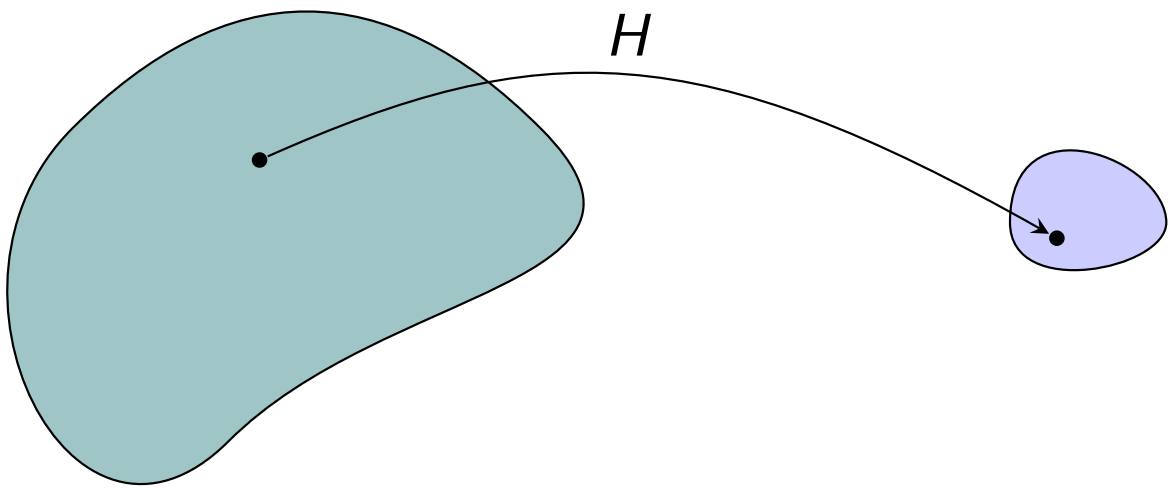
Introduction to cryptographic hashing

Krystian Matusiewicz

PWr, 2010.06.14

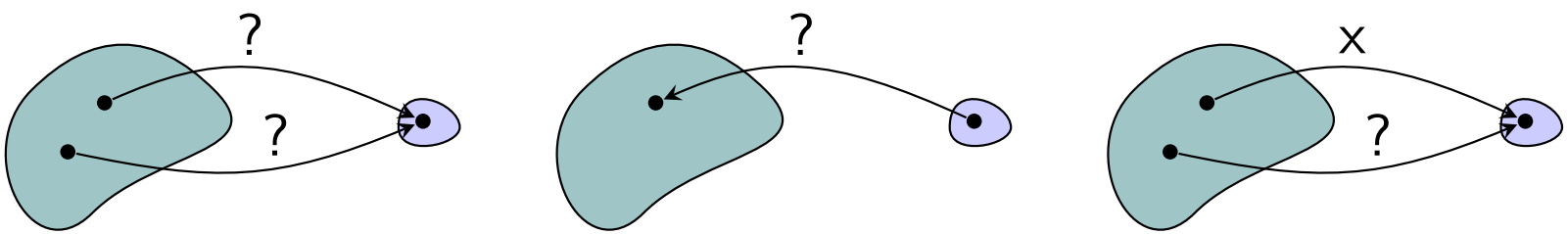
Hash function

- A hash function H maps strings of arbitrary length to short fixed-length bit strings (e.g., 256 bits)
- Provides a "fingerprint" of the message



Security properties

- Collision: distinct x and x' with $H(x) = H(x')$
- Preimage: Given $H(x)$, find x' such that $H(x') = H(x)$
- Second preimage: Given x, find $x' \neq x$ such that $H(x) = H(x')$.



Ideal model: Random Oracle

- Theoretical model of an ideal, structureless function
- A device with memory
- On a query, check if such question was asked before
  - If yes, return the previous answer (consistency)
  - If not, pick a random, uniform value, return it and remember the pair (query, answer)

Birthday attack

- Collision attack (n-bit hash function)
- After q queries we have $\binom{q}{2} = q(q-1)/2$ hash pairs (h, h')
- Probability that $h = h'$ is about 2^{-n}
- Hence, with 2^n pairs, we probably have a collision
- With $q = 2^{n/2}$, we have about 2^n pairs
- Hence, birthday attack in time about $2^{n/2}$.

Brute force preimage and second preimage attacks

- Ideal n-bit hash function
- Best attack: try random messages
- Probability for each is 2^{-n}
- I.e., try 2^n messages.

