



Hash and MAC functions

Luca Veltri

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

Course of Network Security, Spring 2013

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

Hash Function

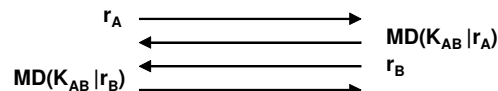
- Also known as Message Digest
- it is a function that takes an input message and produce an output (hash value, or message digest)
- the input can be a variable-length bit string, the output is a fixed-length bit string (e.g. 128 bits)
- It is a one-way function
 - it is not practical to figure out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

$$h=H(m)$$

- e.g. MD2, MD5 (RFC1321), SHA-1, SHA-2

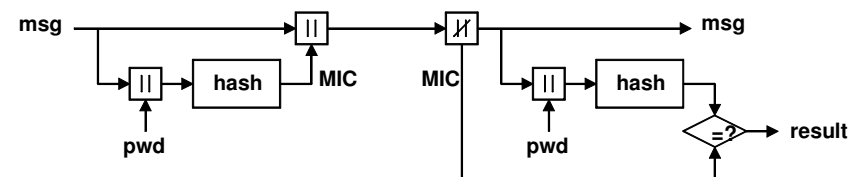
What doing with a Hash

- 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)
- Password Hashing
 - a system may know/store just the hash of a passwd
- 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

- 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



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 \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
 - $b_1=MD(K_{AB}||IV)$, $b_2=MD(K_{AB}|c_1)$, .. , $b_{k+1}=MD(K_{AB}|c_k)$
 - $c_1=m_1\oplus b_1$, $c_2=m_2\oplus b_2$, .. , $c_k=m_k\oplus b_k$

5

Hash function properties

- Input message m of any size
- Output data h of fixed size
- The transformation $H(m)$ is one-way
- It reduces data size, “summarizing” the characteristics of the message
 - allows the detection of possible modifications/errors
- Fast calculation of $h=H(m)$
 - requires low processing resources
- 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)

6

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)
 - this is why message digest functions have output of at least 128 bits (in place of just 64 as for symmetric cryptography)
 - 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

7

About the hash function

- Message digest functions 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)

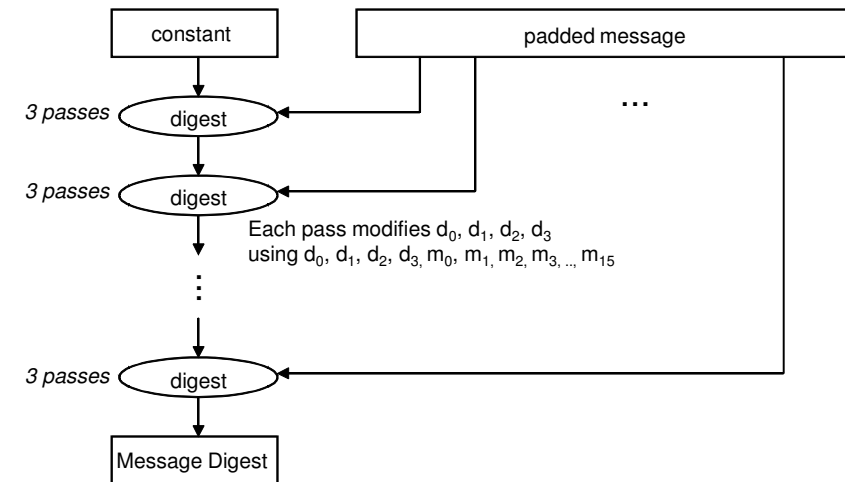
8

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 $N \times 512 - 64$**
 - **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)

