The Hitchhiker's Guide to the SHA-3 Competition

Orr Dunkelman

Faculty of Mathematics and Computer Science
Weizmann Institute of Science

October 4th 2010

Outline

Outline

What is a Cryptographic Hash Function?

A hash function is a function that accepts an input of indefinite length, and outputs a digest of fixed length.

What is a Cryptographic Hash Function?

A cryptographic hash function is a function that accepts an input of indefinite length, and outputs a digest of fixed length securely.

First Introduction to Cryptography

[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.
- 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.)

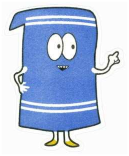
- (Cryptographic) Hash Functions are means to **securely** reduce a string *m* of arbitrarily length into a fixed-length digest.
- The main problem is the definition of securely.
- For signature schemes, three basic requirements exist:
  1. Preimage resistance: given $y = h(x)$, it is hard to find x (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.)

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.

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.
  - In certificates (in the signatures).
  - Inside protocols (used in many "imaginative" ways).
  - ...



What do we Want out of Our Hash Functions?

As hash functions are widely used, various requirements are needed to ensure the security of construction based on hash functions:

- Collision resistance — signatures, bit commitment (for binding), MACs.
- Second preimage resistance — signatures.
- Preimage resistance — signatures (RSA, or other TD-OWP), password files, bit commitment (for hiding).
- Pseudo Random Functions — key derivation, MACs.
- Pseudo Random Oracle — protocols, PRNGs.

What do we Really Want out of Hash Functions?

We want the hash function to behave in a manner which would prevent any adversary from doing anything malicious to the hash function:

- One-wayness (no inversion).
- No collisions (up to the birthday bound).
- No second preimages.
- Outputs which are nicely distributed.
- ...

Therefore, the ideal hash function attaches for each possible message M a random value as $h(M)$. And voilá — a random oracle.

What about Security?

What about Security?

- Collisions exist.

What about Security?

- Collisions exist. Also second preimages.

What about Security?

- Collisions exist. Also second preimages. Also preimages.

What about Security?

- Collisions exist. Also second preimages. Also preimages.
- Finding them is possible.

What about Security?

- Collisions exist. Also second preimages. Also preimages.
- Finding them is possible.
- But should be hard.

What about Security?

- Collisions exist. Also second preimages. Also preimages.
- Finding them is possible.
- But should be hard.

which raises the question:

What about Security?

- Collisions exist. Also second preimages. Also preimages.
- Finding them is possible.
- But should be hard.

which raises the question:

How hard?

Optimal Security of a Hash Function

If $h(\cdot)$ is the ideal hash function (a random oracle):

- Finding a preimage — $O(2^n)$ work (exhaustive search).
- Finding a second preimage — $O(2^n)$ work (exhaustive search).
- Finding a collision — $O(2^{n/2})$ work (birthday attack) [can be done with small memory overhead (Floyd or Nivasch)].

for an n-bit digest size.

Collision Resistance of Hash Functions

Let us try to define the meaning of $h(\cdot)$ being collision resistant.

Collision Resistance of Hash Functions

Let us try to define the meaning of $h(\cdot)$ being collision resistant.

- It is computationally infeasible to find a collision. Formally: There is no efficient algorithm which given h finds collisions.

Collision Resistance of Hash Functions

Let us try to define the meaning of $h(\cdot)$ being collision resistant.

- It is computationally infeasible to find a collision. Formally: There is no efficient algorithm which given h finds collisions.
- $h(\cdot)$ is a hash function. Therefore, necessarily there exist a, b such that $h(a) = h(b)$. Consider the algorithm:

  print a, b.

Collision Resistance of Hash Functions

Let us try to define the meaning of $h(\cdot)$ being collision resistant.

- It is computationally infeasible to find a collision. Formally: There is no efficient algorithm which given h finds collisions.
- $h(\cdot)$ is a hash function. Therefore, necessarily there exist a, b such that $h(a) = h(b)$. Consider the algorithm:

  print a, b.

What Should We Do?

Collision Resistance of Hash Functions (cont.)

- Practical solution — *a* and *b* are unknown. For any specific function finding them takes $O(1)$ anyway. So who cares?

Collision Resistance of Hash Functions (cont.)

- Practical solution — a and b are unknown. For any specific function finding them takes $O(1)$ anyway. So who cares?
- Theoretical solution (I) — let us define a *family* of hash functions, and bundle the collision resistance of one of them to the collision resistance of the family.

Collision Resistance of Hash Functions (cont.)

- Practical solution — a and b are unknown. For any specific function finding them takes $O(1)$ anyway. So who cares?
- Theoretical solution (I) — let us define a *family* of hash functions, and bundle the collision resistance of one of them to the collision resistance of the family.
- Theoretical solution (II) — we do not know the value of a, b for a specific hash function. Thus, let us define a protocol Π, which uses a hash function $h(\cdot)$, such that we can show that every adversary A against Π yields an attack on $h(\cdot)$ [R05].

A(n Extremely) Short History of Hash Functions

1976	Diffie and Hellman suggest to use hash functions to make digital signatures shorter.
1977/8	Rabin's hash function based on DES as a compression function.
1978	Yuval's attack on Rabin's hash (invertible compression functions are not a good idea).
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	Snefru is almost broken by differential cryptanalysis.

A(n Extremely) Short History of Hash Functions

1992	MD5 is introduced by Rivest.
1993	Preneel, Govaerts, Vandewalle study block-cipher based hashing.
1993	den Boer & Bosselaers find collisions in MD5's compression function.
1993	Bellare & Rogaway formally introduce random oracles.
1993	SHA-0 is introduced.
1995	SHA-1 is introduced.
1995	Collisions for MD4 are published.
1997	SHA-0 is broken by Chabaud and Joux.
1999	Dean's long second preimage attack on Merkle-Damgård.

A(n Extremely) Short History of Hash Functions

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.
2005	Kelsey & Scheneier's long second preimage attack on Merkle-Damgård.
2006	Kelsey & Kohno's herding attack.
2006	Better collision attacks on SHA-1.
2007	Preimage attacks on reduced-round SHA-1.
2007	SHA-1 Collision BOINC project starts.
2008	The SHA-3 competition starts . . .

Outline

How to Build a Hash Function

How to Build a Hash Function

?

How to Build a Symmetric-Key Block Cipher-Based Encryption Scheme

1. Design a block cipher.

How to Build a Symmetric-Key Block Cipher-Based Encryption Scheme

1. Design a block cipher (a primitive that accepts a key of fixed length, and encrypts plaintexts of a fixed length).
2. Find a good mode of iteration.

How to Build a Symmetric-Key Block Cipher-Based Encryption Scheme

1. Design a block cipher (a primitive that accepts a key of fixed length, and encrypts plaintexts of a fixed length).
2. Find a good mode of iteration (a method to encrypt messages whose length is different than the block size).
3. Combine the two together.

How to Build a Symmetric-Key Block Cipher-Based Encryption Scheme

1. Design a block cipher (a primitive that accepts a key of fixed length, and encrypts plaintexts of a fixed length).
2. Find a good mode of iteration (a method to encrypt messages whose length is different than the block size).
3. Combine the two together.

Examples of modes of operation: ECB, CBC, CTR, ...

How to Build a Hash Function (part II)

- Design a compression function (a black box that accepts $n + b$ bits and produces n bits).
- Find a good mode of iteration (a way to handle messages of length longer (or shorter) than $n + b$).
- Combine the two.