Typical design method

- Design a *compression function* $f : \{0,1\}^b \to \{0,1\}^n$
- Initial n-bit state value h_0
- Message $M = m_1 \| m_2 \| m_3 \| \cdots \| m_t$, $|m_i| = b - n$
- $h_1 = f(h_0 \| m_1)$, $h_2 = f(h_1 \| m_2)$, ...
- Final value h_t is the hash
- (Padding may be necessary).

Merkle-Damgård

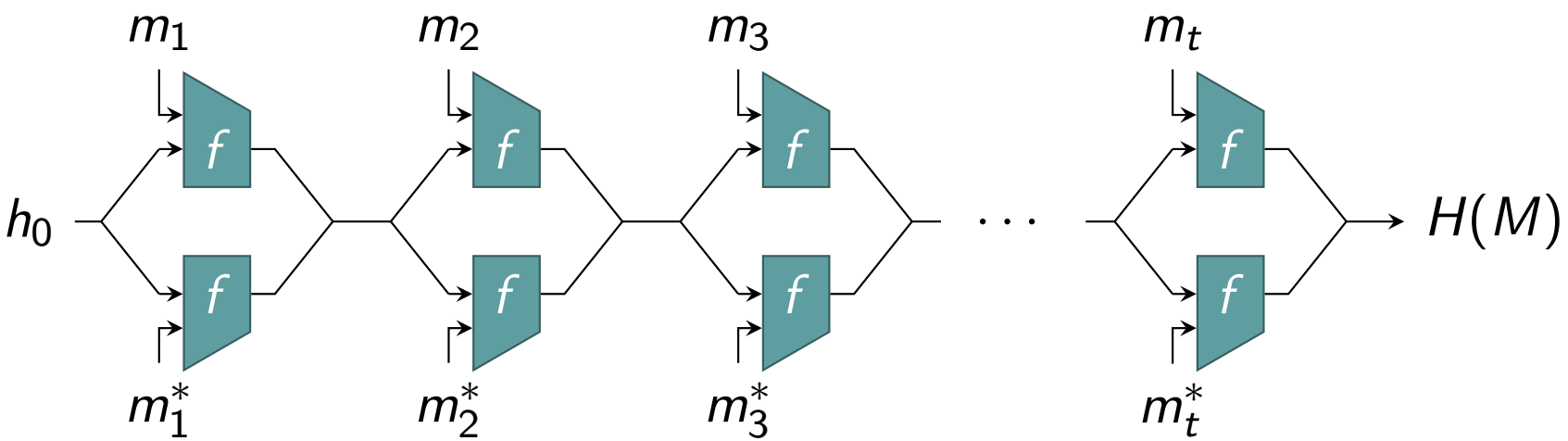
- Take a collision resistant *compression function* f
- Pad message M by appending '0' bits *and* the length $|M|$ (MD-strengthening)
- Iterate as described
- Now you have a collision resistant *hash function*.

Multicollisions

- A set of $r \geq 2$ messages all having the same hash
- Complexity for an ideal hash function:

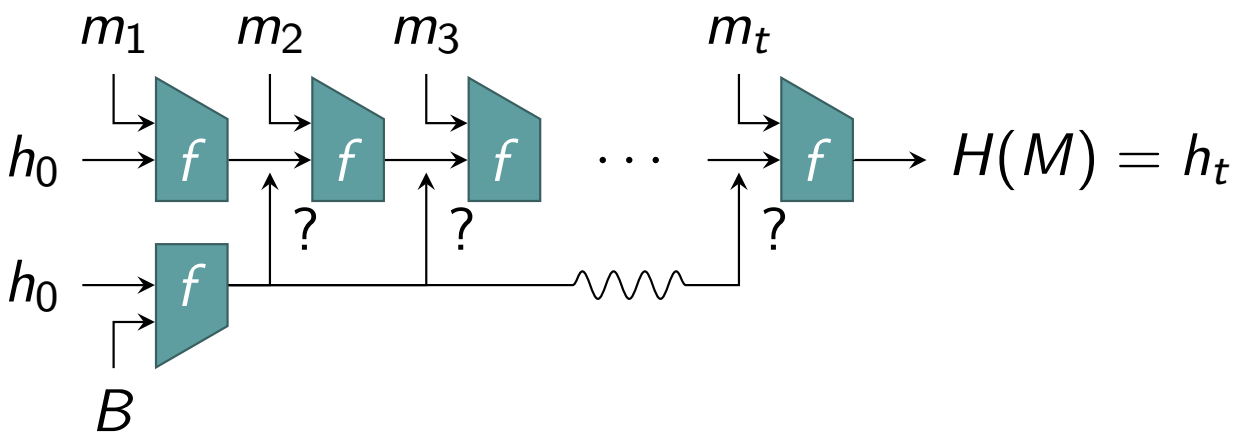
$$2^{n(r-1)/r} \qquad \text{tends to } 2^n \text{ when } r \text{ large}$$

- Joux, 2004: 2^t-collision in time $t2^{n/2}$.



