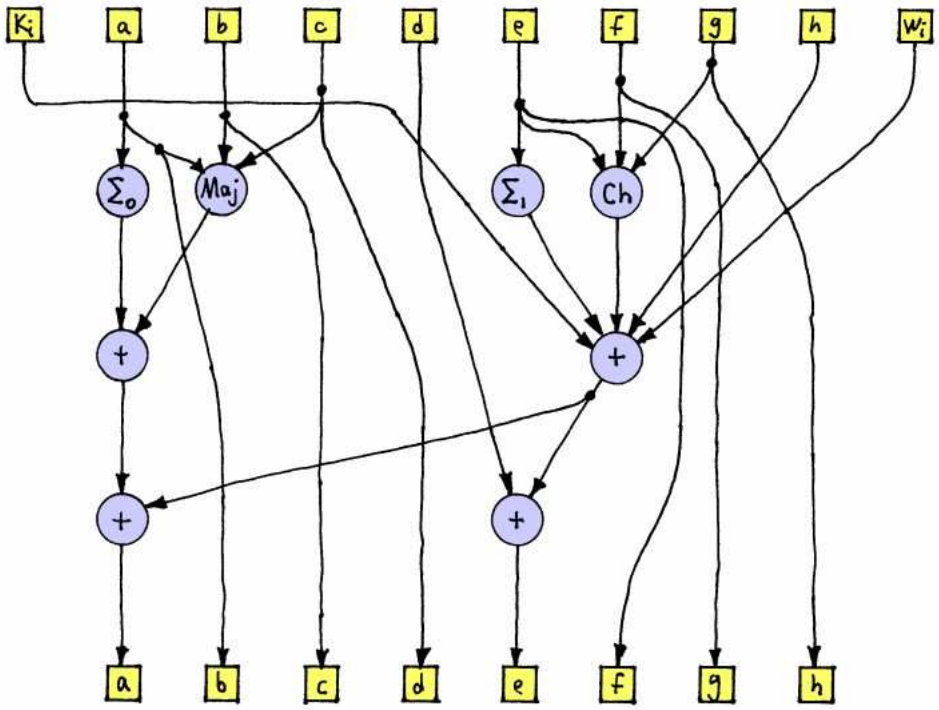


[Stallings]

SHA-2

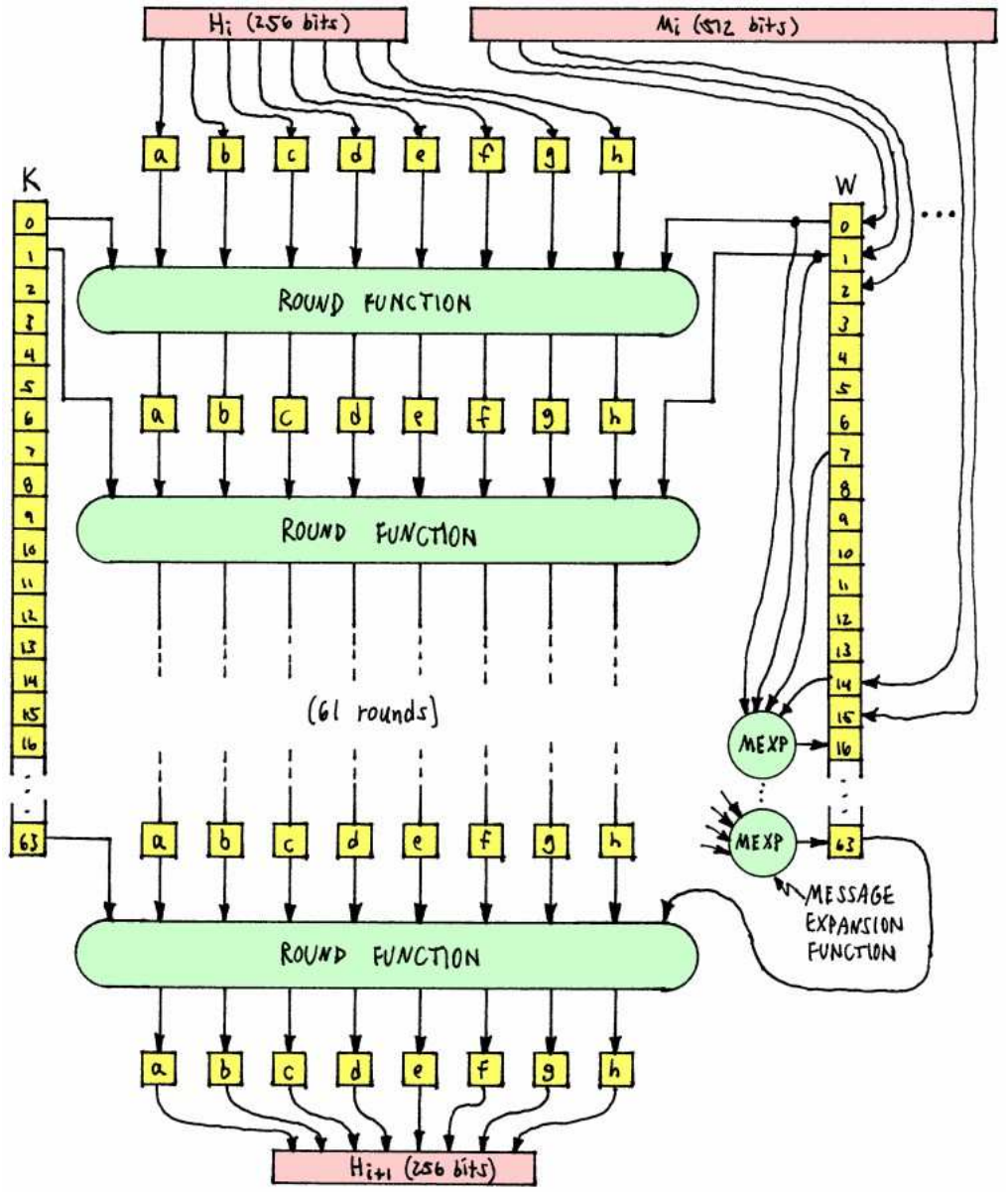
FIPS 180-2, 2002
Modes for 224, 256, 384 and 512 bits

each unit is a 32-bit word



one round of SHA-256 compression
(Alan Kaminsky)

