New trends in the design of cryptographic hash functions

Krystian Matusiewicz

Centre for Advanced Computing Algorithms and Cryptography,
Department of Computing, Macquarie University

WKiOI, KNI KUL, 18 May 2007

Agenda

- Introduction to hash functions
- Dedicated hashes from the MD family
- Potential weaknesses of the MD family
- Some alternative approaches
- New designs: LASH and RC4Hash
- Conclusions

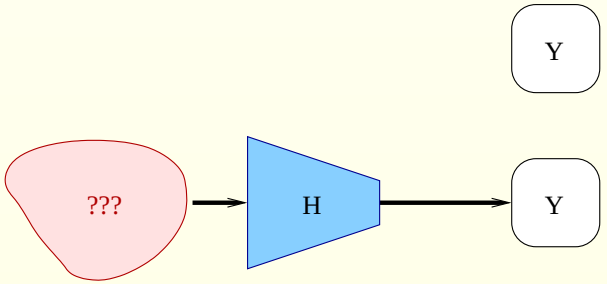
Agenda

- Introduction to hash functions
- Dedicated hashes from the MD family
- Potential weaknesses of the MD family
- Some alternative approaches
- New designs: LASH and RC4Hash
- Conclusions

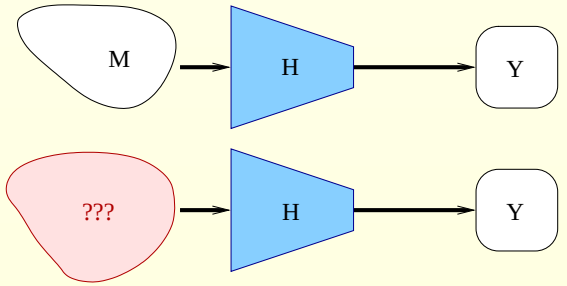
Fundamental properties of cryptographic hash functions

hash function : $H : \{0,1\}^* \rightarrow \{0,1\}^n$

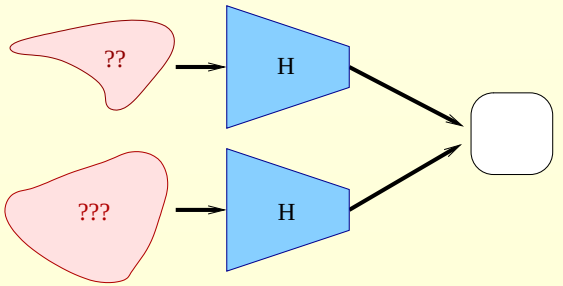
Preimage resistant : Given an output Y of the hash function it is difficult to find any *preimage* - an input X such that $h(X) = Y$.



Second preimage resistant : Given a fixed input X to the hash function and the corresponding output $h(X)$ it is difficult to find a *second preimage* - another input X', $X' \neq X$ such that $h(X) = h(X')$.



Collision resistant : It is hard to find any pair of distinct messages (X, X'), $X \neq X'$ such that $h(X) = h(X')$.



The meaning of cryptographic properties

- preimage resistance $\Rightarrow$ knowledge of a hash value does not reveal anything about the contents of the message

- collision resistance $\Rightarrow$ digest is "almost unique" for the message
  - if a message has been altered, the digest will be different (almost always)
  - if two digests are equal, two corresponding messages are the same (almost for sure)

Digest can be treated as a digital "fingerprint" of a message.

Implicit properties of cryptographic hash functions

In practice quite often we require more from a cryptographic hash function!

- Resistance to near-collisions : it is safe to use truncated hashes
- "Random-looking" output : Generating pseudo-random strings by hashing input data, message authentication codes like HMAC, etc.

Some applications

- digital signatures

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

This is a sample message to be signed. PG Miniconference is great!!!
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.3 (GNU/Linux)

iD8DBQFEjjXm32L5tWc+oHoRAomJAKCpgQnBrMw/sTLqZ99Qt00eejkm6gCgrxve
mXkmY//9J7Nspav5nB+r2wU=
=7Z0s
-----END PGP SIGNATURE-----
```

- password-based user identification