The Merkle-Damgård Construction

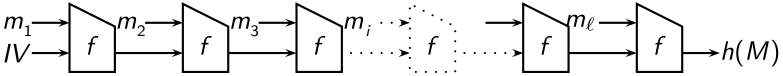
Given a compression function $f : \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$, the Merkle-Damgård hash function H_f is defined as:

1. Pad the message M to a multiple of b (with 1, and as many 0's as needed and the length of the message).
2. Divide the padded message into ℓ blocks $m_1 m_2 \ldots m_\ell$.

The Merkle-Damgård Construction

Given a compression function $f : \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$, the Merkle-Damgård hash function H_f is defined as:

1. Pad the message M to a multiple of b (with 1, and as many 0's as needed and the length of the message).
2. Divide the padded message into ℓ blocks $m_1 m_2 \ldots m_\ell$.
3. Set $h_0 = IV$.
4. For $i = 1$ to ℓ, compute $h_i = f(h_{i-1}, m_i)$.
5. Output h_ℓ (or some function of it).



The Security of the Merkle-Damgård Construction

- Finding a collision in H_f means finding a collision in f.
- Thus, if f is collision-resistant, so is H_f.

The Security of the Merkle-Damgård Construction

- Finding a collision in H_f means finding a collision in f.
- Thus, if f is collision-resistant, so is H_f.
- Also, finding a second preimage in H_f means finding a collision in f.

The Security of the Merkle-Damgård Construction

- Finding a collision in H_f means finding a collision in f.
- Thus, if f is collision-resistant, so is H_f.

- Also, finding a second preimage in H_f means finding a collision in f.
- The same is true for finding a preimage (because you can use it to find a second preimage).

The Security of the Merkle-Damgård Construction

- Finding a collision in H_f means finding a collision in f.
- Thus, if f is collision-resistant, so is H_f.

- Also, finding a second preimage in H_f means finding a collision in f.
- The same is true for finding a preimage (because you can use it to find a second preimage).

To conclude, if f is collision resistant (i.e., it takes $O(2^{n/2})$ invocations to find a collision), then H_f is collision resistant and (second) preimage resistant with security level of $O(2^{n/2})$.

The Security of the Merkle-Damgård Construction

- Finding a collision in H_f means finding a collision in f.
- Thus, if f is collision-resistant, so is H_f.

- Also, finding a second preimage in H_f means finding a collision in f.
- The same is true for finding a preimage (because you can use it to find a second preimage).

To conclude, if f is collision resistant (i.e., it takes $O(2^{n/2})$ invocations to find a collision), then H_f is collision resistant and (second) preimage resistant with security level of $O(2^{n/2})$.

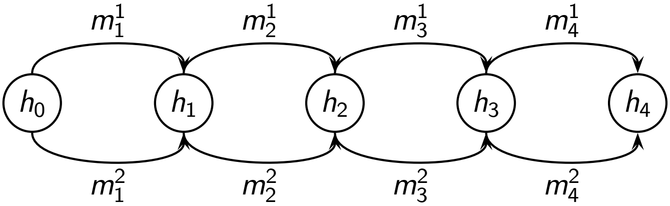
But we want better security guarantees (of $O(2^n)$) for (second) preimage resistance!

Multi-collision Attacks on Iterative Hashing

- Finding 2^t collisions in iterative hash function with chaining value length m_c, takes $O(t \cdot 2^{n/2})$ [J04]

Multi-collision Attacks on Iterative Hashing

- Finding 2^t collisions in iterative hash function with chaining value length m_c, takes $O(t \cdot 2^{n/2})$ [J04]

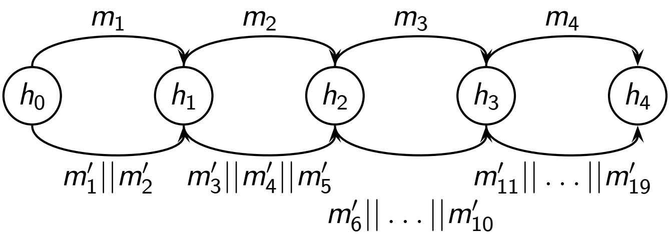


$$\begin{tikzcd}
h_0 \arrow[r, bend left, "m_1^1"] \arrow[r, bend right, "m_1^2"'] & h_1 \arrow[r, bend left, "m_2^1"] \arrow[r, bend right, "m_2^2"'] & h_2 \arrow[r, bend left, "m_3^1"] \arrow[r, bend right, "m_3^2"'] & h_3 \arrow[r, bend left, "m_4^1"] \arrow[r, bend right, "m_4^2"'] & h_4
\end{tikzcd}$$

In an ideal hash function the time complexity should be $O(2^{\frac{2^t-1}{2^t} \cdot n})$.

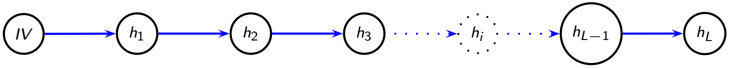
How to Generate an Expandable Messages

- In [KS05] the expandable message is constructed as a multi-collision. In the first block between a message of one block and a message of two blocks. Then, between one block and three blocks, one block and five, etc.


$$\begin{tikzcd}
h_0 \arrow[r, bend left, "m_1"] \arrow[r, bend right, "m'_1||m'_2"'] & h_1 \arrow[r, bend left, "m_2"] \arrow[r, bend right, "m'_3||m'_4||m'_5"'] & h_2 \arrow[r, bend left, "m_3"] \arrow[r, bend right, "m'_6||\ldots||m'_{10}"'] & h_3 \arrow[r, bend left, "m_4"] \arrow[r, bend right, "m'_{11}||\ldots||m'_{19}"'] & h_4
\end{tikzcd}$$

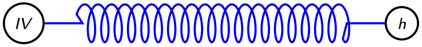
Computational cost: $O(\ell \cdot 2^{n/2} + 2^\ell)$ for an expandable message of lengths between ℓ and $2^\ell + \ell - 1$.

Expandable Message $\longrightarrow$ a Second Preimage Attack


$$\begin{tikzcd} IV \arrow[r] & h_1 \arrow[r] & h_2 \arrow[r] & h_3 \arrow[r, dotted] & h_i \arrow[r, dotted] & h_{L-1} \arrow[r] & h_L \end{tikzcd}$$

Expandable Message → a Second Preimage Attack

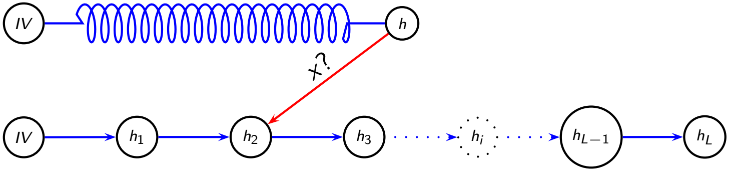
- Generate an expandable message that covers many lengths (up to 2^ℓ), whose output chaining value is h.



$$\begin{tikzcd} IV \arrow[r] & h_1 \arrow[r] & h_2 \arrow[r] & h_3 \arrow[r, dotted] & h_i \arrow[r, dotted] & h_{L-1} \arrow[r] & h_L \end{tikzcd}$$

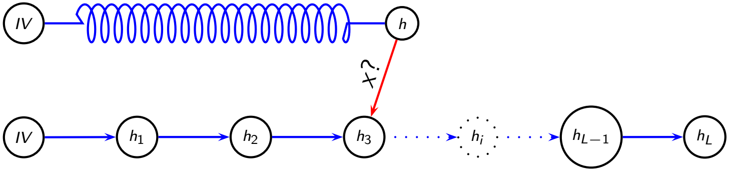
Expandable Message $\longrightarrow$ a Second Preimage Attack

- Generate an expandable message that covers many lengths (up to 2^ℓ), whose output chaining value is h.
- Try to find x, such that $f(h, x) = h_i$ (one of the chaining values computed for the original message).



