

Cryptographic Hash

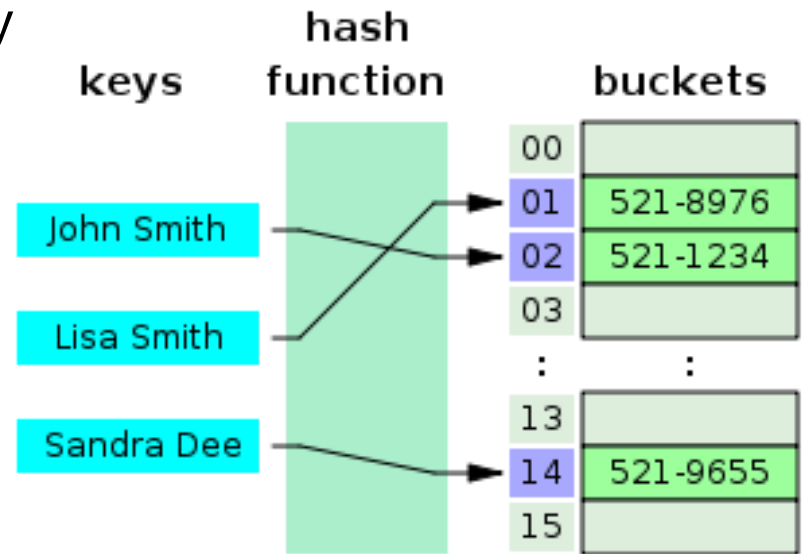
BMEVITMAV52

Information and Network Security

feher.gabor@tmit.bme.hu

Conventional hashing

- Conventional hash functions (bucket hashing)
 - Larger domains are mapped to smaller ranges
 - Speed up data retrieval
 - Reduce memory footprint
 - Collisions reduce efficiency



Cryptographic hashing

- Cryptographic hash functions
 - Creating a representative image
 - Compact representative image
 - Imprint, digital fingerprint, message digest
 - Used for data integrity protection and authentication
 - Collisions make the hash unsecure
 - Message -> hash (hash-code, hash-result, hash-value)

Hash functions

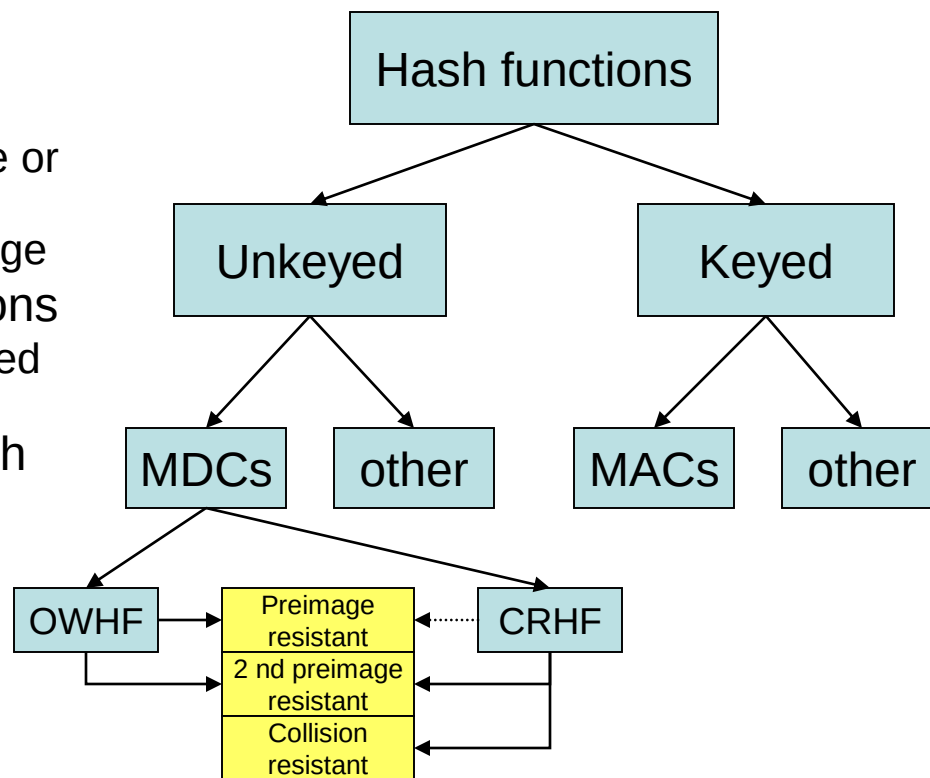
- From any sized input, provide a fixed length string
- Many-to-one function
 - There are collisions
 - Two input has the same output with the probability of 2^{-n}
 - Independent of the size of the input (it should be!)
- Minimum requirements
 - Compression
 - Arbitrary length -> finite length
 - Ease of computation