```
gdm:!!:13245:0:99999:7:::
kmatus:$1$5IwJJ/.O$9Em5P./CiGE48TVO2QWbz/:13245:0:99999:7:::
bogo:$1$MPW3Z.Au$SJ1zrNZUA1qBa8nkUF6Ki.:13245:0:99999:7:::
```

- data integrity

The CD ISO image names are listed below. After downloading, you can verify the file against the file is not corrupted. A full installation will require all of the discs for the desired architecture.

- For x86-compatible (32-bit):
  - **FC5-i386-disc1.iso** (sha1sum: 43546c0e0d1fc64b6b80fe1fa99fb6509af5c0a0)
  - **FC5-i386-disc2.iso** (sha1sum: a85ed1ca5b63e2803f29a33ea6a6bc8eb7f63122)

- many others...

Agenda

- Introduction to hash functions
- Dedicated hashes from the MD family
- Potential weaknesses of the MD family
- Some alternative approaches
- New designs: LASH and RC4Hash

How to construct a hash function from scratch?

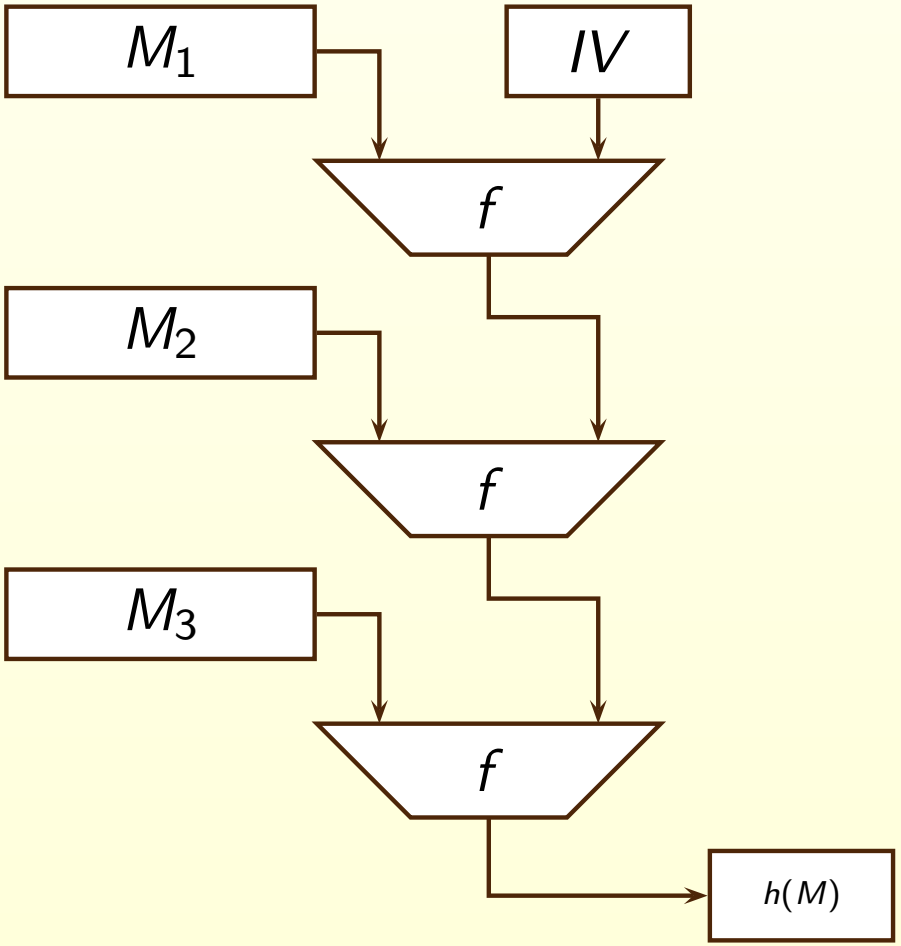
Problem 1: The function has to accept inputs of arbitrary size (in practice at least 2^{64} bits).

Problem 2: The function has to be very efficient in software and hardware

Problem 3: The function has to have desired security properties.

Solution to problem 1: Merkle-Damgård iteration

Compression function - function that maps longer inputs to shorter outputs $f : \{0,1\}^{n+k} \longrightarrow \{0,1\}^k$.



$$
\begin{aligned}
h_0 &\leftarrow IV \\
h_i &\leftarrow f(M_{i-1}||h_{i-1}) \\
&\qquad i = 1, \ldots, d \\
h(M) &:= h_d
\end{aligned}
$$

If the compression function f is secure (one-way and collision-resistant) then the iterative hash function h is also secure.

$\Downarrow$

It is enough to study compression functions.

Solution to problem 2: Use fast processor operations

Pick those processor instructions that are executed fast and operate on many bits at once (i.e. word-based operations)

- bitwise logical operations, i.e. `not`, `and`, `or`, `xor`
- shifts and rotations, i.e. `shl`, `shr`, `rol`, `rot`
- additions and subtractions, `add`, `sub`

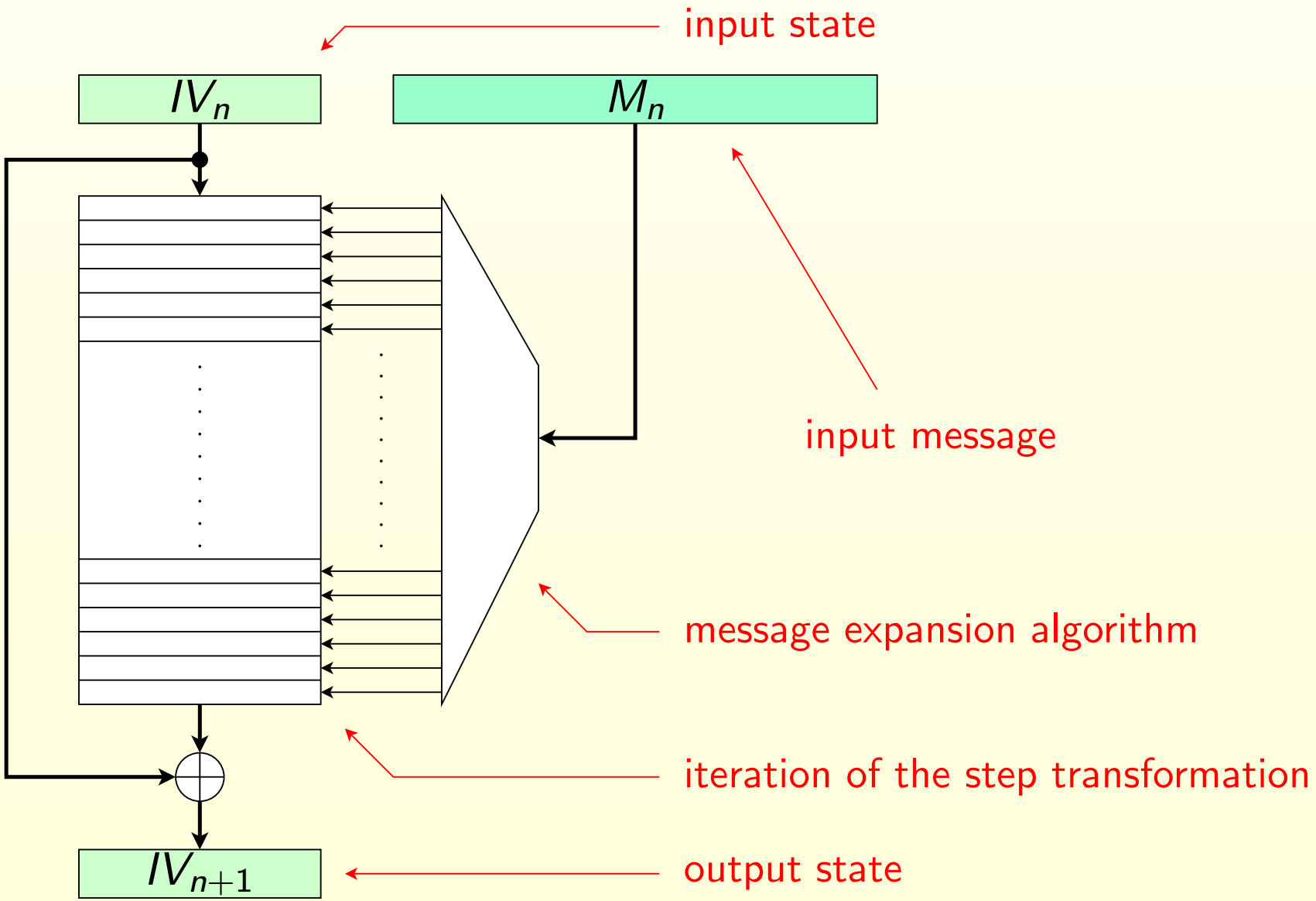
