# Recent Advances in Hash Functions:

# The Way to Go

Eli Biham

Computer Science Department
Technion, Haifa 32000, Israel

June 24, 2005

# History

Two approaches for the design of hash functions are:

1. **Hash functions based on block ciphers** were very popular in the 1980's. Since the block size of the block cipher was smaller than the hash size, each compression function had to call to several applications of the block cipher, so the speed was quite slow. Also, the message was traditionally entered as the key, so the key schedule algorithm had also to be called, which slowed down the function even further.

2. **Dedicated hash functions** use a dedicated design for their compression function, which allows a larger block size (not limited by the key size of the block cipher), and faster speeds.

The second approach is the more popular nowadays.

# Dedicated Hash Functions

Hash functions of the second approach include:

| Function | Hash Size | Design | Broken | Family |
|---|---|---|---|---|
| Snefru (2,4 passes) | 128–224 | 1990 | 1990 | |
| MD4 | 128 | 1990 | 1995 | MD4/SHA |
| MD5 | 128 | 1992 | 2004 | MD4/SHA |
| SHA-0 (SHA) | 160 | 1993 | 1998,2004 | MD4/SHA |
| SHA-1 | 160 | 1995 | 2005 | MD4/SHA |
| Tiger | $\leq$192 | 1995 | | |
| RIPEMD-160 | 160 | 1996 | | MD4/SHA |
| Whirlpool | 512 | 2000 | | |
| SHA-224, SHA-256 | 224, 256 | 2002 | | MD4/SHA |
| SHA-384, SHA-512 | 384, 512 | 2002 | | MD4/SHA |

# Two Kinds of Attacks

1. Differential attack (attacking the compression function)

2. Structural attacks (against Merkle-Damgård)

# Attacks on the MD4/SHA Family

1. Two rounds of MD4 — several papers in 1990–1994
2. Full MD4 — Dobbertin found a pair of messages that differ by 3 bits (additive difference of 1 in one of the message words) (1995)
3. MD5 — collision with a non-standard initial value (Dobbertin, 1996)
4. SHA-0 — Differential attack with complexity $2^{61}$ (1998)

**Recent attacks:** — all are differential

5. SHA-0 — reduced/extended versions, near-collisions (Biham, Chen, 2004)
6. Multi-block, reduced versions of SHA-0 and SHA-1 (Biham, Chen, 2004)
7. SHA-0 — a multi-block collision (Joux et al., 2004)
8. SHA-1 — reduced versions (Rijmen, Oswald, 2004)
9. MD5, MD4, and others — differential attacks (Wang et al., 2004)
10. SHA-0 — a fast collision (Wang et al., 2005)
11. SHA-1 — an attack for finding collisions with complexity about $2^{69}$ (Wang et al., 2005)

# Structural attacks

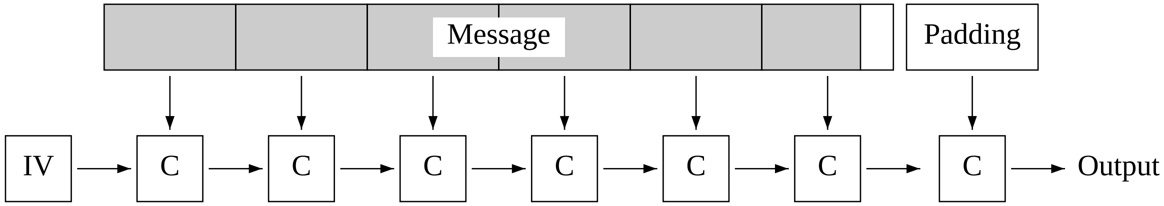
1. One-of-many second preimage attack (Merkle, 1979)
2. Multi collision (Joux, 2004)
3. Fixpoint attacks and second preimages (Dean, 1999)
4. Expandable messages and second preimages of iterative hash functions (Kelsey, Schneier, 2005)
5. It is easy to append suffix to a hash, even if the message is unknown (well known problem for MACs)
6. Related hash function attacks: e.g., SHA-512, SHA-384: replacing the IV, truncating the hash result, also SHA-256 truncated to 160 bits
7. Theoretical studies of hash functions consider a family of hash functions

# Part I

# Structure of Practical Hash Functions

All practical hash functions divide the message into fixed-size blocks, and process each block in turn by a compression function (Merkle-Damgård).

Each application of the compression function generates an intermediate value, which in turn is being input to the next application with the next block.




The message is first padded to become a multiple of a block.

# The Compression Function of SHA-1

1. Divide the 512-bit block into 16 32-bit words: $W_0, W_1, W_2, \ldots, W_{15}$.

2. **Message schedule**: For $t = 16$ to 79 compute

$$\boldsymbol{W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})} \quad \underbrace{\lll \mathbf{1}}_{\uparrow \text{ Only in SHA-1}},$$

   where $X \lll Y$ (cyclicly) rotates $X$ to the left by $Y$ bits.

3. $A = h_0$, $B = h_1$, $C = h_2$, $D = h_3$, $E = h_4$.

4. **80 rounds**: For $t = 0$ to 79 do

   (a) $\boldsymbol{T = A \lll 5 + f_t(B, C, D) + E + W_t + K_t}$.

