



UNIVERSITA' DEGLI STUDI DI PARMA
Dipartimento di Ingegneria dell'Informazione

Hashes and Message Digests

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Corso di Sicurezza nelle reti di telecomunicazioni, a.a. 2006/2007

<http://www.tlc.unipr.it/veltri>



Università degli Studi di Parma
Dipartimento di Ingegneria dell'Informazione

Hash Functions

Hash Function



- Also known as Message Digest
- it is a function that takes an input message and produce an output (hash value, or message digest)
- the input can be a variable-length bit string, the output is a fixed-length bit string (e.g. 128 bits)
- It is a one-way function
 - **it is not practical to figure out which input corresponds to a given output**

$$h=H(m)$$

- e.g. RSA-MD2, RSA-MD5 (rfc1321), SHA-1

2



Università degli Studi di Parma
Dipartimento di Ingegneria dell'Informazione

Hash Functions

Hash function properties

- Il messaggio m in ingresso può essere di qualsiasi lunghezza
- Il messaggio h in uscita ha sempre lunghezza fissa
- La computazione $h=H(m)$ è veloce e poco onerosa
- La trasformazione $H(m)$ è monodirezionale (one-way)
 - **permette rilievo di eventuali alterazioni**
- The message digest should look “randomly generated”
- It must be computationally infeasible to find a message with a given prespecified message digest
- It should be impossible to two find two messages that has the same digest (although the function is not one-to-one)

3



Università degli Studi di Parma
Dipartimento di Ingegneria dell'Informazione

Hash Functions

How many bits should the output have?

- How many bits should the output have in order to prevent someone from being able to find two message with the same hash?
- If the message digest has m bits, then it would take $2^{m/2}$ messages chosen at random (Birthday Paradox)
 - **however sometime it is not sufficient for an attacker to find out just two messages with the same hash; in such case, a brute-force attack requires 2^m searches**
- That is why message digest functions have output of at least 128 bits (in place of just 64 as for symmetric cryptography)

4

About the hash function

- Message digest function are like alchemy
- It's a bunch of steps that each mangle the message more and more
- A plausible way of constructing a message digest function is to combine lots of "perverse" operations
 - **however the message digest should remain easy to compute**
- Often, hash function uses constants (magic numbers)
 - "random numbers"
- Often the algorithm designers specify how they chose a particular number (to prevent suspects on particular properties of the chosen number)
 - π
 - **Published books with random numbers**
(A book has been published in 1939)

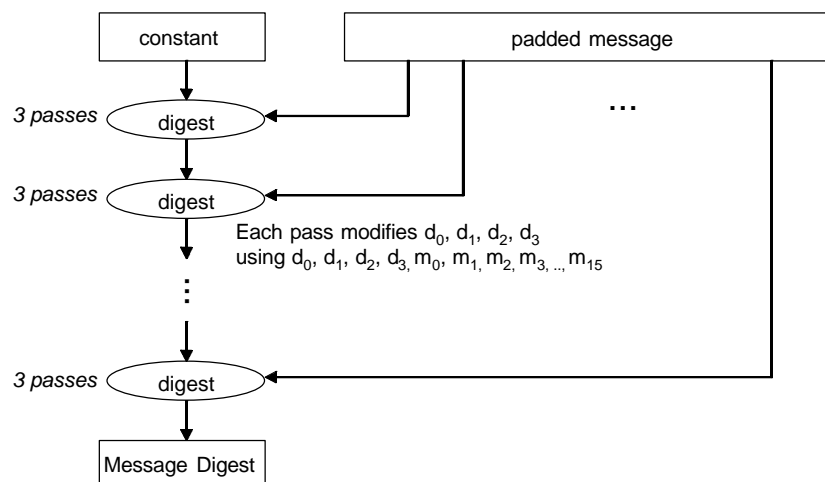
5

MD4

- Was designed to be 32-bit-world-oriented (instead of byte oriented schemes like MD2)
- Can handle message with an arbitrary number of bits
- Produce a 128 bit hash
- Message padding
 - **the message must be a multiple of 512bits (16 words);**
 - **the message is padded by adding one "1" bit and**
 - **padded with "0"s until bit $N \times 512 - 64$**
 - **the remaining 64 bit represent the number of unpadded message bits, mod 2^{64}**
- Message processed in 512-bit blocks (16 words)
- Each step makes three passes over the message block
- Message digest computed on 128-bit quantity (4 words)