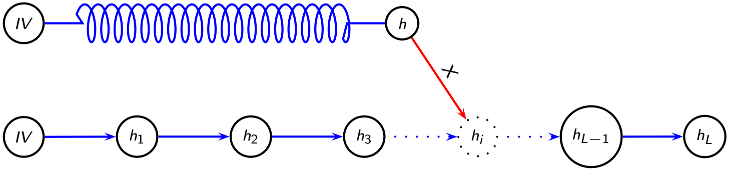
Expandable Message $\longrightarrow$ a Second Preimage Attack

- Generate an expandable message that covers many lengths (up to 2^ℓ), whose output chaining value is h.
- Try to find x, such that $f(h,x) = h_i$ (one of the chaining values computed for the original message).



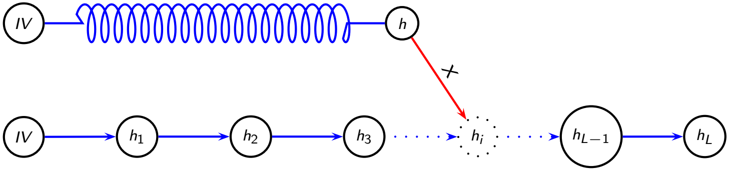
Expandable Message $\longrightarrow$ a Second Preimage Attack

- Generate an expandable message that covers many lengths (up to 2^ℓ), whose output chaining value is h.
- Try to find x, such that $f(h,x) = h_i$ (one of the chaining values computed for the original message).



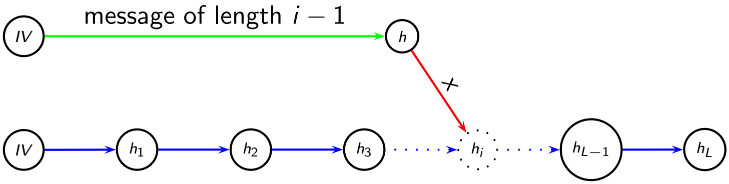
Expandable Message $\longrightarrow$ a Second Preimage Attack

- Generate an expandable message that covers many lengths (up to 2^ℓ), whose output chaining value is h.
- Try to find x, such that $f(h,x) = h_i$ (one of the chaining values computed for the original message).
- Once the "connection" step succeeds, fix the length using the precomputed expandable message.



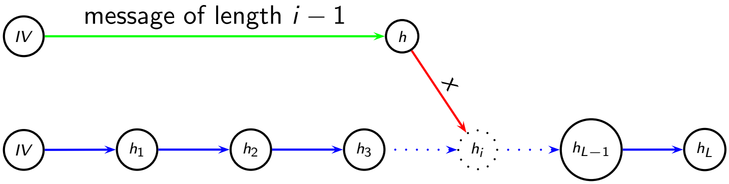
Expandable Message $\longrightarrow$ a Second Preimage Attack

- Generate an expandable message that covers many lengths (up to 2^ℓ), whose output chaining value is h.
- Try to find x, such that $f(h, x) = h_i$ (one of the chaining values computed for the original message).
- Once the "connection" step succeeds, fix the length using the precomputed expandable message.



Expandable Message $\longrightarrow$ a Second Preimage Attack

- Generate an expandable message that covers many lengths (up to 2^ℓ), whose output chaining value is h.
- Try to find x, such that $f(h, x) = h_i$ (one of the chaining values computed for the original message).
- Once the "connection" step succeeds, fix the length using the precomputed expandable message.
- Online time complexity: $O(2^{n-\ell})$.



The MD/SHA-Family

The MD/SHA family is composed of many hash functions with similar design criteria:

- Davies-Meyer transformation of a block cipher into a compression function.
- Merkle-Damgård hash function.
- Simple round functions (with little nonlinearity).
- The nonlinearity is "introduced" bit-by-bit (AND, MAJ operations) and using addition modulo 2^{32}.
- The message expansion (key schedule) is linear (either repetition, or through an LFSR).
- Very software-friendly (not so bad on hardware as well).
- Message block: 512-bit (1024 for SHA-2); Digest size: 128-bit (MD4/5), 160-bit (SHA-0/1), 224/256/384/512 (SHA-2).

The SHA-1 Hash Function (cont.)

- Designed by the NSA, following the structure of MD4 and MD5.
- SHA-1 is a Merkle-Damgård hash function:
  1. **Padding**: Given an m-bit message, a single bit "1" is appended as the $m+1$'th bit and then $(448-(m+1)) \bmod 512$ (between 0 and 511) zero bits are appended. As a result, the message becomes 64-bit short of being a multiple of 512 bits long.
  2. **Merkle-Damgård Strengthening** Append the length: A 64-bit representation of the original length of m is appended, making the result a multiple of 512 bits long.
  3. **Division into Blocks** The result is divided into 512-bit blocks, denoted by $M_1, M_2, \ldots, M_\ell$.

The SHA-1 Hash Function (cont.)

The internal state of SHA-1 is composed of five 32-bit words A, B, C, D and E, used to keep the 160-bit chaining value h_i.

- **Initialization:** The initial value (h_0) is set to

$$\begin{aligned} A &= \texttt{67452301}_x \quad ; & B &= \texttt{EFCDAB89}_x \quad ; \\ C &= \texttt{98BADCFE}_x \quad ; & D &= \texttt{10325476}_x \quad ; \\ E &= \texttt{C3D2E1F0}_x \quad . \end{aligned}$$

- **Compression:** For each block, the compression function $h_i = H(h_{i-1}, M_i)$ is applied on the previous value of $h_{i-1} = (A, B, C, D, E)$ and the message block.
- **Output:** The hash value is the 160-bit value $h_\ell = (A, B, C, D, E)$.

The Compression Function H of SHA-1

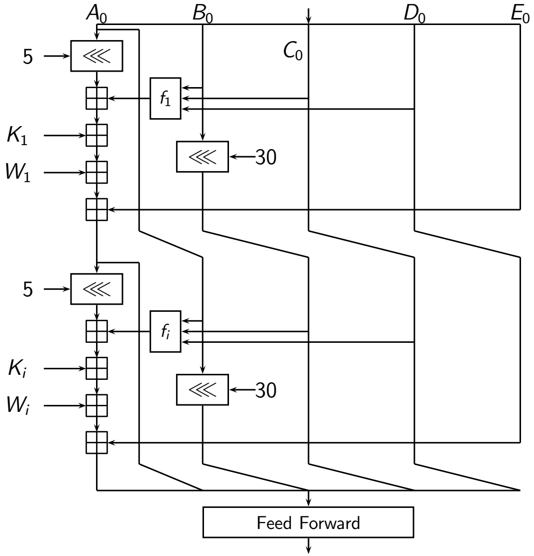
1. Divide M_i into 16 32-bit words: $W_0, W_1, W_2, \ldots, W_{15}$.
2. for $t = 16$ to 79 compute
   $W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1$.
3. Set $(A_0, B_0, C_0, D_0, E_0) \leftarrow h_{i-1}$.
4. For $t = 0$ to 79 do
   1. $T = A_t \lll 5 + f_t(B_t, C_t, D_t) + E_t + W_t + K_t$.
   2. $E_{t+1} = D_t,\ D_{t+1} = C_t,\ C_{t+1} = B_t \lll 30,\ B_{t+1} = A_t,$
      $A_{t+1} = T$.