   (b) $\boldsymbol{E = D,\ D = C,\ C = B \lll 30,\ B = A,\ A = T}$.

5. The output is the five registers $A = A + h_0$, $B = B + h_1$, $C = C + h_2$, $D = D + h_3$, and $E = E + h_4$ (modulo $2^{32}$).

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

6. The function $f_t$ and the values $K_t$ used above are:

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

# How These Attacks Work?

We will concentrate on two levels of the attacks:

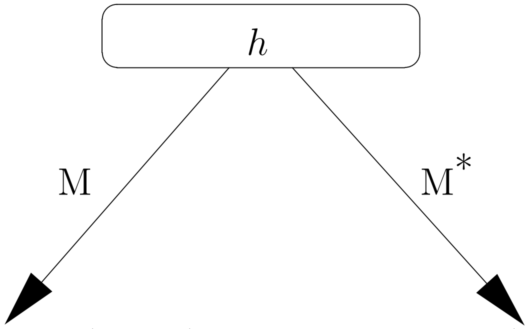
1. The multi-block technique
2. Differential attack on each block

# The Multi-Block Technique

1. Till recently, only one-block collisions were considered $(0 \longrightarrow 0)$

2. **Near-collisions** $(0 \longrightarrow x)$ were studied only recently

> … may be called *certificational properties* – properties which intuitively appear desirable, although they cannot be shown to be directly necessary
>
> ("Handbook of Applied Cryptography", 1996)

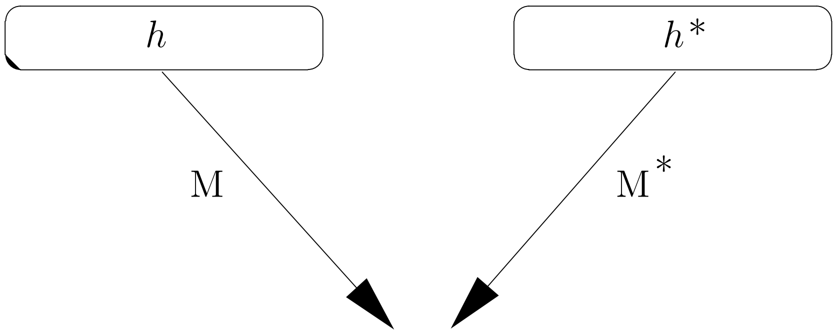



$$\mathrm{compress}(M, h) \quad \approx \quad \mathrm{compress}(M^*, h)$$

# The Multi-Block Technique (cont.)

3. **Pseudo-collisions** ($x \rightarrow 0$) were known, but were not a major research topic:

   > Pseudo-collisions are of much less practical importance than collisions
   >
   > (Dobbertin, "Cryptanalysis of MD5 compress", 1996)

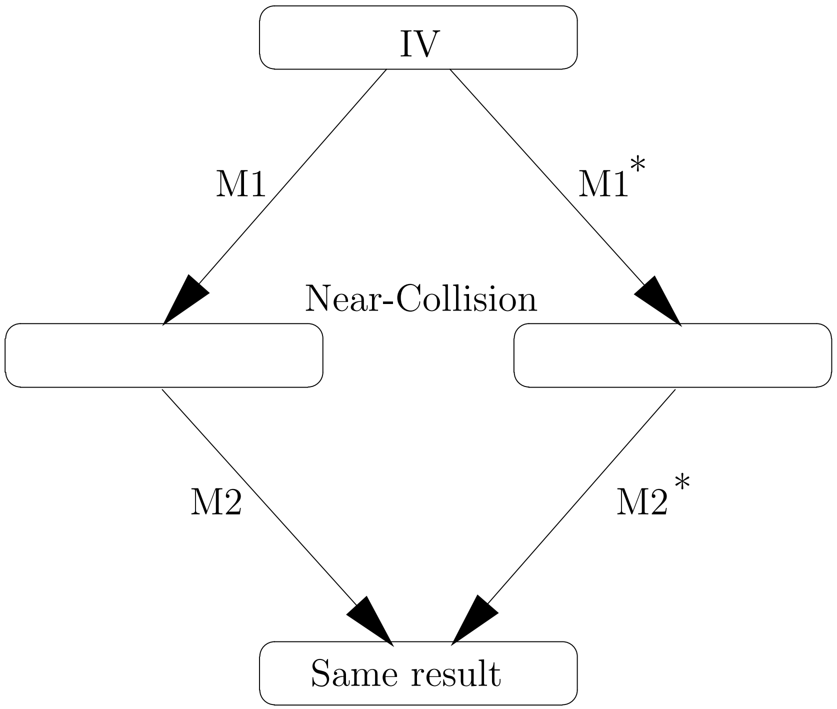



$$\text{compress}(M, h) = \text{compress}(M^*, h^*)$$

4. **Near-Pseudo collisions** ($x \rightarrow y$) were ignored!

It appears that the combination of near-collisions and pseudo-collisions is very strong!

