Cryptography
Cryptographic Hash Functions

Uwe Egly

Knowledge-Based Systems Group
Institute of Information Systems
Vienna University of Technology

Overview

- Hash function (HF) accepts input of arbitrary length
- Returns corresponding fix length hash value (e.g., 160 bits) (imprint, digital fingerprint, message digest)
- Various applications (e.g., make changes in emails detectable)
- Here: Hash functions without a key (unkeyed HV)
- HFs are often constructed using compression functions
- Compression function: $h : \Sigma^m \mapsto \Sigma^n$ with $m, n \in \mathcal{N}$ and $m > n$
- Hashing procedures and their security:
  MD4 (broken), MD5 (insecure/broken), SHA-1 (insecure ?), RipeMD-160 (?)

Attack Against Hash Functions

hash	**attack**			
	author	**type**	**complexity**	**year**
MD4	Dobbertin	collision	2^{22}	1996
	Wang et. al	collision	2^{8}	2005
MD5	dan Boer & Bosselaers	pseudo-collision	2^{16}	1993
	Dobbertin	free-start	2^{34}	1996
	Wang et. al	collision	2^{39}	2005
SHA-0	Chabaud & Joux	collision	2^{61} (theory)	1998
	Biham & Chen	near-collision	2^{40}	2004
	Biham et. al	collision	2^{51}	2005
	Wang et. al	collision	2^{39}	2005
SHA-1	Biham et. al	collision (40 rounds)	very low	2005
	Biham et. al	collision (58 rounds)	2^{75} (theory)	2005
	Wang et. al	collision (58 rounds)	2^{33}	2005
	Wang et. al	collision	2^{63} (theory)	2005

(From: I. Mironov. Hash functions: Theory, attacks, and applications)

Consequence

- 2007: NIST has asked for proposals for replacing current SHA (SHA: standard hashing algorithm)
- Dec 2010: finalist chosen (three rooted in Europe)
  1. Blake (from Switzerland)
  2. Grøstl (TU Graz/TU of Denmark)
  3. Keccak (with J. Daemen from Rijndal/AES)
  4. JH (Singapure)
  5. Skein (Bruce Schneier from the US)
- Winner: Keccak
  (info at `http://keccak.noekeon.org/`)

