The Hitchhiker's Guide to the SHA-3 Competition

Orr Dunkelman

Computer Science Department
University of Haifa

4 July, 2012

Outline

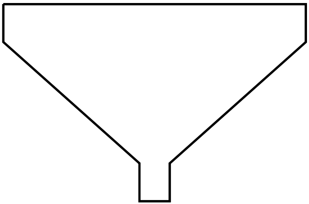
Outline

What is a Hash Function?

[DH76] There is, however, a modification which eliminates the expansion problem when N is roughly a megabit or more. Let g be a one-way mapping from binary N-space to binary n-space where n is approximately 50. Take the N bit message m and operate on it with g to obtain the n bit vector m'. Then use the previous scheme to send m'...

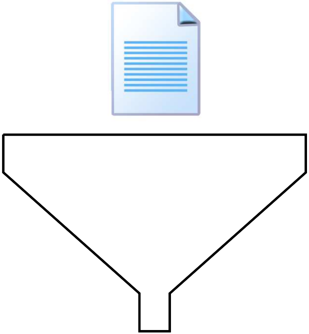
What is a Hash Function? (cont.)

- (Cryptographic) Hash Functions are means to **securely** reduce a string *m* of arbitrarily length into a fixed-length digest.



What is a Hash Function? (cont.)

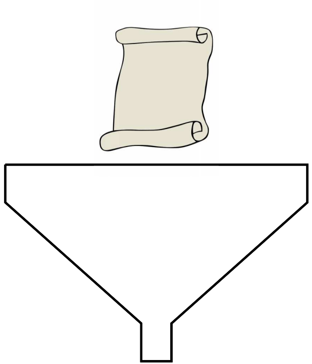
- (Cryptographic) Hash Functions are means to **securely** reduce a string m of arbitrarily length into a fixed-length digest.



0x256C795AC8222D4F90EA836D69687B68

What is a Hash Function? (cont.)

- (Cryptographic) Hash Functions are means to **securely** reduce a string m of arbitrarily length into a fixed-length digest.



0x6CA0B3C905C0DDABA60E08BFA9A9B8BD

What is a Hash Function? (cont.)

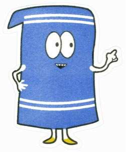
- The main problem is the definition of securely.
- For signature schemes, two basic requirements exist:
  1. Second preimage resistance: given x, it is hard to find x' s.t. $h(x) = h(x')$.
  2. Collision resistance: it is hard to find x_1, x_2 s.t. $h(x_1) = h(x_2)$.

What is a Hash Function? (cont.)

- The main problem is the definition of securely.
- For signature schemes, three basic requirements exist:
  1. Preimage resistance: given $y = h(x)$, it is hard to find x (or x', s.t., $h(x') = y$).
  2. Second preimage resistance: given x, it is hard to find x' s.t. $h(x) = h(x')$.
  3. Collision resistance: it is hard to find x_1, x_2 s.t. $h(x_1) = h(x_2)$.

What is a Hash Function? (cont.)

- Hash functions were quickly adopted in other places:
  - Password files (storing $h(pwd, salt)$ instead of pwd).
  - Bit commitments schemes (commit — $h(b, r)$, reveal — b, r).
  - Key derivation functions (take $k = h(g^{xy} \bmod p)$).
  - MACs (long story).
  - Tags of files (to detect changes).
  - Inside PRNGs.
  - Inside protocols (used in many "imaginative" ways).
  - ...



What is a Hash Function? (cont.)

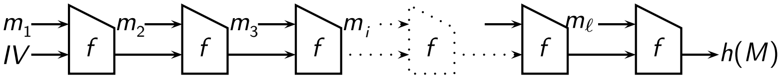
The Hitch Hiker's Guide to the Galaxy has a few things to say on the subject of hash functions.
A hash function, it says, is about the most massively useful thing a cryptographer can have. Partly it has great practical value — you can use it to replace random oracles in real protocols when you need them; you can use them to make signatures faster; you can use it along with salts to have better password files; you can commit to bits using it; you can derive keys using it; produce pseudo random numbers using it; authenticate data with it, and of course, just hash the data when you need a digest.
More importantly, a hash function has immense psychological value. For some reason, if a strag (strag: non-cryptographer) discovers that a cryptographer has his hash function with him, he will automatically assume that he is also in possession of a symmetric-key encryption, a public-key encryption, a voting protocol, a zero-knowledge protocol, etc. etc. Furthermore, the strag will then happily implement for the cryptographer any of these or a dozen other protocols that the cryptographer is too "busy" do himself. What the strag will think is that any cryptographer who can design protocols, follow bits, avoid differentials, and SAT solvers, and still knows where his hash function is is clearly a man to be reckoned with.