Operations like `mul` or look-up tables are tempting (good source of non-linearity) but not always efficient (`mul`) and require memory (tables).

Solution to problem 3

???

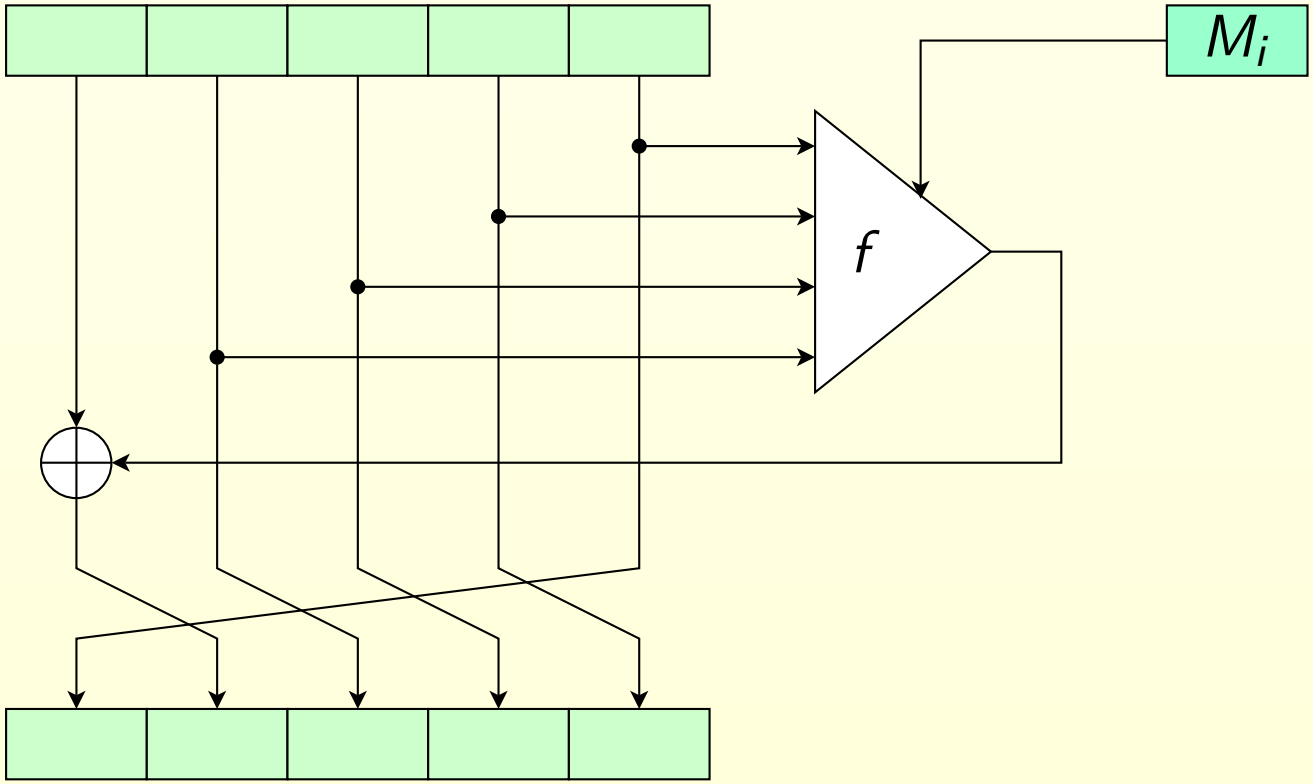
- No solid theory, designs based on engineering intuition and experience.
- Designs “evolving” over time in the environment where cryptanalysts attack existing functions and the best ideas from strongest functions are taken to design new ones

Compression function used by the MD family

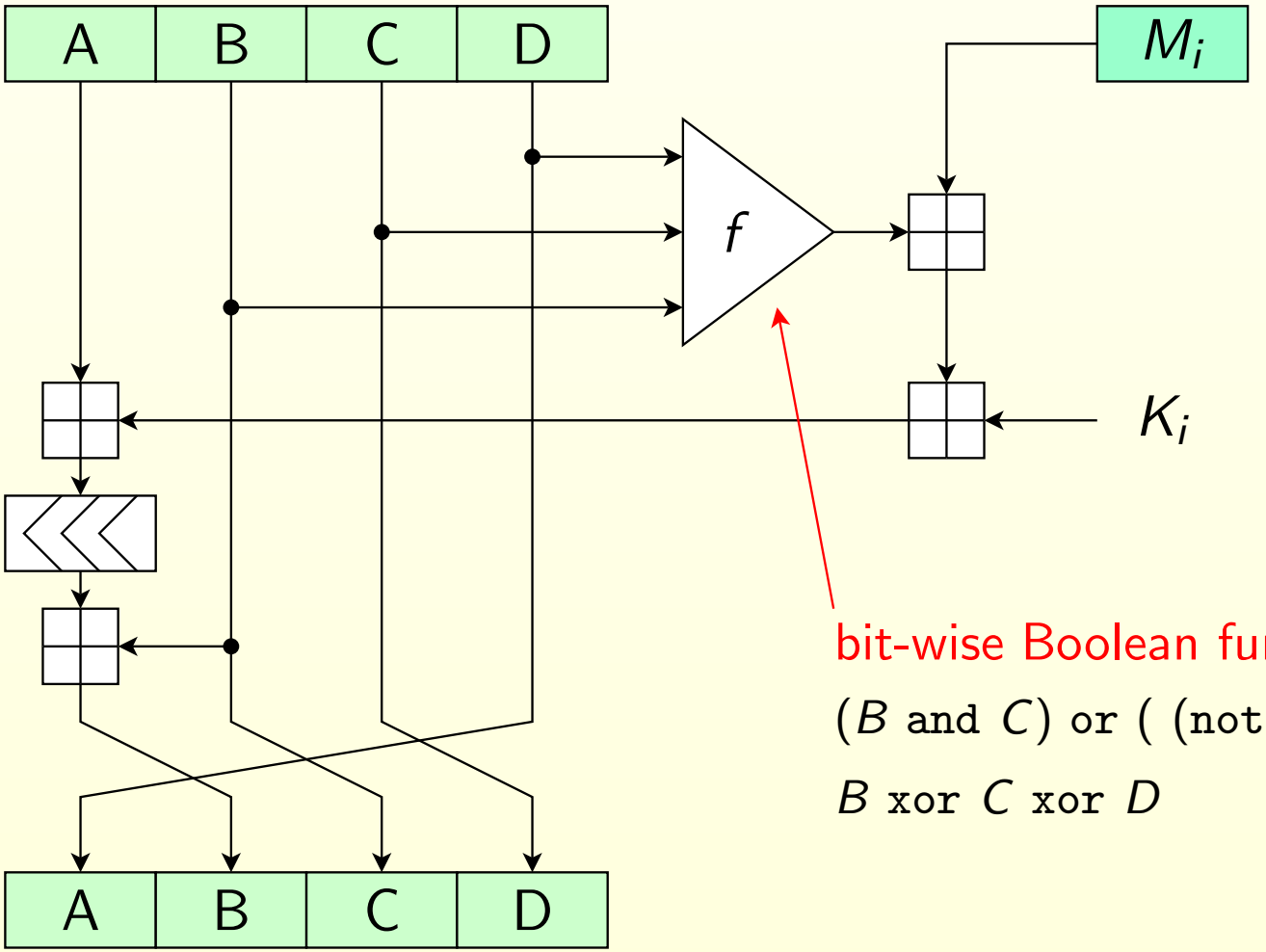


Step transformation

- The step transformation is a message-dependent permutation of input bits.
- It is realised as a source-heavy Unbalanced Feistel Network.



Example : MD5 step transformation



bit-wise Boolean function, e.g.

$(B$ `and` $C)$ `or` (`(not` $B)$ `and` $D)$

B `xor` C `xor` D

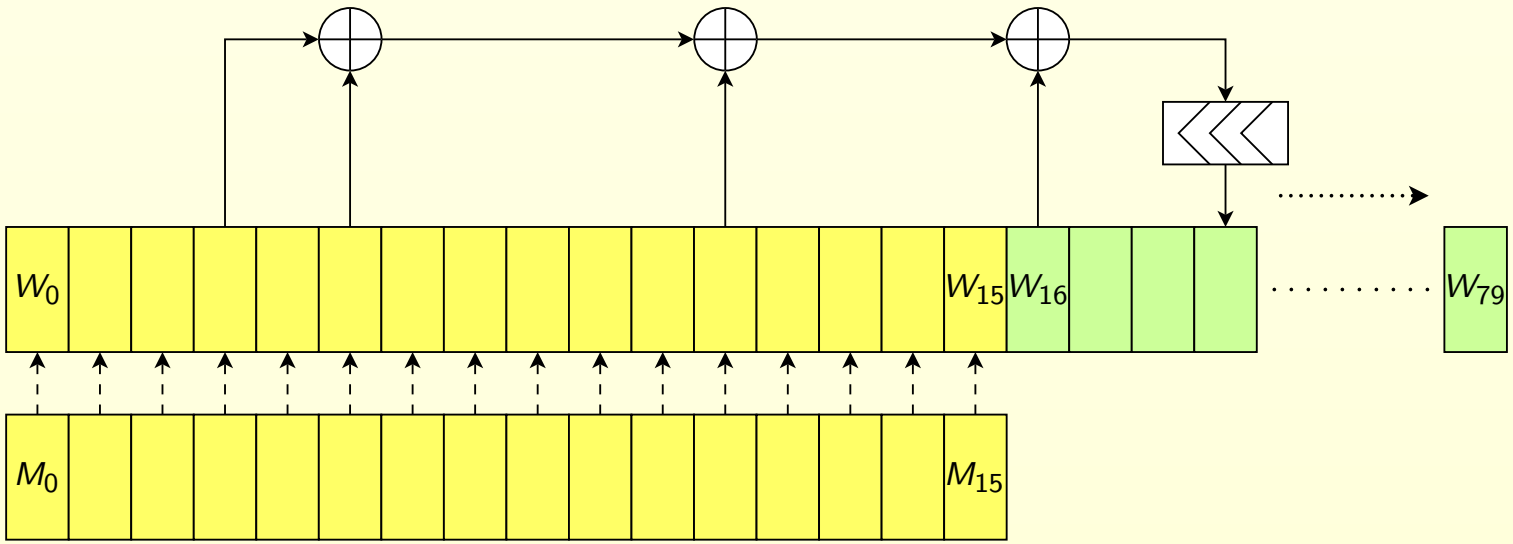
Message expansion algorithms

Produce enough message words to use one in each step

- Permutations of message words: MD4, MD5, HAVAL, RIPEMD
- Recurrent relation : SHA-0, SHA-1, HAS-V, etc

Example (Message expansion in SHA-1)

$$W_i = \begin{cases} X_i & \text{for } i = 0, \ldots, 15 \\ ROL^1(W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) & \text{for } i = 16, \ldots, 79 \end{cases}$$

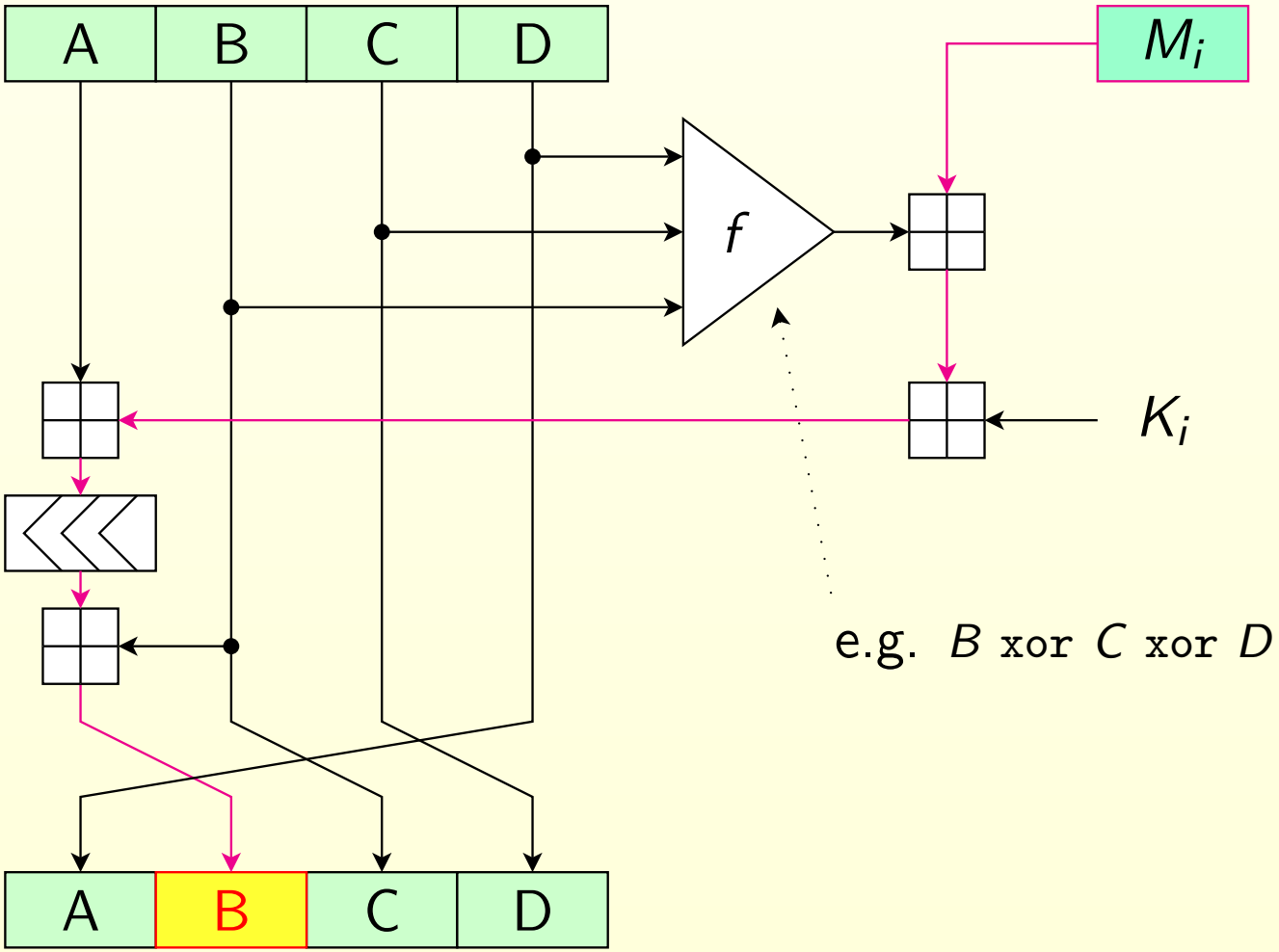


Agenda

- Introduction to hash functions
- Dedicated hashes from the MD family
- Potential weaknesses of the MD family
- Some alternative approaches
- New designs: LASH and RC4Hash
- Conclusions

Potential weaknesses: step transformation

There is a lot of freedom in each step to adjust the value of the target register by manipulating some of the source registers and especially the message word.

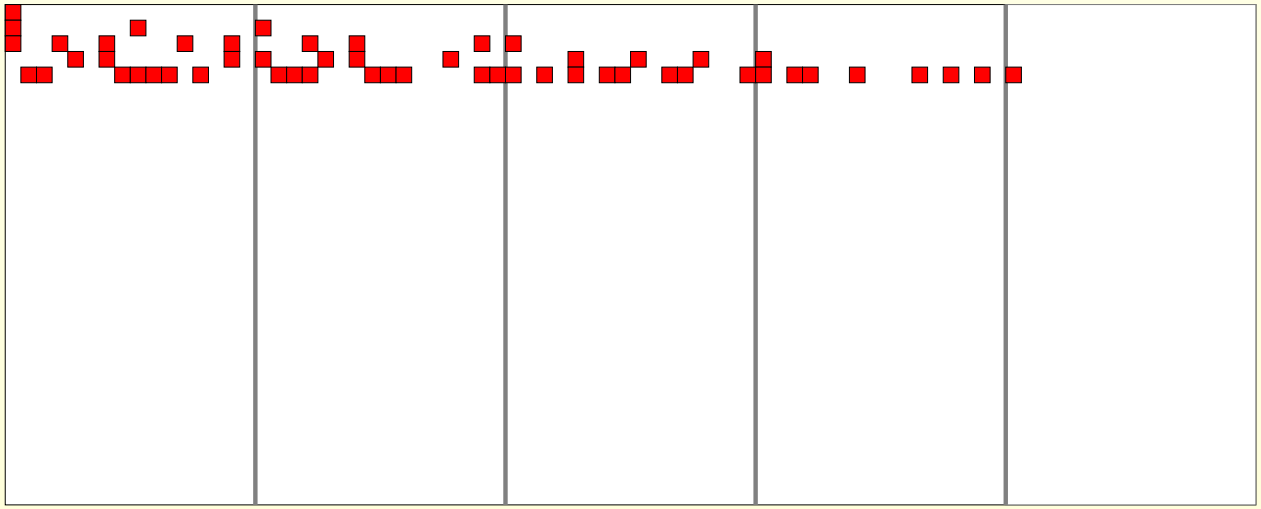


Potential weaknesses: message expansion

In all of the message expansion algorithms it is easy to find very low weight differences.

- In permutation-based expansions: minimum weight of a difference only a few bits
- Even in SHA-1 the lowest weight difference has only 44 bits set