Second preimages for long messages

- Let M be a very long message
- A second message can map to any intermediate value to form a second preimage
- Problem: message lengths don't fit (MD-strengthening)
- Solution?



Solution 1

- Find a *fixed point* in f (Davies-Meyer)
- $f(h, m) = h$
- Repeat m as many times as necessary to make lengths fit
- Sub-problem: $h \neq h_0$
- Sub-solution:
  - find $2^{n/2}$ fixed points, place in list L
  - find linking message block b s.t. $f(h_0, b) \in L$
  - Time about $2^{n/2}$ when Davies-Meyer.

Solution 2

- Yet another application of Joux
- Find colliding messages of lengths 1 and 2 (blocks)
- Find colliding messages of lengths 1 and 3
- Find colliding messages of lengths 1 and 5
- ...
- Find colliding messages of lengths 1 and $2^k + 1$
- Combine blocks to form message of length $k + 1$, $k + 2$, ..., $2^{k+1} + k$ (2^k different message lengths).

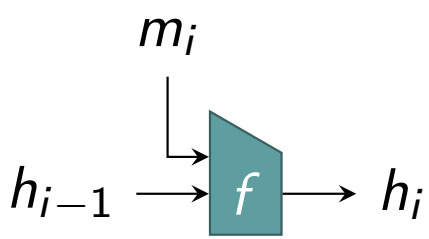
Security reductions

Security reductions: to have a secure hash function, we need a secure compression function

- Collision resistance of order $2^{n/2}$ calls for n-bit output
- (Second) preimage resistance of order 2^n calls

Compression function

$$f : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$$
$$f(h_{i-1}, m_i) = h_i$$



Example: Compression function as secure as factoring

- Proposed by Goldwasser, Micali, Rivest '84
- Let $N = pq$ where p, q are primes $\equiv 3 \pmod 4$
- Compression function $h : \{0,1\} \times Q_R(N) \rightarrow Q_R(N)$ defined as

$$h(x, y) = a_x \cdot y^2 \pmod N$$

  where $Q_R(N)$ is the set of quadratic residues mod N and $a_0, a_1 \in Q_R(N)$ are randomly chosen.
- We can prove that someone who is able to find collisions in that function can also factor the modulus.

Constructions with asymptotic security reductions

Many constructions proposed with reductions to

- factoring
- discrete log problem
- shortest vector problem in a lattice

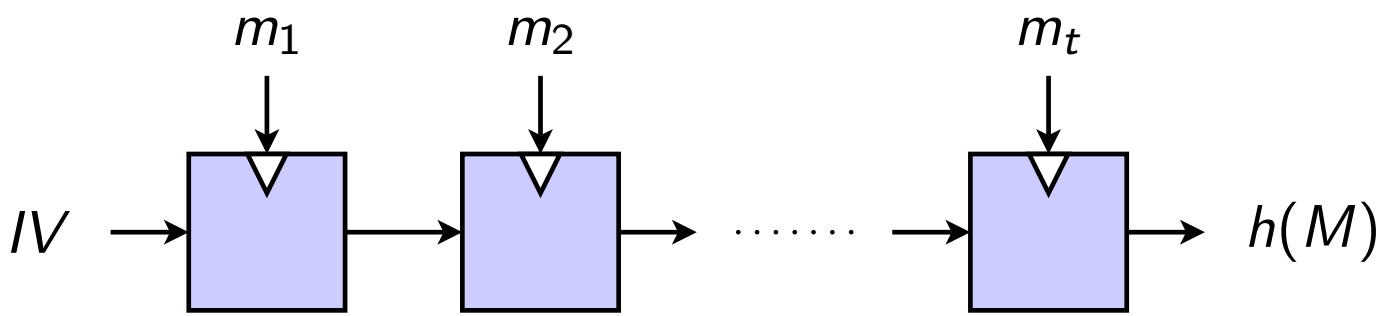
Problems:

- Speed
  (e.g.: long modular mutliplication and squaring)
- Digest length for instances secure in practice
  (e.g.: secure RSA modulus is $\approx$ 2048 bits)
- Structure
  (using RSA signature with factoring-based hash???)