The MD/SHA Family

- Started with Rivest's MD4.
- Following a few cryptanalytic attempts, was upgraded to MD5.
- MD5, also known to many as **md5sum** generate tags of 128 bits.
- Became very popular given its high speed, alleged security, and lack of true competition...
- Later, it was used as the basis for the SHA-0 and SHA-1 hash functions.

The MD5 Hash Function

- To hash a message M the following steps are performed:
  1. M is padded with '1' as many 0's as needed (up to 512) and the original length of M encoded in 64 bits, such that the length of the padded message $pad(M)$ is divisible by 512.
  2. $pad(M)$ is divided into ℓ blocks of 512 bits, i.e., $pad(M) = m_1, m_2, \ldots, m_\ell$.
  3. The 128-bit chaining value h_0 is initialized.
  4. For $i = 1, 2, \ldots, \ell,\ h_i = H(h_{i-1}, m_i)$ (the compression function is applied).
  5. The output is h_ℓ



The MD5 IV

- The internal state (chaining value) of MD5, is treated as four words of 32-bit each: A, B, C, D.
- The initial value h_0 is:

$$
\begin{aligned}
A &= \texttt{0x67452301} \\
B &= \texttt{0xEFCDAB89} \\
C &= \texttt{0x98BADCFE} \\
D &= \texttt{0x10325476}
\end{aligned}
$$

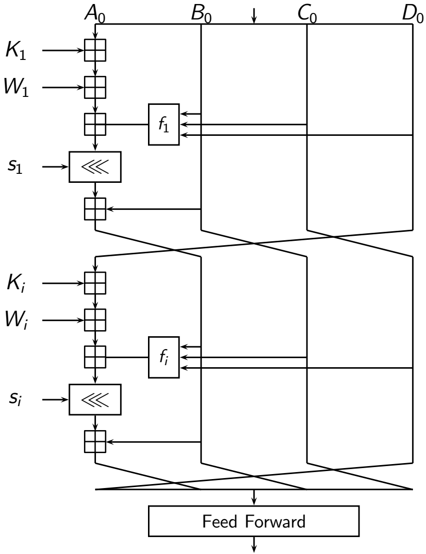
(this initial value is given in a little-endian manner)

The MD5 Compression Function

- Let $h_{i-1} = (A_0, B_0, C_0, D_0)$.
- Let the message block be $M_i = (W_0, W_1, \ldots, W_{15})$
- For $i = 0, 1, \ldots, 63$:
  1. $D_{i+1} \leftarrow C_i$
  2. $C_{i+1} \leftarrow B_i$
  3. $B_{i+1} \leftarrow B_i + (A_i + F_i(B_i, C_i, D_i) + K_i + W_{g(i)}) \lll s_i$
  4. $A_{i+1} \leftarrow D_i$
- $h_i \leftarrow (A_0 + A_{64}, B_0 + B_{64}, C_0 + C_{64}, D_0 + D_{64})$.

All additions are modulo 2^{32}, and $\lll$ stands for rotation to the left.

The MD5 Compression Function



The MD5 Compression Function (cont.)

- Each round, a different message word is used, a different round constant is used, and a different function and rotations:

$$
\begin{array}{llll}
0 \le t \le 15: & f_t(X,Y,Z) = XY \vee (\neg X)Z & g(t) = t \\
16 \le t \le 31: & f_t(X,Y,Z) = XY \vee (\neg Z)X & g(t) = (5 \cdot t + 1) \bmod 16 \\
32 \le t \le 47: & f_t(X,Y,Z) = X \oplus Y \oplus Z & g(t) = (3 \cdot t) \bmod 16 \\
48 \le t \le 63: & f_t(X,Y,Z) = Y \oplus (X \vee \neg Z) & g(t) = (7 \cdot t) \bmod 16
\end{array}
$$

The set of constants K_i is based on sin:

$$K_i = \lfloor |\sin(i+1)| \cdot 2^{32} \rfloor$$

The MD5 Compression Function (cont.)

The rotation constants (s_i) are

Rotation Constants

7	12	17	22	7	12	17	22	7	12	17	22	7	12	17	22
5	9	14	20	5	9	14	20	5	9	14	20	5	9	14	20
4	11	16	23	4	11	16	23	4	11	16	23	4	11	16	23
6	10	15	21	6	10	15	21	6	10	15	21	6	10	15	21

The Shortcomings of the MD/SHA Family

- First of all, these hash functions are Merkle-Damgård ones, susceptible all the attacks on such hash functions.

The Shortcomings of the MD/SHA Family

- First of all, these hash functions are Merkle-Damgård ones, susceptible all the attacks on such hash functions.
- Most of the nonlinearity is introduced either in addition or locally (bitwise operations).

