MAC & Hash Functions

Message Security Requirements

Disclosure — **Message Encryption**

traffic analysis — **Message Encryption**

masquerade — **Message Authentication & Message Integrity**

content modification — **Message Authentication & Message Integrity**

sequence modification — **Message Authentication & Message Integrity**

timing modification — **Message Authentication & Message Integrity**

source repudiation — **Digital Signature**

destination repudiation — **Digital Signature**

Message Authentication

Encryption protects message against passive attack, while Message Authentication protects against active attack.

message authentication is concerned with:

- protecting the integrity of a message
- validating identity of originator
- non-repudiation of origin (dispute resolution)

Approaches for message authentication

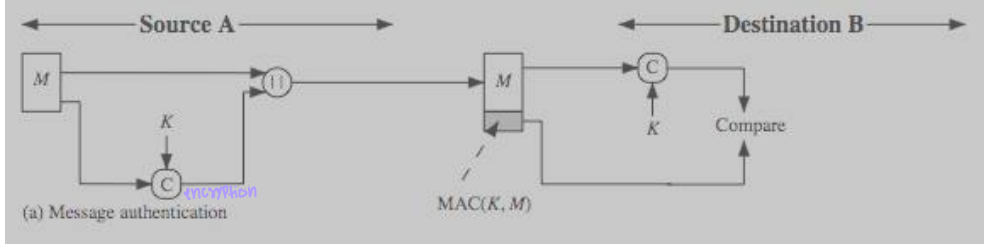
- Authentication using conventional encryption.
  - Symmetric Encryption
- Authentication using Public Key Encryption
- Message Authentication without message encryption.
  - MAC

Message Authentication without message encryption

- Message is not always encrypted, but an authentication tag is generated and appended to the message.
- Authentication and Confidentiality services are combined in a single algorithm by encrypting the message plus the authentication tag.

Message Authentication Code (MAC)

- a small fixed-sized block of data
  - generated from message + secret key
  - MAC = C(K,M)
  - appended to message when sent



Message Authentication Code (MAC)

generated by an algorithm that creates a small fixed-sized block

- depending on both message and some key
- like encryption though need not be reversible

appended to message as a **signature**

receiver performs same computation on message and checks it matches the MAC

Message Authentication Code (MAC)

- If MAC matches then message is authenticated , and this means

1. Message is not altered.
2. Message is coming from alleged user.
3. If message includes a sequence number (as used with TCP) then ordering is guaranteed.

A number of algorithms are used to generate the MAC.

- NIST recommends to use DES to generate an encrypted version of the message and the last number of bits in the ciphertext is used as the code (16 or 32 bits).
- Authentication algorithm needs not to be reversible.

Message Authentication Code (MAC)

as shown the MAC provides authentication

can also use encryption for secrecy

- generally use separate keys for each
- 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)

note that a MAC is not a digital signature

Requirements for MACs

taking into account the types of attacks

need the MAC to satisfy the following:

1. knowing a message and MAC, is infeasible to find another message with same MAC
2. MACs should be uniformly distributed
3. MAC should depend equally on all bits of the message

HMAC Design Objectives

- use, without modifications, hash functions
- allow for easy replaceability of embedded hash function
- preserve original performance of hash function without significant degradation
- use and handle keys in a simple way.
- have well understood cryptographic analysis of authentication mechanism strength

HMAC Algorithm

specified as Internet standard RFC2104

uses hash function on the message:

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

- where K^+ is the key padded out to size
- $opad$, $ipad$ are specified padding constants

overhead is just 3 more hash calculations than the message needs alone

any hash function can be used

- eg. MD5, SHA-1, RIPEMD-160, Whirlpool

HMAC Security

proved security of HMAC relates to that of the underlying hash algorithm

attacking HMAC requires either:

- brute force attack on key used
- birthday attack (but since keyed would need to observe a very large number of messages generated using the same key)

choose hash function used based on speed versus security constraints

One-way hash function

- Alternative to MAC.
- Accept a variable –sized message M as input.
- Produce fixed-sized message digest H(M).
- Unlike MAC: a hash function doesn't take a secret key as input.
- To authenticate a message: the message digest is sent with the message.

Hash functions

A hash function computes a fixed length value from a variable length source

- Example: Check sums in communication protocols
- Indices in databases

More convenient to handle a hash of a document instead of the document itself

We will consider *cryptographically secure hash functions*.