9

MD4 scheme



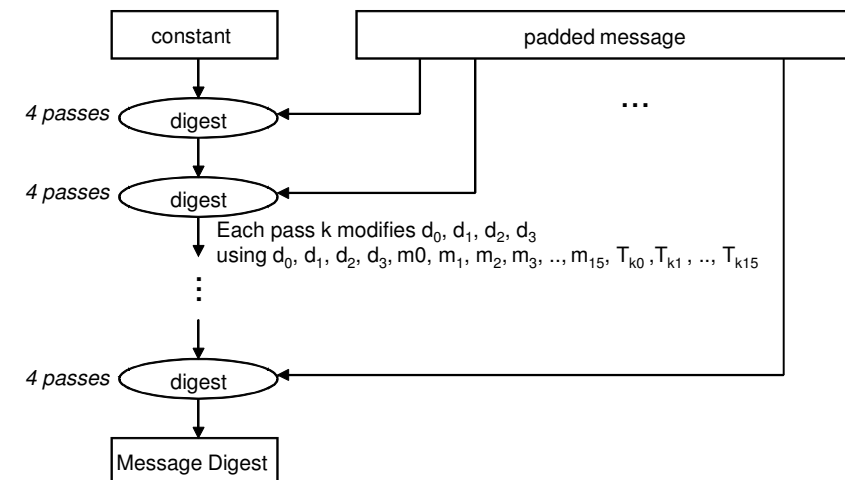
10

MD5

- Designed to be less concerned with speed and more with security, respect to MD4
- Very similar to MD4:
 - **can handle message with an arbitrary number of bits**
 - **produces a 128 bit hash**
 - **32-bit-world oriented (it works on 4 x 32-bit-worlds)**
 - **message processed in 512-bit blocks (16 words)**
 - **message padding**
- 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)**

11

MD5 scheme



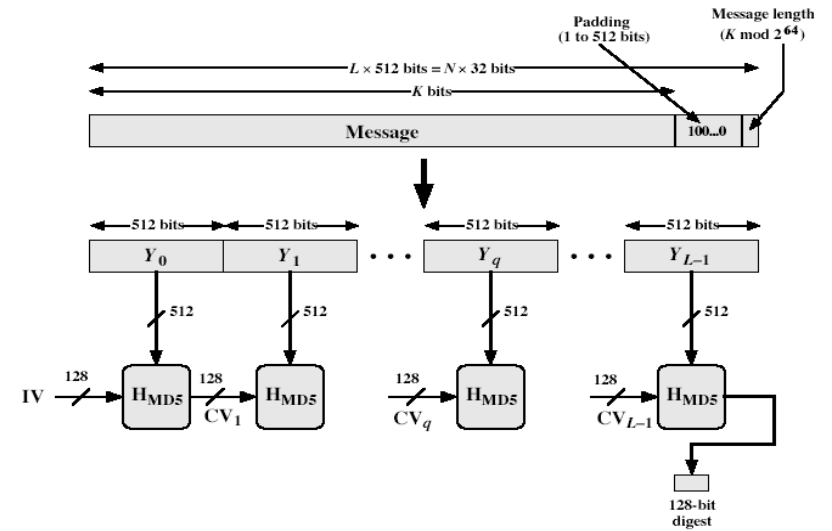
12

MD5 initialization

- Padding
 - **Input message needs to be padded in order to make the length module 512 equal to 448 bit**
 - total length (including padding) becomes 512 minus 64 bit
 - added from 1 to 512 bit as needed
 - padding bits are one "1" followed by zeros
 - **then, 64 bit (512-448) are appended, reporting the message length module 2^{64}**
 - resulting total length becomes 512
- The 128 bit MD buffer, formed by four 32 bit words (A, B, C, D), is initialized to:
 - **A= 01 23 45 67**
 - **B= 89 AB CD EF**
 - **C= FE DC BA 98**
 - **D= 76 54 32 10**

13

MD5 padding



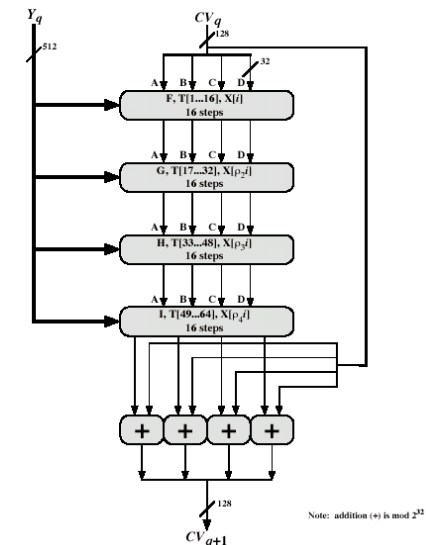
14

MD5 processing

- Message is processed in blocks of size 512 bit (16 word)
- Starting from the initial buffer, 4 processing phases for each block are executed
- For each phase, a different function is used, referred as F, G, H and I
- Each function uses as input:
 - the buffer ABCD of size 128 bit,
 - the current message block Y_q of size 512 bit,
 - 1/4 of a table $T[1..64]$ with 64 values, obtained from the $\sin()$ function
- The output of the fourth is added to the input (module 32), word by word
- The output of the last operation is the resulting message digest