The Shortcomings of the MD/SHA Family

- First of all, these hash functions are Merkle-Damgård ones, susceptible all the attacks on such hash functions.
- Most of the nonlinearity is introduced either in addition or locally (bitwise operations).
- An immediate consequence — easy to approximate the algorithm as a linear.

The Shortcomings of the MD/SHA Family

- First of all, these hash functions are Merkle-Damgård ones, susceptible all the attacks on such hash functions.
- Most of the nonlinearity is introduced either in addition or locally (bitwise operations).
- An immediate consequence — easy to approximate the algorithm as a linear.
- Easy to define the conditions when the approximation holds.

The Shortcomings of the MD/SHA Family

- First of all, these hash functions are Merkle-Damgård ones, susceptible all the attacks on such hash functions.
- Most of the nonlinearity is introduced either in addition or locally (bitwise operations).
- An immediate consequence — easy to approximate the algorithm as a linear.
- Easy to define the conditions when the approximation holds.
- Along with a simple message expansion, relatively slow diffusion, and many cool techniques* one can offer differentials with high probability that lead to collisions.

*multi-block collision, neutral bits, message modification, advance message modification, generalized differentials, amplified boomerang attack.

A(n Extremely) Short History of Hash Functions

1976 Diffie and Hellman suggest to use hash functions to make digital signatures shorter.

1979 Salted passwords for UNIX (Morris and Thompson).

1983/4 Davies/Meyer introduce Davies-Meyer.

1986 Fiat and Shamir use random oracles.

1989 Merkle and Damgård present the Merkle-Damgård hash function.

1990 MD4 is introduced by Rivest.

1990 N-Hash is almost broken by differential cryptanalysis.

1992 MD5 is introduced by Rivest.

1993 Preneel, Govaerts, Vandewalle study block-cipher based hashing.

1993 Bellare & Rogaway formally introduce random oracles.

A(n Extremely) Short History of Hash Functions

1993 SHA-0 is introduced.

1995 SHA-1 is introduced.

1997 SHA-0 is broken by Chabaud and Joux.

1999 Dean's long second preimage attack on Merkle-Damgård.

2001 SHA-2 is introduced.

2004 Joux's multicollision attack.

2004 Wang introduces attacks on MD4, MD5.

2005 Collision attacks on SHA-0 and SHA-1.

2006 Kelsey & Kohno's herding attack.

2007 Preimage attacks on reduced-round SHA-1.

2007 SHA-1 Collision BOINC project starts.

The State of Affairs in 2007

Hash	Collisions	2nd Preimage	Preimage
MD4	By hand	—	—
MD5	2^{24}	—	—
SHA-0 (80 rounds)	2^{39}	up to 50 rounds	up to 50 rounds
SHA-1 (80 rounds)	2^{63}–2^{69}	up to 45 rounds	up to 45 rounds
SHA-256 (64 rounds)	up to 24 rounds	—	—
SHA-512 (80 rounds)	up to 24 rounds	—	—

Our Options

Our Options

Outline

The First Phase of the SHA-3 Competition

- January 2007: NIST announces that a SHA-3 competition will be held. Asks the public for comments.
- November 2007: NIST publishes the official rules of the competition.
- August 2008: First submission deadline.
- October 2008: The *real* deadline.

The First Phase of the SHA-3 Competition

- January 2007: NIST announces that a SHA-3 competition will be held. Asks the public for comments.
- November 2007: NIST publishes the official rules of the competition.
- August 2008: First submission deadline.
- October 2008: The *real* deadline.
- 64 candidates were submitted.
- NIST went over them, and identified 51 which satisfied a minimal set of requirements.

The First Phase of the SHA-3 Competition

- January 2007: NIST announces that a SHA-3 competition will be held. Asks the public for comments.
- November 2007: NIST publishes the official rules of the competition.
- August 2008: First submission deadline.
- October 2008: The *real* deadline.
- 64 candidates were submitted.
- NIST went over them, and identified 51 which satisfied a minimal set of requirements.

Let the games begin!

Welcome to the Wild West

Candidate	Candidate	Candidate	Candidate	Candidate
Abacus	ARIRANG	AURORA	Blake	Blender
BMW	Boole	Cheeta	CHI	CRUNCH
CubeHash	DCH	Dynamic SHA	Dynamic SHA2	ECHO
ECOH	EDON-R	Enrupt	ESSENCE	FSB
Fugue	Grøstl	Hamsi	JH	KECCAK
Khichidi-1	Lane	Luffa	LUX	MCSSHA-3
MD6	MeshHash	NaSHA	NKS2D	SANDstorm
Sarmal	Sgáil	Shabal	SHAMATA	SIMD
Skein	SHAvite-3	Spectral Hash	StreamHash	SWIFFTX
Tangle	TIB3	Twister	Vortex	WaMM
		Waterfall		

What a Break is?

- There is an ongoing debate what a broken hash function is.

What a Break is?