The Compression Function H of SHA-1 (cont.)

5. Output $A = A_0 + A_{80}$, $B = B_0 + B_{80}$, $C = C_0 + C_{80}$, $D = D_0 + D_{80}$, and $E = E_0 + E_{80}$ (modulo 2^{32}).
6. The function f_t and the values K_t used above are:

$0 \leq t \leq 19$:	$f_t(X,Y,Z) = XY \vee (\neg X)Z$	$K_t = \texttt{5A827999}$
$20 \leq t \leq 39$:	$f_t(X,Y,Z) = X \oplus Y \oplus Z$	$K_t = \texttt{6ED9EBA1}$
$40 \leq t \leq 59$:	$f_t(X,Y,Z) = XY \vee XZ \vee YZ$	$K_t = \texttt{8F1BBCDC}$
$60 \leq t \leq 79$:	$f_t(X,Y,Z) = X \oplus Y \oplus Z$	$K_t = \texttt{CA62C1D6}$

The Compression Function H of SHA-1 (cont.)



The Shortcomings of the MD/SHA Family

- Apparently, most of the nonlinearity is introduced either in addition or locally (bitwise operations).

The Shortcomings of the MD/SHA Family

- Apparently, 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

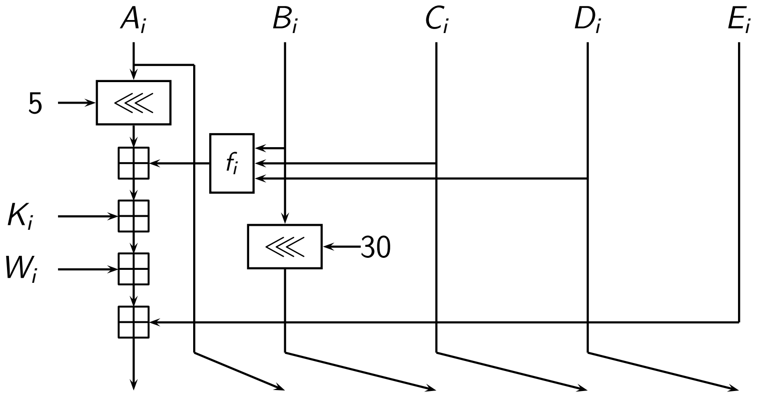
- Apparently, most of the nonlinearity is introduced either in addition or locally (bitwise operations).
- An immediate consequence — easy to approximate the algorithm as a linear.
- Easy to define the conditions on when the approximation holds.

The Shortcomings of the MD/SHA Family

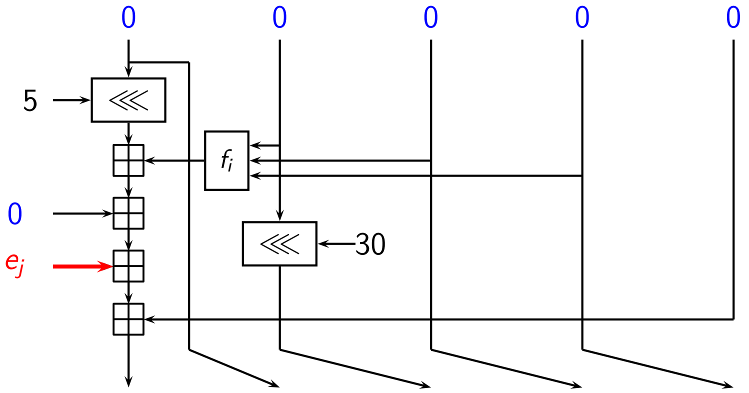
- Apparently, most of the nonlinearity is introduced either in addition or locally (bitwise operations).
- An immediate consequence — easy to approximate the algorithm as a linear.
- Easy to define the conditions on when the approximation holds.
- Along with a simple message expansion, relatively slow 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.

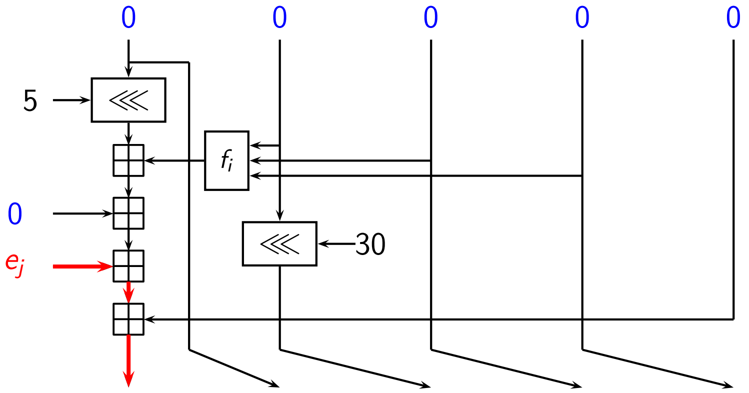
Example: Local Collisions of SHA-0 [CJ97]



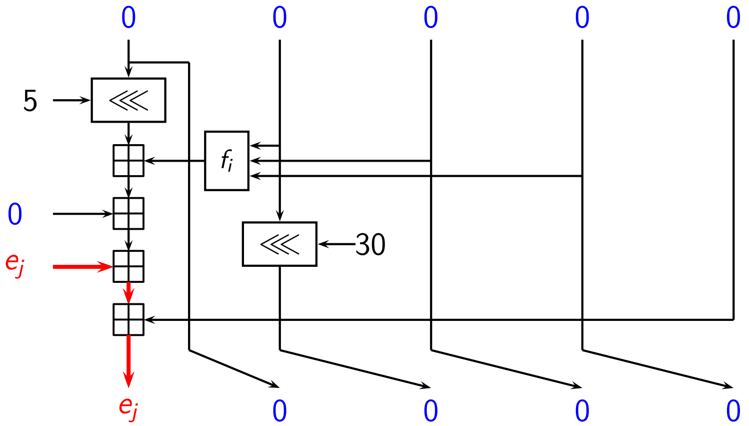
Example: Local Collisions of SHA-0 [CJ97]



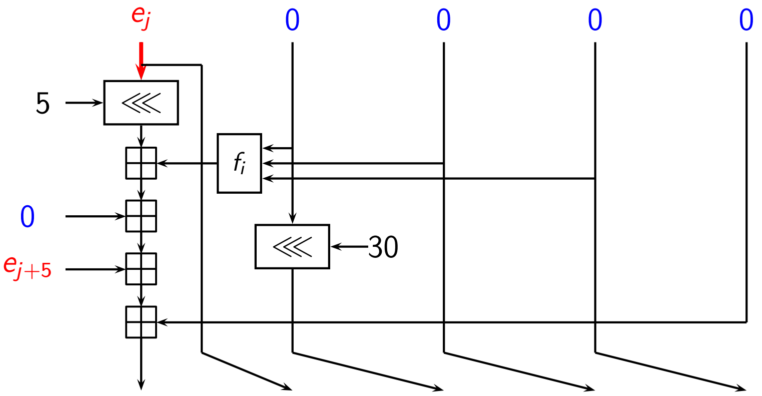
Example: Local Collisions of SHA-0 [CJ97]



