Assignment A Elective - III D

Aim:

To understand different hashing algorithms.

Problem Statement:

Write a program in python/ Java/ Scala/ C++/ HTML5 to implement password data encryption. Use encryption method overloading (any to methods studied)

Input:

Password string.

Output:

Message digest of password.

Theory:

Message digests take the data in a message and generate a block of bits designed to represent the "fingerprint" of the message.

What is a message digest?
A *message digest* is a function that ensures the integrity of a message. Message digests take a message as input and generate a block of bits, usually several hundred bits long, that represents the fingerprint of the message. A small change in the message (say, by an interloper or eavesdropper) creates a noticeable change in the fingerprint.
The message-digest function is a one-way function. It is a simple matter to generate the fingerprint from the message, but quite difficult to generate a message that matches a given fingerprint.
Message digests can be weak or strong. A checksum -- which is the XOR of all the bytes of a message -- is an example of a weak message-digest function. It is easy to modify one byte to generate any desired checksum fingerprint. Most strong functions use hashing. A 1-bit change in the message leads to a massive change in the fingerprint (ideally, 50 percent of the fingerprint bits change).
Increasing power of brute-force attacks lead to evolution in algorithms from DES to AES in block ciphers. Similarly it led to evolution from MD4 & MD5 to SHA-1 & RIPEMD-160 in hash algorithms

Message Digest Algorithms:

MD2 and MD5, which are 128-bit algorithms
· SHA-1, which is a 160-bit algorithm
· SHA-256, SHA-383, and SHA-512, which offer longer fingerprint sizes of 256, 383, and 512 bits, respectively

MD5 and SHA-1 are the most used algorithms.

MD5:

- designed by Ronald Rivest (the R in RSA)
- latest in a series of MD2, MD4
- produces a 128-bit hash value
- until recently was the most widely used hash algorithm
    - in recent times have both brute-force & cryptanalytic concerns
- specified as Internet standard RFC1321

MD5 Overview:

1. pad message so its length is 448 mod 512
    - Padding of 1-512 bits is always used.
    - Padding: 1000….0
2. append a 64-bit length value to message
    - Generate a message with 512L bits in length
3. initialise 4-word (128-bit) MD buffer (A,B,C,D)
4. process message in 16-word (512-bit) blocks:
5. output hash value is the final buffer value

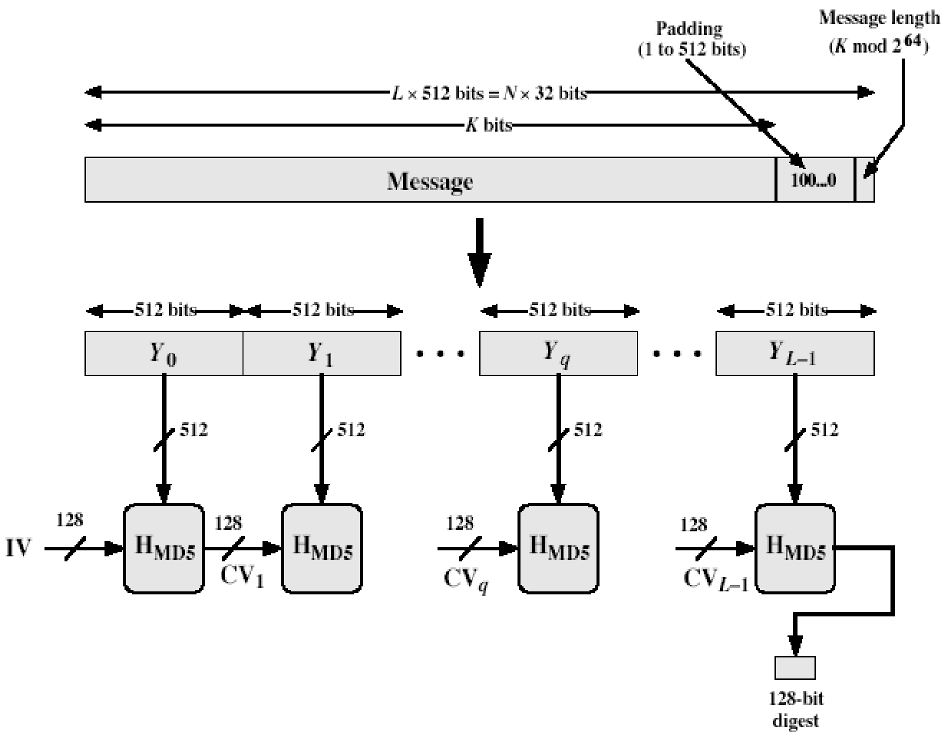

Figure 1: MD5 overview

Strength of MD5:

- Every hash bit is dependent on all message bits
- Rivest conjectures security is as good as possible for a 128 bit hash
    - Given a hash, find a message: O(2^{128}) operations
    - No disproof exists yet
- known attacks are:
    - Berson 92 attacked any 1 round using differential cryptanalysis (but can't extend)
    - Boer & Bosselaers 93 found a pseudo collision (again unable to extend)
    - Dobbertin 96 created collisions on MD compression function for one block, cannot expand to many blocks
    - Brute-force search now considered possible

Secure Hash Algorithm (SHA-1):

- SHA was designed by NIST & NSA in 1993, revised 1995 as SHA-1

- US standard for use with DSA signature scheme
    - standard is FIPS 180-1 1995, also Internet RFC3174
    - nb. the algorithm is SHA, the standard is SHS
- produces 160-bit hash values
- now the generally preferred hash algorithm
- based on design of MD4 with key differences

SHA Overview:

1. pad message so its length is 448 mod 512
2. append a 64-bit length value to message
3. initialise 5-word (160-bit) buffer (A,B,C,D,E) to

   (67452301,efcdab89,98badcfe,10325476,c3d2e1f0)

1. process message in 16-word (512-bit) chunks:
    - expand 16 words into 80 words by mixing & shifting
    - use 4 rounds of 20 bit operations on message block & buffer
    - add output to input to form new buffer value
2. output hash value is the final buffer value

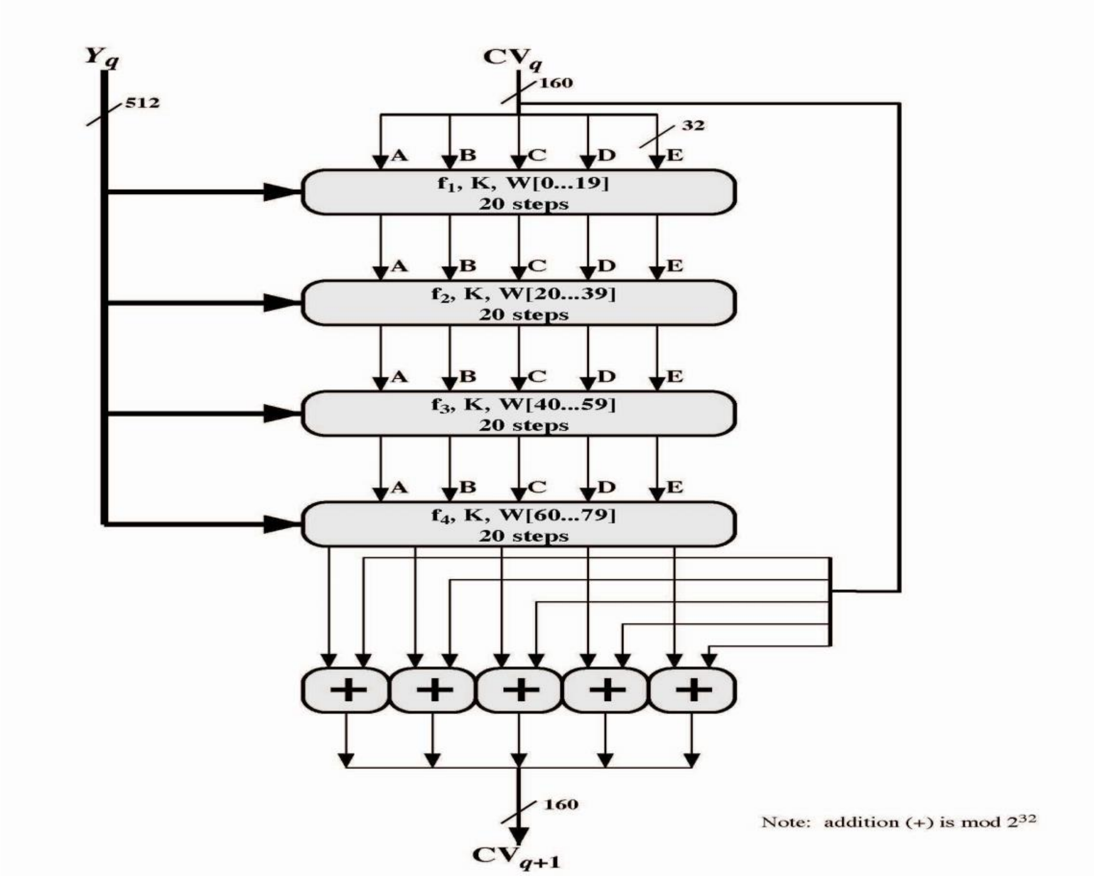

Figure 2: SHA-1 overview

SHA-1 verses MD5:

- brute force attack is harder (160 vs 128 bits for MD5)
- not vulnerable to any known attacks (compared to MD4/5)
- a little slower than MD5 (80 vs 64 steps)
- both designed as simple and compact
- optimised for big endian CPU's (SUN) vs MD5 for little endian CPU's (PC)

The MessageDigest class in Java manipulates message digests. The following methods are used :
· MessageDigest.getInstance("MD5"): Creates the message digest.
· update (plaintext): Calculates the message digest with a plaintext string.
· digest(): Reads the message digest.
If a key is used as part of the message-digest generation, the algorithm is known as a ***message-authentication code*** .

The Mac class manipulates message-authentication codes using a key produced by the KeyGenerator class. The following methods are used in for Message authentication code:
· KeyGenerator.getInstance("HmacMD5") and .generateKey(): Generates the key.
· Mac.getInstance("HmacMD5"): Creates a MAC object.
· init(MD5key): Intializes the MAC object.
· update(plaintext) and .doFinal(): Calculates the MAC object with a plaintext string.

Mathematical Modeling:

Let S be the system that represents the Password Encryption system.

Initially,
S ={_}
Let,
S ={I, O, F}
Where,
I - Represents Input set
O - Represents Output set
F - Represents Function set

Input set - I:
I = {S}
Where,
• S - Represents the input password string.

Output set - O:
O = {Md}
Where,
• Md is message digest of S

Function Set - F:
F = {*F*1, *F*2, *F*3**}**
Where,
• *F*1 - Represents a function for generating encrypted password.
*F*1() -> {Md}
• *F*2 - Represents a function for generating simple password object.
*F*2(S) -> {Pu}
– Pu – Simple unsecured password object.
*F*3(S) -> {Ps}
– Ps - Secured password object.

Conclusion:

Thus, we have studies and implemented password encryptor.