- There is an ongoing debate what a broken hash function is. Even from the theoretical point of view.

What a Break is?

- There is an ongoing debate what a broken hash function is. Even from the theoretical point of view.
  1. Practical.
  2. Close to Practical.
  3. (Time, Memory) is better then for generic attacks (e.g., time-memory tradeoff attacks, birthday attack).
  4. Time $\times$ Memory is less than required in generic attacks.
  5. Money for finding $\{$collision, second preimage, preimage$\}$ in a given time frame is less than for generic attacks.

What NIST Did?

- At that point NIST had 27 broken submissions out of 51.
- They discarded the broken ones (24 left).
- MD6 was withdrawn (23 left).

What NIST Did?

- At that point NIST had 27 broken submissions out of 51.
- They discarded the broken ones (24 left).
- MD6 was withdrawn (23 left).
- To further reduce the list of candidates to about 15, they decided to *not select candidates* which "has no real chance to be selected as SHA-3".

What NIST Did?

- At that point NIST had 27 broken submissions out of 51.
- They discarded the broken ones (24 left).
- MD6 was withdrawn (23 left).
- To further reduce the list of candidates to about 15, they decided to *not select candidates* which "has no real chance to be selected as SHA-3".
- NIST allowed tweaks (small changes which do not invalidate previous analysis).
- And in July 2009 announced the second round candidates.

Outline

Welcome to the Second Round

Candidate	Candidate	Candidate	Candidate	Candidate
Blake	BMW	CubeHash	ECHO	Fugue
Grøstl	Hamsi	JH	KECCAK	Luffa
Shabal	SHAvite-3	SIMD	Skein	

The Second Round Process

- During the second round, all 14 candidates were analyzed.
- Hamsi was the only one that was (marginally) broken.
- Distinguishing properties were reported for the full compression functions of BMW, CubeHash, Grøstl, KECCAK, Luffa, Shabal, SHAvite-3, and SIMD.
- These attacks do not scale to the full hash function (at the moment).
- Attacks on almost the full compression functions of ECHO, Fugue, and Skein were also reported.
- JH and Blake were also analyzed.

The Second Round Process

- During the second round, all 14 candidates were analyzed.
- Hamsi was the only one that was (marginally) broken.
- Distinguishing properties were reported for the full compression functions of BMW, CubeHash, Grøstl, KECCAK, Luffa, Shabal, SHAvite-3, and SIMD.
- These attacks do not scale to the full hash function (at the moment).
- Attacks on almost the full compression functions of ECHO, Fugue, and Skein were also reported.
- JH and Blake were also analyzed.
- Some primitives received less cryptanalytic attention.

The Story of Shabal

- Shabal was submitted with a security proof (compression function is secure $\Rightarrow$ hash function is secure).

The Story of Shabal

- Shabal was submitted with a security proof (compression function is secure $\Rightarrow$ hash function is secure).
- Shabal's compression function can be easily distinguished.

The Story of Shabal

- Shabal was submitted with a security proof (compression function is secure $\Rightarrow$ hash function is secure).
- Shabal's compression function can be easily distinguished.
- Shabal's team fixed the proof.

The Story of Shabal

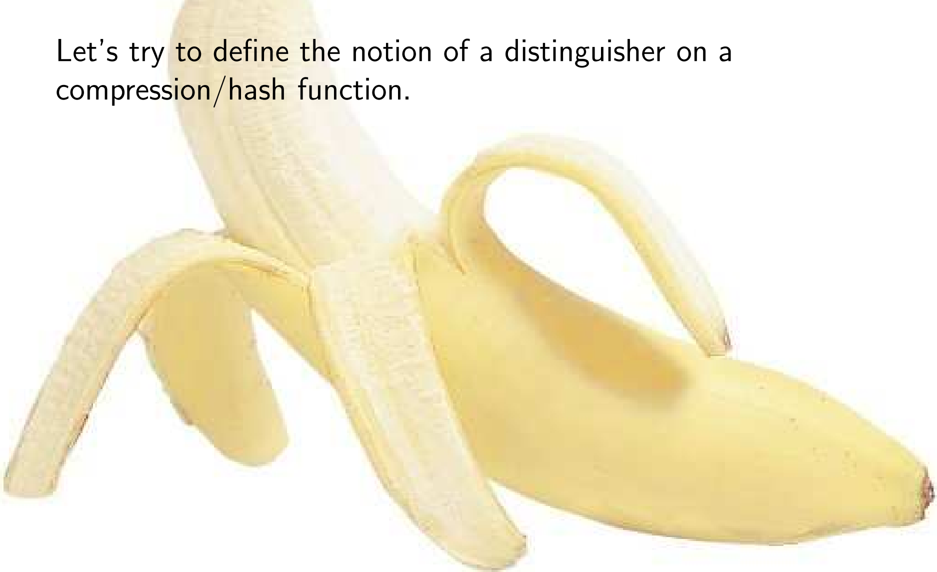
- Shabal was submitted with a security proof (compression function is secure $\Rightarrow$ hash function is secure).
- Shabal's compression function can be easily distinguished.
- Shabal's team fixed the proof.
- A new distinguishing attack on Shabal* is introduced. Where Shabal* is secure according to the new proof...