`http://www.h-online.com/security/news/item/NIST-s-s`

Properties of Cryptographic Hash Functions

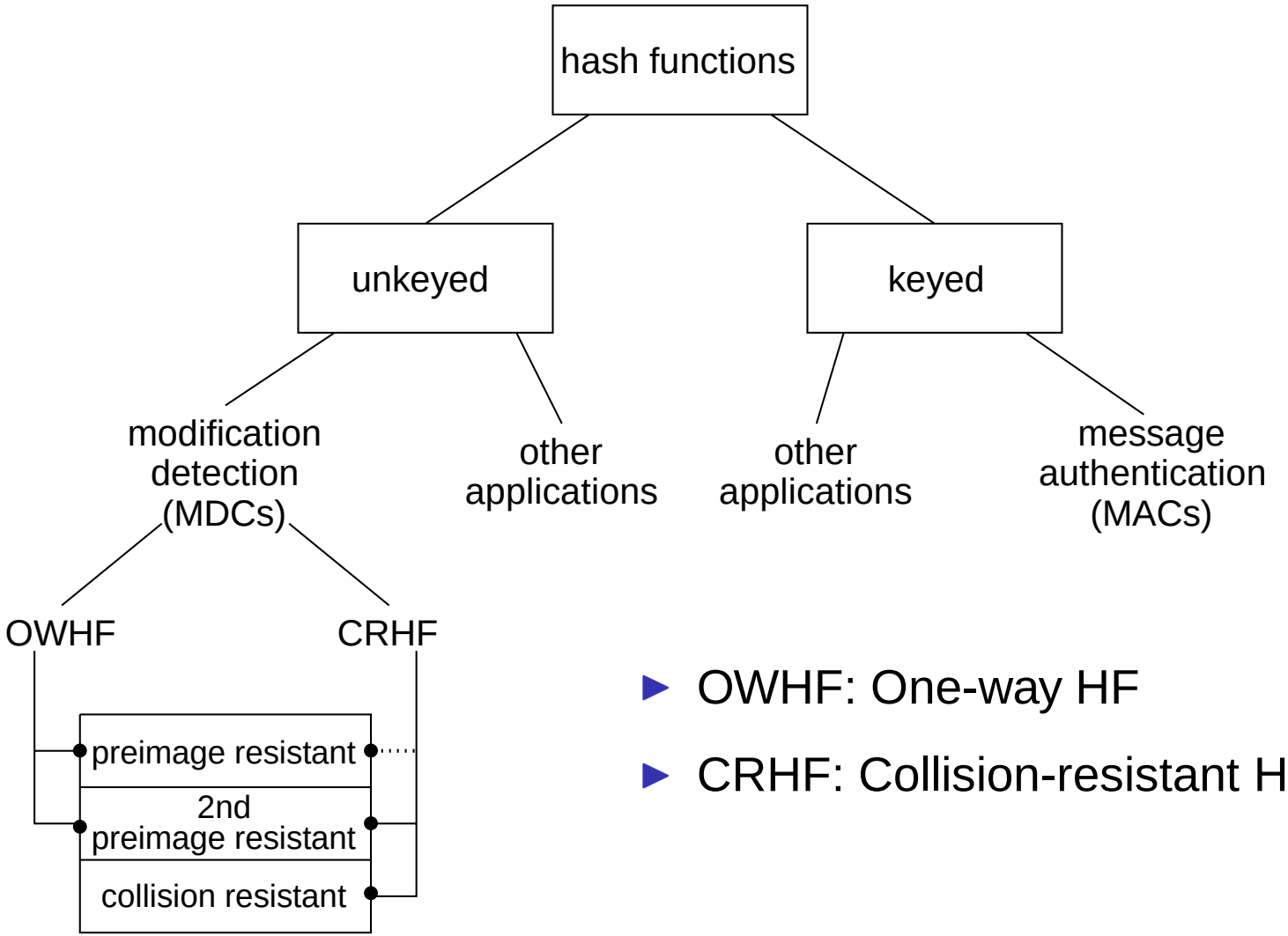
Let p be a message, v the computed (n bit) HV and h a HF

- h maps arbitrary length p to a fixed bitlength v (compression)
- Good HF: maps message uniformly (i.e., with prob. $\frac{1}{2^n}$) to HVs
- For a given p, it is easy to compute v
- For a given v, it is hard to compute p with $h(p) = v$
- For a given p, it is hard to compute q with $h(p) = h(q)$
- Collision of h: Pair of messages p, q with $h(p) = h(q)$
- All compressions functions cause collisions (because compressions functions are not injective)

The Properties in Detail

- Let h be an unkeyed HF, x, x' inputs and y, y' outputs
- Preimage resistance: For essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage x' such that $h(x') = y$ when given any y for which a corresponding input is not known
- 2nd-preimage resistance: It is computationally infeasible to find any second input which has the same output as any specified input, i.e., given x, to find a 2nd-preimage $x' \neq x$ such that $h(x) = h(x')$
- Collision resistance: It is computationally infeasible to find any two distinct inputs x, x' which hash to the same output, i.e., such that $h(x) = h(x')$. (Note that here there is free choice of both inputs.)