Alternative proposals: block-cipher based hashes

- We know how to construct secure block ciphers
- Build compression functions using a secure block cipher

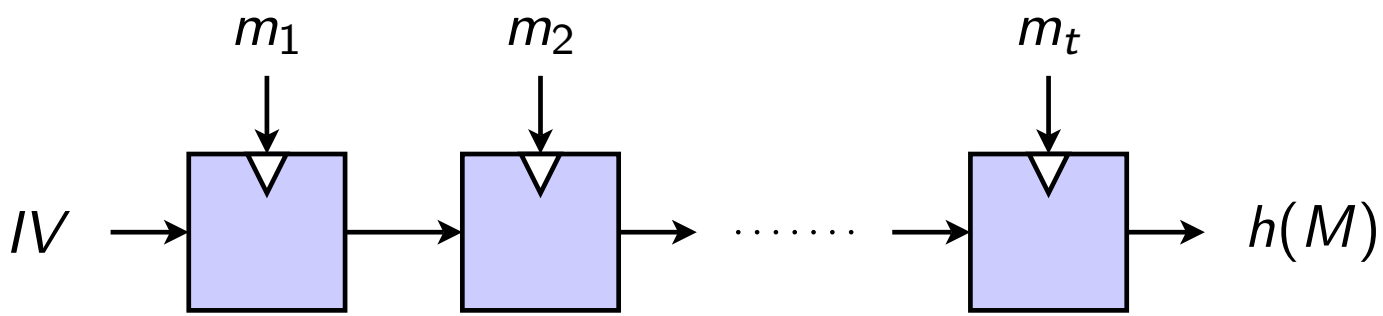
Example: Rabin hash [Rabin 78]



Alternative proposals: block-cipher based hashes

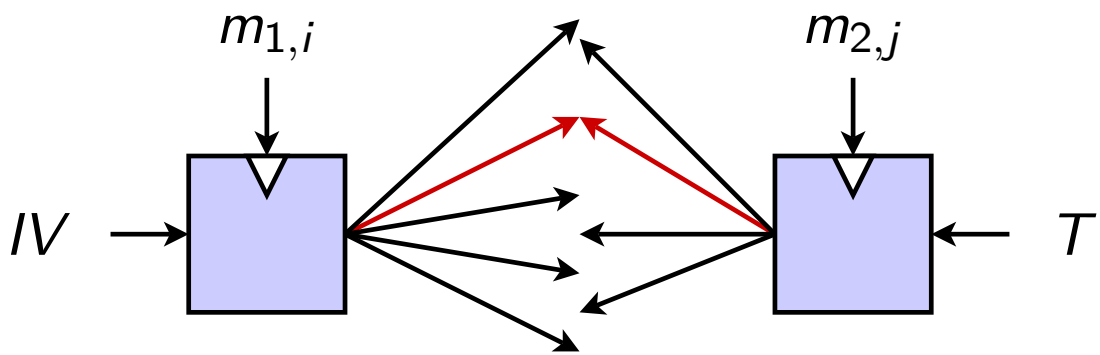
- We know how to construct secure block ciphers
- Build compression functions using a secure block cipher

Example: Rabin hash [Rabin 78]



Can you see a problem?

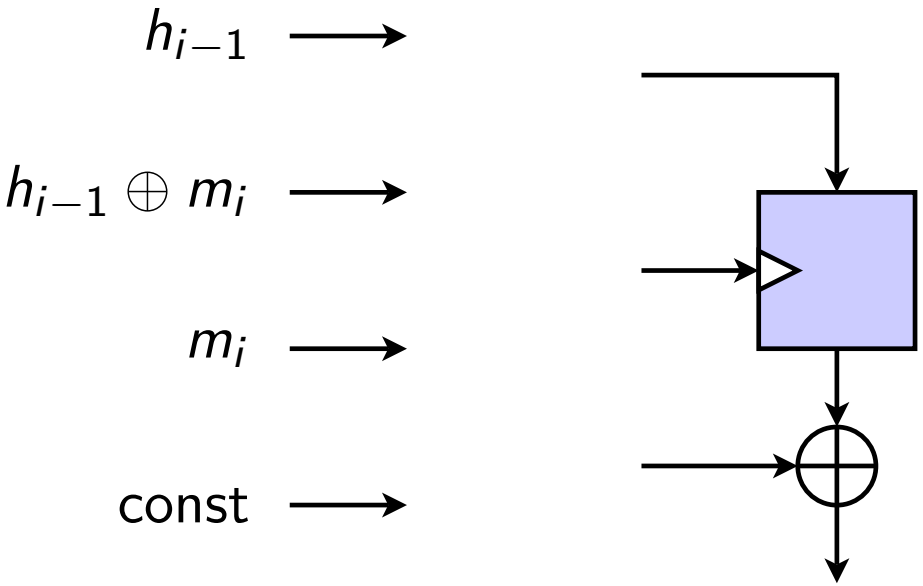
Preimage attack on a Rabin hash



- Block cipher with known key is reversible
- For a given target value T pick $2^{n/2}$ messages $m_{2,i}$ decrypt T and store
- Encrypt IV with $2^{n/2}$ messages $m_{2,i}$ and store results
- Look for a match: by birthday paradox expect to find one
- We found a preimage with complexity $2^{n/2}$, below required 2^n

Systematic approach: PGV modes

- Use block cipher without modifications to construct a secure compression function?
- Consider possible inputs to the compression function...