Size of the hash output

- Hash collision: 2^{-n}
 - Collision should be less than $1:2^{64}$ $n = 64$??? **no!**
- Birthday attack
 - The birthday paradox states that if there are 23 people in a room then there is a chance of more than 50% that at least two of them will have birthday on the same day (For 60 or more people, the probability is greater than 99%)
 - Paradox, since 23 seems to be too small
 - Using it on hash functions, generating N bit hash values there is a chance that there is a collision among $2^{N/2}$ randomly chosen messages
 - n should be 128 or 160 to defeat birthday attack

Hash function groups

- MDC
 - Modification detection code
 - Manipulations detection code or message integrity code
 - Provide a representative image
 - OWHF: One way hash functions
 - Find and input for the specified hash value is difficult
 - CRHF: Collision resistant hash function
 - Find two input that have the same hash value is difficult
- MAC
 - Message authentication code
 - The Integrity and source authentication



Unkeyed hash functions

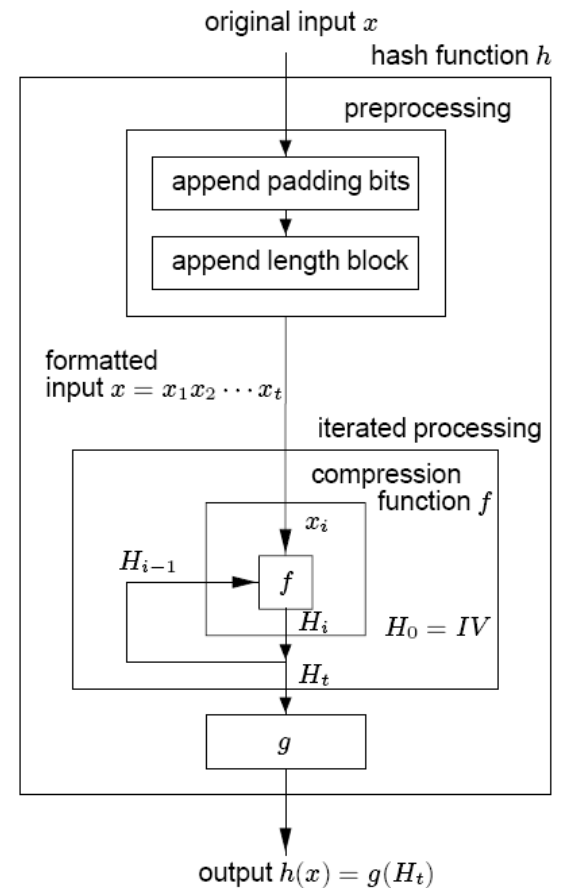
- $h(x)=y$ and $h(x')=y'$
- Preimage resistance (one-way)
 - Computationally infeasible to find preimage x' that $h(x')=y$ for a **given y**
- 2nd preimage resistance (weak collision resistant)
 - Computationally infeasible to find a second preimage that has the same output. For a **given x** , where $h(x)=y$, find x' where $h(x')=y$
- Collision resistance (strong collision resistant)
 - Computationally infeasible to find **any two** distinct input which have the same output $h(x) = h(x')$