Example (Lowest weight difference in SHA-1 msg. exp.)



Situation so far...

- MD5 totally destroyed
- SHA-1 broken theoretically and up to 70 out of 80 steps in practice
- other hashes of similar structure are believed to be susceptible to attacks similar to Wang's

Agenda

- Introduction to hash functions
- Dedicated hashes from the MD family
- Potential weaknesses of the MD family
- Some alternative approaches
- New designs: LASH and RC4Hash

Alternative approaches

- Strive for provable security
- Use something solid and well-tested
- Try something completely new

In search for provable security

Provable security – the hardness of breaking the function is related to some well-studied hard problem in mathematics/computer science.

The best approach, but there are difficulties:

- very difficult to design!
- less efficient implementations
- usually longer hashes are needed to achieve security

Example (VSH - Very Smooth Hash)

A hash function with provable collision resistance (only!). If one could find collisions for this hash efficiently, it would be possible to speed up the number field sieve and thus factorise numbers much faster.

Use something well-studied and well-understood

AES is probably the most studied modern block cipher.

Base the hash function on AES-like structure:

- Whirlpool – a hash function using cipher W (very similar to AES) in Preneel-Miyaguchi mode of hasing.
- Grindahl-256 – a new proposal that reuses some parts of AES to build a compression function that can be used in different modes

Try something completely different

- Very risky to use such designs in practice (so more of academic interest)
- Very interesting – it can be a beginning of a new trend that can mature and give rise to new constructions

B.Schneier in his blog (Nov,1,2005):

> *We simply don't know enough about designing hash functions. What we need is research, random research all over the map. Designs beget analyses beget designs beget analyses.... Right now we need a bunch of mediocre hash function designs. We need a posse of hotshot graduate students breaking them and making names for themselves. We need new tricks and new tools.*