The Story of Shabal

- Shabal was submitted with a security proof (compression function is secure $\Rightarrow$ hash function is secure).
- Shabal's compression function can be easily distinguished.
- Shabal's team fixed the proof.
- A new distinguishing attack on Shabal* is introduced. Where Shabal* is secure according to the new proof...
- Luckily for Shabal — not so easy to get to Shabal*.

To Distinguish or Not to Distinguish

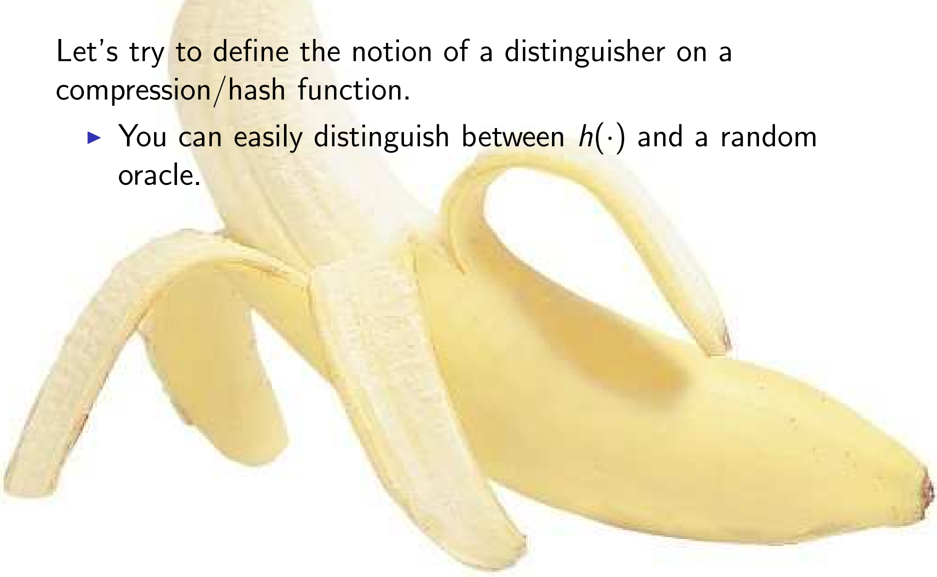
Let's try to define the notion of a distinguisher on a compression/hash function.



To Distinguish or Not to Distinguish

Let's try to define the notion of a distinguisher on a compression/hash function.

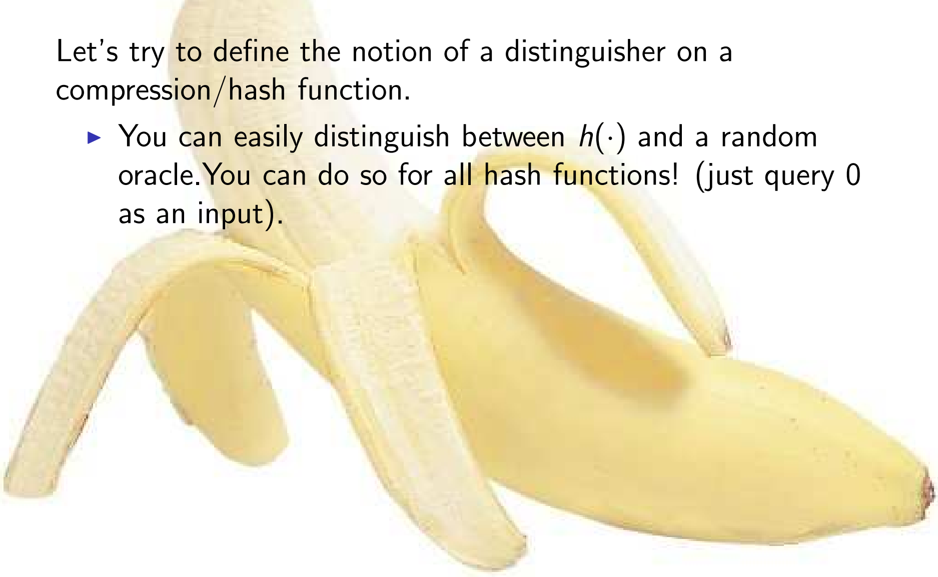
- You can easily distinguish between $h(\cdot)$ and a random oracle.



To Distinguish or Not to Distinguish

Let's try to define the notion of a distinguisher on a compression/hash function.