SHA-2



structure of SHA-2 compression (Alan Kaminsky)

Theory needs your help

Theorem

(most of the time - in various scenarios)
Resistant compression implies resistant hash.
Resistant hash implies resistant compression.

Problem

Find a way to study collision resistant compression using complexity theory.
(more than in CRC Handbook 9.8.2 [13])

Characterize more formally:
"This n-to-m-bit compression needs essentially $2^{m/2}$ tests to find a collision and essentially 2^m effort to find any preimage."

People do it normally in random oracle model in probabilistic combinatorics language.

Birthday Attack

Counting fishes in a lake, Schnabel 1938 [16]

Theorem (Birthday Paradox)
Random sampling of q elements of the domain of size m will produce at least one collision with probability ϵ if

$$q \approx \sqrt{2m \ln \frac{1}{1-\epsilon}}$$

$q \approx 1.17\sqrt{m}$ for $\epsilon = 1/2$

$(m = 365, q = 23)$
Among 23 random people at least two of them have the same birthday with probability at least 1/2.

Birthday Attack

Proof. [19]
The probability of collision ϵ satisfies

$$1-\epsilon = \left(\frac{m-1}{m}\right)\left(\frac{m-2}{m}\right)\cdots\left(\frac{m-q+1}{m}\right)$$

$$= \prod_{i=1}^{q-1}\left(1-\frac{i}{m}\right) \approx \prod_{i=1}^{q-1} e^{\frac{-i}{m}}$$

$$= e^{\sum_{i=1}^{q-1}\frac{-i}{m}} = e^{\frac{-q(q-1)}{2m}}.$$

Hence

$$\frac{-q(q-1)}{2m} \approx \ln(1-\epsilon).$$

After ignoring -1 in $-q(q-1)$
the theorem follows.

Generic Attack

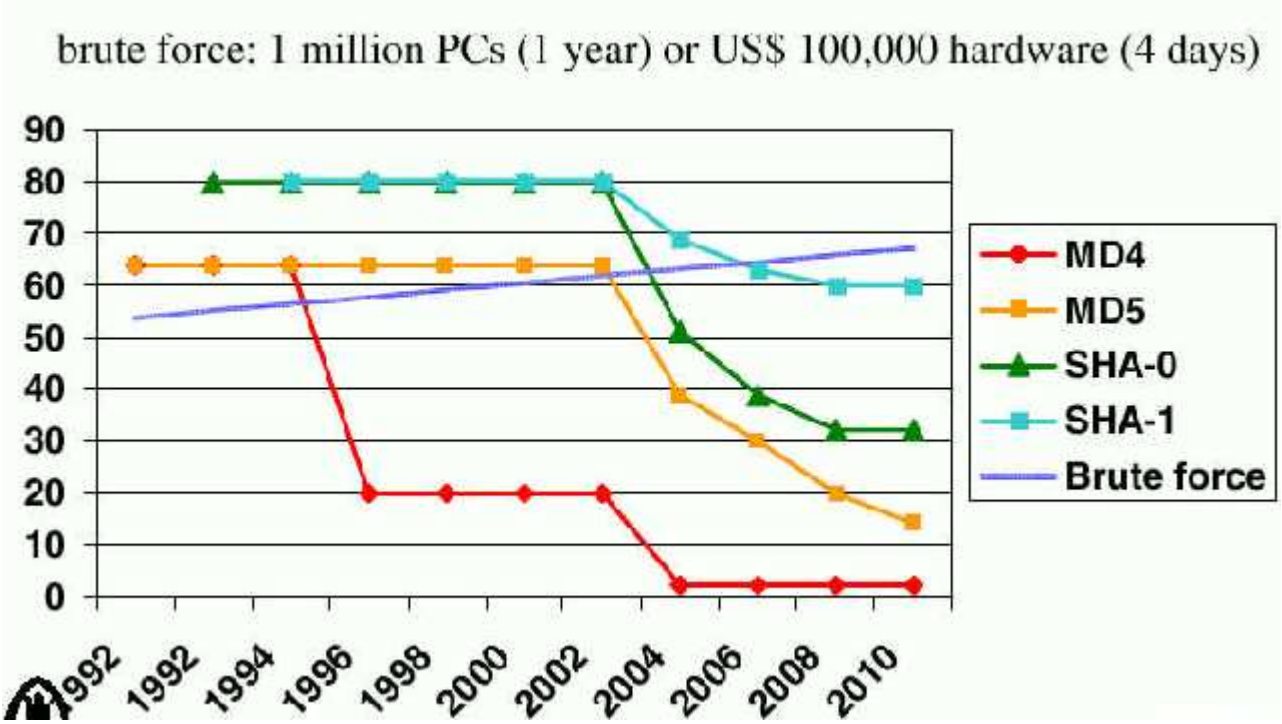
Sheer power of computing

- 1998, effort 2^{56}, **DES**$\downarrow$
- 2010, effort 2^{64} is possible
  $m = 2^{128}$, **MD5**$\downarrow$
- 2020, effort 2^{80} may be feasible
  $m = 2^{160}$, **SHA-1**$\downarrow$
- effort 2^{112}, won't be feasible for long

Conclusion. Requiring $m \geq 224$ for **AHS** seems reasonable (224 is the smallest multiple of 32 which prevents birthday attack well).