Agenda

- Introduction to hash functions
- Dedicated hashes from the MD family
- Potential weaknesses of the MD family
- Some alternative approaches
- New designs: RC4Hash and LASH
- Conclusions

Two new designs: a challenge to a cryptanalyst

The reasons why they seem interesting:

- They are very different from MD functions
- They have nice mathematical formulations
- They don't have rigorous security proofs

RC4Hash

- Presented by Donghoon Chang, Kishan Chand Gupta, Mridul Nandi at INDOCRYPT 2006.
- Inspired by the design of RC4 stream cipher
- Permutation-based primitive

Three main parts of the algorithm:

- Message padding
- Iteration of the compression-like function
- Post-processing

RC4Hash : Message padding

INPUT: M – initial message,
$\ell \in \{16, \ldots, 64\}$ – length of the hash in bytes,
OUTPUT: $M_1, M_2, \ldots, M_t$ – 512-bit blocks of the padded msg

The padded message is first produced as

$$pad(M) := bin_8(\ell) \,||\, M \,||\, 1 \,||\, 0^k \,||\, bin_{64}(|M|)$$

where $bin_8(\cdot)$ returns 8-bit binary representation of the argument and $bin_{64}(\cdot)$ returns 64-bit binary representation of the argument and k is the smallest non-negative integer s.t.
$8 + |M| + 1 + k + 64$ is a multiple of 512.

Next, the padded message $pad(M)$ (which has length $512 \cdot t$) is split into 512-bit blocks M_1, $M_2, \ldots, M_t$.

RC4Hash: Iteration of the compression function C

The function is defined as

$$C : \text{Perm} \times \{0..255\} \times \{0,1\}^{512} \rightarrow \text{Perm} \times \{0..255\}$$

where Perm is the family of all permutations on the set $\{0,\ldots,255\}$. (Basically, Perm$=S_{256}(\{0,1,\ldots,255\})$).
The internal state is a permutation of bytes plus an index (one byte).
The iteration works as follows:

$$(S_0, j_0) := (S^{IV}, 0)$$