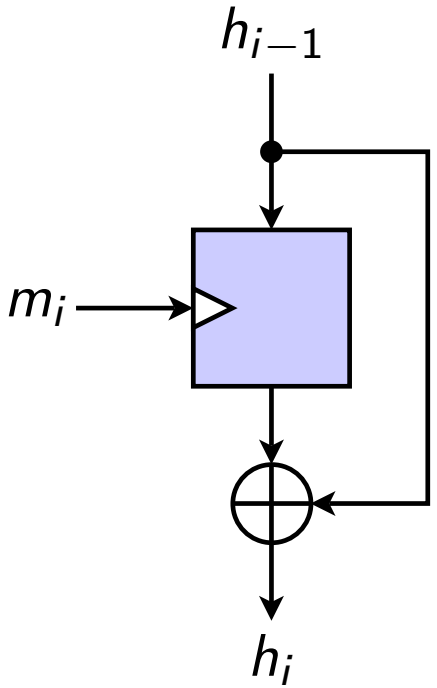


PGV modes

- 4^3 possible combinations - 64 modes in total
- 12 modes have been proven to be secure

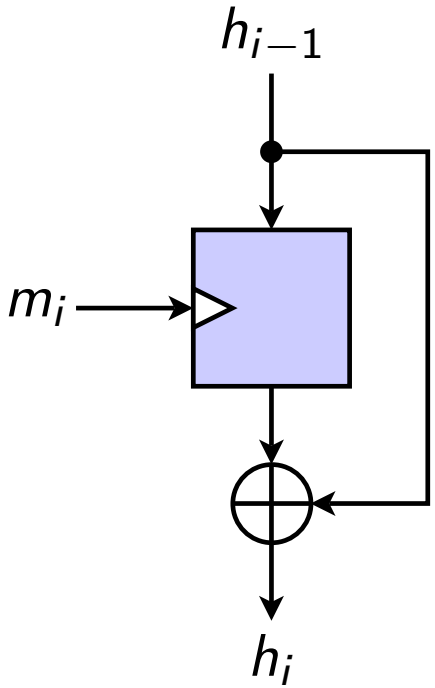
PGV modes

- 4^3 possible combinations - 64 modes in total
- 12 modes have been proven to be secure
- most popular: Davies-Meyer mode



PGV modes

- 4^3 possible combinations - 64 modes in total
- 12 modes have been proven to be secure
- most popular: Davies-Meyer mode



- Nice feature: key can be longer than the block size

One more problem with block ciphers

- Common block ciphers have block length 64 or 128 bits (AES)
- Generic birthday attack works then with complexity 2^{32}, 2^{64}
- Within reach of today's computational possibilities

Two approaches:

- Use two block cipher calls for one compression function

One more problem with block ciphers

- Common block ciphers have block length 64 or 128 bits (AES)
- Generic birthday attack works then with complexity 2^{32}, 2^{64}
- Within reach of today's computational possibilities

Two approaches:

- Use two block cipher calls for one compression function
  - Half the speed of the block cipher

One more problem with block ciphers

- Common block ciphers have block length 64 or 128 bits (AES)
- Generic birthday attack works then with complexity 2^{32}, 2^{64}
- Within reach of today's computational possibilities

Two approaches:

- Use two block cipher calls for one compression function
  - Half the speed of the block cipher
- Design a custom block cipher with large block length

One more problem with block ciphers

- Common block ciphers have block length 64 or 128 bits (AES)
- Generic birthday attack works then with complexity 2^{32}, 2^{64}
- Within reach of today's computational possibilities

Two approaches:

- Use two block cipher calls for one compression function
  - Half the speed of the block cipher
- Design a custom block cipher with large block length

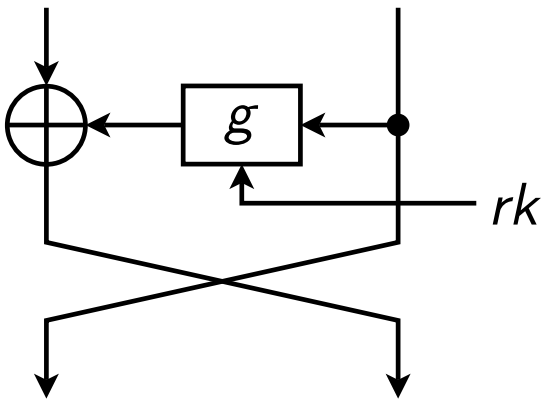
MD (Message Digest) family of compression functions