Hash Functions are Versatile

Hash functions are used for :

- message and file integrity
- secure login
- fingerprints of keys
- authentication
- digital signatures

[From Andreas Klappenecker]

Hash functions, definition

- A hash function is a function f:{0,1}* → {0,1}n.
- The size of the output, n, is a property of the function. Common values are 128, 160 and 256.
- Commonly used hash functions are MD5, SHA and SHA-1

Properties of good hash functions

Let H be a hash function

One-way

- Given x, unfeasible to compute an v such that H(v) = x

Collision-free

- Unfeasible to find x1 and x2 such that H(x1) = H(x2) and x1 ≠ x2

Desirable properties of hash function

1. It should be possible to efficiently compute the hash value z=H(m) of a message m.

2. Given the hash value z=H(m), it should be computationally infeasible to find m. A function with this property is called a one-way function.

3. Given a message m, it should be infeasible to find another message m' such that H(m)=H(m').

4. It should be infeasible to find two messages m and m' such that H(m)=H(m').

Property 3) is known as weak collision resistance, and

Property 4) is known as strong collision resistance.

[From Andreas Klappenecker]

Hash Functions & Authenticated Encryption

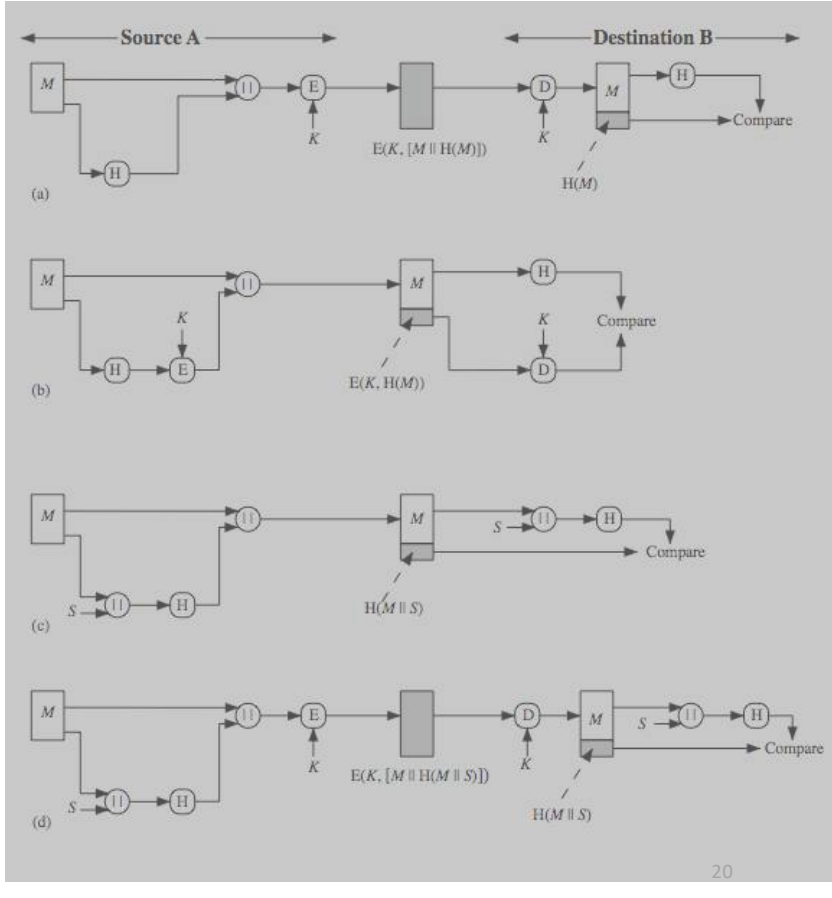
simultaneously protect confidentiality and authenticity of communications

- often required but usually separate

approaches

- Hash-then-encrypt: E(K, (M || H(M))
- MAC-then-encrypt: E(K2, (M || MAC(K1, M))
- Encrypt-then-MAC: (C=E(K2, M), T=MAC(K1, C)
- Encrypt-and-MAC: (C=E(K2, M), T=MAC(K1, M)

Hash Functions & Message Authent-ication



Hash Function Requirements

Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value h, it is computationally infeasible to find y such that H(y) = h.
Second preimage resistant (weak collision resistant)	For any given block x, it is computationally infeasible to find y ! x with H(y) = H(x).
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) such that H(x) = H(y).
Pseudorandomness	Output of H meets standard tests for pseudorandomness