Collision resistance

- Collision resistance implies 2nd preimage resistance
 - Proof: Assume h is collision resistant but not 2nd preimage resistant
 - For a given x one can find x' , where $h(x)=h(x')$
 - x and x' a collision pair
- Collision resistance does not guarantee preimage resistance
 - Proof: assume g is collision resistant with output of n bit
 - Output of the h hash function is $n+1$ bit:
 - If x is n bit long: $h(x) = 1|x$
 - Otherwise: $0|g(x)$
 - h is also collision resistant
 - h is not preimage resistant

Iterated hash functions

- Input is divided into fixed length blocks
 - Use padding
 - Padding with 0s
 - Padding with 1 then 0s
- f is the compression function
 - $H_i = f(H_{i-1}, x_i)$
 - n bit chaining variable
 - H_0 is the IV
 - Should be known at the other side
- g is an optional transformation
 - From n bit produces m bit



Merkle-Damgård strengthening

- MD-strengthening
 - Before hashing, append length block, containing the binary representation of the input's length
 - Presumes $\text{length} < 2^{\text{block size}}$
 - Any collision resistant compression function can be extended to a collision resistant hash function
 - Using the iterated approach

Cascading hash functions

- If $h_1(x)$ and $h_2(x)$ are collision resistant hash functions then $h(x) = h_1(x) || h_2(x)$ is also a collision resistant hash function
 - Increase the strength of the hash function

Block cipher based hash functions

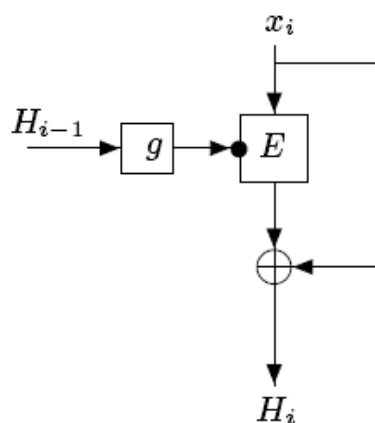
- Usually block cipher is present (SW or HW)
- h is an iterated hash function. In each iteration f perform s block encryptions to get the successive n -bit block
 - The rate of h is $1/s$
 - k : key size
 - n : block size
 - m : hash size

Hash function	(n, k, m)	Rate
Matyas-Meyer-Oseas	(n, k, n)	1
Davies-Meyer	(n, k, n)	k/n
Miyaguchi-Preneel	(n, k, n)	1
MDC-2 (with DES)	$(64, 56, 128)$	$1/2$
MDC-4 (with DES)	$(64, 56, 128)$	$1/4$

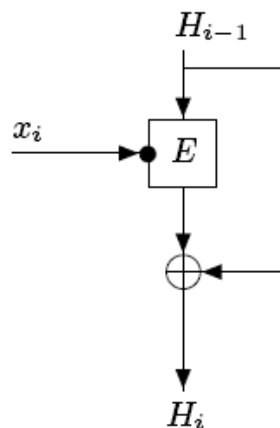
Single-length MDCs

- Components:
 - n-bit Block cipher
 - Function g that maps n-bit blocks to the key
 - Fixed initial value (IV) for the block cipher