6

MD4 scheme



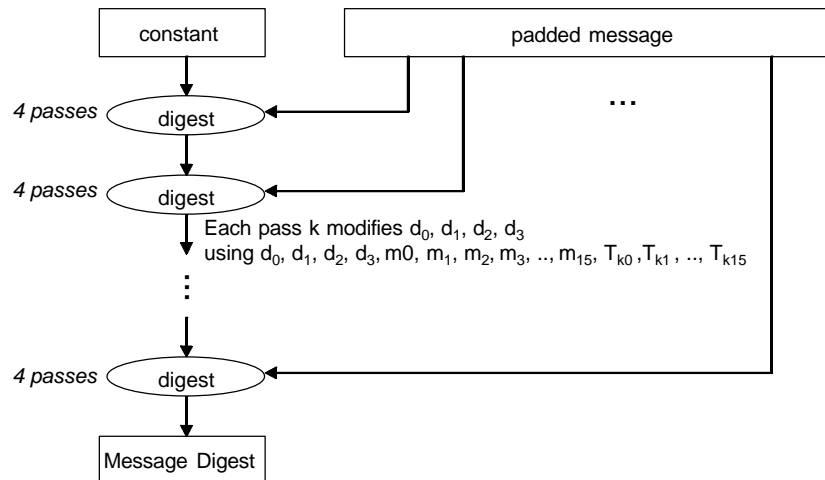
7

MD5

- Designed to be less concerned with speed and more with security
- Very similar to MD4; the main differences are:
 - **4 passes over each 128-bit (16-byte/4-word) chunk**
 - **different functions**
 - **uses a different constant T for each message word for each pass**
(4 passes x 16 message words = 64 32-bit constants)
- The message padding is the same as in MD4

8

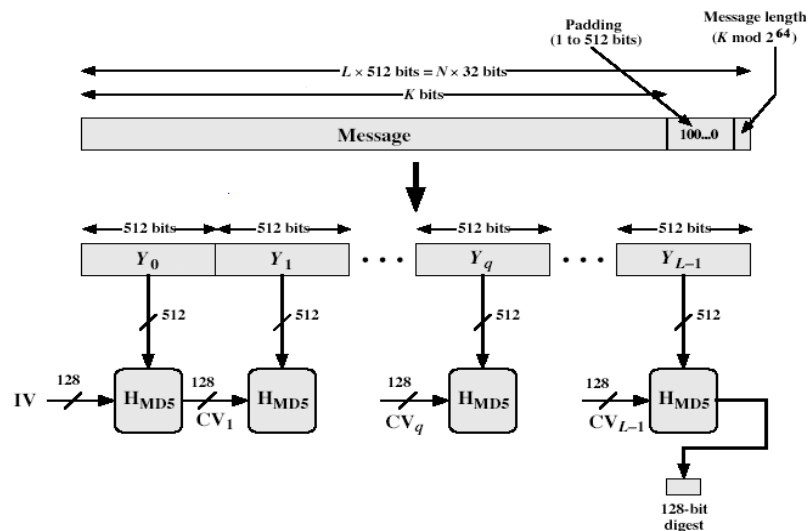
MD5 scheme



MD5 initialization

- Padding
 - **il messaggio viene sempre completato con bit di riempimento in modo in modo che la lunghezza modulo 512 sia 448 bit**
 - ovvero una lunghezza multipla di 512 meno 64 bit
 - vengono aggiunti da 1 a 512 bit
 - i bit di riempimento sono un 1 seguito da zeri
 - **vengono aggiunti 64 bit in cui viene inserita la lunghezza del messaggio modulo 2^{64}**
 - si ottiene così una stringa di lunghezza multipla di 512
- Inizializzazione del buffer MD di 128 bit composto da 4 word da 32 bit (A, B, C, D) prefissati
 - **A= 01 23 45 67**
 - **B= 89 AB CD EF**
 - **C= FE DC BA 98**
 - **D= 76 54 32 10**

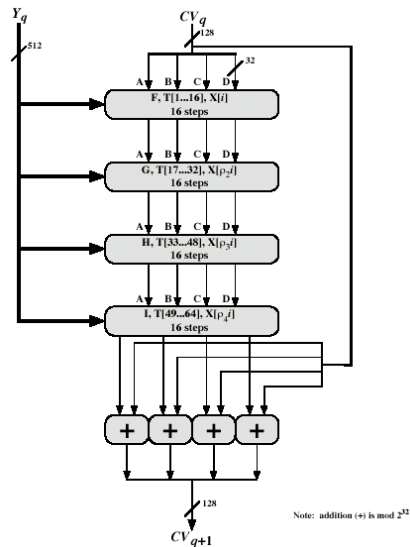
MD5 padding



MD5 processing

- Elaborazione del messaggio in blocchi da 512 bit (16 word)
- A partire dal buffer iniziale, per ogni blocco si effettuano 4 fasi di elaborazione
- In ognuna di esse viene eseguita 1 funzione differente, indicata rispettivamente con F, G, H e I
- Ciascuna fase utilizza in ingresso
 - il buffer ABCD da 128 bit,
 - il blocco corrente Y_q da 512 bit,
 - 1/4 di una tabella di 64 valori $T[1..64]$ basati sulla funzione seno (valori tabellati)
- l'uscita della quarta fase viene sommata word a word con l'ingresso (somma modulo 32)
- l'uscita dell'ultima elaborazione il message digest finale