Attacks on Hash Functions

- Brute-force attacks and cryptanalysis
- a preimage or second preimage attack
  - find y s.t. $H(y)$ equals a given hash value
- collision resistance
  - find two messages x & y with same hash so $H(x) = H(y)$
- cryptanalytic attacks exploit some property of algorithm, so faster than exhaustive search

Example

- Suppose a hash function H produces n bit values.
- Compose a document nice treaty and about 2n/2+1 semantically equivalent versions.
- Similarly, compose an evil treaty and about 2n/2+1 semantically equivalent versions.
- With probability ½ or more there will be a version of the nice treaty and a version of the evil treaty that have the same hash value.

[From Andreas Klappenecker]

UNIX Password System

Uses modified DES as if it were a hash function

- Encrypt NULL string using password as the key
  - Truncates passwords to 8 characters!
- Artificial slowdown: run DES 25 times
- Can instruct modern UNIXes to use MD5 hash function

Problem: passwords are not truly random

- With 52 upper- and lower-case letters, 10 digits and 32 punctuation symbols, there are 948 ≈ 6 quadrillion possible 8-character passwords
- Humans like to use dictionary words, human and pet names ≈ 1 million common passwords

[From A. Nascimento]

Unix passwords

httpd:Nologin:100:22:httpd:/usr/users/httpd:/bin/sh

guest:41LYDCYHYJzHQ:200:15:Guest:/usr/users/guest:/bin/tcsh

oracle:Nologin:201:200::/usr/users/oracle:/bin/tcsh

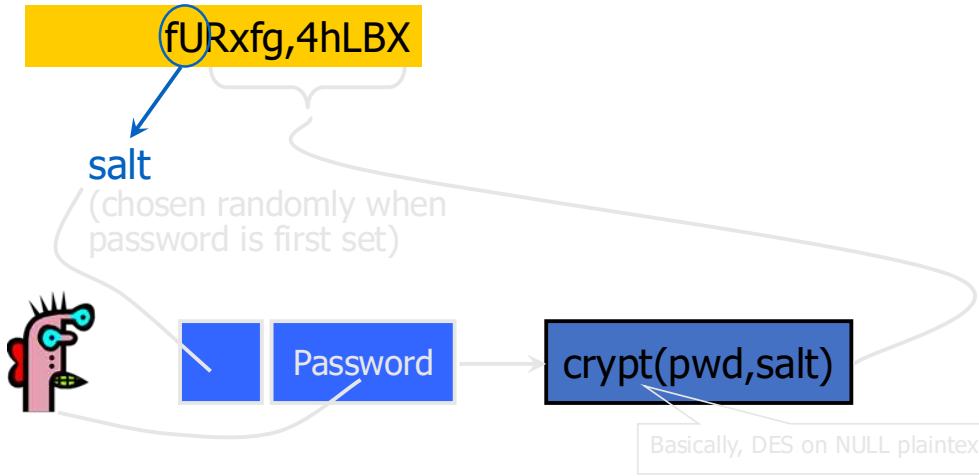
mysql:LS6qP.LbvchSk:202:202::/usr/users/mysql:/bin/tcsh

Andris:Ie7K1yjGLDqsw:203:203::/usr/users/Andris:/bin/tcsh

Password length up to 8 characters, encrypted by 1-way hash function crypt(3).

Are they safe?

Salt



- Users with the same password have <u>different</u> entries in the password file
- Dictionary attack is still possible!

