Hashes and Message Digests

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

Corso di Sicurezza nelle reti, a.a. 2009/2010

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

Hash Function

- Also known as Message Digest
- it is a function that takes an imput 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 pratical to figure out which input corresponds to a given output**

h=H(m)

- e.g. MD2, MD5 (RFC1321), SHA-1, SHA-1

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)
- Riduce le dimensioni del messaggio, riassumendo le caratteristiche di un messaggio
  - **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)

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)

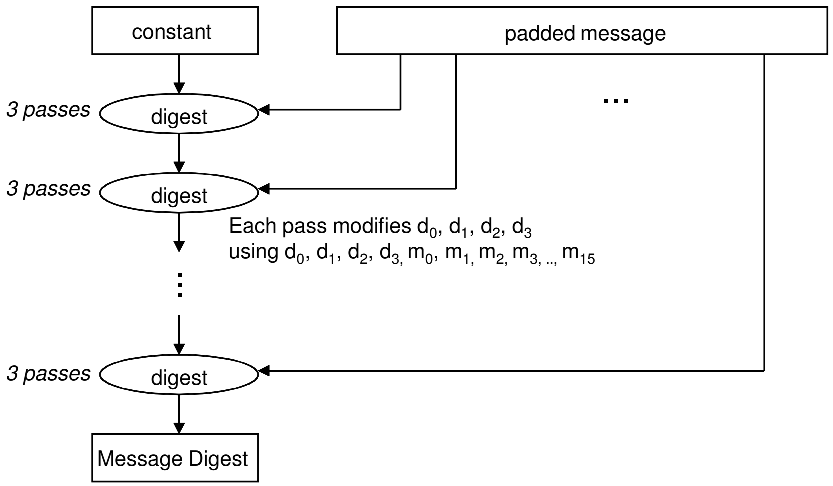
About the hash function

- 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)
  - **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)

MD4

- Designed by Ronald L. Rivest of MIT
- Can handle message with an arbitrary number of bits
- Produce a 128 bit hash
  - **32-bit-world-oriented (instead of byte oriented schemes like MD2)**
- 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 Nx512-64**
  - **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)