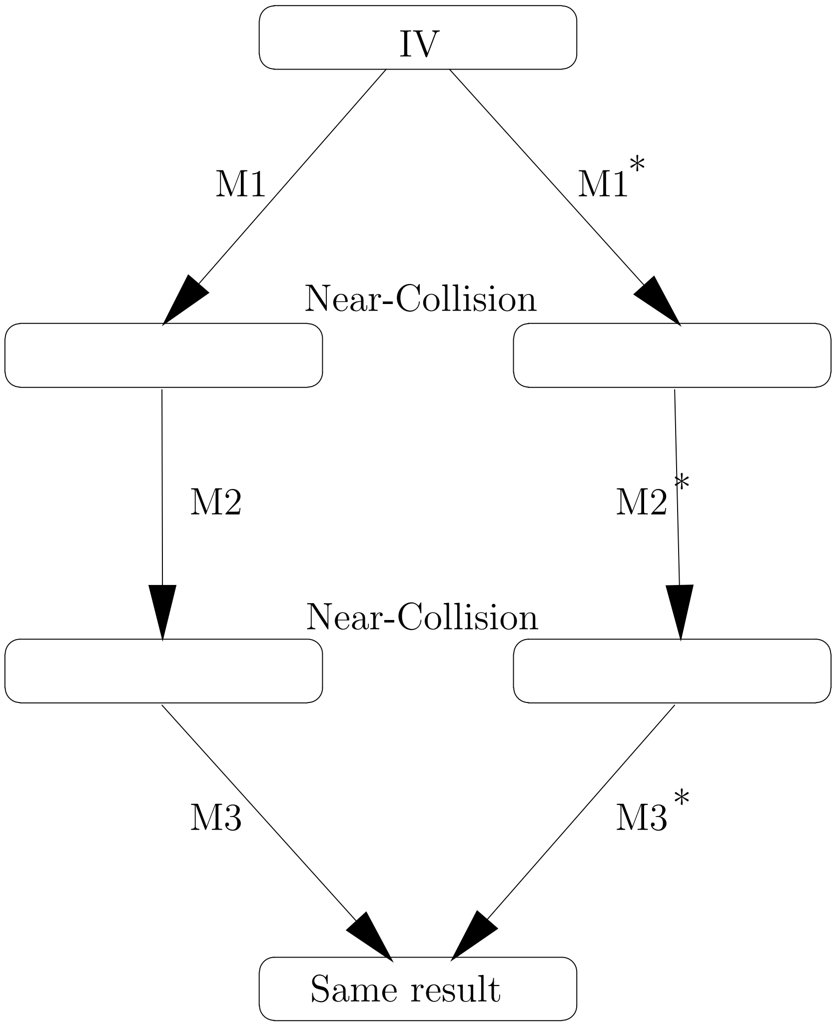
# The Multi-Block Technique (cont.)




The two colliding messages are

$$M = M1 \| M2, \qquad\qquad M^* = M1^* \| M2^*$$

# The Multi-Block Technique (cont.)



# Four-Block Attack of SHA-0

The best one-block attack that follows the limitations of CJ/BC attack, has a complexity of $2^{58}$. With 4-block attack it is reduced to $2^{51}$ (80,000 CPU hrs.).

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A766A602 | B65CFFE7 | 73BCF258 | 26B322B3 | D01B1A97 | 2684EF53 | 3E3B4B7F | 53FE3762 |
| 24C08E47 | E959B2BC | 3B519880 | B9286568 | 247D110F | 70F5C5E2 | B4590CA3 | F55F52FE |
| EFFD4C8F | E68DE835 | 329E603C | C51E7F02 | 545410D1 | 671D108D | F5A4000D | CF20A439 |
| 4949D72C | D14FBB03 | 45CF3A29 | 5DCDA89F | 998F8755 | 2C9A58B1 | BDC38483 | 5E477185 |
| F96E68BE | BB0025D2 | D2B69EDF | 21724198 | F688B41D | EB9B4913 | FBE696B5 | 457AB399 |
| 21E1D759 | 1F89DE84 | 57E8613C | 6C9E3B24 | 2879D4D8 | 783B2D9C | A9935EA5 | 26A729C0 |
| 6EDFC501 | 37E69330 | BE976012 | CC5DFE1C | 14C4C68B | D1DB3ECB | 24438A59 | A09B5DB4 |
| 35563E0D | 8BDF572F | 77B53065 | CEF31F32 | DC9DBAA0 | 4146261E | 9994BD5C | D0758E3D |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A766A602 | B65CFFE7 | 73BCF258 | 26B322B1 | D01B1AD7 | 2684EF51 | BE3B4B7F | D3FE3762 |
| A4C08E45 | E959B2FC | 3B519880 | 39286528 | A47D110D | 70F5C5E0 | 34590CE3 | 755F52FC |
| 6FFD4C8D | 668DE875 | 329E603E | 451E7F02 | D45410D1 | E71D108D | F5A4000D | CF20A439 |
| 4949D72C | D14FBB01 | 45CF3A69 | 5DCDA89D | 198F8755 | AC9A58B1 | 3DC38481 | 5E4771C5 |
| 796E68FE | BB0025D0 | 52B69EDD | A17241D8 | 7688B41F | 6B9B4911 | 7BE696F5 | C57AB399 |
| A1E1D719 | 9F89DE86 | 57E8613C | EC9E3B26 | A879D498 | 783B2D9E | 29935EA7 | A6A72980 |
| 6EDFC503 | 37E69330 | 3E976010 | 4C5DFE5C | 14C4C689 | 51DB3ECB | A4438A59 | 209B5DB4 |
| 35563E0D | 8BDF572F | 77B53065 | CEF31F30 | DC9DBAE0 | 4146261C | 1994BD5C | 50758E3D |

common $h_4 =$ C9F16077 7D4086FE 8095FBA5 8B7E20C2 28A4006B

# Differential Attack on Each Block

1. All the attacks of Joux et al., Biham & Chen, and Wang et al. are differential.

2. They use the fact that by changing a small number of message bits, it is possible to cancel the difference after a few rounds, or keep the Hamming distance low.