Preimage attacks are much more difficult, MD5 and SHA-1 are still strong.

Best potential attacks



Complexity of collision attacks

(Bart Preneel, 2010)

Chinese attacks on MD5/SHA-1

Wang, Yu, Yin (1995 - 2004 - 2006)
Probabilistic differential cryptanalysis found collisions in **MD5** and other hashes [22-24].

Collisions for full 80 rounds CAN
be found (still not done) with

$$2^{80} \rightarrow 2^{69} \rightarrow 2^{63}$$

SHA-1 computations.

Attacks on MD5/SHA-x (1996-2007)

- Collisions for **MD5** in seconds
- Collisions for **SHA-1** likely soon (?)
- **SHA-2** not (yet) threatened
- Preimages still almost hopeless

Attacks on MD5 (2005, 2008)

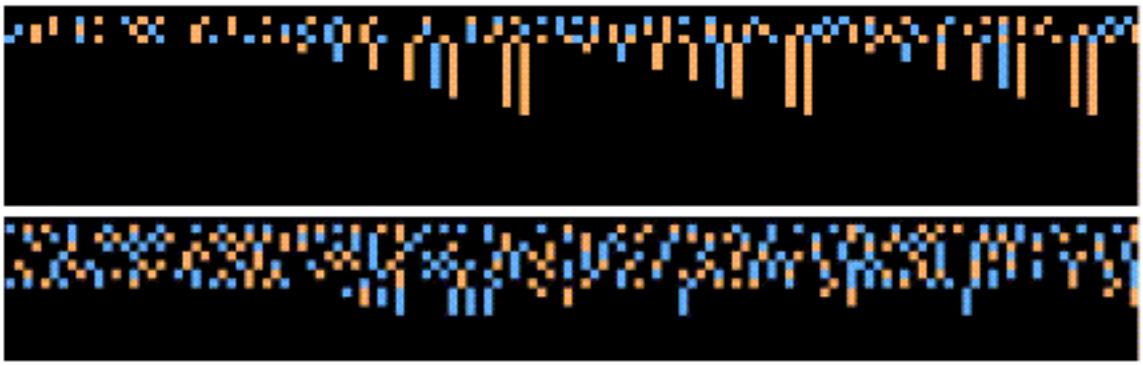
Faking X.509 certificates,
chosen prefix collisions -
Stevens, Lenstra, de Weger

For any P, P' find S, S' such that

$$MD5(P \parallel S) = MD5(P' \parallel S')$$

Consecutive blocks of suffixes target to eliminate specific bit differences.

Converging paths



time down, black the same, colors 0-1, two cases

.

False Alarm?

```
CRYPTO-GRAM, March 15, 2005
Bruce Schneier, <http://www.schneier.com>


                    SHA-1 Broken

SHA-1 has been broken.  Not a reduced-round version.
Not a simplified version.  The real thing.
```

Still waiting for the real thing ...

Collision search for SHA-1 using the distributed platform BOINC at GUT began August 8, 2007, ... abandoned May 12, 2009 due to lack of progress.

Recommendations

- no more MD5
- SHA-1 out by 2010 (NIST call in 2007)
- in each case analyze which type of resistance is needed, if preimage then SHA-1 may stay around longer

New hash needed

- Must be resistant to all known attacks
- Secure hashes from modular number theory are possible, but painfully slow
- Rather one parameterized hash than several special purpose hashes
- Take constants from math (like fractional part of $\sqrt[3]{p_i}$ in SHA-2, p_i the i-th prime).