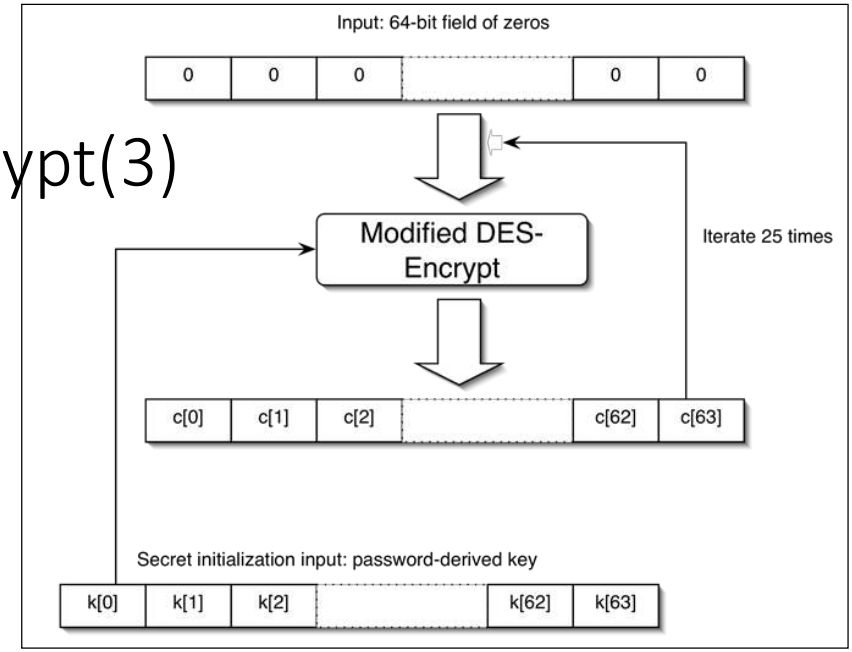
[From A. Nascimento]

Unix passwords - crypt(3)

The salt introduces disorder in the DES algorithm in one of 16777216 or 4096 possible ways (ie. with 24 or 12 bits: if bit i of the salt is set, then bits i and i+24 are swapped in the DES E-box output).

The DES key is used to encrypt a 64-bit constant using count iterations of DES. The value returned is a null-terminated string, 20 or 13 bytes (plus null) in length, consisting of the setting followed by the encoded 64-bit encryption.

Unix crypt(3)

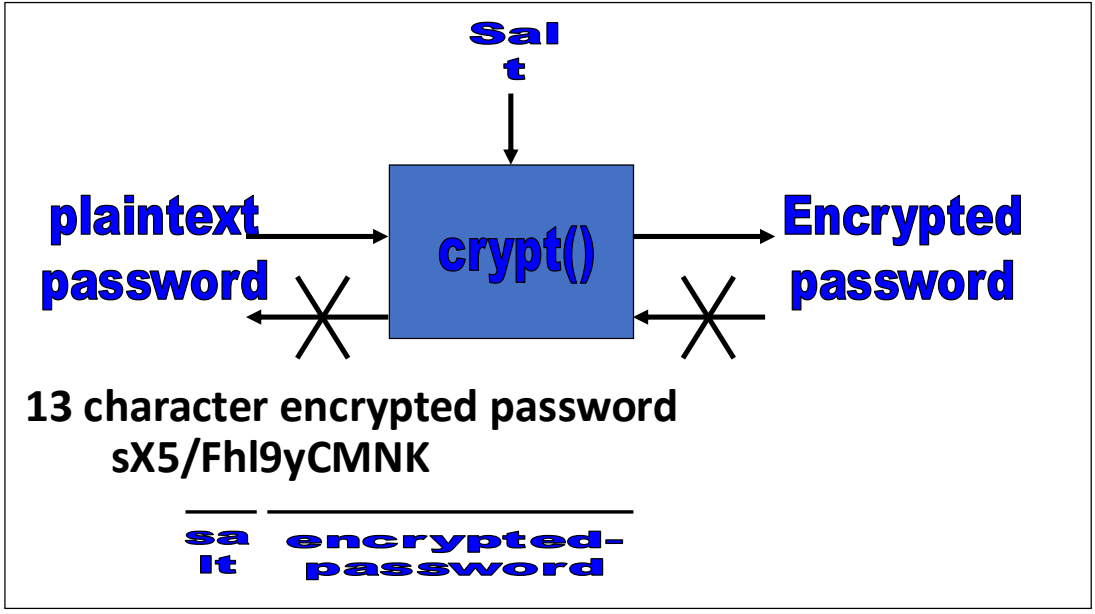


Modifications to use larger passwords were adopted. These separate the password in groups of 8 characters, generating the first key from the first group, and then XOR-ing keys for subsequent groups into the DES encryption of the current key using itself as a key.