3. Differences are usually defined as the XOR of the value in one run and the corresponding value in the other run (or alternatively additive difference or multiplicative difference).

4. Wang defined a new kind of difference in which the difference under two operations is used simultaneously.

# Differential Attack on Each Block (cont.)

**Example**: In SHA-1:

**80 rounds**: For $t = 0$ to $79$ do

1. $\boldsymbol{T = A \lll 5 + f_t(B, C, D) + E + W_t + K_t.}$
2. $\boldsymbol{E = D,\ D = C,\ C = B \lll 30,\ B = A,\ A = T.}$

By careful selection of differences in the $W_t$'s, the created differences can be canceled — **this cancellation process is probabilistic**.

# Differential Attack on Each Block (cont.)

Moreover, the LFSR in the message schedule

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1$$

ensures that the differences will have correction patterns in all their occurrences (but the product probability may be low).

In the case of SHA-0 it is particularly easy, as there is no diffusion between different bit locations (no $\lll 1$).

# Example: 34-Round SHA-1

For attacking 34-round SHA-1 we use the following differences (HW= 2 from round 20 and on):

| Rnd | D.Vec | D&C | Rnd | D.Vec | D&C | Rnd | D.Vec | D&C |
|---|---|---|---|---|---|---|---|---|
| -5 | 00000000 | | 8 | 00000000 | 80000003 | 21 | 00000000 | 00000040 |
| -4 | 00000000 | | 9 | 00000002 | 40000002 | 22 | 00000002 | 00000000 |
| -3 | 00000000 | | 10 | 00000000 | C0000040 | 23 | 00000000 | 80000040 |
| -2 | 00000000 | | 11 | 00000000 | C0000002 | 24 | 00000000 | 80000002 |
| -1 | 00000000 | | 12 | 00000000 | 80000000 | 25 | 00000000 | 00000000 |
| 0 | 00000002 | 00000002 | 13 | 00000000 | 80000000 | 26 | 00000000 | 80000000 |
| 1 | 00000000 | 00000040 | 14 | 00000002 | 80000002 | 27 | 00000000 | 80000000 |
| 2 | 00000002 | 00000000 | 15 | 00000000 | 00000040 | 28 | 00000000 | 00000000 |
| 3 | 00000000 | 80000040 | 16 | 00000000 | 00000002 | 29 | 00000000 | 00000000 |
| 4 | 00000002 | 80000000 | 17 | 00000000 | 80000000 | 30 | 00000000 | 00000000 |
| 5 | 00000000 | 00000040 | 18 | 00000000 | 80000000 | 31 | 00000000 | 00000000 |
| 6 | 00000003 | 80000001 | 19 | 00000000 | 80000000 | 32 | 00000000 | 00000000 |
| 7 | 00000000 | 00000060 | 20 | 00000002 | 00000002 | 33 | 00000000 | 00000000 |

Collisions (after the round) are marked in **pink**. The message difference is marked in blue.

# Example: 34-Round SHA-1 (cont.)

1. The following messages collide after 34 rounds of SHA-1:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| F1641C2B | 242BFDB5 | EAE01E30 | F4BBBA6F | 18D45E8E | DE68AEBA | 74EC8CF9 | FC204957 |
| 45AAA8BF | 1CD3AE7D | D845C2F3 | AC737464 | F25BEBBB | BE5FFF1D | 2ADB2818 | 0B1D13FB |
| F1641C29 | 242BFDF5 | EAE01E30 | 74BBBA2F | 98D45E8E | DE68AEFA | F4EC8CF8 | FC204937 |
| C5AAA8BC | 5CD3AE7F | 1845C2B3 | 6C737466 | 725BEBBB | 3E5FFF1D | AADB281A | 0B1D13BB |

   We found **millions** of such collisions

2. We also have **over 100** near-collisions with 40 conforming rounds, including three-bit difference after 40 rounds, 9 after 43, and 17 after 44

3. And even messages with **43 conforming rounds**

4. There is a lot of freedom for the attacker in this 34-round attack, and thus we can also find...

# Collisions of 34-Round SHA-1 in ASCII

The following colliding pair of messages consists entirely of letters in ASCII:

Message 1: `IkGDqVMwISGGcBMpNHMYavPTsmUlykPTzokJOkwnrSgJSfDmlpeqsmDzWbAjmNxP`

Message 2: `IkgDqRMwISGGcFEpNHEYarPTsmMlymPTzoSJOkSnrWkJSfhmlpmqsmLzWbijmJxP`

In hex the messages are:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 496B4744 | 71564D77 | 49534747 | 63424D70 | 4E484D59 | 61765054 | 736D556C | 796B5054 |
| 7A6F6B4A | 4F6B776E | 7253674A | 5366446D | 6C706571 | 736D447A | 5762416A | 6D4E7850 |
| 496B6744 | 71524D77 | 49534747 | 63464570 | 4E484559 | 61725054 | 736D4D6C | 796D5054 |
| 7A6F534A | 4F6B536E | 72576B4A | 5366686D | 6C706D71 | 736D4C7A | 5762696A | 6D4A7850 |

To get this effect we had to rotate the locations of the disturbances, which caused some increase in the complexity.