MD5 processing



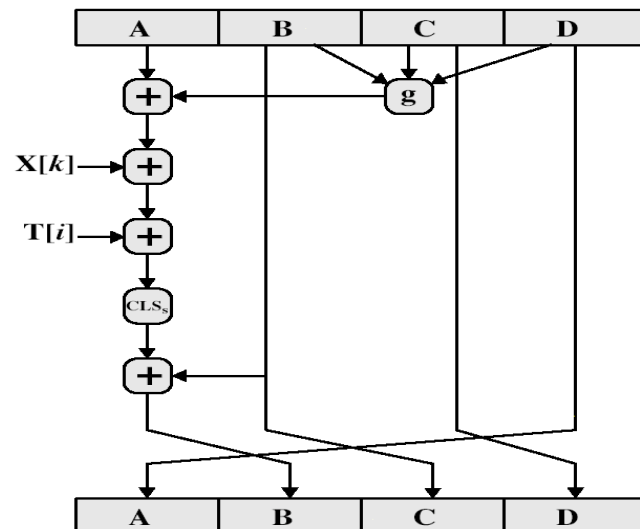
13

MD5 processing (passes)

- $A = B + ((A + g(B, C, D) + X[k] + T[i] \ll s))$
- Pass 1
 $g(x, y, z) = F(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$
- Pass 2
➤ $G(x, y, z) \dots$
- Pass 3 ...
➤ $H(x, y, z) \dots$
- Pass 4 ...
➤ $I(x, y, z) \dots$

14

MD5 processing (passes)



15

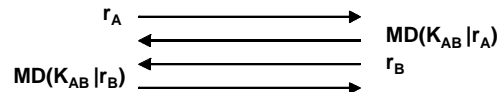
SHS

- Secure Hash Standard (SHS) proposed by NIST as message digest function
- Takes a message of length at most 2^{64} bits and produce 160-bit output (160 bit = 5-word block)
- Pad the message as in MD4 and MD5 (except that the message is limited to 2^{64} bits)
➤ **it is not a problem, since it would take several hundred years to transmit at 10Gb/s and it would take even longer (hundreds of centuries) to compute SHS at 100MIPS ;-)**
[P.S. if you can't say something in less than 264 bits, you shouldn't say it at all.]
- Operates in stages (as MD4, MD5)
➤ **Makes 5 passes for each block of data (3 in MD4 and 4 in MD5)**
➤ **Uses a different 160-bit mangle function in each stage**
- Little slower than MD5 and (presumably) little more secure

16

What doing with a Hash

- Password Hashing
 - a system may know/store just the hash of a passwd
- Message fingerprint
 - maintaining a copy of a message digest of some data/program in place of the copy of the entire data (for integrity check)
- Digital signature
 - Signing the MD of a message instead of the entire message
 - for efficiency (MDs are easier to compute than public-key algorithms)
- Authentication
 - similar to secret key cryptography



17

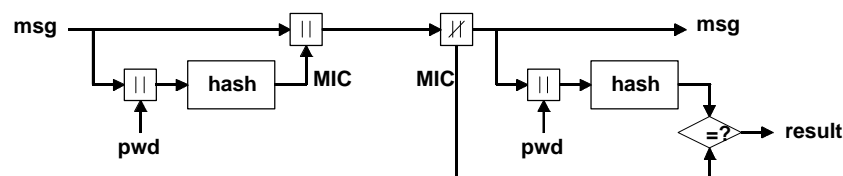
What doing with a Hash

- Encryption
 - encryption should be easy with MD, but what about decryption? ;-)
 - generating a one-time pad
 - just as OFB, generating a pseudorandom bit stream and encrypting the message just by a simple \oplus
 - the pseudorandom stream is generated starting from a MD of a secret: $b_1 = MD(K_{AB} || IV)$, $b_2 = MD(K_{AB} || b_1)$, .., $b_{k+1} = MD(K_{AB} || b_k)$
 - same problems as OFB
 - Mixing in the plaintext
 - as in CFB, the plaintext is mixed in the bit stream generation
 - $b_1 = MD(K_{AB} || IV)$, $b_2 = MD(K_{AB} || c_1)$, .., $b_{k+1} = MD(K_{AB} || c_k)$
 - $c_1 = m_1 \oplus b_1$, $c_2 = m_2 \oplus b_2$, .., $c_k = m_k \oplus b_k$