Simplified Classification of Hash Functions

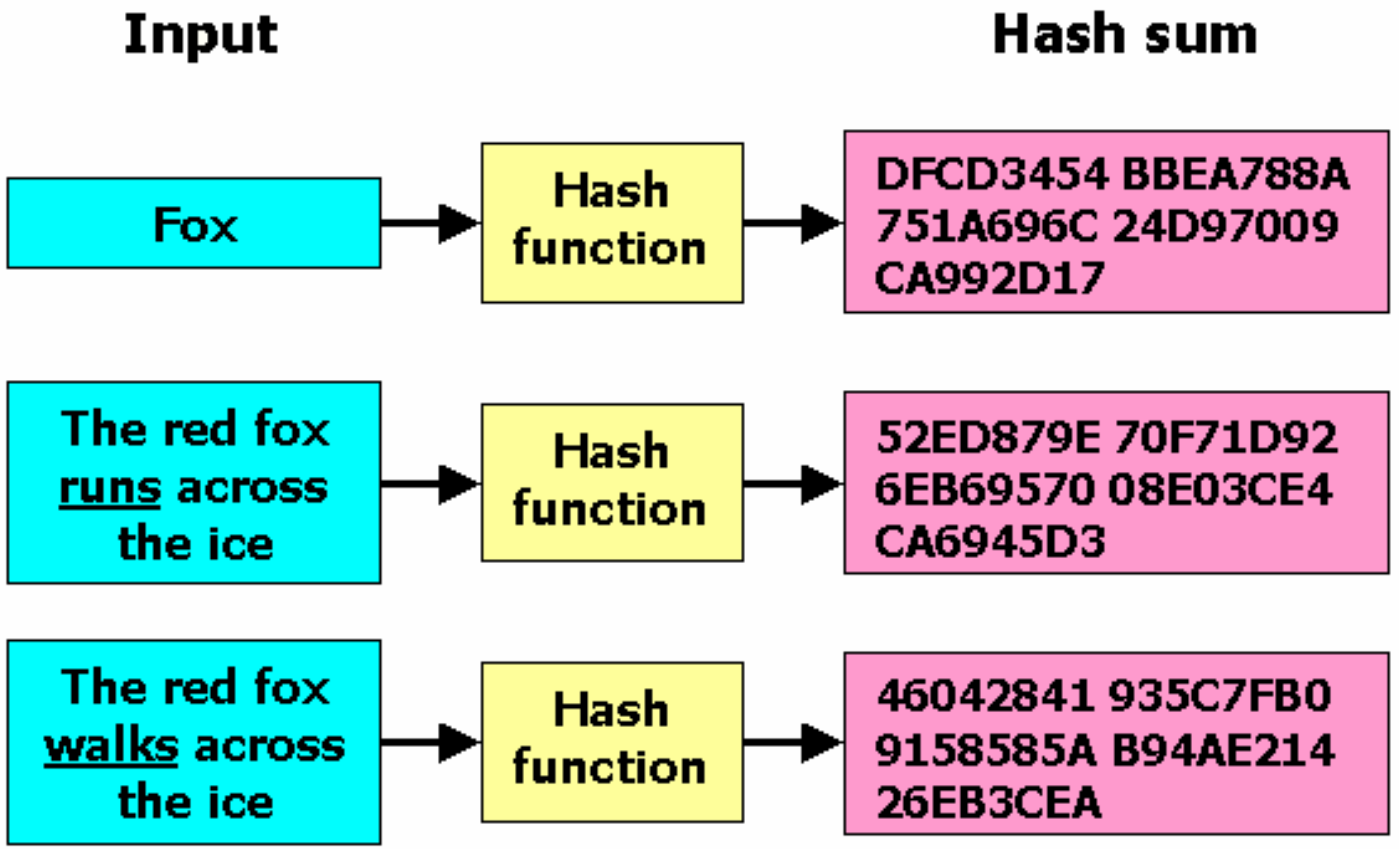


- OWHF: One-way HF
- CRHF: Collision-resistant HF

Modification Detection Codes (MDCs)

- Main application of MDCs: Provide data integrity
- HV provides message digest or finger print of larger data
  - Construct message digest of, e.g., a software distribution
  - Modification of data can be detected: compute HV and compare it with the original
  - HV of the original data has to be write-protected
  - The data/programs can be given away
- OWHFs are preimage and 2nd preimage resistant
- CRHFs are 2nd preimage and collision resistant
  (In practice, often also preimage resistant)
- Use CRHF, if attacker can choose msg to provocate a collision

Applying SHA-1 on Different Messages



Birthday Paradox (1)

How many persons are required to be in a room such that the probability that two have the same birthday is $\geq \frac{1}{2}$?

- There are n birthdays and k persons are in the room
- Elementary event: $(g_1, \ldots, g_k) \in \{1, 2, \ldots, n\}^k$
  i th person has birthday g_i $(1 \leq i \leq k)$
- n^k elementary event, all equally probable (with $\frac{1}{n^k}$)
- Look for probability p such that ≥ 2 persons have the same birthday
- $q = 1 - p$: probability s.t. all persons' birthday is different
- How many tuples $(g_1, \ldots, g_k)$ are there with different g_i?
  There are: $\prod_{i=0}^{k-1}(n - i) = |E|$

Birthday Paradox (2)

$$q = \frac{|E|}{n^k} = \frac{1}{n^k}\prod_{i=0}^{k-1}(n-i) = \prod_{i=0}^{k-1}\frac{(n-i)}{n}$$
$$= \prod_{i=0}^{k-1}\left(1-\frac{i}{n}\right)$$