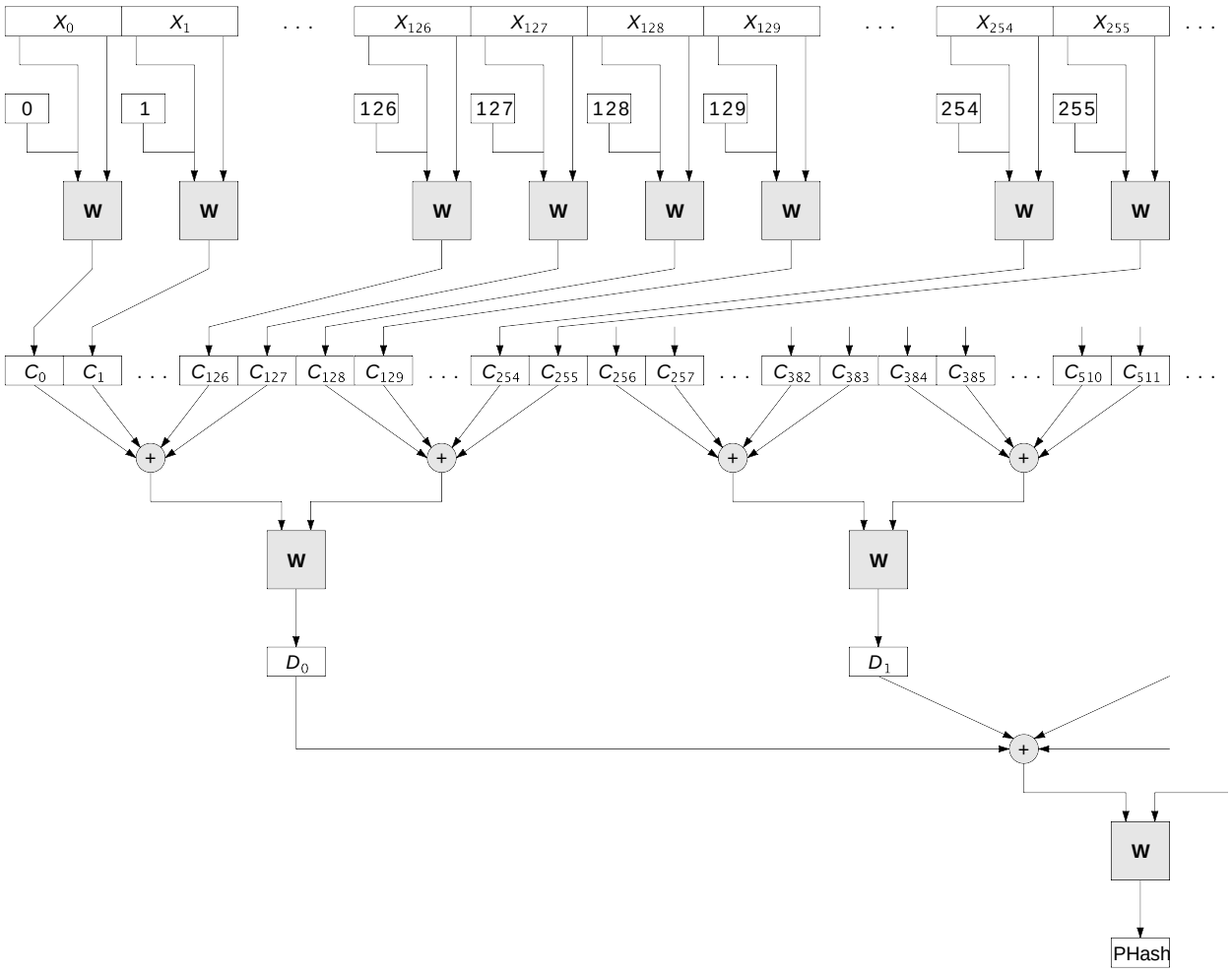
  DES/SHA-1 constants are a mystery.

Block cipher based design?

- Could be massively parallelized,
  NONE of the standard hashes can.
- Immune to linear/differential
  cryptanalysis, good confusion/diffusion
- Uses better understood components
- Incrementability. Small length-preserving
  message changes permit fast hash update.
- For software parallelizability use some tree
  result collection (MD6 or Skein).
- Tree-hashing: built-in or an afterthought
  mode of operation.

The dilemma of
SEQUENTIAL vs. PARALLELIZABLE

PHASH



Part of the PHASH computation tree with cipher W, $L = 512$ and $R = 128$

(KaminskyR 2008, [7])

NIST

SHA-3 Competition

2007-2012

SHA-3 acceptability requirements

2007 - call for proposals
2008 - submissions due October 31

- **A.1**
  Free worldwide.
- **A.2**
  Implementable on varied hardware and software platforms.
- **A.3**
  Must support 224, 256, 384 and 512 bit digests, and messages of at least up to 2^{64} bits.

SHA-3 submission requirements

- **B.1** Completely specified, rationale given for choices made, attack scenarios and resistance analysis, parameterizable
- **B.2** Source in ANSI C
- **B.3** Time and space requirements for hardware and software for 8-, 32- and 64-bit platforms
- **B.4** Documentation in English
- **B.5** Issued or pending patents
- **B.6** Self-evaluation

SHA-3 evaluation criteria

- **C.1**
  Security
- **C.2**
  Cost (time and space complexity)
- **C.3**
  Algorithm and implementation characteristics (flexibility, parameterizable, easy to parallelize, and ... simplicity)

Hash Function Candidate competition timeline [11]

- 2007, 1-3Q - minimum requirements
- 2008, October 31 - submissions deadline
- 2009, public comments period
  2Q - First **HFC** Conference [26]
  Leuven, BE, February 25-28
- 2010, public comments period
  3Q - Second **HFC** Conference [27]
  Santa Barbara, CA, August 23-24
  4Q - final round begins
- 2011, 4Q - end of public comments
- 2012, 1Q - Final **HFC** Conference
  2Q - select the winner
  3Q - draft documents
  public comments, tuning up
  4Q - **SHA-3** proposed to
  the Secretary of Commerce

SHA-3 Round-2/3 Candidates

November 2010, Round-2, 14 candidates [25]
December 2011, Round-3, 5 finalists [31]

What	Who	Where*
BLAKE	Aumasson+	CH
BMW	Knapskog	NO
CubeHash	Bernstein	IL
ECHO	Gilbert+	FR
Fugue	Jutla + IBM	NY
Grøstl	Knudsen+	DK
Hamsi	Küçük (f)	BE
JH	Wu	SG
Keccak	Daemen+ (m)	BE
Luffa	Watanabe+	JP
Shabal	Misarsky+	FR
SHAvite-3	Dunkelman+	Israel
SIMD	Leurent	FR
Skein	Schneier+	US/UK

*main countries/states of the main submitter

Hash Function Competition 2011+

- 2011, February, Round-3 NIST Report

  Big boss: Bill Burr
  Contents boss: John Kelsey

- Race is getting hot!