# Collisions of 34-Round SHA-1 in ASCII (cont.)

The following collision contains a more meaningful text (case-sensitive):

Message 1: `I Am OilMANgujnPay916472136314$USAkNOWwTkjepMFXGlmfHNGcpodElGfvL`

Message 2: `I am KilMANgunfPay11607213.312$USASNOWSTknipMFtGlmnHNGkpodmlGbvL`

And also this pair:

Message 1: `OhG, not this mess,age notThat onenot U, oh noHRtBMTkKllLlIluvpB`

Message 2: `Ohg, jot this$eess$aga notLhar oneVot q, kd nodRtBETkKdlLlalurpB`

# Weaknesses of the MD4/SHA Family

1. Slow Diffusion in the Message Schedule

2. Slow Diffusion in the Main Function

3. Differential Properties

4. Existence of Neutral Bits

The next slides elaborate on these issues.

# Slow Diffusion in the Message Schedule

1. In MD4 and MD5, the message schedule is very simple: the extra $W_t$'s are copies of the message words in different order.

2. In SHA-0 and SHA-1, the message schedule expands the messages linearly, with slow diffusion.

3. Moreover, the differential corrective patterns are not destroyed by the expansion — the expansion has correct patterns

# Slow Diffusion in the Main Function

1. In all the MD4/SHA family, most registers are just **copied** in each round, i.e., all their differential and other properties are kept unchanged.

2. In all the MD4/SHA family, the diffusion (avalanche effect) is very **slow**, due to the used operations (addition, XOR, and bitwise operations).

3. There are no operations that allow a bit to strongly affect many others at once.

# Differential Properties

1. Differential cryptanalysis works when the attacker can predict the evolution of differences with a high probability.
2. Especially when registers are just copied.
3. It is easy to cancel a difference in the state by creating another difference in the messages.
4. Wang's breakthrough defines a new kind of differences that study simultaneously XOR differences along with additive differences. It is **usually very difficult** to analyze both of these kinds simultaneously, but when registers are copied, it is **easy!**
5. Even if these registers would be XORed with a fixed value between rounds, many of Wang's improvements would not work! (and even Biham/Chen and Joux's results would also be affected)
6. **If** a **better diffusion** would be made in the rounds, these attacks would **not work**.

# Existence of Neutral Bits

Given a pair of messages, **neutral bits** are bits that do not affect the differences of the data for many rounds.

It was recently observed that neutral bits exist for SHA-0 and SHA-1.

They were used to reduce the complexity of attacks against these hash functions.

The design criteria of block ciphers, e.g., strong avalanche criterion and strong diffusion (also called wide trail strategy in AES), ensure that neutral bits cannot exist.

# Our Results on SHA-1 Variants

The following table summarizes Hamming weights (from round 20 and on) for the best differences we could find:

| Rounds | SHA-1 | | | NO-IF | | |
|---|---|---|---|---|---|---|
| | HW | 2B | NC | HW | 2B | NC |
| 34 | **2** | | | 2 | | |
| 35 | 7 | 6 | | **4** | | |
| 36 | **7** | **3** | | 5 | | |
| 37 | 11 | 9 | | **5** | | |
| 38 | 12 | 7 | | 8 | | |
| 39 | 12 | 11 | | 8 | | |
| 40 | 19 | **5** | 5 | 11 | 5 | 5 |
| 41 | 17 | 14 | 6 | 12 | 10 | 6 |
| 42 | 17 | 14 | 7 | 13 | 11 | 7 |
| 43 | 17 | 15 | 8 | 17 | 13 | 7 |
| 44 | 19 | 17 | 9 | 15 | 15 | 8 |
| 45 | 25 | 16 | 10 | 15 | 15 | 10 |
| 46 | 25 | 18 | 10 | 23 | 13 | 10 |
| 47 | 26 | 23 | 12 | 24 | 21 | 12 |

| Rounds | SHA-1 | | | NO-IF | | |
|---|---|---|---|---|---|---|
| | HW | 2B | NC | HW | 2B | NC |
| 48 | 28 | 25 | 13 | 14 | 14 | 13 |
| 49 | 32 | 22 | 15 | 14 | 14 | 14 |
| 50 | 35 | 29 | 16 | 14 | 14 | 14 |
| 51 | 38 | 26 | 19 | 15 | 15 | 15 |
| 52 | 42 | 32 | 19 | 16 | 16 | 15 |
| 53 | 42 | 32 | 20 | $\Longleftarrow$ 16 | 16 | 16 |
| 54 | 39 | 42 | 24 | 36 | 34 | 16 |
| 55 | 39 | 48 | 27 | 39 | 38 | 16 |
| 56 | 41 | 39 | 28 | 41 | | |
| 57 | 61 | 56 | 29 | 42 | | |
| 58 | 58 | 52 | 29 | $\Longleftarrow$ 42 | 17 | |
| 59 | 64 | 53 | 29 | 51 | | |
| 60 | 45 | 45 | | 29 | | |
| 61 | 45 | 38 | | 30 | | |

The probabilities are expected to be on the order of $2^{-3\text{HW}}$ (depending on various details).