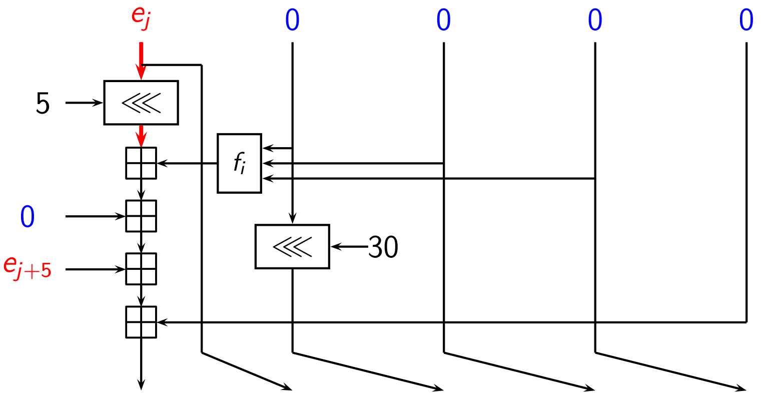
Example: Local Collisions of SHA-0 [CJ97]



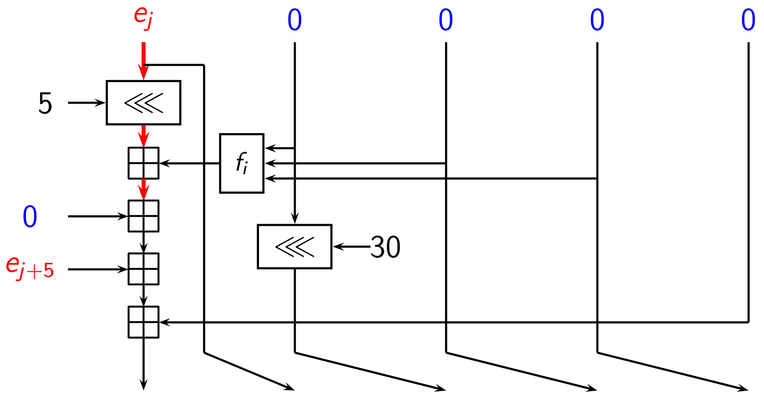
Example: Local Collisions of SHA-0 [CJ97]



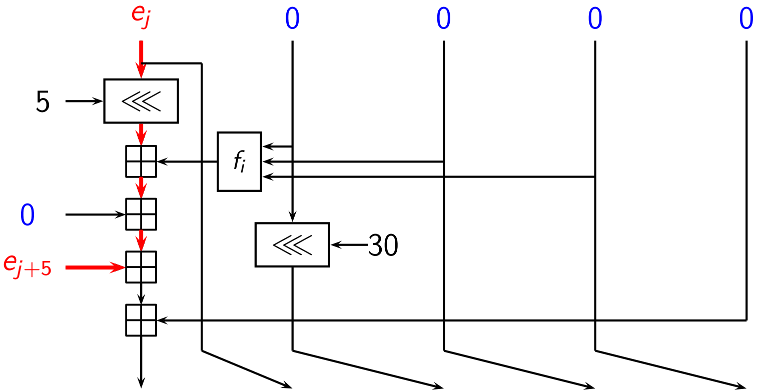
Example: Local Collisions of SHA-0 [CJ97]



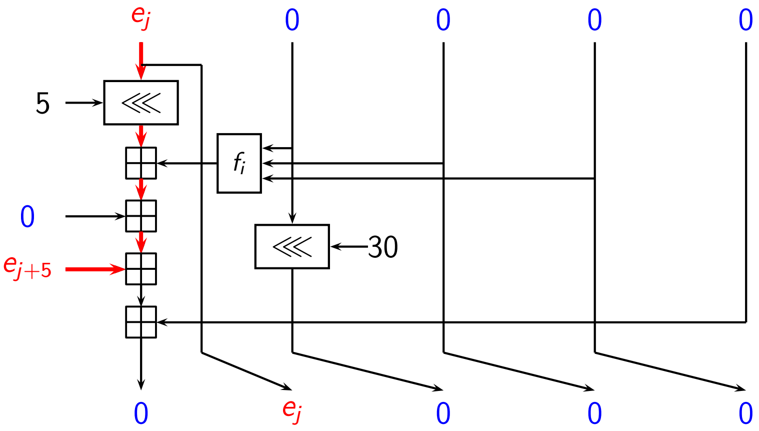
Example: Local Collisions of SHA-0 [CJ97]



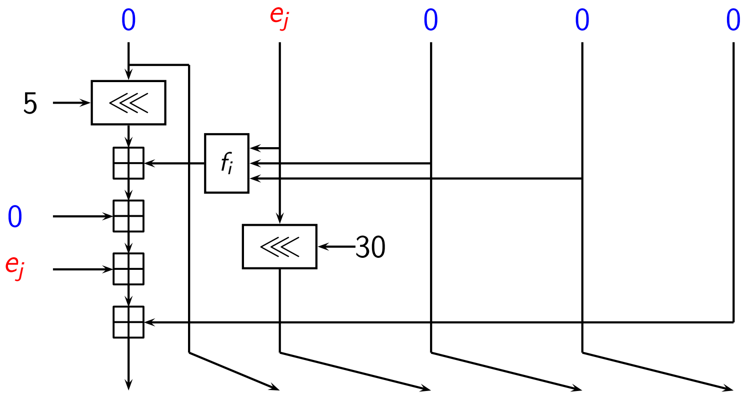
Example: Local Collisions of SHA-0 [CJ97]



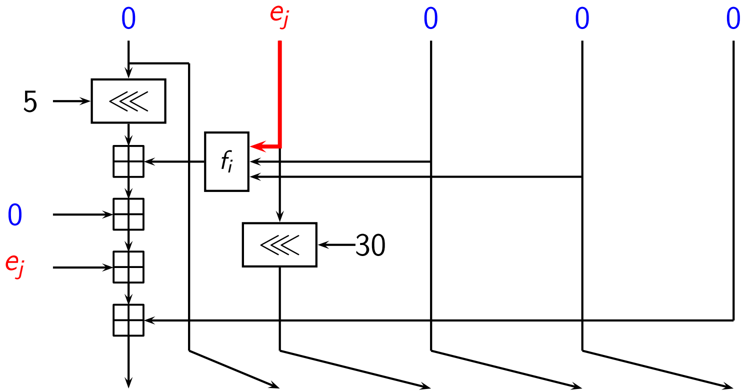
Example: Local Collisions of SHA-0 [CJ97]



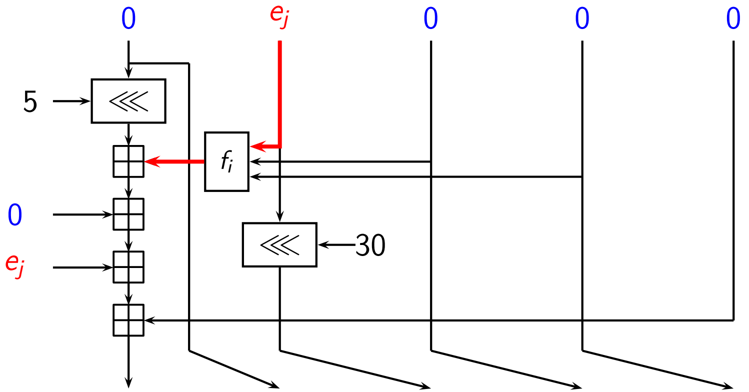
Example: Local Collisions of SHA-0 [CJ97]