With $1+x \leq e^x$ for any real number x, we obtain

$$q \leq \prod_{i=0}^{k-1} e^{-i/n} = e^{\sum_{i=0}^{k-1} -i/n} = e^{-k(k-1)/(2n)}$$

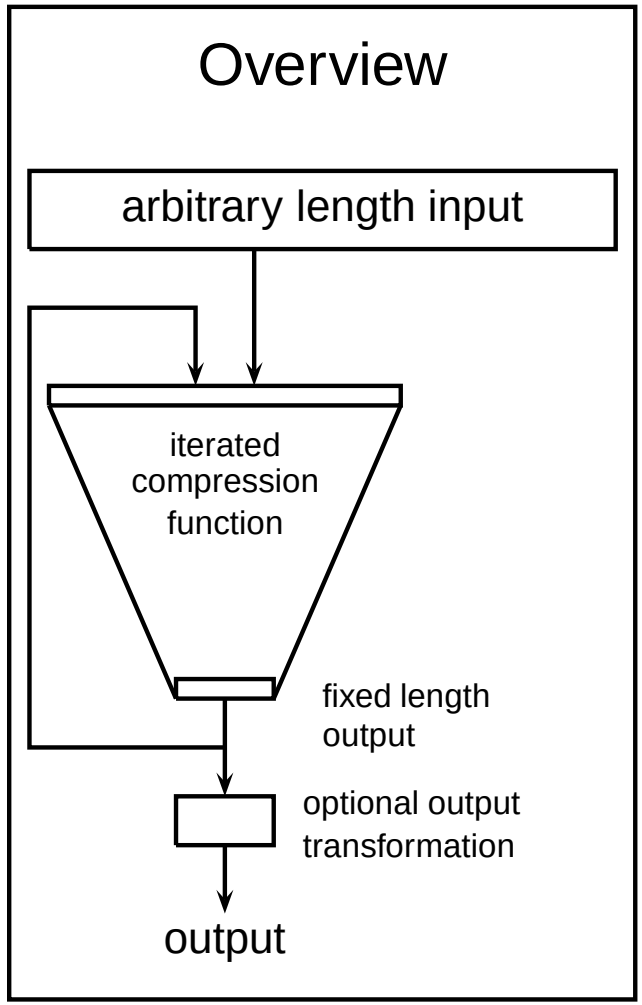
If $k \geq (1+\sqrt{1+8n\ln 2})/2$, then $q \leq \frac{1}{2}$ holds and $p \geq \frac{1}{2}$ follows

Hence, a little bit more than $\sqrt{n}$ persons are sufficient for $p \geq \frac{1}{2}$