# Free-Start collisions

In some cases, the middle rounds of the characteristics may have a very low probability, while the first rounds have a higher probability.

In such cases, searching for free-start collisions may be much faster than finding collisions:

1. Set the values of registers $A$, $B$, $C$, $D$, $E$ in some intermediate rounds.
2. search for a good message that satisfies that next (or previous) 16–20 rounds (by selecting the 16 words of the expanded messages that correspond to these rounds)
3. Apply the attack backwards towards the initial values
4. Compute the (non-standard) initial value
5. Apply the attack forward as usual

It appears that in this case the number of neutral bits is much larger than in the regular case of search for collisions, and that it is easy to set the neutrality threshold to 10 rounds more than in the search for collisions.

# Conclusions from Block Ciphers

1. The same kind of problems existed in the design of **block ciphers** before differential cryptanalysis was introduced.

2. The conclusion that we learned in block cipher cryptanalysis was that there must be a fast diffusion every round of the cipher, and this is the way all recent block ciphers work (e.g., Rijndael/AES, Serpent, RC6, IDEA)

3. **Related-key differential attacks** on block ciphers were not considered till recently as a very important threat when discussing the security of block ciphers. However, in the case of hash functions, this is the exact kind of attacks that make the highest threat, as the message in the hash function is considered the equivalent of the key of the block cipher. Nowadays, every block cipher is designed to be protected against such attacks.

# A Few Additional Comments

1. The general structure of MD4/SHA family is not good enough

2. In order to strengthen this family, the design of SHA-256, and SHA-512 became too complicated, and still the diffusion is slow (*)

3. The parallel runs of RIPEMD protect against naive applications of the attacks, but still the general structure is not optimal (*)

(*) At this moment, no attacks against SHA-256, SHA-512, and RIPEMD-160 are known.

# Strong Design Criteria — Block Cipher Technology

**Conclusion**: For hash functions we should urgently adapt all the design techniques that are used in block ciphers!

Every hash function should be verified by all the tests that designers do for block ciphers.

I call it **strong design criteria** or **block cipher technology**.

- It includes: S boxes, linear transformations, variable rotations, fast and strong diffusion, etc., along with analysis based on the current state of the art in cryptanalysis of block ciphers.

**Example**: In block cipher every designer shows that every data or subkey bit affects all the data after a few rounds (completeness). In MD4/SHA family this is far from being the case.

# Comparison

| Function | Hash Size | Design | Broken | Family | Speed (cy/byte) Pentium | Alpha |
|---|---|---|---|---|---|---|
| Snefru (2,4 passes) | 128–224 | 1990 | 1990 | Special | | |
| MD4 | 128 | 1990 | 1995 | MD4/SHA | 4.7 | 6.3 |
| MD5 | 128 | 1992 | 2004 | MD4/SHA | 7.2 | 10 |
| SHA-0 | 160 | 1993 | 1998,2004 | MD4/SHA | 15 | 9.8 |
| SHA-1 | 160 | 1995 | 2005 | MD4/SHA | 15 | 11 |
| Tiger (and Tiger2) | $\leq$192 | 1995 | | **BC technology** | 21 | 5.2 |
| RIPEMD | 160 | 1996 | | MD4/SHA | 18 | 24 |
| Whirlpool | 512 | 2000 | | **BC technology** | 79 | 35 |
| SHA-224, SHA-256 | 224, 256 | 2002 | | MD4/SHA | 39 | 29 |
| SHA-384, SHA-512 | 384, 512 | 2002 | | MD4/SHA | 83 | 16 |

Speed measurements are from NESSIE performance document.

# An Additional Note

It was observed many years ago that most attacks find collisions of intermediate hash values.

Thus, intermediate values are better to be even longer than the hash values.

In particular, the compressed result can be truncated to form the hash value (e.g., Tiger, SHA-224, SHA-384).

# Why wasn't it Found Earlier?

We had to know it earlier!

1. The main ideas of the recent attacks on hash functions are differential, and were known in block ciphers years ago
2. Differential attacks were applied against hash functions already in 1990
3. Near and Pseudo collisions were mentioned years ago
4. In 1997–2003 there were (almost) no published works on analysis of hash functions in the literature
5. It looks like many of the recent advances had to be found many years ago, even more than a decade ago

# Why wasn't it Found Earlier? (cont.)

But,

1. Block ciphers received much attention, and had an interesting framework
2. Block cipher cryptanalysis techniques were partially used against hash functions, but the general framework was not adapted
3. Related-key attacks were initially viewed as a secondary kind of attack, and their merge with differential cryptanalysis was late and slow
4. There is no parallel to multi-block attacks in block ciphers
5. Large projects, such as AES, shifted the focus of researchers even further towards block ciphers
6. There were widely accepted strong candidates that everybody considered unbreakable (like it was the case for DES in the 1980's)
7. The widely used hash functions had a huge number of rounds, which seemed to hide any possible weakness
8. And, since nobody published, nobody else did