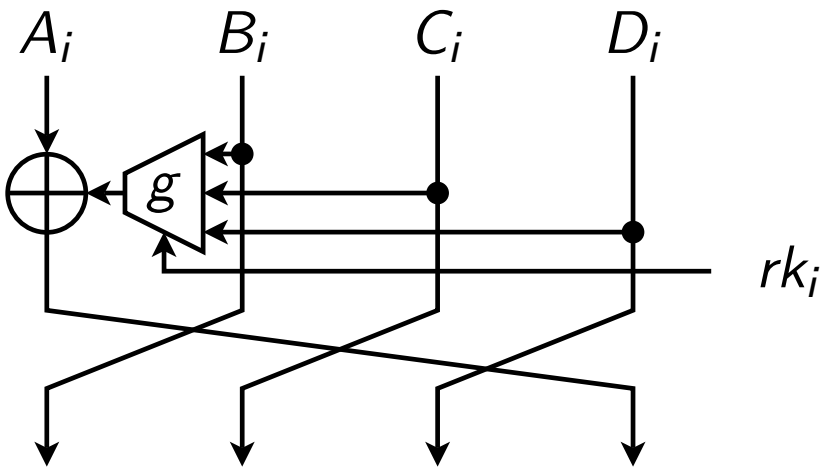
- Direction set by R. Rivest's designs MD4 and MD5
- Followed by SHA-0, SHA-1, HAVAL and many others

Basic design choices:

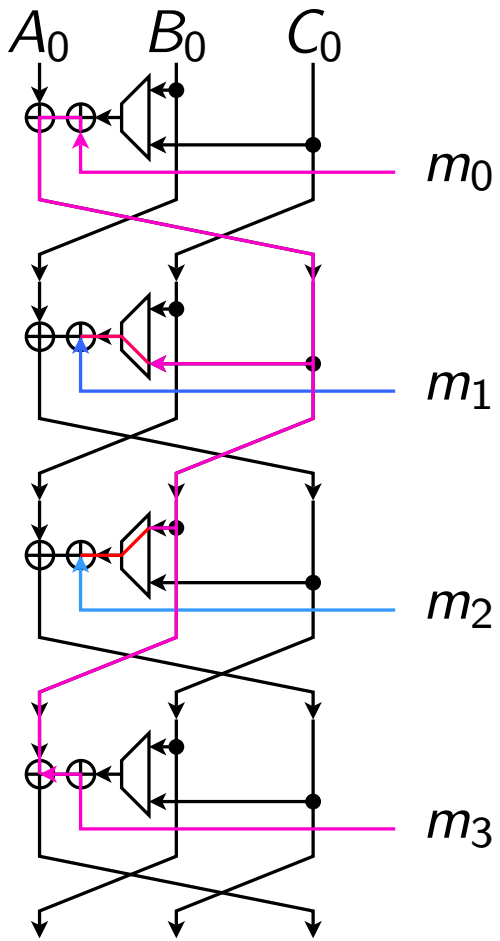
- SPEED in software
- 32-bit word oriented operations
- larger block size

Unbalanced Feistel Network



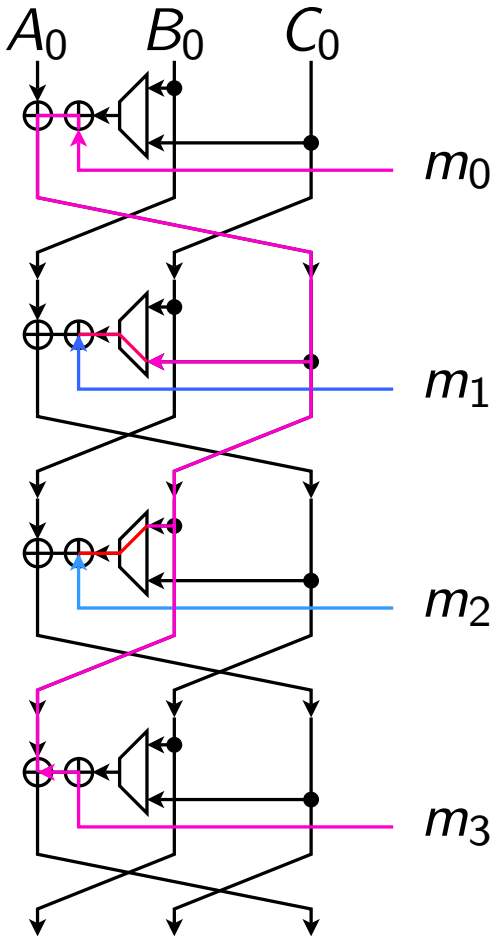


Disturbance-corrections strategy



Introduce a difference in m_0
Correct difference in step 1 with m_1
Correct difference in step 2 with m_2
Correct difference in step 3 with m_3