15

MD5 processing



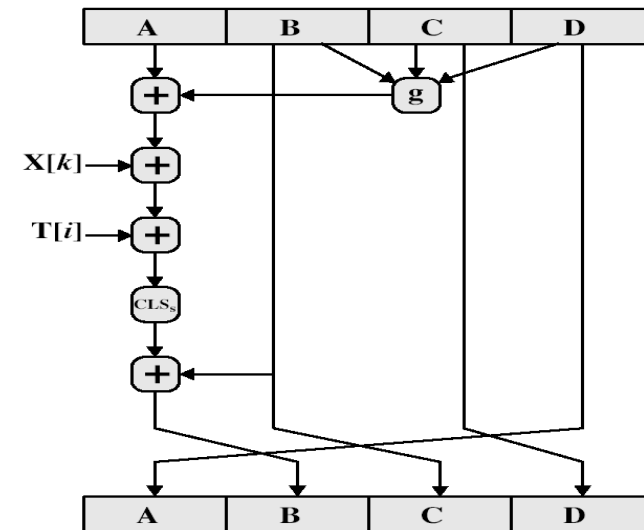
16

MD5 processing (4 passes)

- $A = B + ((A + g(B, C, D) + X[k] + T[i]) \lll s)$
- Pass 1
 $g(x, y, z) = F(x, y, z)$
- Pass 2
 $g(x, y, z) = G(x, y, z)$
- Pass 3 ...
 $g(x, y, z) = H(x, y, z)$
- Pass 4 ...
 $g(x, y, z) = I(x, y, z)$
- with:
 - $F(X, Y, Z) = XY \vee \text{not}(X) Z$
 - $G(X, Y, Z) = XZ \vee Y \text{not}(Z)$
 - $H(X, Y, Z) = X \text{ xor } Y \text{ xor } Z$
 - $I(X, Y, Z) = Y \text{ xor } (X \vee \text{not}(Z))$

17

MD5 processing (16 steps)



18

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.**

19

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-224/224	256	512	$2^{64} - 1$	32
	SHA-256/384	512	1024	$2^{128} - 1$	64

20

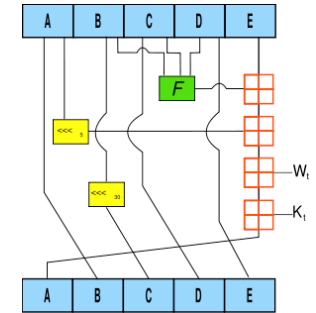
SHA-1

- 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 $(2^{64} - 1)$ 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

21

SHA-1 (cont.)

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

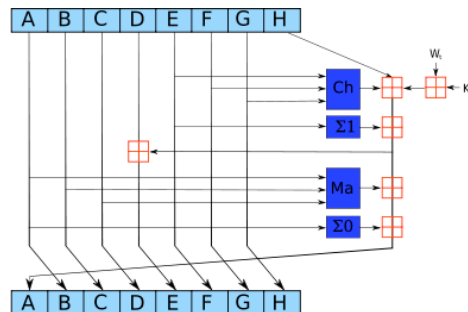


- Little slower than MD5 and (presumably) little more secure

22

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



23

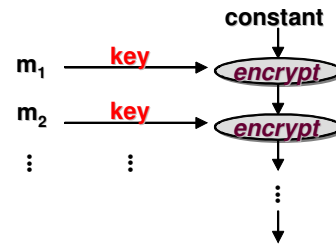
Future of SHA

- 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 was due October 31, 2008
 - the proclamation of a winner and publication of the new standard are scheduled to take place in 2012

24

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



25

Using secret key algorithm as Hash Function

- Example: the original 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

26

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:\$1\$BZftq3sP\$xEeZmr2fGEnKjVAXzjQo68:12747:0:99999:7:::

- where \$1\$ indicates the use of MD5-crypt, while BZftq3sP is the base-64 encoding of the salt and xEeZmr2fGEnKjVAXzjQo68 is the password hash

27

Message Authentication
(data origin authentication, integrity check)