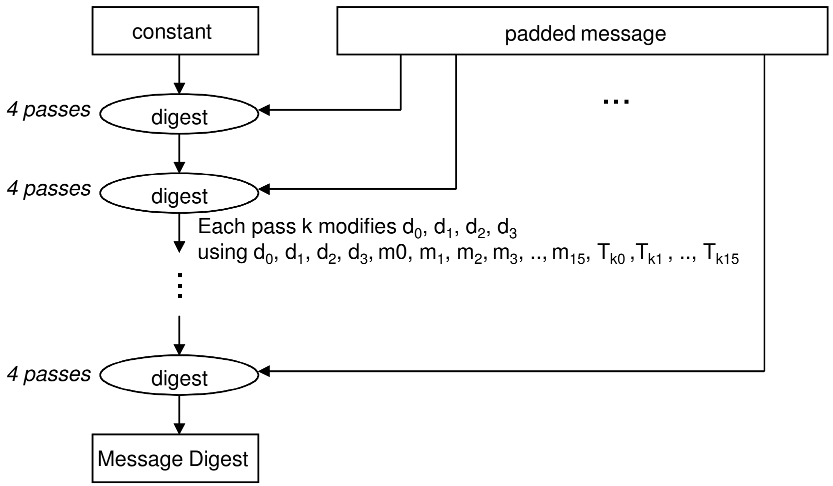
MD4 scheme



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

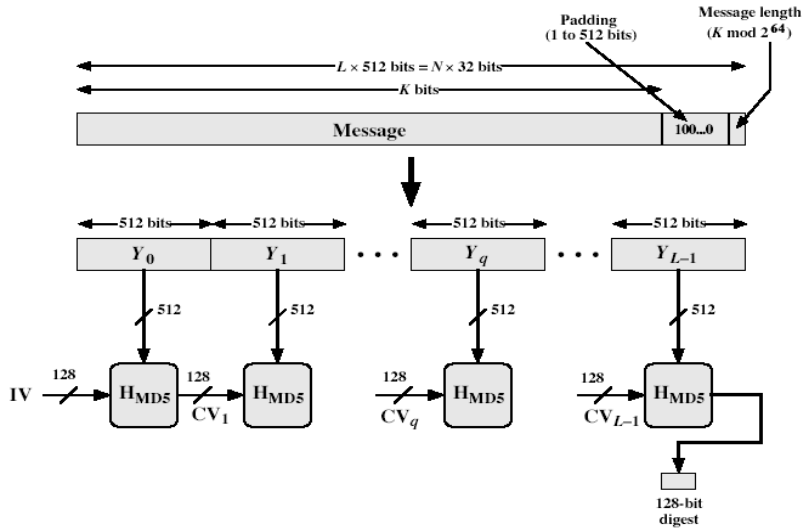
MD5 scheme



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`**

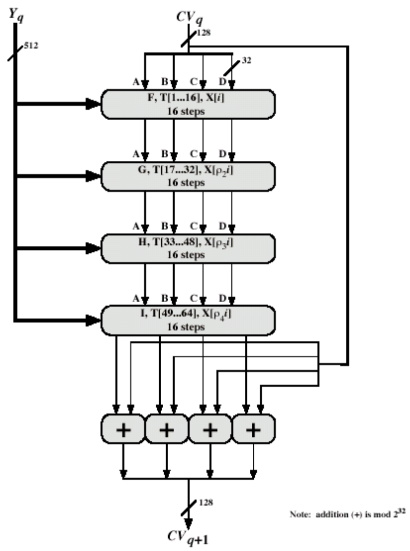
MD5 padding



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 e' il message digest finale

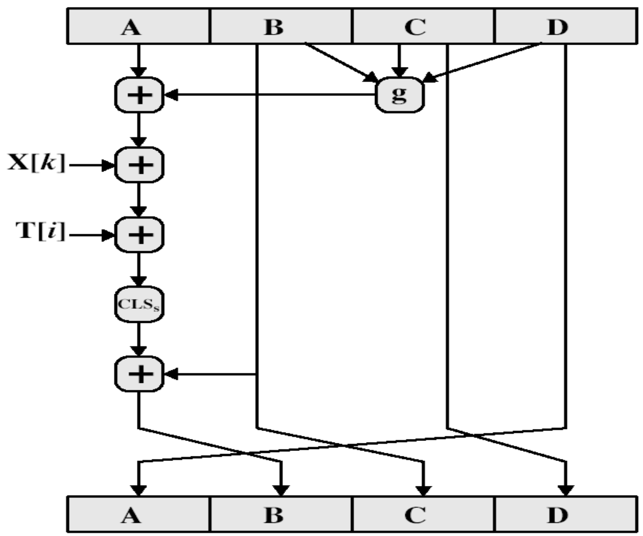
MD5 processing



MD5 processing (4 passes)

- A = B+((A+g(B,C,D)+X[k]+T[i]<<<s)
- Pass 1
  g(x,y,z) = F(x,y,z)=(x ∧ y) ∨(¬x ∧ z)
- Pass 2
  - **G(x,y,z)..**
- Pass 3 …
  - **H(x,y,z)..**
- Pass 4 …
  - **I(x,y,z)..**

MD5 processing (16 steps)



Secure Hash Standard (SHS/SHA)

- Set of cryptographically secure hash algorithms specified by NIST as message digest functions
- The original specification of the algorithm was published in 1993 as the Secure Hash Standard, FIPS PUB 180, by NIST (SHA-0)
  - **Secure Hash Algorithm (SHA)**
- Successively revised by the following standards
  - **SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512**
  - **the latter four variants are sometimes collectively referred to as SHA-2**
  - **SHA-1 (and SHA) produces a message digest that is 160 bits long**
  - **the other algorithms produce digests that are respectively 224, 256, 384, 512 bits long**
- SHA-1 is employed in several widely used security applications and protocols
  - **TLS/SSL, PGP, SSH, S/MIME, IPsec, etc.**

SHA standards

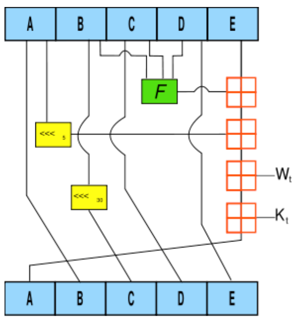
Algoritmo e variante		Dimensione dell'output (bit)	Dimensione dello stato interno (bit)	Dimensione del blocco (bit)	Max. dimensione del messaggio (bit)	Dimensione della *word* (bit)
SHA-0		160	160	512	$2^{64}-1$	32
SHA-1		160	160	512	$2^{64}-1$	32
SHA-2	*SHA-256/224*	256/224	256	512	$2^{64}-1$	32
	SHA-512/384	512/384	512	1024	$2^{128}-1$	64

SHA-1

- SHA-0 was superseded by the revised version SHA-1, published in 1995
  - **SHA-1 differs from SHA-0 only by a single bitwise rotation in the message schedule of its compression function**
  - **this was done, according to the NSA, to correct a flaw in the original algorithm which reduced its cryptographic security**
- SHA-1 (as well as SHA-0) produces a 160-bit (5-word blocks) digest from a message with a maximum length of (264 - 1) bits
  - **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 SHA-1 at 100MIPS**

SHA-1 (cont.)