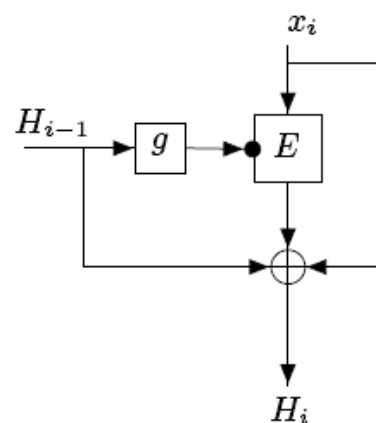
Matyas-Meyer-Oseas



Davies-Meyer

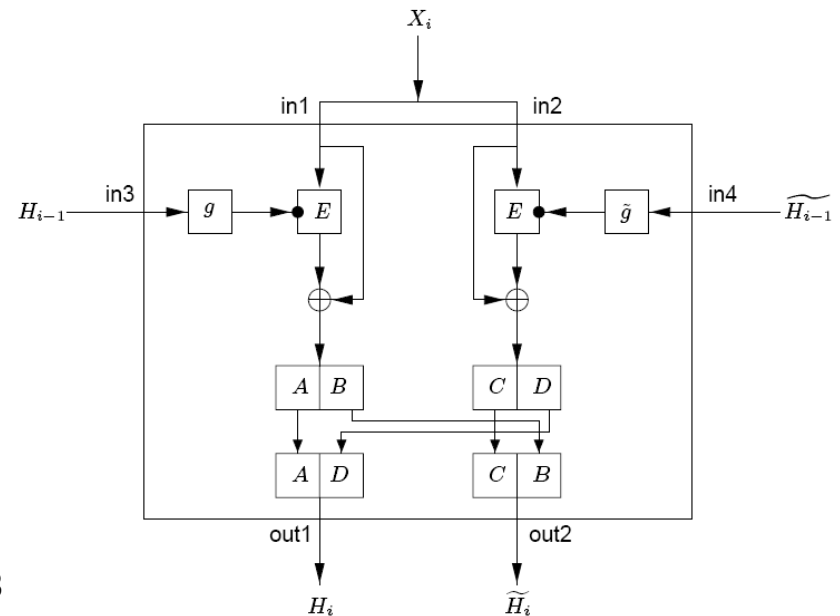


Miyaguchi-Preneel



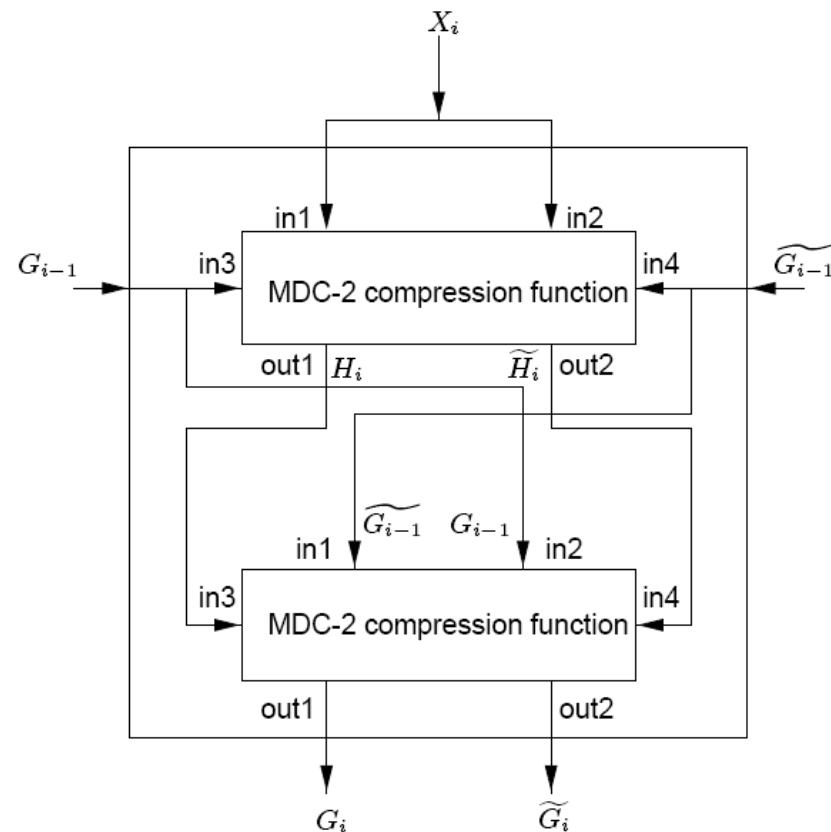
Double-length MDCs (MDC-2)

- More encryption operation during one iteration
- MDC-2:
 - Use DES
 - Combination of Matyas-Meyer-Oseas
 - Key generation:
 - $g(U) = u_1 \mathbf{10} u_4 u_5 u_6 u_7 u_9 u_{10} \dots u_{63}$
 - $g'(U) = u_1 \mathbf{01} u_4 u_5 u_6 u_7 u_9 u_{10} \dots u_{63}$



Double-length MDCs (MDC-4)