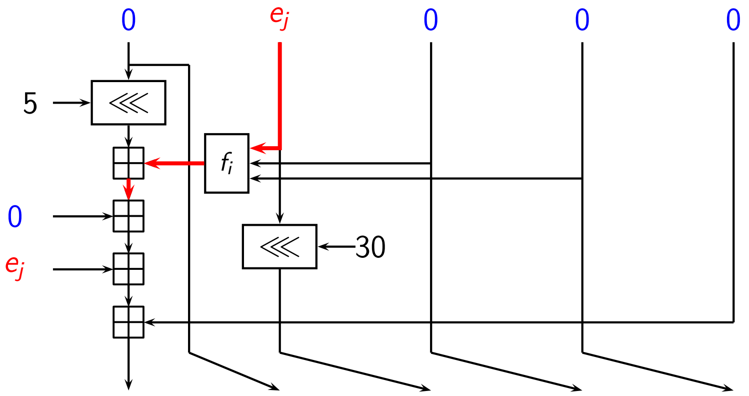
Example: Local Collisions of SHA-0 [CJ97]



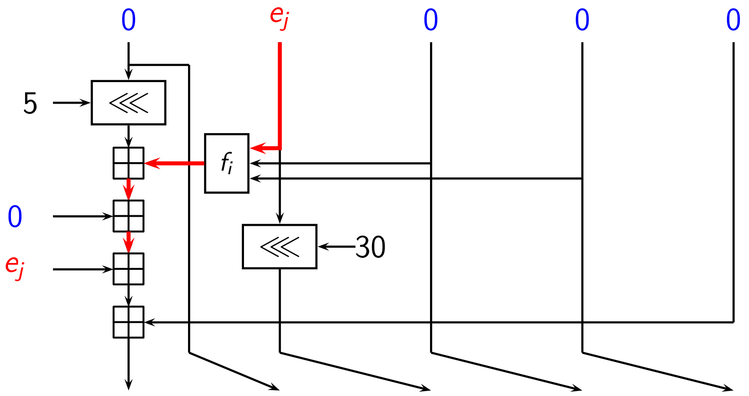
Example: Local Collisions of SHA-0 [CJ97]



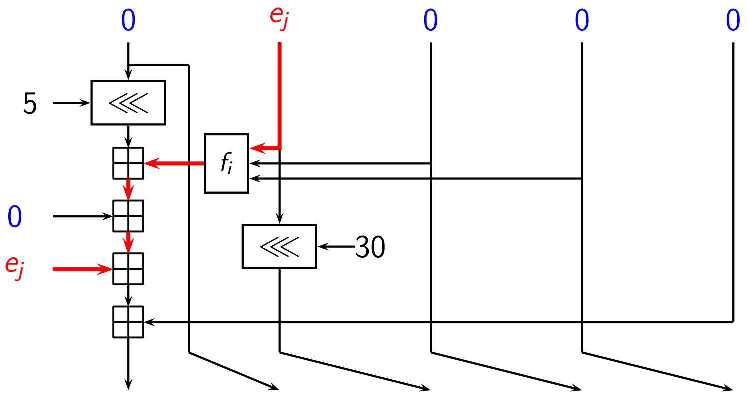
Example: Local Collisions of SHA-0 [CJ97]



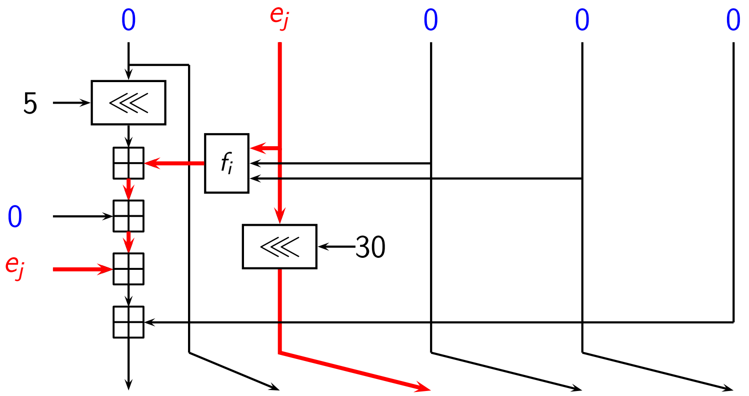
Example: Local Collisions of SHA-0 [CJ97]



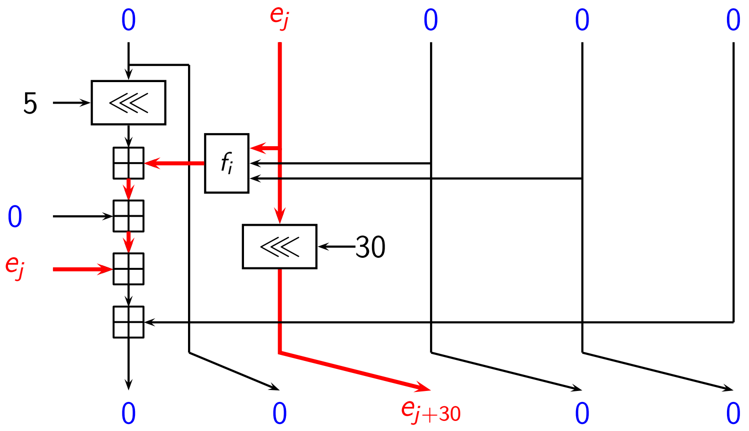
Example: Local Collisions of SHA-0 [CJ97]



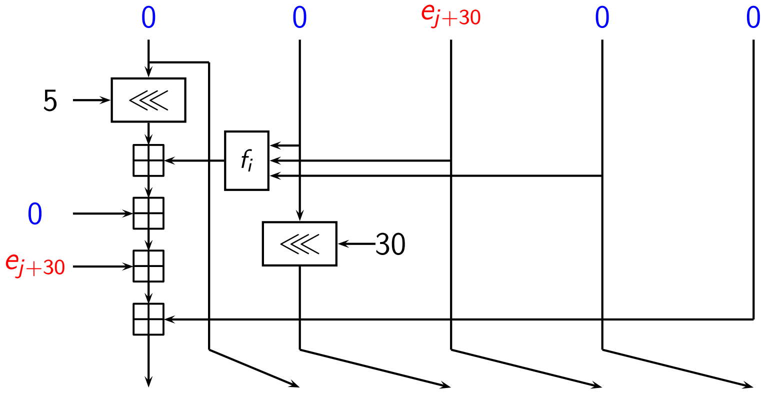
Example: Local Collisions of SHA-0 [CJ97]



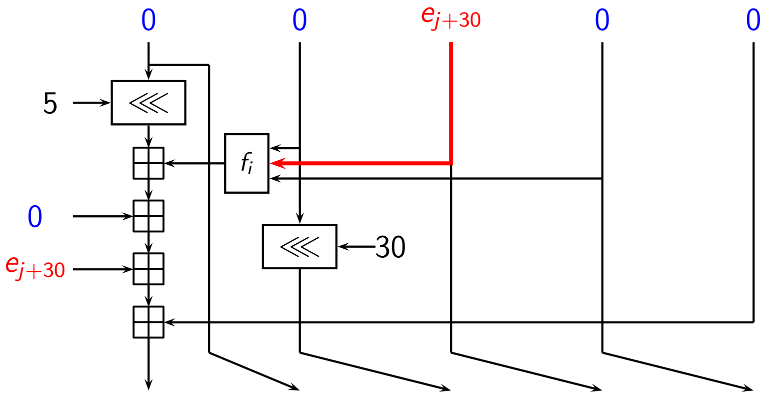
Example: Local Collisions of SHA-0 [CJ97]