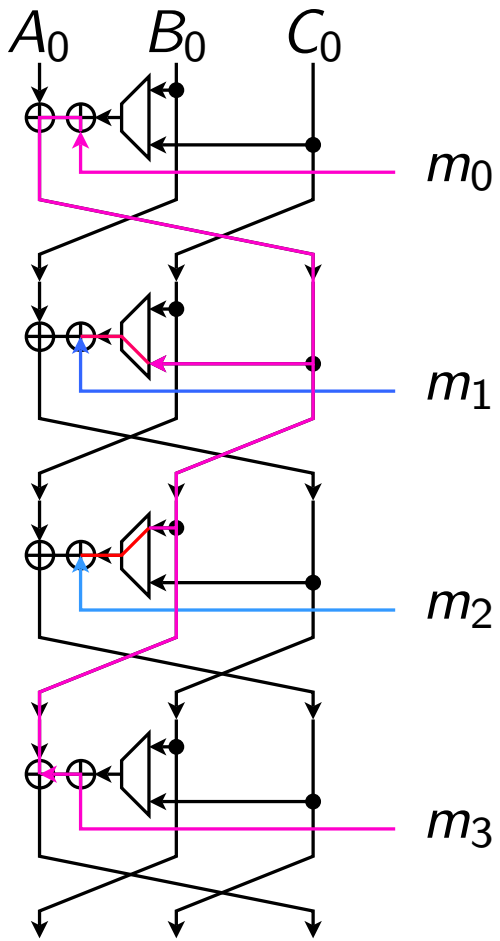
Disturbance-corrections strategy



Introduce a difference in m_0
Correct difference in step 1 with m_1
Correct difference in step 2 with m_2
Correct difference in step 3 with m_3

Exercise: Find collisions for 16 steps of SHA-1.