  Subscribe to hash-forum@nist.gov
  Email listproc@nist.gov, in the body

  subscribe hash-forum "your name"

- 2011

  become an expert in one finalist function

- 2012

  become an expert in the winner function

- 2013+

  help new SHA-3 to spread around

SHA-3 Zoo

Hosted in Austria by
the Graz University of Technology [29]

- It aims to provide an overview of design and cryptanalysis of all submissions.
- Informal collection of documents, attacks, papers and opinions on SHA-3 hash candidates.
- Extensive benchmarking of SHA-3 hardware implementations.
- NIST doesn't provide anything similar.

eBash

Hosted by Daniel Bernstein from
the University of Illinois at Chicago [30]

djb@cr.yp.to

- Part of a large system benchmarking cryptographic functions.
- By far the most extensive uniform software benchmarking of all SHA-3 candidates, on many architectures.
- In places questionable taste:

  ... suppose you want to submit your MD7 software to eBash, then do the following ...

Hints by John Kelsey, NIST, August 2010

Security

- no earth-shaking results
- will be used as KDF, PRF, PRNG
- new Haifa designs with salt, sponges
- do we count papers?
- rebound attacks
- do pseudocollisions matter?
- is there a "narrow pipe problem"?
- symmetries
- randomness

Hints by John Kelsey, NIST

What will matter for the final choice?

First call: security, cost, algorithm

Other factors

- diversity - not all eggs in one basket
- well interpreted number of papers
- 64-bit is becoming a standard
- SHA-3 will compete with SHA-2
- possible AES-chip-like solutions
- dual signatures
- updates to DSA and HMAC

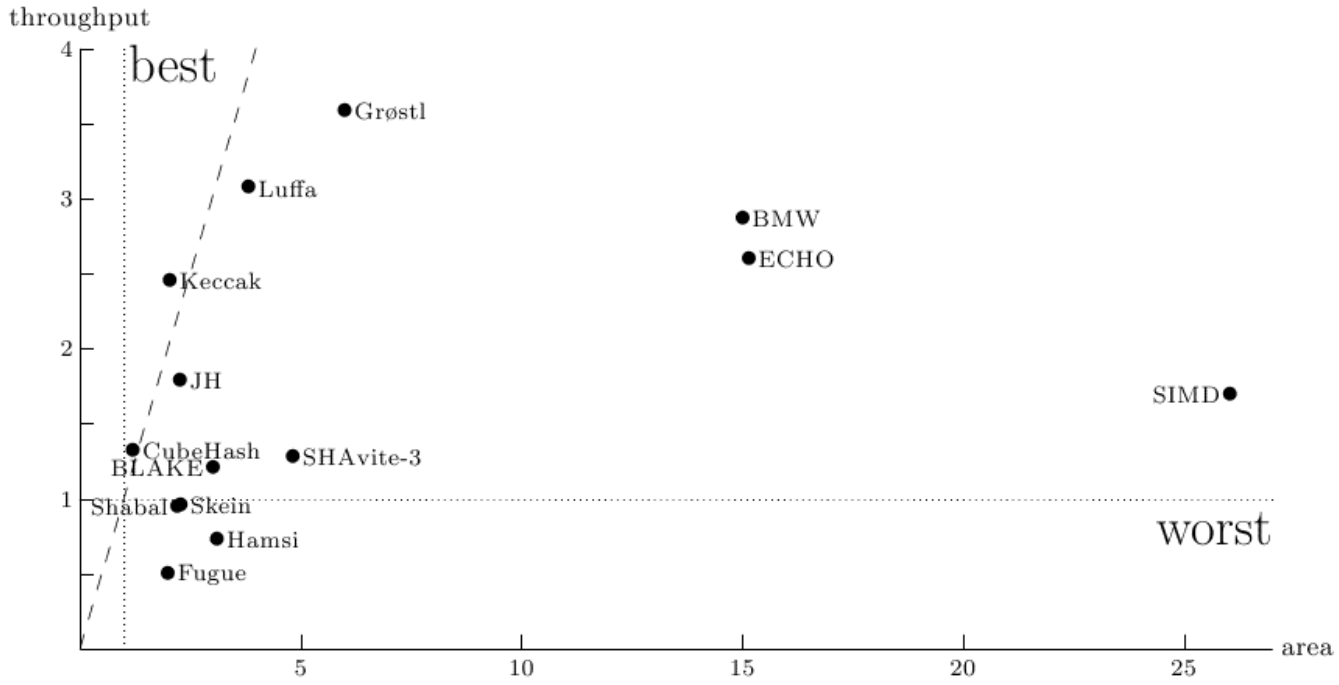
Hints by John Kelsey, NIST

Performance (narrow, wide, sponge)

Desktop	ASIC throughput	FPGA (ratio)
– BMW	– Luffa	– Keccak
– Shabal	– Keccak	– Cube
– Skein	– Cube	– Luffa
– SIMD	– Hamsi	– JH
– Luffa	– Blake	– Grostl
– Keccak	– Grostl	– Shabal
– Blake	– SHAvite3	– Blake
– JH	– JH	– Skein
– Cube	– BMW	– SHAvite3
– Grostl	– Shabal	– Fugue
– Hamsi	– Skein	– Hamsi
– Shavite3*	– Echo	– BMW
– Echo*	– Fugue	– Echo
– Fugue	– SIMD	– SIMD

Tricky Comparisons

Performance, Daniel Bernstein



area - FPGA slices relative to SHA-2/512

throughput - Gbit/sec relative to SHA-2/512

Hash Summaries

The following 14 slides contain summaries of the Round-2 hash function candidates, in alphabetical order, as announced by NIST in the report NISTIR 7620, September 2009 [25]

Status Report on the First Round of the SHA-3 Cryptographic Hash Algorithm Competition

Use SHA-3 Zoo website to access the documentation of all candidates

`http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo`

BLAKE

BLAKE is a HAIFA [24] hash algorithm whose compression function is based on using a keyed permutation in a Davies-Meyer-like construction [25]. The keyed permutation is based on the internals of the ChaCha [26] stream cipher, extended over a large state, and derives its nonlinearity from the overlap of modular addition and XOR operations. The most innovative part of BLAKE is its keyed permutation.