[From B.Madeiros]

UNIX Password security

Overview of Unix encrypted passwords using crypt(3)



Unix passwords - salt

httpd:Nologin:100:22:httpd:/usr/users/httpd:/bin/sh
guest:41LYDCYHYJzHQ:200:15:Guest:/usr/users/guest:/bin/tcsh
oracle:Nologin:201:200::/usr/users/oracle:/bin/tcsh
mysql:LS6qP.LbvchSk:202:202::/usr/users/mysql:/bin/tcsh
Andris:Ie7K1yjGLDqsw:203:203::/usr/users/Andris:/bin/tcsh

13 characters (Base64 - 6 bit),78 bits

64 bit hashed value
12 bits of "salt"

Dictionary Attack

Password file /etc/passwd is world-readable

- Contains user IDs and group IDs which are used by many system programs

Dictionary attack is possible because many passwords come from a small dictionary

- Attacker can compute H(word) for every word in the dictionary and see if the result is in the password file
- With 1,000,000-word dictionary and assuming 10 guesses per second, brute-force online attack takes 50,000 seconds (14 hours) on average
  - This is very conservative. Offline attack is much faster!

[From A. Nascimento]

Dictionary Attack – some numbers

Typical password dictionary

- 1,000,000 entries of common passwords
  - people's names, common pet names, and ordinary words.
- Suppose you generate and analyze 10 guesses per second
  - This may be reasonable for a web site; offline is much faster
- Dictionary attack in at most 100,000 seconds = 28 hours, or 14 hours on average

If passwords were random

- Assume six-character password
  - Upper- and lowercase letters, digits, 32 punctuation characters
  - 689,869,781,056 password combinations.
  - Exhaustive search requires 1,093 years on average

[From J.Mitchell]

Hash Algorithms

- **The message digest algorithm MD5 by Ron Rivest with 128 bit hash values.**

- **The secure hash algorithm SHA-1. It was developed by NSA and standardized by NIST. This algorithm uses 160 bit hash values encoded in 5 x 32 bit words.**

- **The family SHA-256, SHA-384, SHA-512 of hash functions that are supposed to be used with AES. They will be part of the NIST Cryptographic Toolkit.**

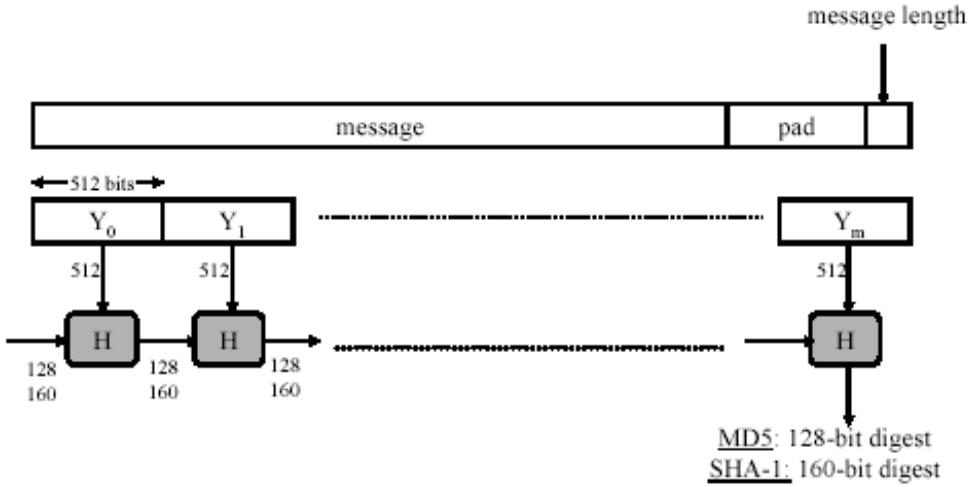
Why are these bit lengths used?

Collisions in SHA-1 can be found by 2^{63} attempts
Collision in MD5 can be found in 8 hours using a notebook PC...

[From Andreas Klappenecker]

Hash Algorithms - general structure

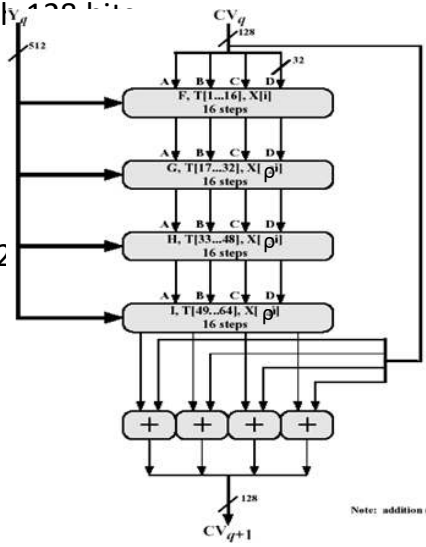
At the end of the four rounds, the result is added to the previous values of ABCD.