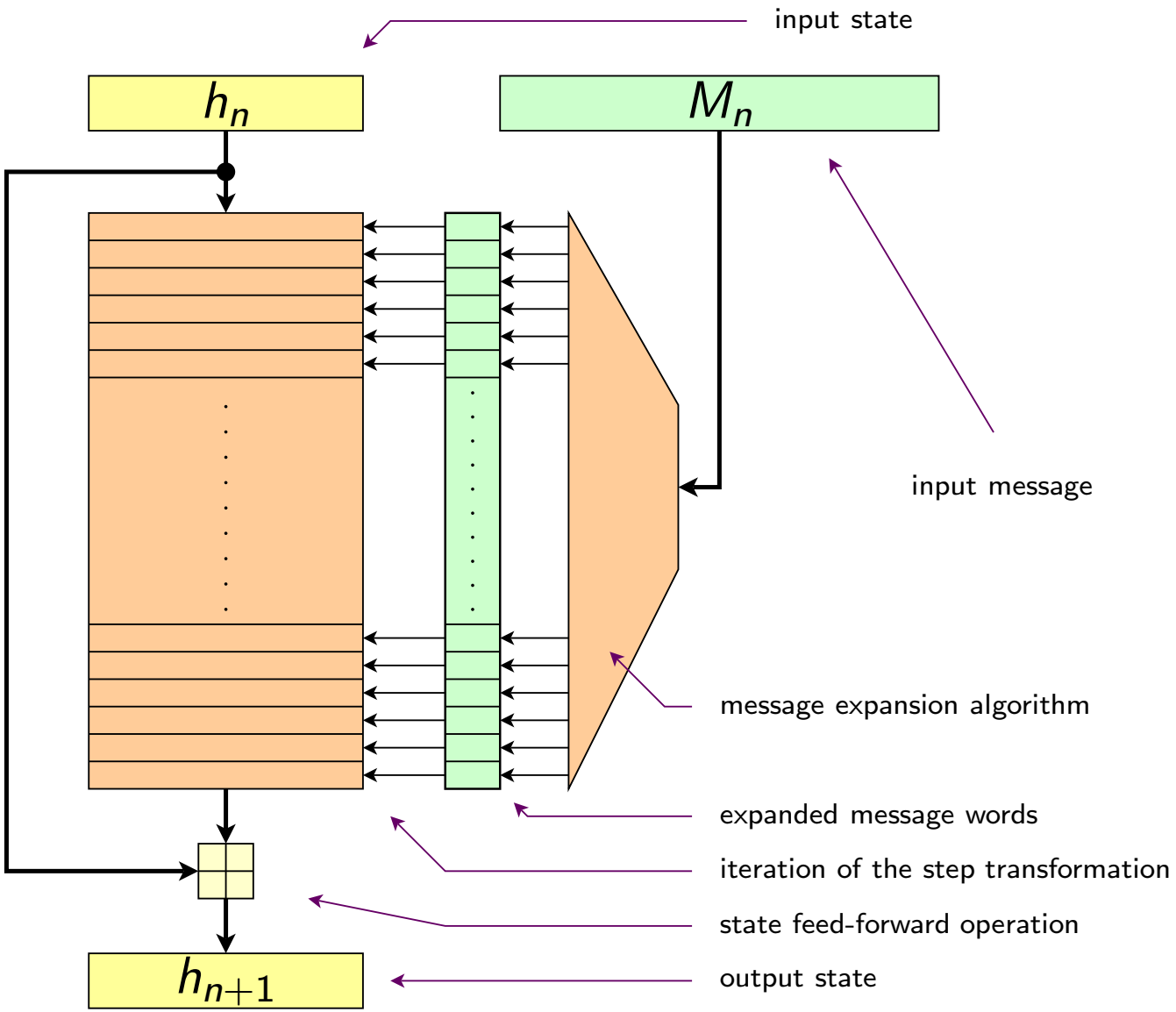
Disturbance-corrections strategy



Introduce a difference in m_0
Correct difference in step 1 with m_1
Correct difference in step 2 with m_2
Correct difference in step 3 with m_3

Exercise: Find collisions for 16 steps of SHA-1.

Lesson: We need a complex mapping between input words and "round keys" – something called **message expansion**


input state
h_n
M_n
input message
message expansion algorithm
expanded message words
iteration of the step transformation
state feed-forward operation
output state
h_{n+1}

The world before Wang (until 2004)

hash	designed	attacks
MD4	1990	Dobbertin, 1996 (collisions, example)
MD5	1992	den Boer, Bosselaers, 1993 (pseudo-collisions, example)
SHA-0	1993	Chabaud, Joux, 1998 (collisions, theory)
SHA-1	1995	no attack
SHA-2	2002	no attack

The world after Wang (after 2004)

hash	designed	Wang-inspired attacks
MD4	1990	collisions by hand calculation
MD5	1992	collisions in ms on a computer
SHA-0	1993	collisions in $\approx 1hr$
SHA-1	1995	attacked, 2^{69}
SHA-2	2002	no attack

Many other hash functions similar to MD5/SHA-1 were subsequently broken using Wang's method.

Main ideas of Wang: modular differentials

Consider

$$a = 12 = 1100_b$$
$$b = 9 = 1001_b$$

- XOR difference: $a \oplus b = 0101$
- modular difference $a - b = 12 - 9 = 3$
- signed binary differences $a \pm b = (0, -1, 0, 1)$

Main ideas of Wang: modular differentials

Consider

$$a = 12 = 1100_b$$
$$b = 9 = 1001_b$$

- XOR difference: $a \oplus b = 0101$
- modular difference $a - b = 12 - 9 = 3$
- signed binary differences $a \pm b = (0, -1, 0, 1)$

Note that having signed binary difference we can determine both XOR and modular differences!

Signed binary differences

Analysis of MD-inspired hashes has to consider differential behaviour of

- XOR operations $\oplus$
- bit rotations
- bitwise Boolean functions
- modular additions $\boxplus$

difference	XOR	rotation	Boolean f.	addition
XOR	easy	easy	medium	hard
modular	hard	medium	hard	easy
signed binary	easy	easy	medium	medium

Example: signed binary differences and XOR

a	b	$a \oplus b$
$0 \rightarrow 1$	$0 \rightarrow 1$	0
$1 \rightarrow 0$	$1 \rightarrow 0$	0
$0 \rightarrow 1$	$1 \rightarrow 0$	1
$1 \rightarrow 0$	$0 \rightarrow 1$	1
$0 \rightarrow 1$	0	$0 \rightarrow 1$
$0 \rightarrow 1$	1	$1 \rightarrow 0$
$1 \rightarrow 0$	0	$1 \rightarrow 0$
$1 \rightarrow 0$	1	$0 \rightarrow 1$

Acknowledgments

- Some slides were borrowed from Søren S. Thomsen's lecture "Advanced Topics in Cryptology: Cryptanalysis of iterated hash functions"