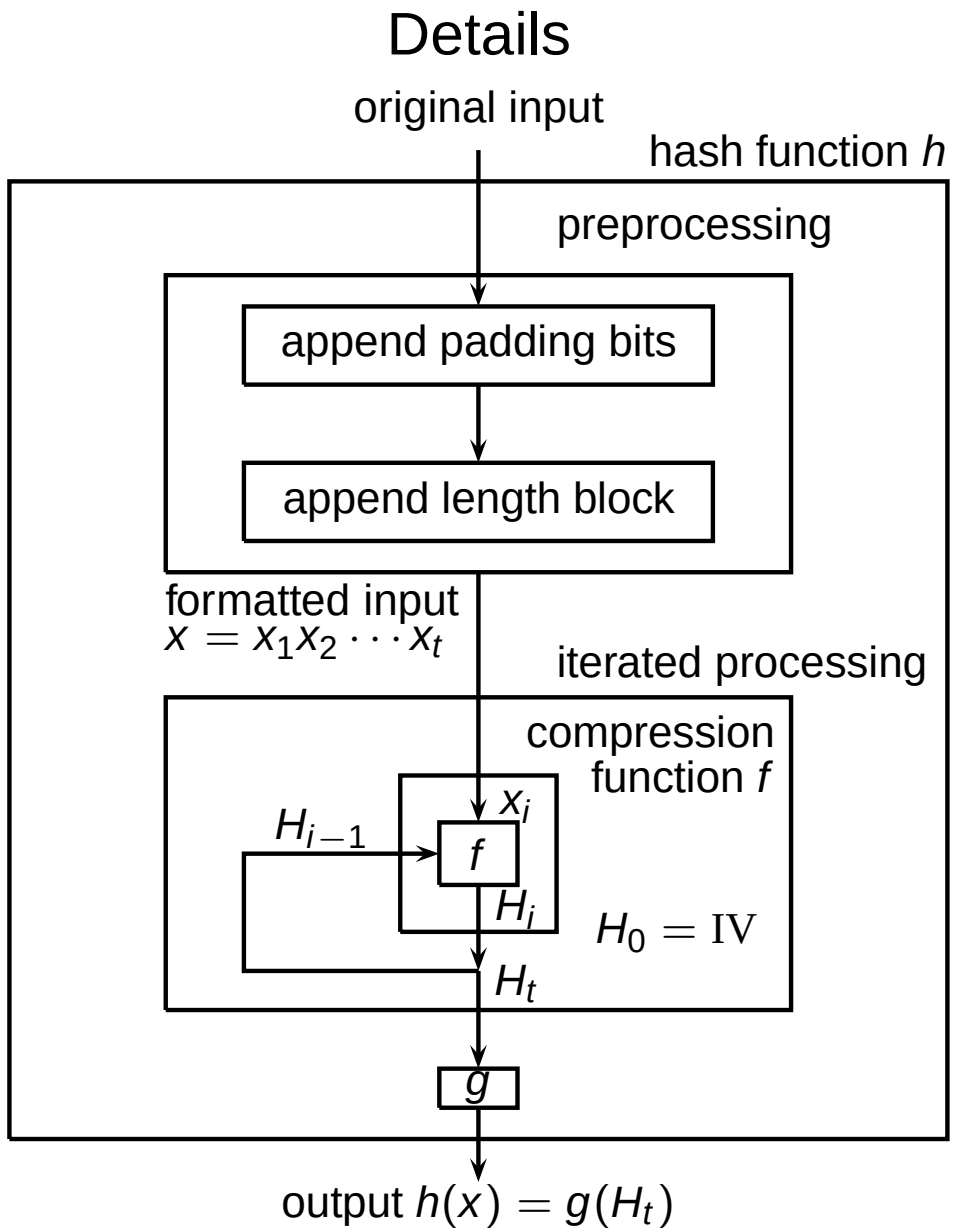
Birthday Attack

- Attack against a hash function (try to find collisions)
- Basic Idea:
  1. Generate and store all HVs in a given time interval
  2. Sort the HVs and search for collisions
- Hash values = birthdays and $\Sigma = \{0, 1\}$
- Assumption: Random choice of strings from Σ^*, all HVs are equally probable
- Choose randomly $k \geq (1 + \sqrt{1 + (8 \ln 2)|\Sigma|^n})/2)$ ele from Σ^*
- Then probability p that 2 have the same HV is $\geq \frac{1}{2}$
- That is, if we choose $\approx 2^{n/2}$ elements, then $p \geq \frac{1}{2}$ holds
- In current procedures: $n = 160$ or more

Overview: How Hash Functions Work



The Detailed View How Hash Functions Work



Hash Functions

- Preprocessing
  - Divide hash input x into t fixed-length r-bit blocks x_i
  - Padding: Fill last block with padding bits
  - Often for security reasons: Add original message size in last (or extra) block
- Apply the compressions fct $f : \Sigma^r \mapsto \Sigma^m$ to each block x_i
- Recurrence equation:
  $H_0 = \mathrm{IV}, \qquad H_i = f(H_{i-1}, x_i)\ (1 \leq i \leq t), \qquad h(x) = g(H_t)$
- IV is initialization value for the hash (for the first “round”)
- g is the (optional) output transformation
  (Reduction to k output bit)

Constructing HFs From Compression Functions

Any collision resistant compression function f can be extended to a collision resistant hash function (CRHF) h