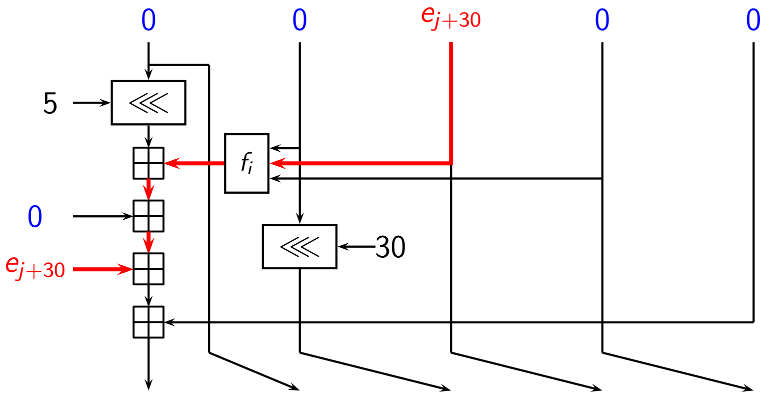
Example: Local Collisions of SHA-0 [CJ97]



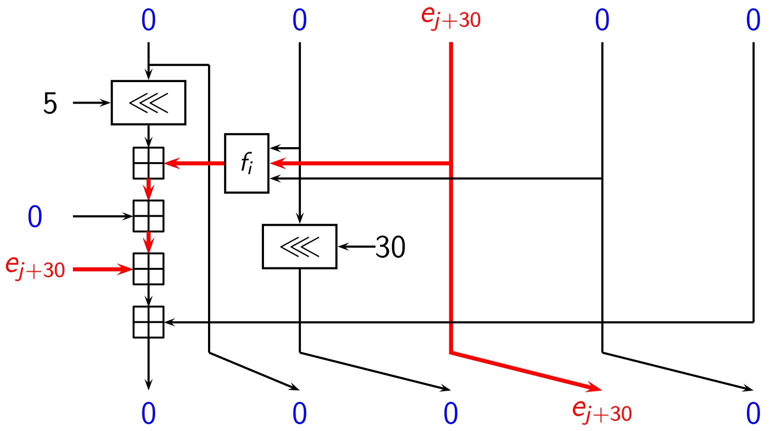
Example: Local Collisions of SHA-0 [CJ97]



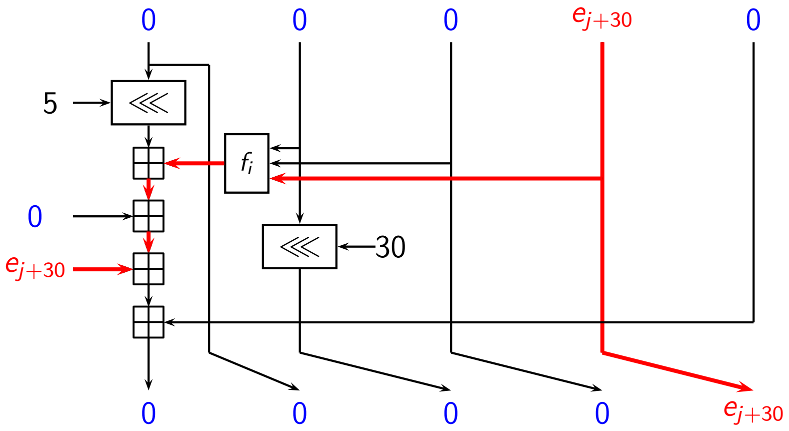
Example: Local Collisions of SHA-0 [CJ97]



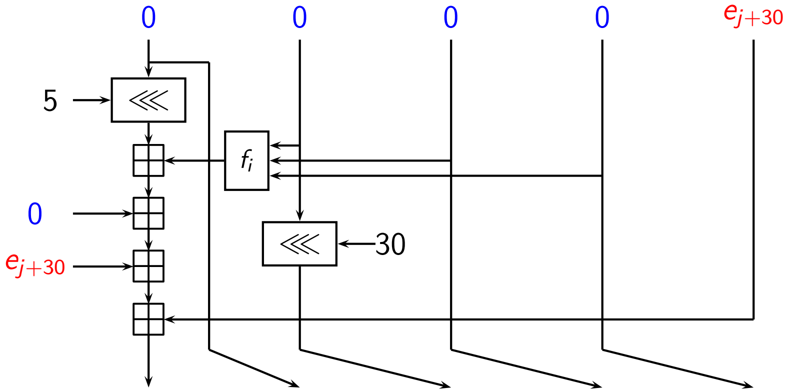
Example: Local Collisions of SHA-0 [CJ97]



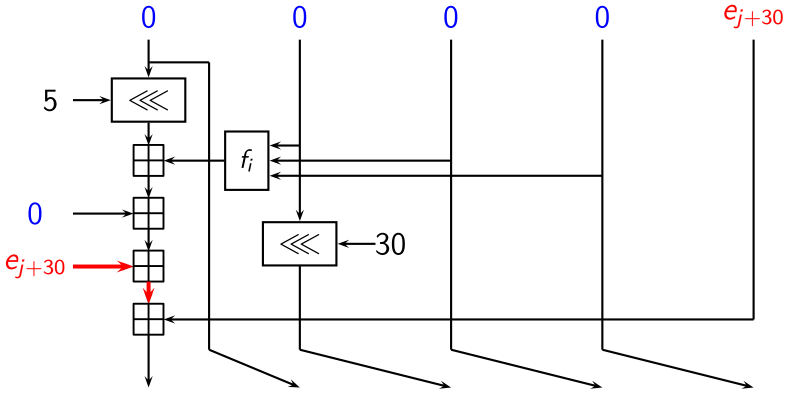
Example: Local Collisions of SHA-0 [CJ97]



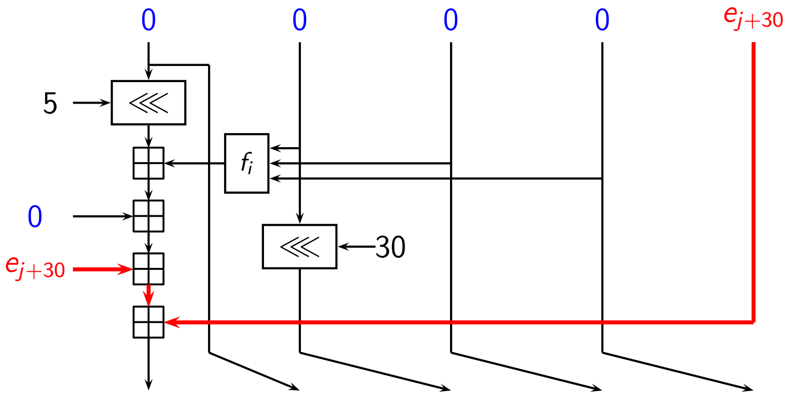
Example: Local Collisions of SHA-0 [CJ97]



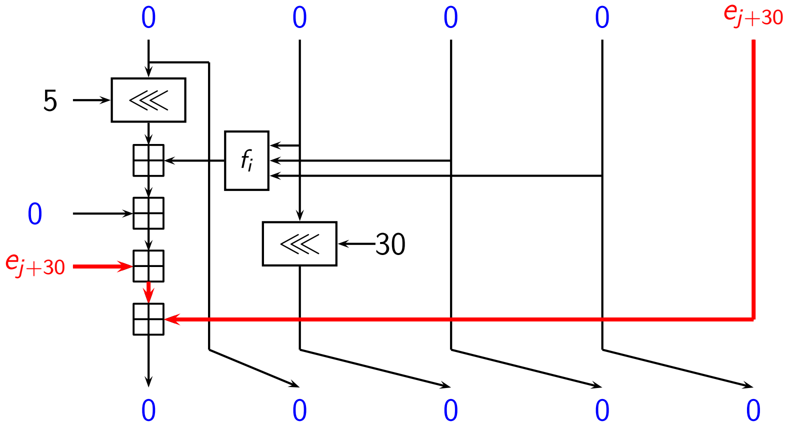
Example: Local Collisions of SHA-0 [CJ97]