$$(S_m, j_m) := C(S_{m-1}, j_{m-1}, M_m) \quad \text{for } m = 1,\ldots,t$$

The compression function C

INPUT: (S, j) – compression state,
$X[0..63]$ – 512 bits of the message block seen as 64 bytes
OUTPUT: (S, j) – new compression state,

for i=0 to 255 **do**
  $j \leftarrow (j + S[i] + X[r(i)]) \mod 256$
  $Swap(S, i, j)$
end for
return (S, j)

Here $r : \{0..255\} \longrightarrow \{0..64\}$ is a function that plays the role of message byte reordering.

Note that in fact this procedure multiplies the initial permutation by a number of transpositions!

RC4Hash: post-processing

- Compute $S_{t+1} \leftarrow S_0 \circ S_t$ and $j_{t+1} \leftarrow j_t$,
- The final hash value h is

$$h \leftarrow HBG_\ell(OWT(S_{t+1}, j_{t+1}))$$

where *OWT* (One-Way Transformation) and *HBG* (Hash Byte Generation) are two functions.

$OWT(S, j)$

$Temp1 \leftarrow S$
for $i = 0$ to 511 **do**
  $j \leftarrow (j + S[i \mod 256]) \mod 256$
  $Swap(S, i, j)$
end for
$Temp2 \leftarrow S$
$S \leftarrow Temp1 \circ Temp2 \circ Temp1$
return (S, j)

$HBG_\ell(S, j)$

for $i = 1$ to ℓ **do**
  $j \leftarrow (j + S[i]) \mod 256$
  $Swap(S, i, j)$
  $Out[i] \leftarrow S[(S[i] + S[j]) \mod 256]]$
end for
return Out

Why is it interesting?

- Nice formulation in terms of operations in the symmetric group S_{256}
- Properties of such transformations are not yet studied (e.g. similar VMPC transformation)
- Can we break it or show some security properties in a formal way?

LASH hash function

LASH-x is a family of hash functions producing x bit digest where $x = 160, 256, 384, 512$.
The key parameter is $n = 640, 1024, 1536, 2048$ (basically, $n = 4 \cdot x$). Then we define $m = n/16$.

The algorithm can be divided into three parts.

- Message padding
- Iteration of the compression function
- Final transformation

LASH message padding

- Initial message M, $|M| = l$ is padded by a single '1' bit and enough zeros to obtain the total length equal to a multiple of $8 \cdot m$ bits.