18

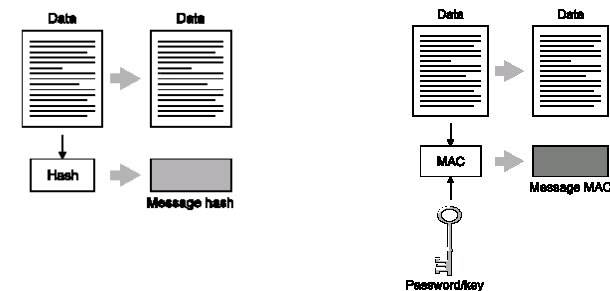
What doing with a Hash

- Computing a MIC (Message Integrity Check)
 - the obvious thought is that $MD(m)$ is a MIC for m , but it isn't; anyone can compute $MD(m)$
 - the way is to send also a (shared) secret
 - if the secret is put at the beginning ($MD(K, m)$), such MD algorithm might become weak since the attacks may continue the MD computation adding a padding
 - however putting the secret at the end might expose the secret
 - a solution could be sending just one half of the hash



19

Hash MIC: Hash Message Authentication Code (H-MAC)



Con una funzione di hash, chiunque può modificare il messaggio iniziale e produrre un nuovo hash

H-MAC (RFC2104) è l'applicazione di una funzione di hash in combinazione con una chiave segreta: solo chi possiede la chiave può generare l'hash

20

Message Authentication Code (MAC)

- a MAC is a cryptographic checksum, generated by an algorithm that creates a small fixed-sized block
 - **depending on both message and a secret key K**
 - $MAC = C_K(M)$
 - **condenses a variable-length message M to a fixed-sized authenticator**
 - it need not be reversible
- **signature**
- receiver performs same computation on message and checks it matches the MAC
- provides assurance that message is unaltered and comes from sender

21

HMAC

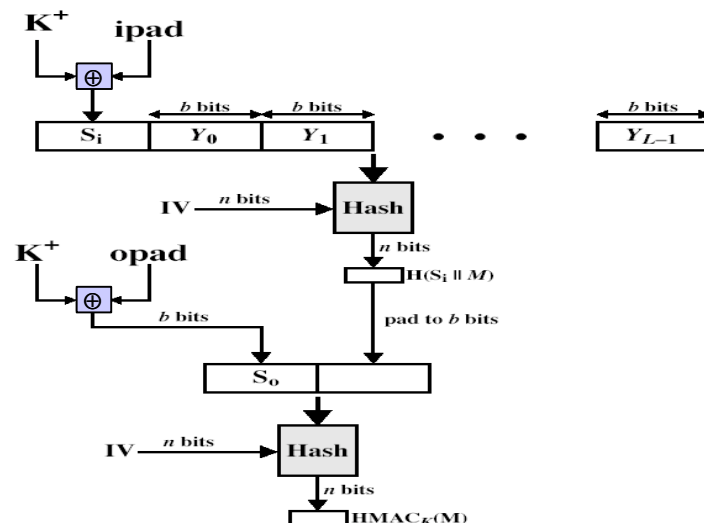
- Specified as Internet standard RFC2104
- Uses hash function on the message:

$$HMAC_K = \text{Hash}[(K^+ \text{ XOR } \text{opad}) \parallel \text{Hash}[(K^+ \text{ XOR } \text{ipad}) \parallel M]]$$

where K^+ is the key padded out to size
and opad, ipad are specified padding constants
- Overhead is just 3 more hash calculations than the message needs alone
- Any of MD5, SHA-1, RIPEMD-160 can be used

22

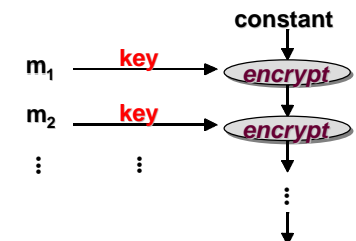
HMAC



23

Using secret key algorithm as Hash Function

- A hash algorithm can be replaced by a block ciphers
 - using $H_0=0$ and zero-pad of final block
 - compute: $H_i = E_{M_i}[H_{i-1}]$
 - and use final block as the hash value
 - similar to CBC but without a key
- resulting hash can be too small (64-bit)
- not very fast to compute



24

Using secret key algorithm as Hash Function

- Example: the UNIX password hash
 - **first convert the passwd (the message) into a secret key**
 - the 7bit ASCII codes of the first 8 chars form the 56bit key
 - **the key is used to encrypt the number 0 with a modified DES**
 - the modified DES is used to prevent HW accelerators designed to DES to be used to reverse the passwd hash
 - the modified algorithm uses a 12-bit random number stored with the hash