Example: Local Collisions of SHA-0 [CJ97]



Example: Local Collisions of SHA-0 [CJ97]



The Current State of Affairs

Hash	Collisions	Second Preimage	Preimage
MD4	By hand	2^{102}	2^{102}
MD5	2^{16}	$\approx 2^{128}$	$\approx 2^{128}$
SHA-0 (80 rounds)	2^{39}	up to 52 rounds	up to 52 rounds
SHA-1 (80 rounds)	$\approx 2^{60.3}$	up to 48 rounds	up to 48 rounds
SHA-256 (64 rounds)	up to 24 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

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.

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	

Security Evaluation

- Besides Hamsi, no attacks on the full hash functions on the other 13 candidates are expected*.
- Some attacks on the full compression function exist (BMW, CubeHash, Grøstl, KECCAK, Luffa, Shabal, SHAvite-3, SIMD).
- These attacks do not scale to the full hash function (at the moment).
- Some attacks on almost the full compression function (ECHO, FUGUE, Skein).
- Some primitives received little 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 fix 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 fix 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 fix 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*.

Performance Evaluation — Software

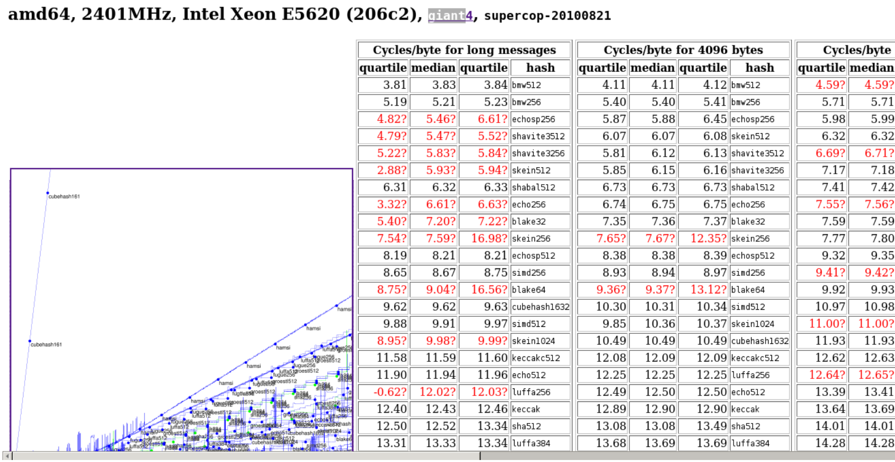
- Some teams had many people on them. Some not.
- All teams submitted C code, but not all submitted assembler code, or optimized per-platform code.
- Some teams supply measurements using method A, some by using method B, . . .
- Some teams supply measurements on a machine type A, some machine type B,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

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

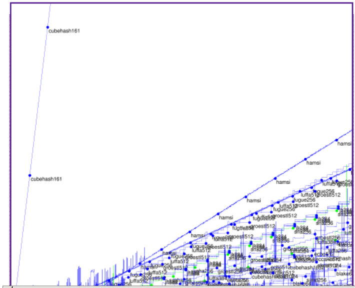
Performance Evaluation — Software (cont.)

- 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

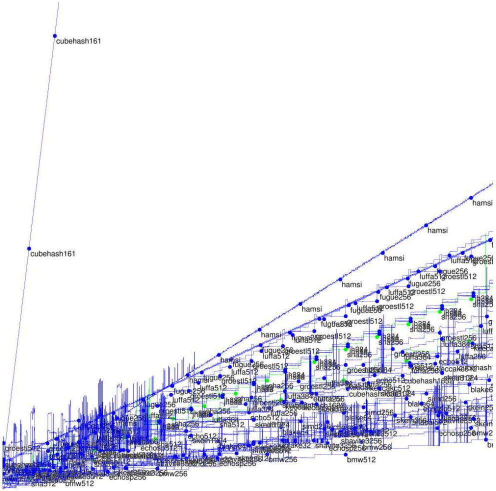


amd64, 2401MHz, Intel Xeon E5620 (206c2), giant4, supercop-20100821



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.)



Software Performance — Recent 32-bit Platforms

1. BMW-256 (6.9–9.7 cpb), BLAKE-32 (7.0–10.1 cpb), SIMD-256 (8.8–13.2 cpb), CubeHash16/32-256 (9.3–13.0 cpb)
2. Luffa-256 (12.3–18.5 cpb), SHA-256 (14.8–19.8 cpb), FUGUE-256 (15.7–21.1 cpb), KECCAK (16.4–21.8 cpb), Skein-256 (18.0–20.8 cpb)
3. JH-256 (16.6–29.2 cpb), Grøstl-256 (20.3–27.0 cpb)
4. Hamsi (24.5–32.8 cpb), SHAvite-3_{256} (25.6–31.7 cpb), ECHO-256 (29.2–35.9 cpb)

Software Performance — Recent 32-bit Platforms

1. BMW-512 (4.2–5.7 cpb), Shabal-512 (7.7–12.4 cpb)
2. CubeHash16/32-256 (9.3–13.0 cpb), SIMD-512 (10.4–14.6 cpb), BLAKE-64 (12.6–16.1 cpb), Skein-512 (14.2–17.6 cpb), SHA-512 (15.5–20.6 cpb), KECCAK (17.2–23.1 cpb)
3. JH-512 (16.7–29.2 cpb), Luffa-512 (22.3–27.4 cpb)
4. Grøstl-512 (29.4–37.7 cpb)
5. SHAvite-3_{512} (41.7–49.0 cpb)
6. ECHO-256 (50.8–60.68 cpb)

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".

The Third Round

- In a few months, NIST would announce the five (± 1) finalists.

The Third Round

- In a few months, NIST would announce the five (± 1) finalists.
- The remaining finalists will be attacked, analyzed, implemented, and assessed for another year and a half.
- Then NIST will pick the new SHA-3.

SHA-3 — My Guesses

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 Guesses

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 — My Guess (Compression Functions)

- Performance not much worse than SHA-256/-512.
- Implementable on 8-bit platforms.
- ASIC speeds that can reach 5 Gbps.
- Possible to implement with “restricted” memory.
- RFID **will not** play any role.
- Good differential and linear properties.
- Known and well-understood components (e.g., XOR vs. addition).

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?

How Can You Help?

- Analysis — try to break candidates.
- Implementation — try to implement:
  1. Optimize previous implementations.
  2. New machines and platforms.
  3. Hardware implementations — ASICs and FGPAs.
  4. Side-channel resistant implementations

How Can You Help — Inside Linux?

- In the latest stable kernel (2.6.35.7), I've (quickly) counted three implementations of MD5. Why?
- Make the necessary changes so that MD5 certificates will be phased out.
- Replace MD5 by good hash functions.
- And prepare for the day that SHA-1 will be insufficient.
- Lay the foundations for crypto agility in the kernel.

Questions?

Thank you for your Attention!