- Padded message is split into $k = \lceil l/8m \rceil$ blocks
- an extra block is appended that contains the original lengh of the message l
- blocks are fed to the compression function one by one

LASH: Iteration of the compression function

Message blocks $M_1, M_2, \ldots, M_k, M_{k+1}$ are processed as follows

$$
\begin{aligned}
r_0 &\leftarrow [0, 0, \ldots, 0] \\
r_i &\leftarrow f(r_{i-1}, M_i) \quad \text{for } i = 1, \ldots, k+1
\end{aligned}
$$

LASH compression function

Let $r, s \in \mathbb{Z}_{256}^m$ be two vectors of bytes. Define

$$rep(\cdot) : \mathbb{Z}_{256}^m \rightarrow \mathbb{Z}_{256}^{8m}$$

as the function that returns binary representation (i.e. sequence of 0's and 1's) of the bytes provided as the argument. For example,

$$rep([255, 1, 128]) = [1, 1, 1, 1, 1, 1, 1, \ 1, 0, 0, 0, 0, 0, 0, 0, \ 0, 0, 0, 0, 0, 0, 0, 1]$$

LASH compression function

The compression function $f : \mathbb{Z}_{256}^{2m} \rightarrow \mathbb{Z}_{256}^{m}$ is defined as

$$f(r, s) = (r \oplus s) + H \cdot (rep(r)||rep(s))^T$$

where

- $\oplus$ means byte-wise XOR (i.e. $[255, 1] \oplus [127, 3] = [128, 2]$,
- $+$ denotes byte-wise addition modulo 256 (i.e. $[255, 1] + [127, 3] = [126, 4]$)
- $rep(\cdot)$ is the representation function defined before,
- $||$ means concatenation of vectors,
- H is a special circulant matrix of dimensions $m \times n$.

LASH: matrix H

$$H = \begin{pmatrix} a_0 & a_{n-1} & a_{n-2} & \dots & a_2 & a_1 \\ a_1 & a_0 & a_{n-1} & \dots & a_3 & a_2 \\ a_2 & a_1 & a_0 & \dots & a_4 & a_3 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ a_{m-1} & a_{m-2} & a_{m-3} & \dots & a_{m+1} & a_m \end{pmatrix}$$

Values $a_0 \dots, a_{n-1}$ are residues modulo 256 of numbers $y_0, \dots, y_{n-1}$ generated using "Pollard rho" PRNG

$$y_0 = 54321$$

$$y_i = (y_{i-1}^2 + 2) \mod (2^{31} - 1) \quad \text{for } i = 1, \dots, n-1$$

Example (mini compression function of LASH)

Minimal example: assume bytes have 4 bits and we compress four such minibytes $r = (15, 12)$, $s = (11, 1)$ to two-minibyte output:

$$f(r,s) = ([15,12] \oplus [11,1]) + \begin{pmatrix} 1 & 6 & 11 & 10 & 15 & 13 & 7 & 4 \\ 4 & 1 & 6 & 11 & 10 & 15 & 13 & 7 \end{pmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \hline 1 \\ 1 \\ 0 \\ 0 \\ \hline 1 \\ 0 \\ 1 \\ 0 \\ \hline 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

LASH: final transformation

After processing the last block and getting as a result the final block of m bytes the digest is obtained by taking only 4 most significant bits of each byte.

For example, when $n = 640$ the state of the function is 40 bytes and the final digest is 20 bytes $= 160$ bits long.

Why is it interesting?

- Finding collisions for the matrix multiplication part, i.e.

$$H \cdot [rep(r)||reps(s)]^T$$

  is proved to be very difficult (as difficult as worst instances of some NP-hard problems)
- But there is that "mode of operation" added and the hash is chopped at the end, so maybe it's not as secure as the underlying problem?
- Nice mathematical theory behind it (lattice reduction problems - geometry of numbers!)

Agenda

- Introduction to hash functions
- Dedicated hashes from the MD family
- Potential weaknesses of the MD family
- Some alternative approaches
- New designs: LASH and RC4Hash
- Conclusions

NIST competition for hash functions

National Institute of Standards and Technology announced a public competition for the new hashing algorithms similar to the one that was used to select AES.

- Planned deadline for submission of new hashes: 3rd quarter of 2008.
- The winner is planned to be announced in 4th quarter of 2011

Hash functions researchers have a lot of work (and fun) before them!

Conclusions

- Cryptographic hash functions have many applications
- Designing secure hash functions seems to be a formidable task
- Much more research into hashing is nedeed
- Breaking ad hoc designs is fun!
- With the current NIST competition hash function research is going to be a realy hot topic in the next few years

Thank you!