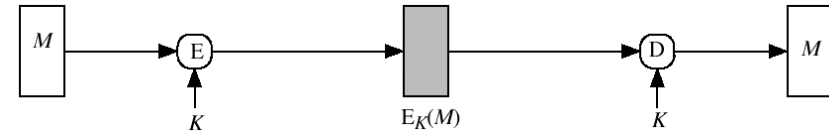
Message Authentication

- Message authentication is concerned with:
 - protecting the integrity of a message
 - validating identity of originator
 - non-repudiation of origin (dispute resolution)
- Three alternative approaches:
 - use of secret or public key encryption algorithms
 - use of encryption and hash algorithms
 - use of ad-hoc (secret key based) Message Authentication Code (MAC) algorithms

29

Msg. Auth. - Secret-key Encryption

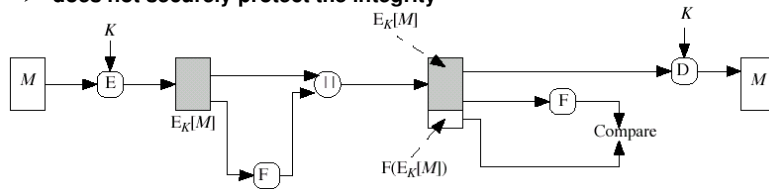
- Symmetric encryption:
 - encryption provides both confidentiality and origin authentication
 - however, need to recognize corrupted messages (MIC)



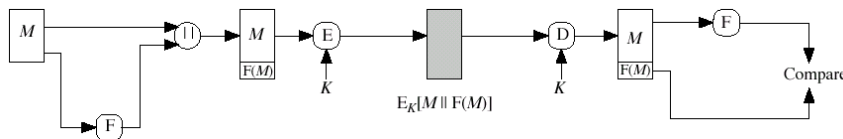
30

Msg. Auth. - Secret-key Encryption + Hash

- External error control (checksum):
 - does not securely protect the integrity



- Message Integrity Check (MIC):
 - example through internal error control - Manipulation Detection Code (MDC)



31

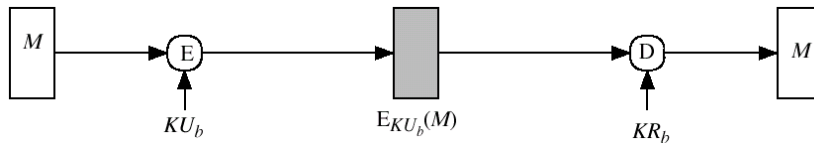
Msg. Auth. - Asymmetric Cryptography

- if public-key encryption is used
 - encryption with public key provides no proof of sender (no sender authentication)
 - since anyone potentially knows public-key
 - both secrecy and authentication if
 - sender "signs" message using their private-key
 - then encrypts with recipients public key
 - problems
 - the result is the same cost of two public-key encryption
 - need to recognize corrupted messages for integrity check

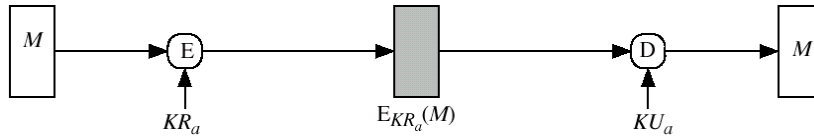
32

Msg. Auth. - Asymmetric Cryptography (cont.)

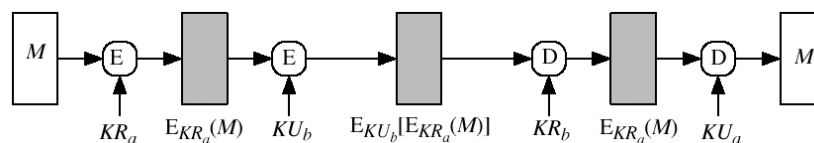
Public-key encryption: confidentiality



Public-key encryption: authentication/signature

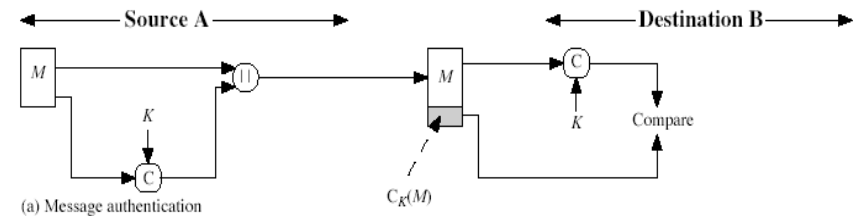
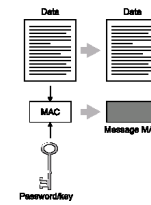


Public-key encryption: confidentiality + authentication/signature



33

Message Authentication Code (MAC)



34

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 needs not be reversible
 - is a many-to-one function
 - potentially many messages have same MAC
 - but finding these needs to be very difficult
- appended to message as a **signature**
- receiver performs same computation on message and checks it matches the MAC
- provides assurance that message is unaltered and comes from sender

35

Message Authentication Code (cont.)