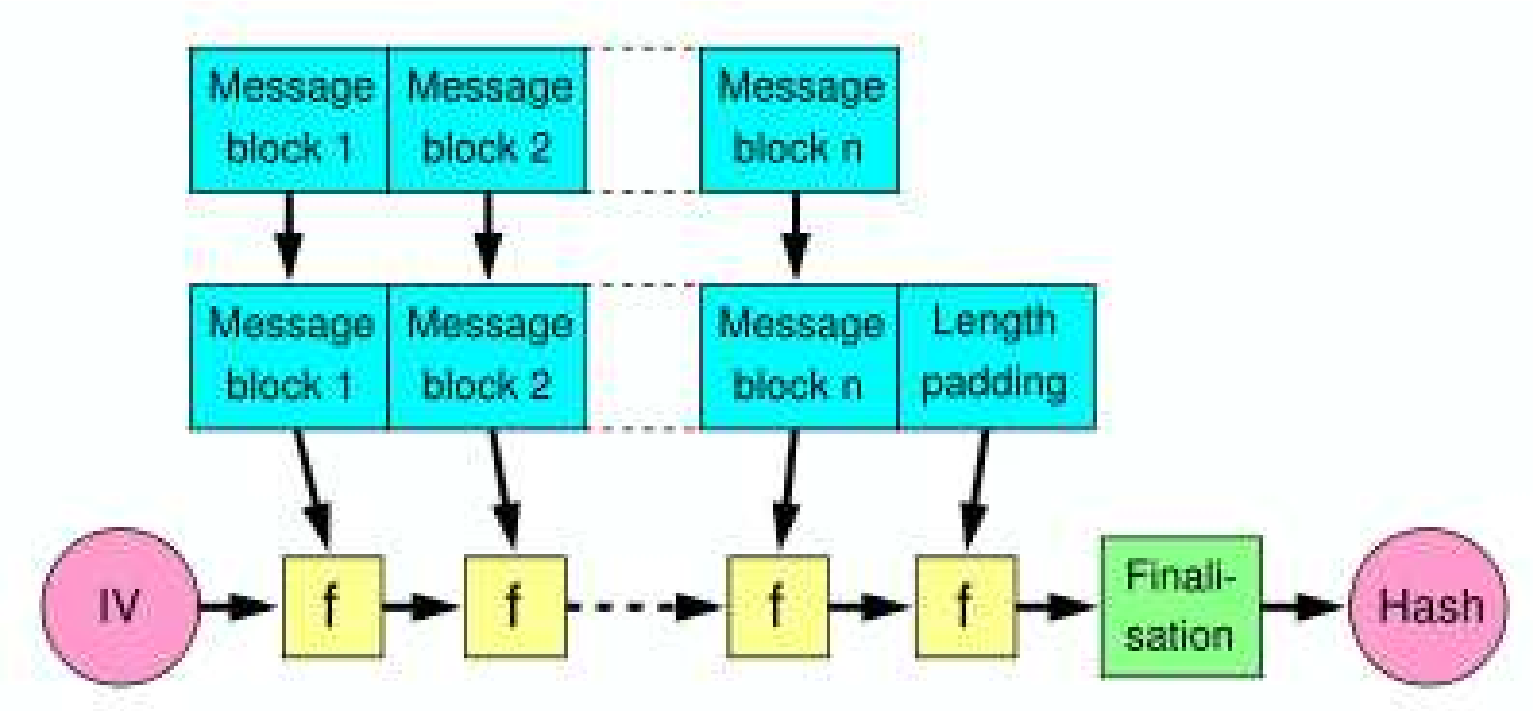
---

Algorithm 1: Merkle's meta-method for hashing

Input: collision resistant compression function f
Result: collision resistant unkeyed hash function h

1. Suppose f maps $(n + r)$-bit inputs to n-bit outputs (for concreteness, consider $n = 128$ and $r = 512$). Construct a hash function h from f, yielding n-bit hash-values, as follows.
2. Break an input x of bitlength b into blocks $x_1 x_2 \cdots x_t$ each of bitlength r, padding out the last block x_t with 0-bits if necessary.
3. Define an extra final block $x_t + 1$, the length-block, to hold the right-justified binary representation of b (presume that $b < 2^r$).
4. Letting 0^j represent the bitstring of j 0's, define the n-bit hash value of x to be $h(x) = H_{t+1} = f(H_t \| x_{t+1})$ computed from:
   $H_0 = 0^n$; $\quad H_i = f(H_{i-1} \| x_i)$, $\quad 1 \leq i \leq t + 1$

Another Presentation of Merkle's Meta-Method



MD-Strengthening and Padding Methods

- MD-strengthening: (MD for Merkle-Damgård)
  Before hashing a msg $x = x_1 x_2 \cdots x_t$ (where x_i is a block of bitlength r appropriate for the relevant compression function) of bitlength b, append a final length-block, x_{t+1}, containing the right-justified binary representation of b. (This presumes $b < 2^r$.)