[From Andreas Klappenecker]

MD5 Message Digest Algorithm

- It compresses messages of 512 bits length into a hash of length 128 bits
- A message of arbitrary length is padded to length
  - k = 448 mod 512
- A 64 bit string describing the length of the
- message is added. The message length is now a multiple of 512
- The hashing is done block-by-block.

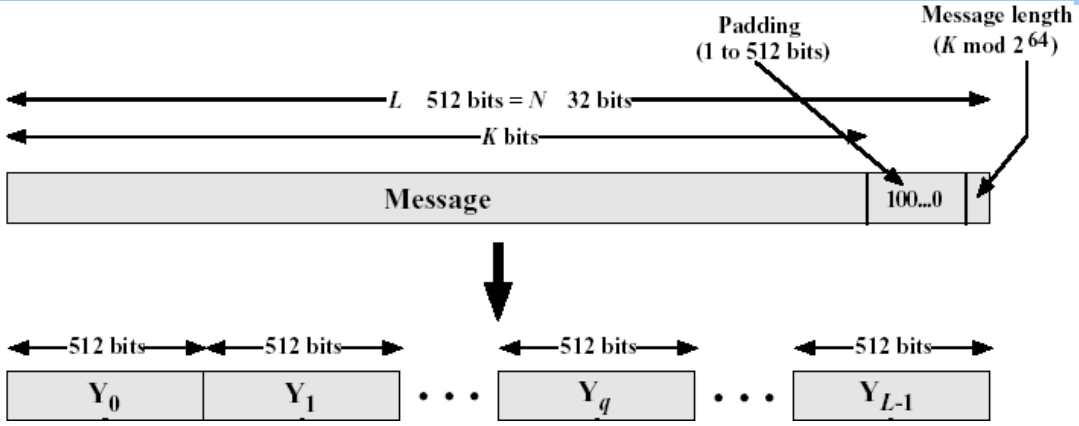


[From Andreas Klappenecker]

MD5 Message Digest Algorithm

Step 1: Append padding bits

- Padded so that its bit length ≡ 448 mod 512 (i.e., the length of padded message is 64 bits less than an integer multiple of 512 bits)
- Padding is always added, even if the message is already of the desired length (1 to 512 bits)
- Padding bits: 1000….0 (a single 1-bit followed by the necessary number of 0-bits)

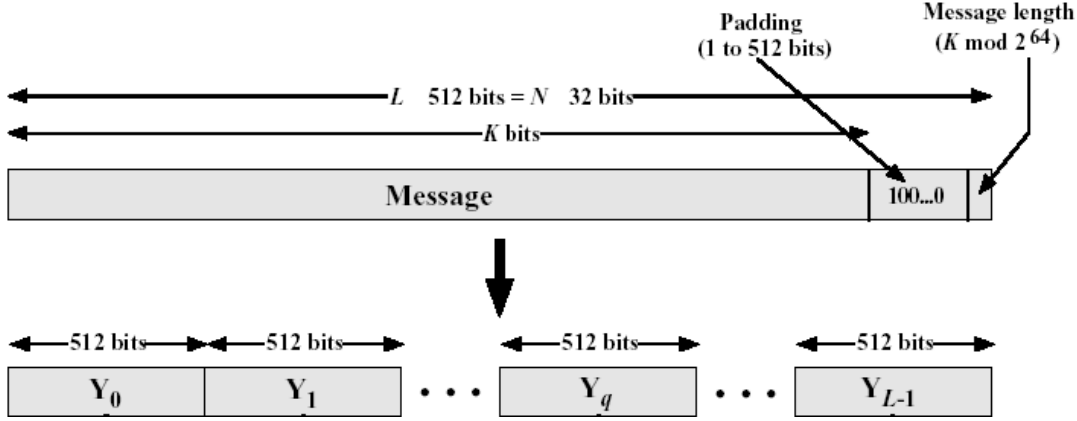


[From H. Yoon]