- In case secrecy is also required
 - use of encryption with separate key
 - can compute MAC either before or after encryption
 - is generally regarded as better done before
- why use a MAC?
 - sometimes only authentication is needed
 - sometimes need authentication to persist longer than the encryption (eg. archival use)
- MAC is similar but not equal to digital signature

36

Requirements for MACs

- MAC functions have to satisfy the following requirements:
 - **knowing a message and MAC, is infeasible to find another message with same MAC**
 - **MACs should be uniformly distributed**
 - **MAC should depend equally on all bits of the message**

37

Using Symmetric Ciphers for MACs

- Can use any block cipher chaining mode and use final block as a MAC
- Data Authentication Algorithm (DAA) is a widely used MAC based on DES-CBC
 - **using IV=0 and zero-pad of final block**
 - **encrypt message using DES in CBC mode**
 - **and send just the final block as the MAC**
 - or the leftmost M bits of final block
- But final MAC is now too small for security (≤ 64 bit)

38

MAC Security

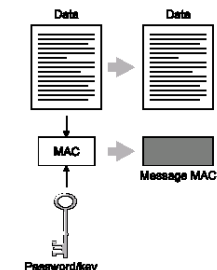
- Cryptanalytic attacks
 - **like block ciphers, brute-force attacks are the best alternative**
- Transient effect
 - **message authentication, as opposed to encryption, has a "transient" effect**
 - **a published breaking of a message authentication scheme would lead to the replacement of that scheme, but would have no adversarial effect on information authenticated in the past**
 - **this is in contrast with encryption, where information encrypted today may suffer from exposure in the future if, and when, the encryption algorithm is broken**

39

Hash Message Authentication Code (H-MAC)

H-MAC (RFC 2104)

- Mechanism for message authentication using cryptographic hash functions in combination with a secret shared key
 - **only who knows the secret key can compute the hash**
- HMAC can be used with any iterative cryptographic hash function, e.g., MD5, SHA-1
 - **the cryptographic strength of HMAC depends on the properties of the underlying hash function**



40

HMAC

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

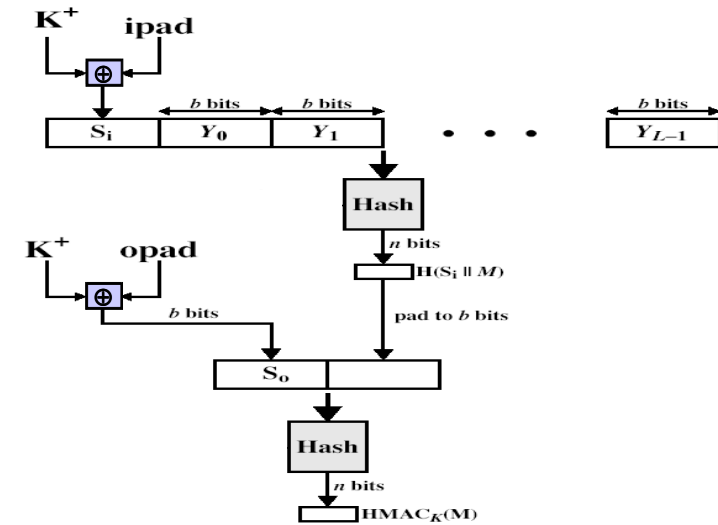
$$\text{HMAC}_K = \text{Hash}[(K^+ \text{ XOR opad}) \parallel \text{Hash}[(K^+ \text{ XOR ipad}) \parallel M]]$$

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

41

HMAC



42

Truncated HMAC

- A well-known practice with MACs is to truncate the output of the MAC and output only part of the bits
 - advantages: less information on the hash result available to an attacker**
 - disadvantages less bits to predict for the attacker**
- It is recommended to let the output length t be not less than half the length of the hash output and not less than 80 bits
- Sometimes HMAC that uses a hash function H with t bits of output is denoted as HMAC-H- t
 - example, HMAC-SHA1-80 denotes HMAC computed using the SHA-1 function and with the output truncated to 80 bits**

43