- Two padding methods (n is the desired block size)
  1. Append to msg x as few (possibly zero) 0-bits as necessary to obtain x' whose bitlength is a multiple of n
  2. Append to msg x a single 1. Then apply padding method 1

- Method 1 is ambiguous (0s at the end from padding or in x?)

- Method 2 is not ambiguous

How to Construct Hash Functions

- Three possibilities how hash functions can be constructed:
  1. Construct HFs based on block ciphers (BCs)
  2. Construct HFs from scratch (customized HFs)
  3. Construct HFs based on modular arithmetic (not discussed)
- Motivation for the BC-based approach: Reuse of software
  If implemented BC is available, then construction of HF is easy
- In general, it is not clear what requirements of a BC is sufficient to construct secure HFs

Hash Functions Based on Block Ciphers

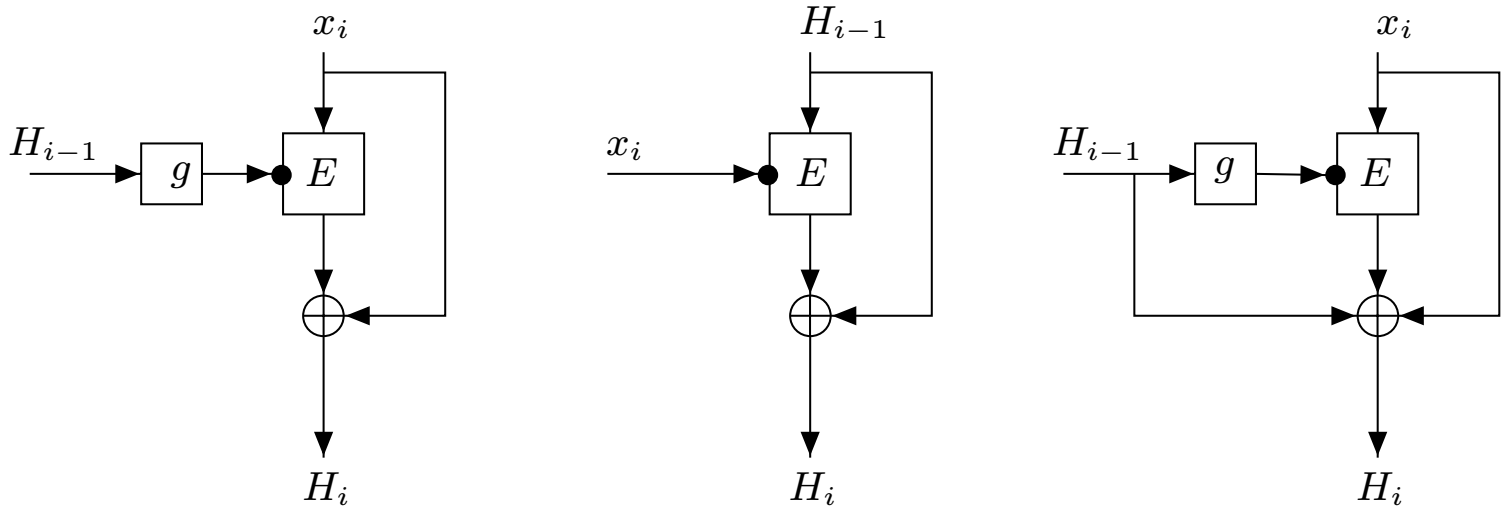
- (n,r) block cipher E: defines an invertible function from n-bit plaintexts to n-bit ciphertexts using an r-bit key. $E_k(x)$ denotes the encryption of x under key k.
- Distinguish between single-length and double-length HVs
  - Single: HV has as many bit as the blocksize (=n-bit)
  - Double: HV has twice as many bit as the blocksize (=2n-bit) (not discussed in the following)
- Rate of h

  Let h be an iterated HF constructed from a BC with compression fct f which performs s block encryptions to process each successive n-bit message block. Then the rate of h is $1/s$.