BLAKE's performance is quite good. It has modest memory requirements and appears to be suitable for a wide range of platforms.

The most significant cryptanalytic results against BLAKE are those that attacked the reduced round versions [27, 28] and appear to pose no threat to the design.

.

Blue Midnight Wish (BMW!)

BMW is a wide-pipe Merkle-Damgård hash construction [29] with an unconventional compression function, where the nonlinearity is derived from the overlap of modular addition and XOR operations. The most innovative parts of the design are the compression function construction and the design of the permutations; much of the design is novel and unique amongst the second-round candidates.

BMW has very good performance and appears to be suitable for a wide range of platforms. It has modest memory requirements.

The most serious cryptanalytic results against BMW are from impractical pseudo-collision attacks, and practical near-collision attacks [30]. These results raise questions about the security of the design.

.

CubeHash (5 dimensional cube)

CubeHash is a sponge-like hash algorithm that is based on a fixed permutation. The permutation is extremely simple and elegant, using only additions, XORs, and rotations in a fixed and simple pattern. All nonlinearity in the hash algorithm is derived from the overlap of modular additions and XOR operations. The novel part of CubeHash is the fixed permutation.

CubeHash has two tunable parameters, and its original proposed set of parameters led to extremely poor performance. A consistent problem in evaluating CubeHash has been uncertainty about those parameters. The designer has now proposed a set of parameters (16, 32) which provide very good performance with the use of SIMD instructions. CubeHash has relatively modest memory requirements and appears to be suitable for implementation on a wide range of platforms.

CubeHash has received a large amount of external analysis, probably due to the simplicity of its design and the flexibility offered to attackers by the two different tunable parameters. This made a strong argument in favor of advancing CubeHash to the next round–it appeared to be the best- understood of the candidates. The best-known attacks are ... (snip)

Of these, we find the semi-free-start collision and the symmetry properties to be the most troubling at this time. The CubeHash submission package identifies these issues and argues that exploiting these properties, given the large state and relatively small injection of message data before each permutation, is about as hard as a brute-force collision search. Relatively simple tweaks could also remove the symmetry properties from the algorithm.

.

Echo

ECHO is a wide-pipe hash algorithm following the HAIFA construction. Its compression function uses a keyed permutation; the counter and salt are used as the key, and the message and chaining value are used as inputs to the permutation. The permutation is quite novel, using a 2048-bit AES-like permutation in which the role of the substitution-box (S-box) [36] is played by a single AES round. The AES S-box provides all nonlinearity in this hash algorithm. By far the most interesting and unique part of this hash algorithm is the super-AES keyed permutation.

ECHO has acceptable performance on current high-end platforms, but requires hardware AES support to achieve impressive performance. ECHO requires a considerable amount of memory, but is expected to be otherwise suited for constrained platforms and hardware implementations.

The only known analytical result is a highly impractical distinguishing attack on the underlying permutation of a reduced round (7 out of 8) version of ECHO [37]. This attack appears to pose no threat to the overall ECHO design. We hope that the selection of ECHO as a second-round candidate will lead to more analysis of this unique hash algorithm design.

.

Fugue

Fugue is a variant of a sponge construction. Its compression function is based on a nonlinear shift register, maintaining a large state (thirty 32-bit words for the 256-bit version). The shift register incorporates a strengthened variant of the AES round function; all other operations are linear. Thus, all nonlinearity in this design is derived from the AES S-box. The most novel part of this design is the shift-register-based compression function, for which proofs and bounds on its differential probabilities were provided.

The performance of Fugue is acceptable, although the efficiency of current implementations is not particularly impressive. The use of SIMD instructions could yield better performance, although how much the performance could be improved is unknown at this time. Hardware support for AES may also improve its performance somewhat, but the impact of this will be limited, because of the use of the variant-AES round function. Fugue also maintains a large state, which may make it difficult to implement in constrained platforms.

We are aware of no external analysis of Fugue. We hope that its selection as a second-round candidate will lead to more analysis of this interesting hash algorithm.

.

Grøstl (austrian bigos)

Grøstl is a wide-pipe Merkle-Damgård hash construction with post-processing. Its compression function is a novel construction, involving two AES-like fixed permutations. All nonlinearity in the design is derived from the AES S-box. The most innovative part of Grøstl's design is the compression function construction.

Grøstl's performance is acceptable, but not especially impressive. Performance may be increased using hardware AES support, although the extent of these gains is unknown at this time. It has modest memory requirements.

The most serious attack on Grøstl is a semi-free-start collision attack on a reduced round variant that breaks 7 out of 10 rounds [37]. This attack raises some question about the security margin of the design.

.

Hamsi (anchovy - sardela)

Hamsi is a Merkle-Damgård hash construction with post-processing to block length-extension attacks. The compression function is constructed on a fixed permutation; the message is expanded using an error-detecting code to fill half the input block to the permutation, with the other half filled by the hash chaining value. The result is truncated and XORed with the hash chaining input, which is similar to the method used in Snefru. The Hamsi fixed permutation is a substitution-permutation network (SP network) [38], combining a single 4-bit S-box (taken from Serpent, and implemented using bit-slicing) with a linear mixing operation. All nonlinearity in the design is derived from that one S-box. The most innovative part of this design is the compression function construction; this is quite different from any other second-round candidate.

Hamsi requires the use of SIMD instructions to achieve acceptable performance in software. It has modest memory requirements.

The only results on Hamsi of which we are aware at present demonstrate low algebraic degree in the outputs of the compression function; whether this has any security implications for the hash algorithm is unclear.