- 2 MDC-2 blocks

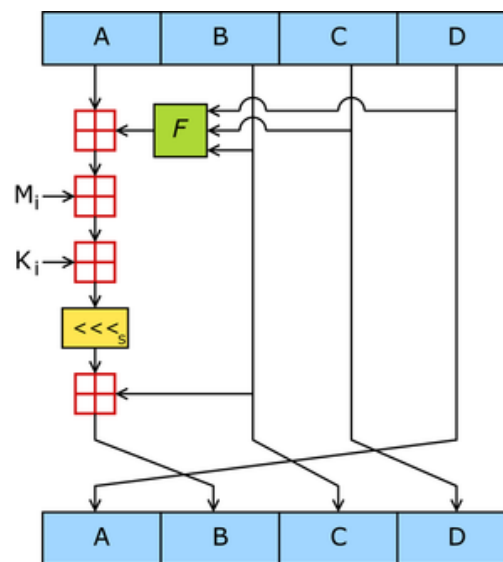


MD4 and MD5

- Designed “from scratch”
 - Optimized performance
- Message Digest 4 – MD4
 - Ron Rivest – 1990
 - 128 bit hash function (512 bit internal blocks)
 - Designed for 32 bit architectures
 - Goal: breaking would require roughly brute-force effort – FAILED (1991)
- Message Digest 5 – MD5
 - Ron Rivest – 1991
 - Strengthened MD4
 - Widespread use
 - 128 bit hash function (512 bit internal blocks)
 - Flaws:
 - 1996: design flaw found (not fatal)
 - 2004: collision using a cluster computer
 - 2006: collision within one minute on a notebook (tunneling)

MD5 algorithm

- Padding to 512 bit blocks
 - Bit 1, followed by 0s
 - The last 64 bit is the message length
- Iterations
 - 4 rounds, 16 repetition
 - 128 output: 4 states (32 bit)
 - Nonlinear function F
 - M : message in 32 bits
 - K : operation dependant key



$$F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$

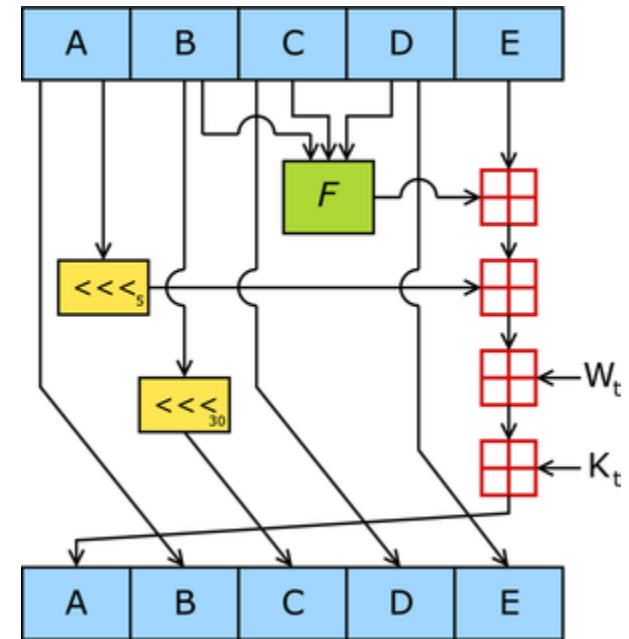
$$G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$

$$H(X, Y, Z) = X \oplus Y \oplus Z$$

$$I(X, Y, Z) = Y \oplus (X \vee \neg Z)$$

SHA algorithms

- SHA (SHA-0) was published in 1993
 - Flaw: collision in 2^{39} steps (2005)
- Secure Hash Algorithm – SHA-1
 - U.S. National Security Agency – 1995
 - Based on MD4
 - NIST proposal
 - 160 bit hash function (512 bit internal blocks)
 - Flaw: collision in 2^{69} steps (2005), 2^{61} steps (2012)
- SHA-2: SHA-224, 256, SHA-384, SHA-512
 - Same Merkle-Damgård engine as SHA-1
 - Still considered secure
- SHA-3: Keccak
 - Alternative engine