Single-Length MDCs Based on Block Ciphers

- Some components used in the following:
  1. A generic n-bit BC E_K parameterized by a symmetric key K
  2. A function g which maps n-bit inputs to keys K for E (if keys for E are also of length n, g might be the identity function)
  3. A fixed (usually n-bit) initial value IV, suitable for use with E

Constructions of Matyas-Meyer-Oseas, Davies-Meyer and Miyaguchi-Preneel



The Output of Single-Length MDCs Based on BCs

- Output H_t of Matyas-Meyer-Oseas hash:
  $H_0 = \mathrm{IV} \qquad H_i = E_{g(H_{i-1})}(x_i) \oplus x_i \quad (1 \leq i \leq t)$
- Output H_t of Davies-Meyer hash:
  $H_0 = \mathrm{IV} \qquad H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1} \quad (1 \leq i \leq t)$
- Output H_t of Miyaguchi-Preneel hash:
  $H_0 = \mathrm{IV} \qquad H_i = E_{g(H_{i-1})}(x_i) \oplus x_i \oplus H_{i-1} \quad (1 \leq i \leq t)$

The Hash Function Whirlpool

- Designed by V. Rijmen (TU Graz) and P. Barreto
- Takes messages with less than 2^{256} bit
- Produces a message digest of 512 bit
- Name inspired by “M51 (Whirlpool) Galaxy in Canes Venatici”

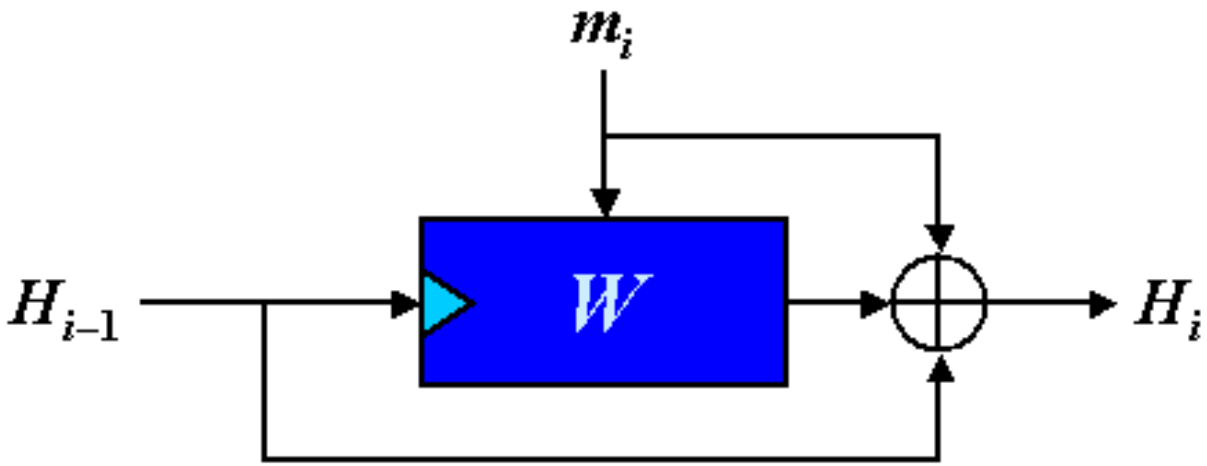
See `http://en.wikipedia.org/wiki/Whirlpool_(cryptography)` for further details

Some information taken from `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html`

The Hash Function Whirlpool (2)

- Uses MD-strengthening and Miyaguchi-Preneel scheme
  - Uses padding method 2.
  - Introduce the bit value 1 after the message string
  - Then add 0s and the 256 bit length block afterwards
  - The length of the final input message m is a multiple of 512
  - Decompose m into 512 bit blocks $m_1, m_2, \ldots, m_t$
- Generate a sequence of 512 bit hash values
  $H_0 = \text{IV} = \text{“}0 \ldots 0\text{”}, H_1, H_2, \ldots, H_t$
- Compute H_i by encrypting m_i using H_{i-1} as key, and XOR the resulting ciphertext with both H_{i-1} and m_i
- The Whirlpool message digest is H_t
- The underlying cipher is W, a 512 bit variant of AES

A Schematic View to Whirlpool



- Function *g* in the Miyaguchi-Preneel scheme is the identity
- Reference implementations for Whirlpool are available (on the referenced www page above)

Final Comments

- NO detailed description of MD5, SHA-1, etc. because of
  - time constraints
  - good descriptions are available at Wikipedia, in HAC, in Schneier's book, etc.
- Thank your for coming and your enthusiasm
- Constructive critics and proposals for improvements welcome