**Experiment No. 5**

**Aim:** Demonstrate and test the integrity of the message using MD-5, SHA-1, For varying message size and analyze the performance of the two protocols. Use crypt APIs.

**Learning Objectives:** To be able to apply the knowledge of Hashing techniques.

**Related Theory:**

**Hash Functions:**

* Mathematical function that converts a numerical input value into another compressed numerical value.
* The input to the hash function is of arbitrary length but output is always of fixed length.
* Values returned by a hash function are called **message digest** or simply **hash values**.
* Hash used to detect changes to message

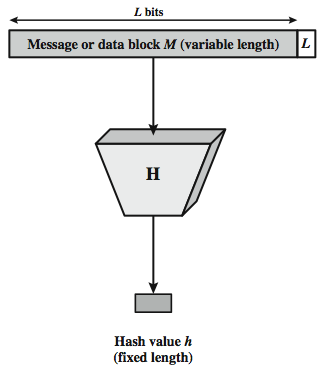


Fig. Cryptographic Hash Function

Above figure depicts the general operation of a cryptographic hash function. Typically, the input is padded out to an integer multiple of some fixed length (e.g., 1024 bits) and the padding includes the value of the length of the original message in bits. The length field is a security measure to increase the difficulty for an attacker to produce an alternative message with the same hash value.

**Applications of Hash Functions**:  
Password Storage

* + Hash functions provide protection to password storage.
  + Instead of storing password in clear, mostly all login processes store the hash values of passwords in the file.
  + The Password file consists of a table of pairs which are in the form (user id, h(P)).

Message Authentication:

* + Verifies the integrity of the message

Digital signatures

Intrusion detection and virus detection

* + keep & check hash of files on system

Pseudorandom function (PRF) or pseudorandom number generator (PRNG)

* + PRF is used for generating symmetric keys

**Popular Hash Functions:**Message Digest (MD)

* + The MD family comprises of hash functions MD2, MD4, **MD5** and MD6

Secure Hash Function (SHA)

* + Family of SHA comprise of four SHA algorithms; SHA-0, SHA-1, SHA-2 (SHA-224, SHA-256, SHA-384, and **SHA-512**) and SHA-3.

**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

-nb. 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**

In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512. Collectively,  these hash algorithms are known as SHA-2. These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1, hence analyses should be similar. A revised document was issued as FIP PUB 180-3 in 2008, which added a 224-bit version. SHA-2 is also specified in RFC 4634, which essentially duplicates the material in FIPS 180-3, but adds a C code implementation.