Keyed hash functions

- Primary purpose: message authentication
 - MAC: Message Authentication Code
 - Hash function with two input: the key and the message
 - $y = h(x, k)$
 - Ease of computation
 - Compression
 - Key non-recovery
 - From one or more message-MAC pairs it is unfeasible to get the key
 - Computation resistance
 - From one or more message-MAC pairs it is unfeasible to get a new message-MAC pair

Secret prefix method

- $\text{MAC}_k(x) = \text{hash}(k|x)$
- Insecure with iterated hash functions!
 - x and M are known ($M = h(k|x)$)
 - Producing MAC on $x|y$ is possible:
 - $M' = f(M, y)$, where f is the compression function

Secret suffix method

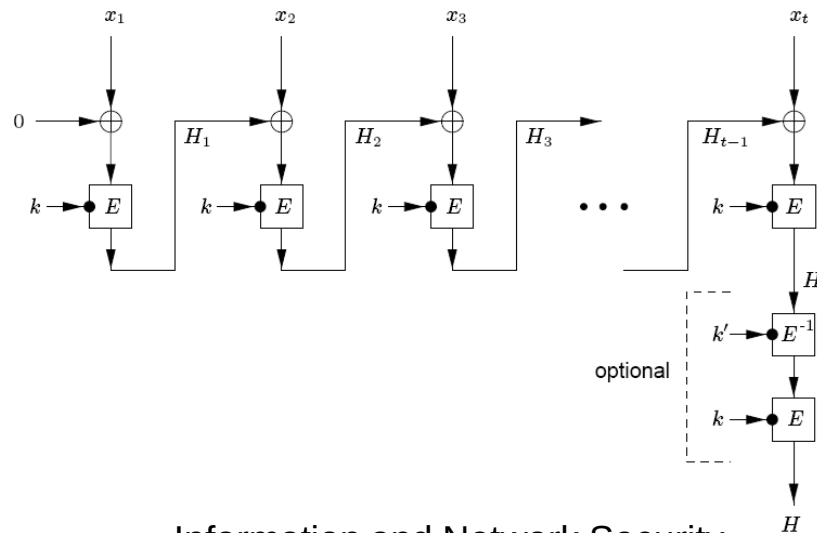
- $\text{MAC}_k(x) = \text{hash}(x|k)$
 - May be insecure if collision is found
 - If $h(x)=h(x')$ then $\text{MAC}_k(x) = \text{MAC}_k(x')$
- Weakness
 - MAC depends on the last chaining variable

HMAC

- Keyed-hash message authentication code
- Mihir Bellare, Ran Canetti, and Hugo Krawczyk (1996)
 - $\text{HMAC}(h,k,m) = h((k \otimes \text{opad}) \parallel h((k \otimes \text{ipad}) \parallel m))$
 - $\text{ipad (inner)} = 0x363636\dots36$ (a whole block)
 - $\text{opad (outer)} = 0x5c5c5c\dots5c$ (a whole block)
- HMAC-SHA1
 - 512 bit block size, 160 bit hash size
- HMAC-MD5
 - 512 bit block size, 128 bit hash size

CBC-MAC

- Based on cipher block chaining
 - Block cipher: DES
 - Block size: 64 bit
 - MAC key: DES key (56 bit)
 - Padding and blocking
 - Additional security: Use an other key and play 3DES
 - The final output is the hash value (64 bit)