.

JH

JH uses a novel construction, somewhat reminiscent of a sponge construction, to build a hash algorithm out of a single, large, fixed permutation. The fixed permutation is an SP network, combining two 4-bit S-boxes with a set of linear mixing operations and bit permutations. All nonlinearity in this design is derived from the S-boxes. The most innovative part of this design is the compression function construction, which XORs a 512-bit message block into the left half of the input of the fixed permutation, and then XORs the same message block into the right half of the output of the fixed permutation. The design of the fixed permutation is also new.

JH's performance is good, and has modest memory requirements. Unlike most second-round candidates, all output sizes of JH use the same function, but with different initial hash values and different amounts of truncation at the end.

The most serious cryptanalytic result on JH is a theoretical preimage attack on the 512-bit version [39, 40], which is barely cheaper than a brute force attack. As this attack does not appear to threaten the design, it does not concern us. However, the compression function construction of JH is not well-understood, and the submitter did not provide a great deal of analysis of this construction.

.

Keccak

Keccak follows the sponge construction and uses a large fixed permutation. The permutation can be seen as an SP-network with 5-bit wide S-boxes, or as a combination of a linear mixing operation and a very simple nonlinear mixing operation. The construction of the permutation is the most innovative part of the Keccak design.

Keccak performs well on high-end platforms and is expected to perform well across a wide range of platforms, as well as in dedicated hardware. The hash algorithm has modest memory requirements. Unlike most second-round candidates, Keccak uses a single design for all hash outputs.

The most significant cryptanalytic result on Keccak of which we are aware are distinguishing attacks against reduced round versions of the permutation [41]; however, these do not appear to threaten the security of the hash algorithm.

.

Luffa

Luffa is a variant of the sponge construction, using a linear mixing operation and several fixed 256-bit permutations in place of a single wider permutation. The fixed permutations are SP networks, which combine linear mixing operations with a single 4-bit wide S-box, and this S-box provides all nonlinearity in the design. The most innovative part of Luffa is the sponge construction.

Luffa provides good performance on high-end platforms and appears suitable for a wide variety of platforms. Substantial parts of the design are the same for different output sizes.

The most significant cryptanalytic result on Luffa of which we are aware is a pseudo-preimage attack on the squeezing steps of Luffa-384 and Luffa-512 [42]. This is a consequence of the structure of Luffa (XORing 256-bit permutation results together to generate an output) and does not appear to lead to a threat to the security of the hash algorithm.

.

Shabal

Shabal is a hash algorithm that is constructed using a novel chaining mode, which can be seen as a variant of a wide-pipe Merkle-Damgrd hash construction. Its compression function is similarly innovative, based on a feedback shift register construction that combines the several inputs provided by the chaining mode efficiently. Nonlinearity in Shabal is derived from the overlap of XOR, modular addition, and bitwise AND operations. The entire design is very different from any other second-round candidate and has many new ideas.

Shabal's performance is good. However, it requires more working memory than most of the second-round candidates. The same internal function is used for all output sizes of Shabal.

Several observations regarding Shabal's compression function have been published, including powerful distinguishing attacks on the keyed permutation that forms its core [43, 44]. However, the attacks have not been claimed to directly threaten the security of the hash algorithm, and the submitters have modified the security proof of their chaining mode to require weaker assumptions that are not invalidated by the attacks. Nonetheless, the distinguishing attacks remain a concern.

.

SHAvite-3

SHAvite-3 is a HAIFA hash algorithm. The compression function is a keyed permutation that is used in the Davies-Meyer construction. The keyed permutation is a balanced Feistel network [45] (for the 256-bit case) or a pair of interwoven balanced Feistel networks (for the 512-bit case), with the F-function constructed from the AES round function. All nonlinearity in the whole construction relies upon the AES S-box. The most innovative part of the design is the decision to construct the keyed permutation in this way; however, SHAvite-3 is a conservative design, with relatively little new about it.

SHAvite-3 has acceptable performance on current high-end platforms, but hardware AES support could have a large impact on its performance, since the AES round function is used directly. Shavite-3 has modest memory requirements.

The most serious cryptanalytic results on SHAvite-3 are large numbers of zero pseudo-preimages for the compression function [46, 47]. However, these require the use of a specific counter value, which is used only for the final message block, where the pseudo-preimages apparently cannot be constructed. While this result appears to pose no direct threat to SHAvite-3, this unexpected property of the compression function is a source of concern, especially given the fact that the offending counter value is used in Shavite-3's current construction.

.

SIMD

SIMD is a wide-pipe Merkle-Damgrd hash construction. Its compression function is constructed from a keyed permutation, in a variant of the Davies-Meyer construction. The keyed permutation is the most innovative part of this design; it uses a linear code with provable diffusion properties as the ”key schedule,” and uses four unbalanced Feistel networks that are reminiscent of the MD4 [48] and MD5 [49] round functions in an interleaved way as its round function. The nonlinearity in this design is provided by the overlap of modular addition and XOR operations and from the bitwise nonlinear functions.

SIMD can achieve very good performance, but only when vector instructions are available. It also has relatively large memory requirements, which raises concerns about its suitability for constrained platforms.

At present, we are aware of no analysis that raises questions about SIMD’s security.

.

Skein (flock of geese, oblong ball of yarn)

Skein is a variant of a Merkle-Damgård hash construction that is based on a novel tweakable block cipher and chaining mode. The compression function is used in a variant of the Matyas-Meyer-Oseas [25] construction that is appropriate for a tweakable block cipher, and the submission provides proofs that the construction is secure, assuming a secure compression function and tweakable block cipher. The block cipher (called "Threefish") is constructed from a large number of very simple rounds and uses only three 64-bit operations–modular addition, bitwise XORing, and rotation. All nonlinearity in the hash algorithm is provided by the overlap of modular addition and XOR operations. The most innovative parts of Skein are the Threefish block cipher and the chaining mode.