# Summary of Part I

1. The attacks against block ciphers and hash functions are similar in nature
2. Hash functions have to be designed by the same technology as block ciphers, and the same principles and design criteria
3. Only a few current hash functions are already designed in this way (Tiger, Whirlpool)
4. I believe that the newer SHA's and RIPEMD are stronger than those that were broken, but the structure is still inferior, and thus the confidence in them should be lower
5. The standards should adopt long-standing hash functions that use block cipher technology, and **new** such **designs**
6. The MD4/SHA family should be discontinued

# Part II: A Framework for Iterative Hash Functions

**HAIFA**

**HA**sh **I**terative **F**r**A**mework

# Attacks against the Merkle-Damgård Construction

Let $\boldsymbol{m}$ be the hash size, and $\boldsymbol{m_c}$ be the chaining value size. Let $\boldsymbol{n}$ be the number of blocks of a given message $\boldsymbol{M}$, and let $\boldsymbol{k}$ be the number of given messages (for one-of-many attacks).

1. Easily invertible compression functions (the backward attack) — $2^{m_c/2}$ (late 1970's)
2. One-of-many preimage attacks, one-of-many second preimage attacks — $2^m/k$, $2^{m/2}$ (Merkle 1979)
3. Multi-collision — $O(2^{m_c/2})$ (Joux, 2004)
4. Fixpoint and second preimage attacks (long messages) — $2^{m_c}/n$, $O(2^{m_c/2})$ (Dean, 1999) [mainly for Davies-Meyer functions]
5. One-of-many second preimage attacks — $2^{m_c}/kn$, $O(2^{m_c/2})$ (Dean, 1999)
6. Expandable messages and second preimage attacks (long messages) — $O(2^{m_c}/n)$, $O(2^{m_c/2})$ (Kelsey, Schneier, 2005) [any iterative hash function]
7. Applicable also to one-of-many second preimage attacks (they missed to mention) — $O(2^{m_c}/kn)$, $O(2^{m_c/2})$

# Possible Conclusions

We have to accept that second preimages are as easy to find as collisions.

Maybe $m_c$ should be further increased if we require that the security against second preimage attacks should be like against preimage attacks?

Possible partial solution: Bound the length of a message by at most $2^{m/2}$ blocks (or bits) — does not help, as the current limit is even smaller ($2^{64}$ bits$= 2^{55}$ blocks).

# Fix-Points

The standard protection against backward attacks on hash functions with easily invertible compression functions involves mixing the input to the output of the function (Davies-Meyer).

Though the function becomes difficult to invert with this solution, it is still easy to find fix-points:

1. Select the output before the feedback to be zero.
2. Invert the functions to get the input chaining value $h_{i-1}$.
3. The output chaining value equals the input chaining value $h_i = h_{i-1}$.

**Fix-points can be used to find second preimages.**

# Fix-Points (cont.)

**Protection**: Either

1. Apply the mixing more than once, say every 20 rounds in SHA-1.
2. Mix the chaining value into the expanded message (e.g., like a few additional message words, or by XORing them to a few words)
3. In the design of a new compression function, there is a lot of freedom for such a mixing.

- The designer must check that the mixing he suggests does not weaken the function.

This protection against easily-to-find fixpoints does not help against expandable messages.

# Protection Against the Second Preimage Attacks

Kelsey and Schneier proposed to add the block index as an input to the compression function, i.e.,

$$h_i = C(h_{i-1}, M_i, i),$$

instead of

$$h_i = C(h_{i-1}, M_i).$$

But the block index is never kept nowadays in the state of the hash.

We propose to use the current bit count as an additional input

$$h_i = C(h_{i-1}, M_i, \#\text{bits}),$$

where #bits is the number of already hashed bits, including this block.

This #bits is already computed by every hash function, for computing the length for the padding.

## Protection Against the Second Preimage Attacks (cont.)

This solution protects against the second preimage attacks, as each computed block becomes **static** — cannot be used in any other location of any message, just at the original location.

It also ensures that there will not be cycles of chaining values (even if there is some value twice, e.g., as a fix-point, it will not be repeated).

Notice that it is **not needed to increase** the chaining size $m_c$ any more.

Notice that since the chaining size does not necessarily need to be increased, such functions can better fit in small restrictive hardware, e.g., smartcards, with a better security.

# Protection Against Message Expansion

The previous solution also protects against the ability to append data at end of a message, without knowing the message (e.g., when the hash function is used for MAC).

In block $i$ ($i < n$),

$$\begin{aligned} \#\text{bits} &= 512i, \\ h_i &= C(h_{i-1}, M_i, 512i). \end{aligned}$$

In the padding block(s), and in particular in the last block $i = n$:

$$\begin{aligned} \#\text{bits} &= \text{length} \neq 512i, \\ \#\text{bits} &= \text{length} \leq 512i - 65, \\ h_i &= C(h_{i-1}, M_i, \text{length}) \neq C(h_{i-1}, M_i, 512i). \end{aligned}$$

## Protection Against Message Expansion (cont.)

But, in order to append another block to the message, the attacker will have to know

$$C(h_{n-1}, M_n, 512n),$$

rather than

$$C(h_{n-1}, M_n, \text{length}),$$

so it is no longer possible to append extra blocks!