- You can easily distinguish between $h(\cdot)$ and a random oracle.You can do so for all hash functions! (just query 0 as an input).

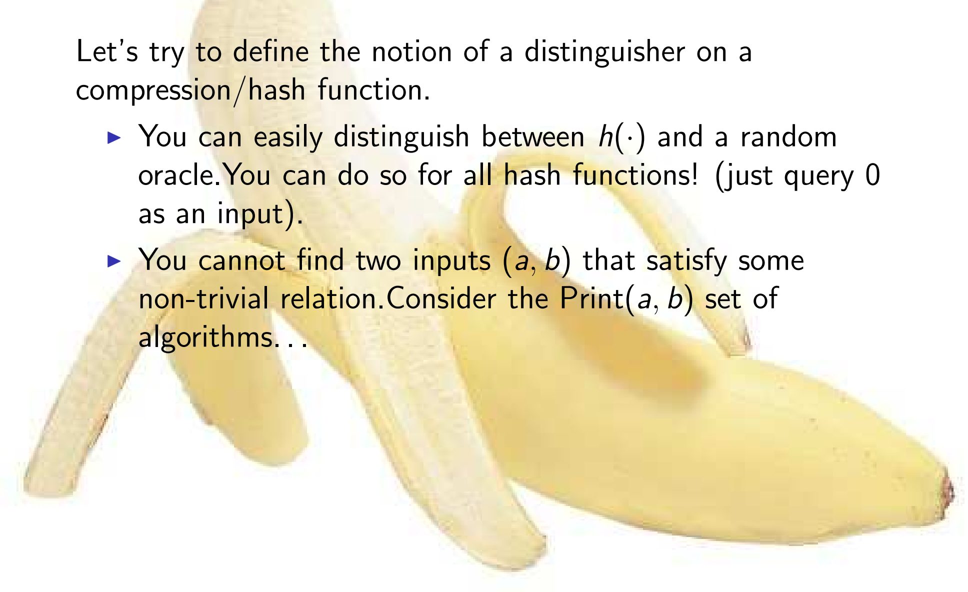


To Distinguish or Not to Distinguish

Let's try to define the notion of a distinguisher on a compression/hash function.

- You can easily distinguish between $h(\cdot)$ and a random oracle.You can do so for all hash functions! (just query 0 as an input).
- You cannot find two inputs (a, b) that satisfy some non-trivial relation.

To Distinguish or Not to Distinguish



Let's try to define the notion of a distinguisher on a compression/hash function.

- You can easily distinguish between $h(\cdot)$ and a random oracle.You can do so for all hash functions! (just query 0 as an input).
- You cannot find two inputs (a, b) that satisfy some non-trivial relation.Consider the $\text{Print}(a, b)$ set of algorithms...

To Distinguish or Not to Distinguish

Let's try to define the notion of a distinguisher on a compression/hash function.

- You can easily distinguish between $h(\cdot)$ and a random oracle. You can do so for all hash functions! (just query 0 as an input).
- You cannot find two inputs (a, b) that satisfy some non-trivial relation. Consider the Print(a, b) set of algorithms...
- Known-key distinguisher approach: It is possible to find a set of inputs that satisfy some relation in the output, faster than for a random oracle.

To Distinguish or Not to Distinguish

Let's try to define the notion of a distinguisher on a compression/hash function.

- You can easily distinguish between $h(\cdot)$ and a random oracle.You can do so for all hash functions! (just query 0 as an input).
- You cannot find two inputs (a, b) that satisfy some non-trivial relation.Consider the Print(a, b) set of algorithms. . .
- Known-key distinguisher approach: It is possible to find a set of inputs that satisfy some relation in the output, faster than for a random oracle.
- . . . and if you do not like this name, feel free to use: pseudo-distinguisher or . . . bananas.

Performance Evaluation — Software

- Some teams had many people on them. Some not.
- All teams submitted C code, but not all submitted assembler code, or optimized per-platform code.
- Some teams supply measurements using method A, some by using method B, ...
- Some teams supply measurements on a machine type A, some machine type B, ...
- Some teams used compiler X, some Y, ...
- Some teams had ...

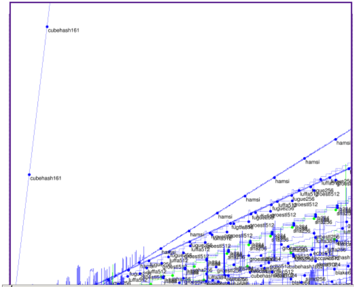
So how can you compare the speed?!?!?

Performance Evaluation — Software (cont.)