MD5 Message Digest Algorithm

Step 2: Append length

- 64-bit length: contains the length of the original message modulo 2^{64}
- The expanded message is $Y_0, Y_1, \ldots, Y_{L-1}$; the total length is L × 512 bits
- The expanded message can be thought of as a multiple of 16 32-bit words
- Let M[0 … N-1] denote the word of the resulting message, where N = L × 16



[From H. Yoon]

MD5 Message Digest Algorithm

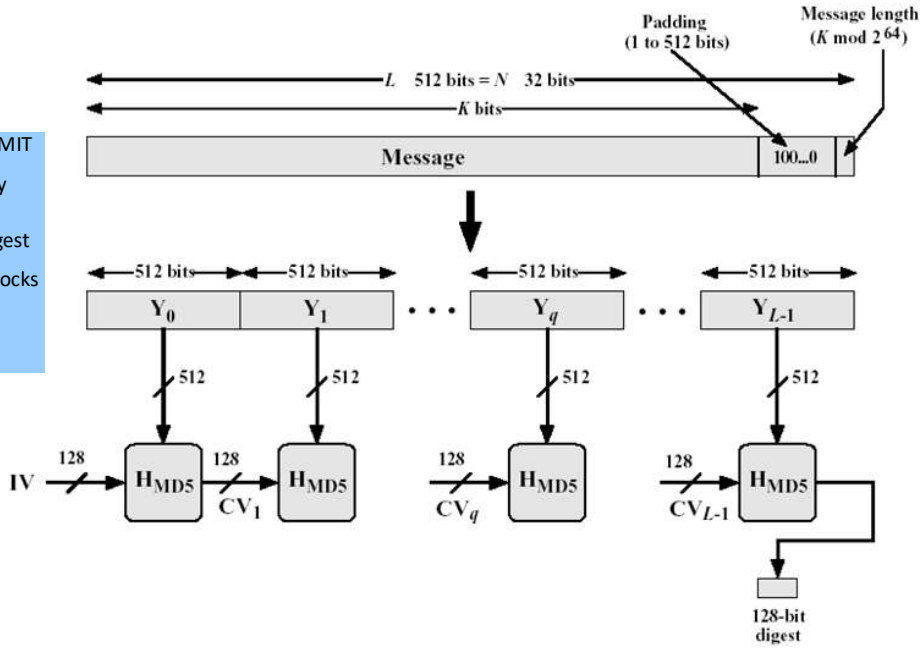
Developed by Ron Rivest at MIT

Input: a message of arbitrary length

Output: 128-bit message digest

32-bit word units, 512-bit blocks

Son of MD2, MD4



[From H. Yoon]

MD5 Reference

- A detailed description of MD5 is contained in RFC1321.
- Hans Dobbertin has shown that MD5 is not collision resistant, so it is not advisable to use this algorithm.
- It is used in IPSec and other protocols**.**

[From Andreas Klappenecker]

MD4

Precursor to MD5

Design goals of MD4 (which are carried over to MD5)

- Security
- Speed
- Simplicity and compactness
- Favor little-endian architecture

[From H. Yoon]

MD4

Main differences between MD5 and MD4

1. A fourth round has been added.
2. Each step now has a unique additive constant.
3. The function g in round 2 was changed from (bc v bd v cd) to (bd v c d') to make g less symmetric.
4. Each step now adds in the result of the previous step. This promotes a faster "avalanche effect".
5. The order in which input words are accessed in rounds 2 and 3 is changed, to make these patterns less like each other.
6. The shift amounts in each round have been approximately optimized, to yield a faster "avalanche effect." The shifts in different rounds are distinct.

[From H. Yoon]

Secure Hash Algorithm

- SHA originally designed by NIST & NSA in 1993
- was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174
  - Note, the algorithm is SHA, the standard is SHS
- based on design of MD4 with key differences
- produces 160-bit hash values
- recent 2005 results on security of SHA-1 have raised concerns on its use in future applications

Revised Secure Hash Standard

- NIST issued revision FIPS 180-2 in 2002
- adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1
- hence analysis should be similar
- but security levels are rather higher

SHA Versions

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message digest size	160	224	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size	512	512	512	1024	1024
Word size	32	32	32	64	64
Number of steps	80	64	64	80	80

Secure Hash Algorithm (SHA)

Developed by NIST (National Institute of Standards and Technology)

- Published as a FIPS PUB 180 in 1993
- A revised version is issued as FIPS PUB 180-1
- Generally referred to as SHA-1