[similar property would not hold if the block index would be used]

# Variable Hash Size

In some functions the hash size can be truncated (e.g., Tiger), or uses a truncated version of another function (e.g., SHA-384).

In some of these cases the initial value is replaced to ensure that the output will be different.

We propose a general construction for such functions.

# Variable Hash Size (cont.)

Let $h_i = C(h_{i-1}, M_i)$ be the compression function of some hash function with $m_c$ bits of chaining, and let $0 < m \leq m_c$ be the (possibly) truncated output size.

Let $IV_m$ be the initial value for the function with the $m$ bits of output.

We propose to select a global $IV$, and compute

$$IV_m = C(m, IV, 0)$$

(or $IV_m = C(m, IV)$), where $m$ is padded in some standard way to fit a block.

# Variable Hash Size (cont.)

**Advantages**:

1. A general method for truncation of the hash size for any hash function
2. $IV_m$ can be computed in advance, so that no extra time is required to hash a message (just like it is done in SHA-384, SHA-512)
3. On the other hand, applications that need several hash sizes, can compute $IV_m$ on the fly, or actually compute

$$\text{truncate}_m(\text{HashFromIV}(m \| M)).$$

# Variable Hash Size (cont.)

**A potential problem**:

1. The compression function is the same for all these functions
2. An attacker may wish to have the same hash value for two different hash sizes (up to truncation)
3. If the attacker has the freedom to select different prefixes, he may apply a collision attack (or birthday attack) on the first few blocks, getting the same chaining value. The rest of the chaining values will be the same, as well as the hash value (up to truncation).
4. In order to solve this possible problem, we propose to change the padding algorithm slightly to include the hash size as well:
   (a) append a single 1, and may 0's.
   (b) **append hash size**.
   (c) append length.

# Variable Hash Size (cont.)

5. It may also be useful to input the hash size as an additional input to the compression function, but at this moment it seems unnecessary.

# Last Modification: Salts: Family of Hash Functions

All the solutions shown till now cannot protect against one-of-many second preimage attacks.

Observe that theoretic definitions of hash functions define **families** of hash functions $h_i()$, $i \in \{0, 1, \ldots, l\}$.

We can adopt families of hash functions as a solution: Users will select a one function of the family every time they compute hash, either at random, or by incrementing by one, or as the frame number or sequence number of the message that is transmitted (there is such a number in every secure communication frame) or of the document (many signature implementations keep sequential numbers anyway), or date, or combinations of those, etc.

[I may have called the member number an $IV$ in the sense of IV in a stream cipher, but the term is already used; will call it **salt** instead, as in UNIX password hashing]

Such a modified definition of a cryptographic hash function should protect against one-of-many preimages, and against other attacks that use several messages generated by the legal user.

# Last Modification: Salts: Family of Hash Functions (cont.)

The salt should be used as an additional input to the compression function (but not change the IV), and should be added to the padding.

Some applications that cannot select a hash function from a family, due to application dependent requirements, may select a fixed member (e.g., salt=0).

The cost of this solution is negligible.

In some sense, a salt can be viewed as a long-term non-colliding value that protects different hashes from colliding, just like the serial number does for different blocks in the same message.

The salt can also be used as a key for keyed hash functions — may be possible to design unified HASH/MAC designs. We did not explore that yet.

**Open question**: What is the required size for the salt? 128 bits? 64 bits? other?

# New Designs

New designs of hash functions can easily apply these guidelines, with compression functions of the form

$$h_i = C(h_{i-1}, M_i, \#\mathrm{bits}, \mathrm{salt}),$$

where $h_0 = IV_m$ is

$$IV_m = C(m, IV, 0, 0),$$

and with padding that include a single 1, many 0's, salt, $m$, and the length of the message.

Note that at the last block #bits=length, thus the length is mixed twice (once through the padding, and once through #bits). In some designs, the mixing of #bits may be light, so a strong mixing of the length through the padding may be necessary. In designs where the mixing of #bits is as strong as the mixing of the message, the length can be removed from the padding.

# Old Designs

Notice that the change of initial values for different hash sizes can easily be combined into any existing iterative function, by prepending a block with the hash size $m$ to any message (and truncating the result to $m$ bits). The change of the padding is however more complicated.

All the changes that add parameters to the compression functions can instead put the extra values as part of the message block, effectively reducing the block size by 64–128 bits.

Some implementations may prefer to add a block with that extra information every 128 blocks, for example, making a tradeoff between the extra protection and the cost in speed and complexity of the application.

The extra fields of the padding can be added at the end of the message before the original padding, without being counted in #bits.

# Summary of Part II

1. We proposed the HAIFA framework for design of new iterative hash functions: a mode of operation with a modified kind of compression functions.
2. This framework protects against most known generic attacks, with a negligible effect on performance.
3. A few attacks can only be protected by increasing $m_c$ (e.g., multi-collisions) — in case they are considered a security risk, then $m < m_c$ should be selected. However, $\boldsymbol{m = m_c}$ is still OK against second preimage attacks.
4. Existing hash functions can be used at some additional cost.
5. We recommend designing new hash function using this framework
6. It may be discussed whether all or only some of the suggestions should be applied.
7. We recommend that functions that do not follow these guidelines (e.g., all existing ones) will be discontinued in the long term.
8. [Some of these changes may be appropriate for block ciphers as well].

# The End