- eBASH — An effort to run everything everywhere.
  1. Strong points: lots of machines, easy to submit a new implementation.
  2. Weak points: still someone needs to implement, takes time for new implementations to be measured, some measurements are inconsistent.
  3. Measurement method can be "attacked": submit a hash function with a message block size of 16,000 bytes.
- sphlib — An effort to implement everything by one guy (without using per-CPU optimization) in C.
  1. Strong point: portable code is sometimes important.
  2. Weak points: based on a one-man show (who is actually a submitter of Shabal), why not to use per-CPU optimizations? why only C?

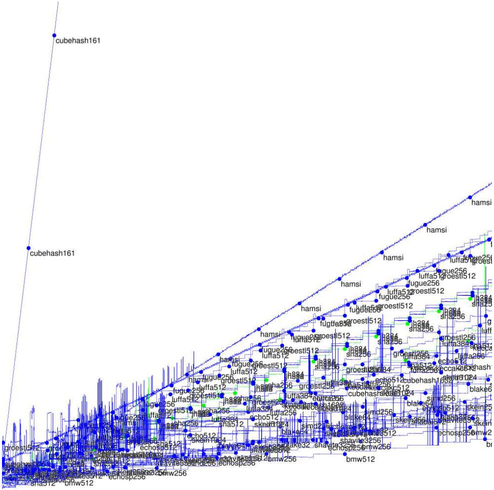
eBASH — A Glimpse

amd64, 2401MHz, Intel Xeon E5620 (206c2), giant4, supercop-20100821



Cycles/byte for long messages				Cycles/byte for 4096 bytes				Cycles/byte	
quartile	median	quartile	hash	quartile	median	quartile	hash	quartile	median
3.81	3.83	3.84	bmw512	4.11	4.11	4.12	bmw512	4.59?	4.59?
5.19	5.21	5.23	bmw256	5.40	5.40	5.41	bmw256	5.71	5.71
4.82?	5.46?	6.61?	echosp256	5.87	5.88	6.45	echosp256	5.98	5.99
4.79?	5.47?	5.52?	shavite3512	6.07	6.07	6.08	skein512	6.32	6.32
5.22?	5.83?	5.84?	shavite3256	5.81	6.12	6.13	shavite3512	6.69?	6.71?
2.88?	5.93?	5.94?	skein512	5.85	6.15	6.16	shavite3256	7.17	7.18
6.31	6.32	6.33	shabal512	6.73	6.73	6.73	shabal512	7.41	7.42
3.32?	6.61?	6.63?	echo256	6.74	6.75	6.75	echo256	7.55?	7.56?
5.40?	7.20?	7.22?	blake32	7.35	7.36	7.37	blake32	7.59	7.59
7.54?	7.59?	16.98?	skein256	7.65?	7.67?	12.35?	skein256	7.77	7.80
8.19	8.21	8.21	echosp512	8.38	8.38	8.39	echosp512	9.32	9.35
8.65	8.67	8.75	simd256	8.93	8.94	8.97	simd256	9.41?	9.42?
8.75?	9.04?	16.56?	blake64	9.36?	9.37?	13.12?	blake64	9.92	9.93
9.62	9.62	9.63	cubehash1632	10.30	10.31	10.34	simd512	10.97	10.98
9.88	9.91	9.97	simd512	9.85	10.36	10.37	skein1024	11.00?	11.00?
8.95?	9.98?	9.99?	skein1024	10.49	10.49	10.49	cubehash1632	11.93	11.93
11.58	11.59	11.60	keccakc512	12.08	12.09	12.09	keccakc512	12.62	12.63
11.90	11.94	11.96	echo512	12.25	12.25	12.25	luffa256	12.64?	12.65?
-0.62?	12.02?	12.03?	luffa256	12.49	12.50	12.50	echo512	13.39	13.41
12.40	12.43	12.46	keccak	12.89	12.90	12.90	keccak	13.64	13.69
12.50	12.52	13.34	sha512	13.08	13.08	13.49	sha512	14.01	14.01
13.31	13.33	13.34	luffa384	13.68	13.69	13.69	luffa384	14.28	14.28

eBASH — A Glimpse (cont.)



Performance Evaluation — Hardware

- Less people working on hardware implementation.
- More optimization targets (throughput vs. size vs. energy consumption)
- More technologies (ASIC vs. FPGA).
- Less common to share the "code".