Input: a message with a maximum length of less than 2^{64} bits

Output: 160-bit message digest

32-bit word units, 512-bit blocks

4 rounds × 20 steps per block

Closely models MD4

Slower, stronger than MD5

[From H. Yoon]

SHA-1 logic

The overall structure and logic is similar to MD5

Step 1: Append padding bits

Step 2: Append length

Step 3: Initialize MD buffer

- 160-bit buffer (five 32-bit registers A,B,C,D,E) is used to hold intermediate and final results of the hash function
- A,B,C,D,E are initialized to the following values
- A,B,C,D = same as in MD5, E = C3D2E1F0
- Stored in *big-endian* format (most significant byte of a word in the low-address byte position)
    - E.g. word E: C3 D2 E1 F0 (low address … high address)

[From H. Yoon]

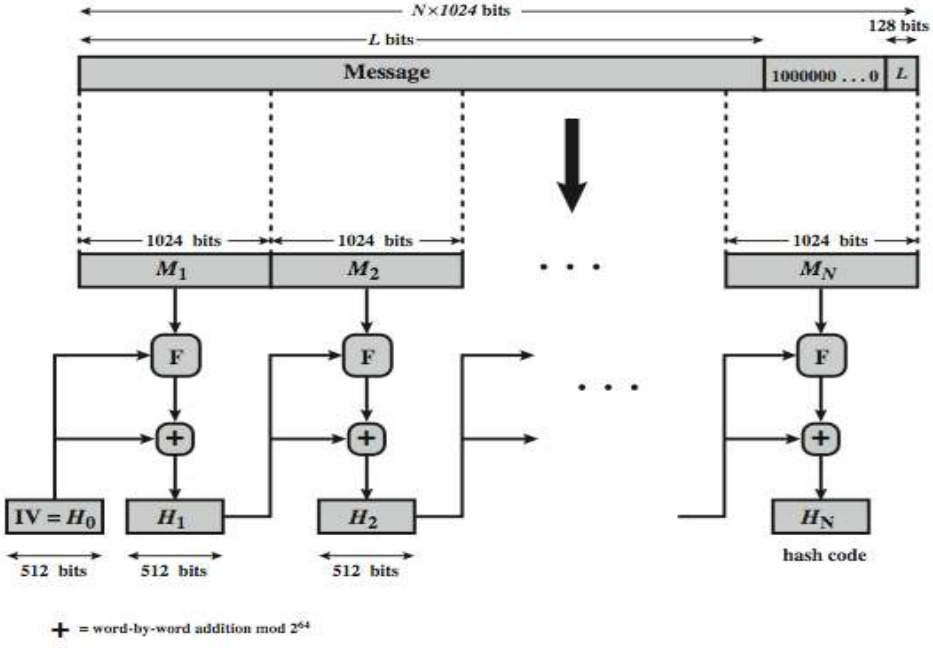
Step 4: Process message in 512-bit (16-word) blocks

- Heart of the algorithm called a *compression function*
- Consists of 4 rounds of processing of 20 steps each
- The 4 rounds have a similar structure, but each uses a different *primitive logical functions*, referred to as f_1, f_2, f_3, and f_4
- Each round takes as input the current 512-bit block (Y_q), 160-bit buffer value ABCDE and updates the contents of the buffer
- Each round also uses the additive constants K_t, where $0 \le t \le 79$ indicates one of the 80 steps across 4 rounds
- In fact only 4 constants are used:
- The output of 4th round (80th step) is
- added to the CV_q to produce CV_{q+1}

Step Number	Hexadecimal	Inetger Paqrt of
$0 \le t \le 19$	K_t = 5A827999	$[2^{30} \times \sqrt{2}]$
$20 \le t \le 39$	K_t = 6ED9EBA1	$[2^{30} \times \sqrt{3}]$
$40 \le t \le 59$	K_t = 8F1BBCDC	$[2^{30} \times \sqrt{5}]$
$60 \le t \le 79$	K_t = CA62C1D6	$[2^{30} \times \sqrt{10}]$

[From H. Yoon]

SHA-512 Overview



SHA-512 Compression Function

heart of the algorithm

processing message in 1024-bit blocks

consists of 80 rounds

- updating a 512-bit buffer
- using a 64-bit value Wt derived from the current message block
- and a round constant based on cube root of first 80 prime numbers