Skein has good performance on high-end platforms, particularly in 64-bit mode, and is also expected to perform well in constrained platforms and in dedicated hardware implementations. It has modest memory requirements and benefits from the pipelining used in modern processors.

The most significant cryptanalytic results on Skein are distinguishing attacks against reduced- round versions of Threefish; these do not appear to pose a threat to the full hash algorithm at this time.

.

References from hash summaries point to those listed in the NIST report [25].

References-1

1. Mihir Bellare, Roch Guérin and Philip Rogaway, XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions, *LNCS* **963**, (1995) 15–28.
2. Mihir Bellare and Daniele Micciancio, A new paradigm for collision-free hashing: incrementality at reduced cost, *LNCS* **1233**, (1997) 163–192.
3. Daniel J. Bernstein, ChaCha, a variant of Salsa20, Jan 28, 2008, `http://cr.yp.to/chacha/chacha-20080128.pdf`
4. Eli Biham, Recent Advances in Hash Functions: The Way to Go, slides from various conference presentations, 2005.
5. Eli Biham and Orr Dunkelman, A Framework for Iterative Hash Functions – HAIFA, NIST Second Cryptographic Hash Workshop, Santa Barbara, August 2006, `http://csrc.nist.gov/groups/ST/hash/documents/DUNKELMAN_NIST3.pdf`
6. John Black, Martin Cochran and Trevor Highland, A Study of the MD5 Attacks: Insights and Improvements, *LNCS* **4047**, (2006) 262–277.
7. Alan Kaminsky and Stanisław Radziszowski, A Case for a Parallelizable Hash, *Proceedings of MILCOM'2008*, San Diego, CA.
8. Vlastimil Klima, posts at the NIST Cryptographic Hash Project forum, 2007, `hash-forum@nist.gov`.
9. Joel Lathrop, Cube Attacks on Cryptographic Hash Functions, May 21, 2009, `http://www.cs.rit.edu/~jal6806/thesis/thesis.pdf`

References-2

10. National Institute of Standards and Technology, Cryptographic Toolkit, Secure Hashing, specification of SHA-2 standards, 2002, http://csrc.nist.gov/CryptoToolkit/tkhash.html
11. National Institute of Standards and Technology, Tentative Timeline of the Development of New Hash Functions, 2007, http://www.csrc.nist.gov/pki/HashWorkshop/timeline.html
12. MD5, http://en.wikipedia.org/wiki/MD5
13. Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, *CRC Handbook of Applied Cryptography*, CRC Press 1996. http://www.cacr.math.uwaterloo.ca/hac
14. Vincent Rijmen and Paulo S. L. M. Barreto, The WHIRLPOOL Hash Function (2003), http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html
15. Akashi Satoh, Hardware Architecture and Cost Estimates for Breaking SHA-1, *LNCS* **3650**, (2005) 259–273.
16. Zoe Emily Schnabel, The Estimation of the Total Fish Population of a Lake, *American Mathematical Monthly*, **45** (6), (1938) 348–352.
17. Bruce Schneier, *Applied Cryptography*, second edition, John Wiley & Sons, 1996.
18. William Stallings, The Whirlpool Secure Hash Function, *Cryptologia*, **30**, (2006) 55–67.

References-3

19. Douglas R. Stinson, *Cryptography: Theory and Practice*, third edition, CRC Press 2006.
20. Douglas R. Stinson, Some observations on the theory of cryptographic hash functions, *Designs, Codes and Cryptography* **38**, (2006) 259–277.
21. Janusz Stokłosa, Ochrona danych i zabezpieczenia w systemach teleinformatycznych, Wydawnictwo Politechniki Poznańskiej, 2005.
22. Xiaoyun Wang and Hongbo Yu, How to Break MD5 and Other Hash Functions, *LNCS* **3494**, (2005) 19–35.
23. Xiaoyun Wang, Hongbo Yu and Yiqun Lisa Yin, Efficient Collision Search Attacks on SHA-0, *LNCS* **3621**, (2005) 1–16.
24. Xiaoyun Wang, Yiqun Lisa Yin and Hongbo Yu, Finding Collisions in the Full SHA-1, *LNCS* **3621**, (2005) 17–36.

Prediction (August 2010)

Keccak will win

Meta References

Sources to other SHA-3 references:

25. NIST report NISTIR 7620, Status Report on the First Round of the SHA-3 Cryptographic Hash Algorithm Competition, September 2009, http://csrc.nist.gov/groups/ST/hash/sha-3

26. The First SHA-3 Candidate Conference, K.U. Leuven, Belgium, Feb. 25-28, 2009, http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/Feb2009

27. The Second SHA-3 Candidate Conference, Santa Barbara, CA, Aug. 23-24, 2010, http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010

28. NIST Hash Forum http://csrc.nist.gov/groups/ST/hash/email_list.html

29. ECRYPT SHA-3 Zoo http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo

30. eBASH: ECRYPT Benchmarking of All Submitted Hashes, http://bench.cr.yp.to/ebash.html

31. NIST report NISTIR 7764, Status Report on the Second Round of the SHA-3 Cryptographic Hash Algorithm Competition, February 2011, http://csrc.nist.gov/groups/ST/hash/sha-3

Revisions

Revision #1, March 2007
Revision #2, August 2008
Revision #3, November 2010
Revision #4, March 2011