Outline

SHA-3 Finalists

In December 2010, NIST have selected five finalists for the SHA-3 competition:

SHA-3 Finalists

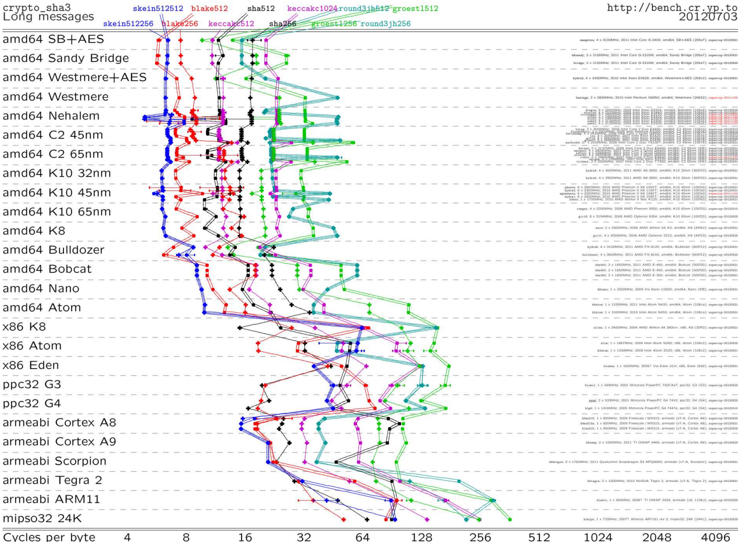
In December 2010, NIST have selected five finalists for the SHA-3 competition:

1. BLAKE
2. Grøstl
3. JH
4. KECCAK
5. Skein

The SHA-3 Finalists

- Each of the five finalists has different design methodology:
  - Narrow pipe (Haifa/UBI): BLAKE and Skein,
  - Double pipe: Grøstl and JH,
  - Sponge: KECCAK
- Each of them relies on different "security" mechanisms:
  - ARX: BLAKE, KECCAK, and Skein,
  - S-boxes: Grøstl and JH

Software Performance — eBASH



The eXtenral Benchmarking eXtension Project

- 8-bit platforms are not as extinct as many people believe them to be . . .
- The new SHA-3 would need to run on these platforms as well.
- The XBX project aims at being the eBASH extension to the 8-bit microcontrollers world.
- In general, Blake, Skein, and KECCAK are leading in performance.

The Security of the SHA-3 Finalists

- Of the 5 finalists, two have distinguishing properties for the full "compression" function:
  1. KECCAK (a zero sum distinguisher, in time complexity of 2^{1579}),
  2. JH (a rebound distinguisher, in time complexity of 2^{304}).
- While they somewhat invalidate the security proofs of JH and KECCAK, none of these attacks are considered as a real threat to the underlying hash functions.

The Security of the SHA-3 Finalists (cont)

Best known attacks against the finalists at the moment:

Candidate	Collision	2nd Preimage	Preimage	Distinguishing
Blake (14–16 rounds)	$5^\star$	2.5	2.5	8–10
Grøstl (10–14 rounds)	$3/6^\star$	—	—	9–10
JH (42 rounds)	16 *	—	—	42
KECCAK (24 rounds)	4	6–8	3	24
Skein (72–80 rounds)	—	37	37	34

SHA-3 — My Guess

Things which will label this entire thing as a waste of resources:

- Selecting something which offers less security than "optimal".
- Selecting something much slower than SHA.
- If performance requirements much larger than SHA.

SHA-3 — My Guess

Things which will label this entire thing as a waste of resources:

- Selecting something which offers less security than "optimal".
- Selecting something much slower than SHA.
- If performance requirements much larger than SHA.

In other words, NIST will pick the fastest secure-enough SHA-3 finalist.

SHA-3 — The True Waste of Effort

- SHA-3 took quite a lot of effort — analysis and implementation.
- Many cryptanalysts spent a lot of time designing their own submission.
- Then, they worked hard on breaking other SHA-3 candidates.

SHA-3 — The True Waste of Effort

- SHA-3 took quite a lot of effort — analysis and implementation.
- Many cryptanalysts spent a lot of time designing their own submission.
- Then, they worked hard on breaking other SHA-3 candidates.
- Hence, little time to work on SHA-1/SHA-2 ...

SHA-3 — The True Waste of Effort

- SHA-3 took quite a lot of effort — analysis and implementation.
- Many cryptanalysts spent a lot of time designing their own submission.
- Then, they worked hard on breaking other SHA-3 candidates.
- Hence, little time to work on SHA-1/SHA-2 ...
- What if this is all a scheme to make cryptanalysts work hard to extend SHA-1/2's lifetime?

The Current State of Affairs

Hash	Collisions	2nd Preimage	Preimage
MD4	By hand	2^{102}	2^{102}
MD5	2^{16}	$\approx 2^{124}$	$\approx 2^{124}$
SHA-0 (80 rounds)	2^{39}	up to 52 rounds	up to 52 rounds
SHA-1 (80 rounds)	2^{57}–2^{69}	up to 48 rounds	up to 48 rounds
SHA-256 (64 rounds)	up to 27 rounds	up to 43 rounds	up to 43 rounds
SHA-512 (80 rounds)	up to 24 rounds	up to 46 rounds	up to 46 rounds

SHA-3: To be Selected in August 2012...

Questions?

Thank you for your Attention!