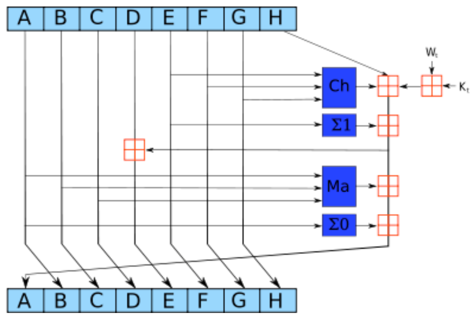
- Based on principles similar to those used by MD4 and MD5 message digest algorithms Pad the message as in MD4 and MD5 (except that the message is limited to 2^{64} bits)
- Operates in stages (as MD4, MD5)
  - **Makes 5 passes for each block of data (3 in MD4 and 4 in MD5)**
  - **Uses a different 160-bit mangle function in each stage**



- Little slower than MD5 and (presumably) little more secure

SHA-2

- SHA-224, SHA-256, SHA-384, and SHA-512
  - **FIPS PUB 180-2 standard in 2002 (SHA-224 variant in 2004)**
- SHA-256 and SHA-512 are computed with 32- and 64-bit words, respectively
  - **use different shift amounts and additive constants**
  - **different number of rounds**
- SHA-224 and SHA-384 are simply truncated versions of the first two, computed with different initial values

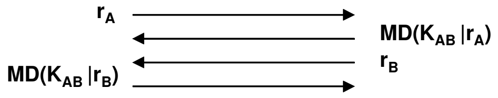


Future of SHA

- SHA-1 has been compromised
- SHA-2 security is not yet as well-established
  - **not received as much scrutiny as SHA-1**
  - **although no attacks have yet been reported, SHA-2 is algorithmically similar to SHA-1**
- An open competition for a new SHA-3 function has been started by NIST on November 2, 2007
  - **similar to the development process for AES**
  - **submissions are due October 31, 2008**
  - **the proclamation of a winner and publication of the new standard are scheduled to take place in 2012**

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**

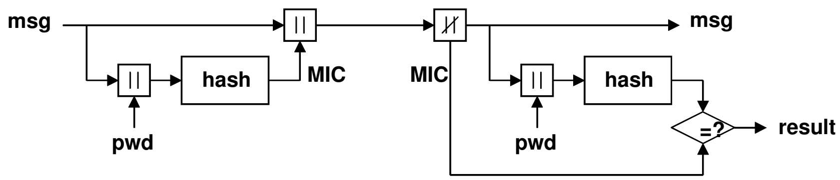


What doing with a Hash

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

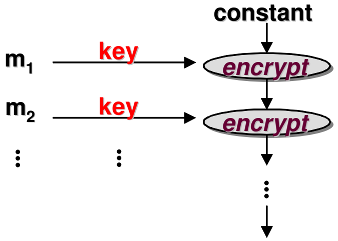
What doing with a Hash

- Computing a MIC (Message Integrity Check) or MAC
  - **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 solutin could be sending just one half of the hash



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



Using secret key algorithm as Hash Function

- Example: the orginal UNIX password hash (crypt function)
  - **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**
    - 25 DES passes are performed
    - 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 (salt)
  - **the salt and the final ciphertext are base64-encoded into a printable string stored in the password or shadow file**
- Currently, the most common crypt function used by Unix/Linux systems supports both the original DES-based and hash-based algorithms (e.g. MD5-crypt function), where common hash function such as MD5 or SHA-1 are used
  - **such functions generally allow users to have any length password (> 8bytes), and do not limit the password to ASCII (7-bit) text**

Unix password hashing

- The MD5-crypt function is really not a straight implementation of MD5
  - **first the password and salt are MD5 hashed together in a first digest**
  - **then 1000 iteration loops continuously remix the password, salt and intermediate digest values**
  - **the output of the last of these rounds is the resulting hash**
- A typical output of the stored password together with username, salt, and other information is:

alice:1BZftq3sP$xEeZmr2fGEnKjVAxzj:12747:0:99999:7:::

  - **where 1 indicates the use of MD5-crypt, while BZftq3sP is the base-64 encoding of the salt and xEeZmr2fGEnKjVAxzjQo68 is the password hash**