CBC-MAC weakness

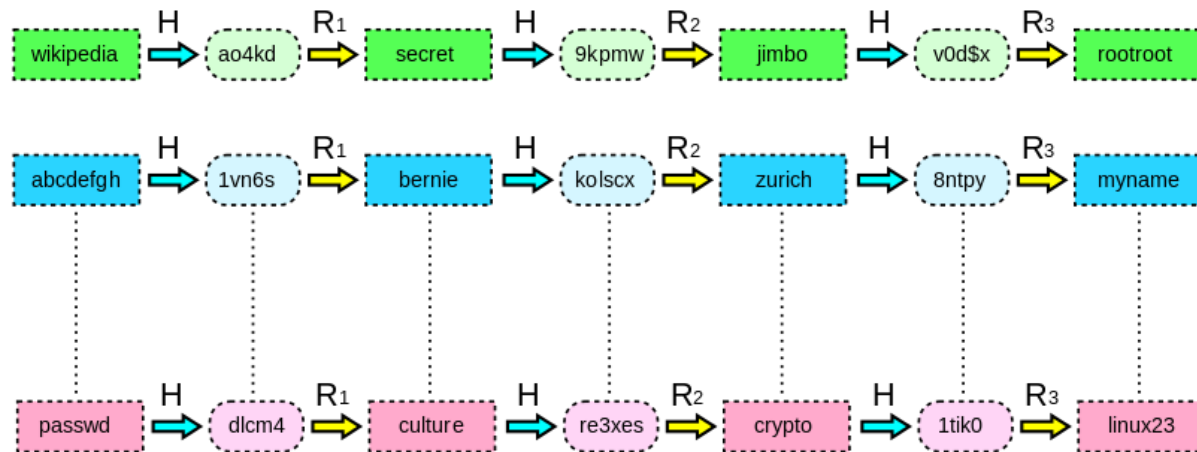
- Existential forgery of CBC-MAC
 - x_1 and MAC_1 pair is known
 - Request CBC-MAC for MAC_1 : $MAC_2 = E_k(E_k(x_1))$
 - MAC_2 is a MAC for $x_1|000\dots0$ as well !
 - x_1, MAC_1 and x_2, MAC_2 are known one block messages + hashes
 - Request CBC-MAC for $x_1|z$: $MAC_z = E_k(E_k(x_1) \oplus z)$
 - MAC_z is also MAC for $x_2|MAC_1 \oplus z \oplus MAC_2$
- Use fixed number of blocks
- MD-strengthening might help

Rainbow tables

- Inverse hash function is theoretically possible with stored hash inputs and outputs
 - Huge tables, not feasible
- Storing just some of these values, which are the inputs and outputs of hash chains
 - Ideally the chains are loop free and not merging to each other
 - TB large tables for given input sets, but now it is already feasible!

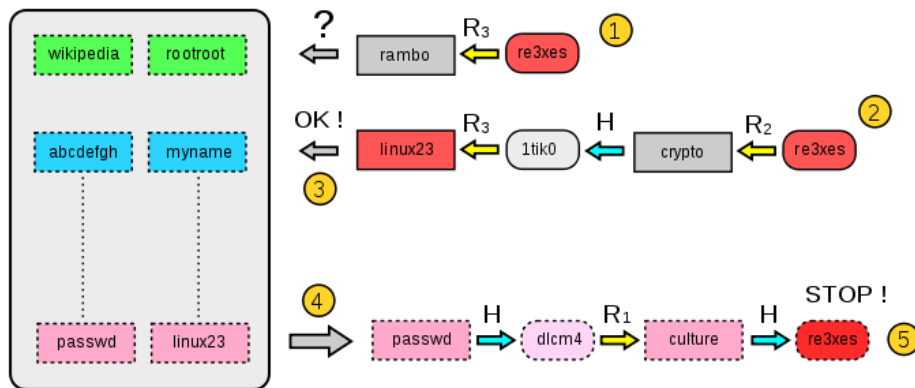
Rainbow table construction

- Reduction function: R
 - Convert hash output into an possible input
 - Multiple reduction functions in order to avoid loops and merges: $R_1, R_2, \dots R_k$



Rainbow tables

- Attacking with rainbow tables
- Example
 - Find input for hash output re3xes



- 1-2. Test with the end of the chains (k tests)
3. Find a match in the table
4. Start the chain from the table's input word
5. Hash until a we found the input

Defense against rainbow tables

- Make the input larger
 - Slating
 - `saltedhash(password) = hash(password || salt)`
 - Makes the input larger
 - Salt is not secret! But different for each hash
 - Key stretching
 - Use iterations to make the computation longer
 - Reduces brute force rate and increase the time to build a rainbow table

References

- Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, “Handbook of Applied Cryptography”, CRC Press, ISBN: 0-8493-8523-7
 - <http://www.cacr.math.uwaterloo.ca/hac/>
- Wikipedia - The free encyclopedia
 - <http://www.wikipedia.org/>