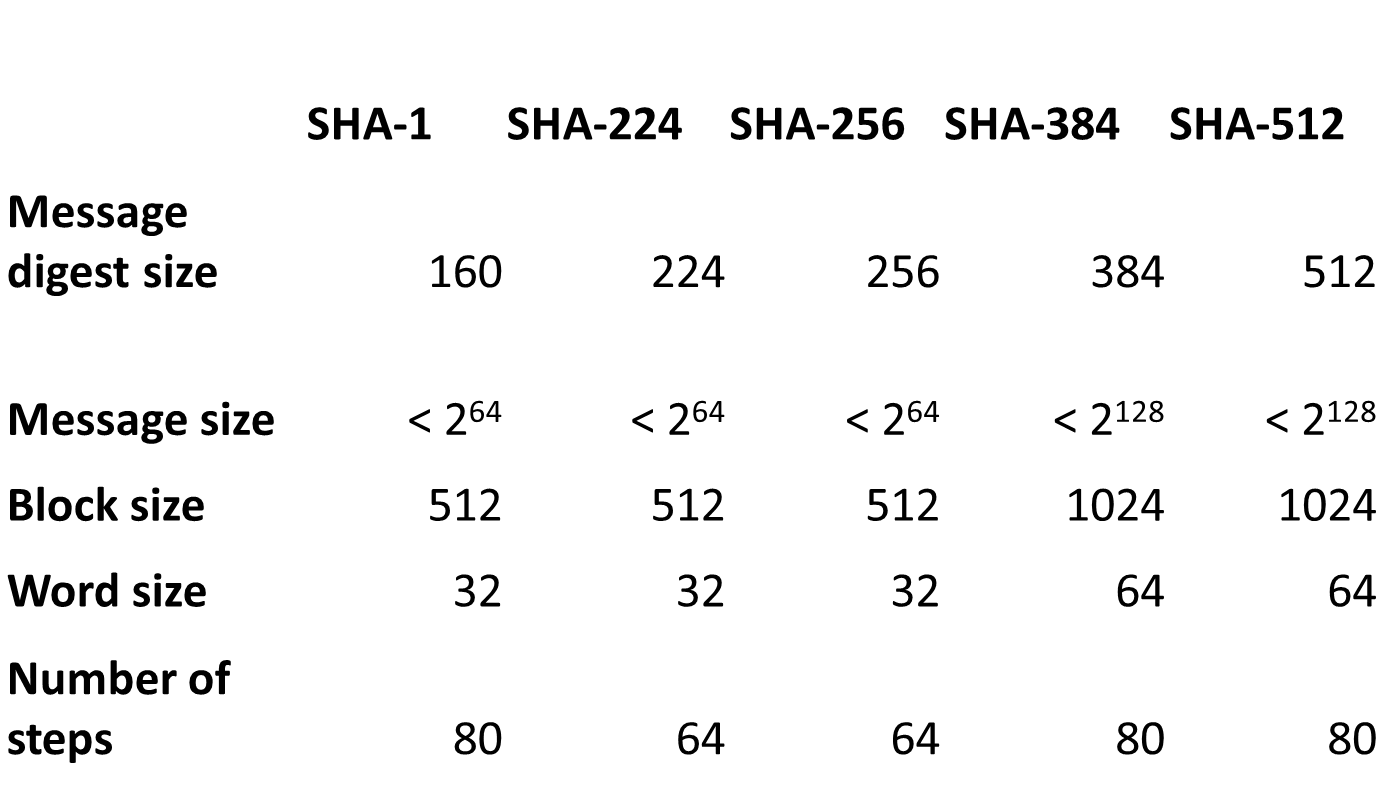
In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on the other SHA versions by 2010.

* 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 512:**

Step 1: Add Padding bits (Block size 1024 bits)

Step 2: Append 128 bits – Actual size of P.T.

Step 3: Initialize the buffers

             (8 buffers- a,b,c,d,e,f,g,h)

             buffer size is 64 bits

Step 4: Processing of each block of P.T. in 80 rounds

Step 5: Output stored in buffer is Hash code (128 bits)

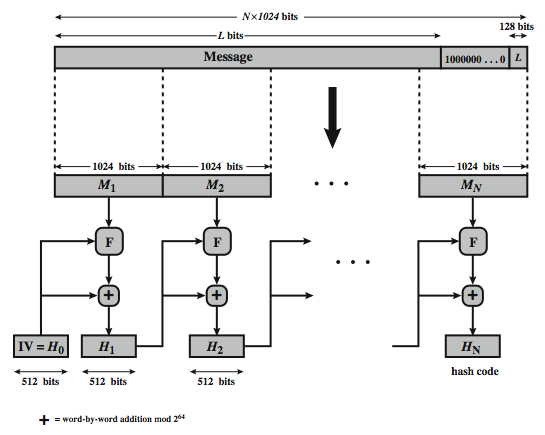


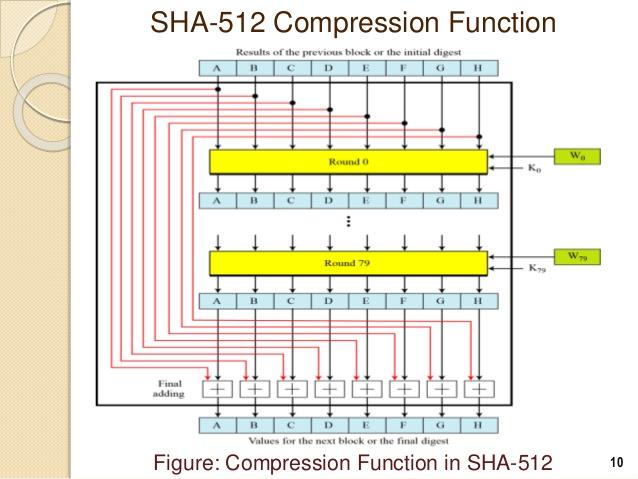
Fig. SHA-512 Overview

**SHA-512 Compression Function**

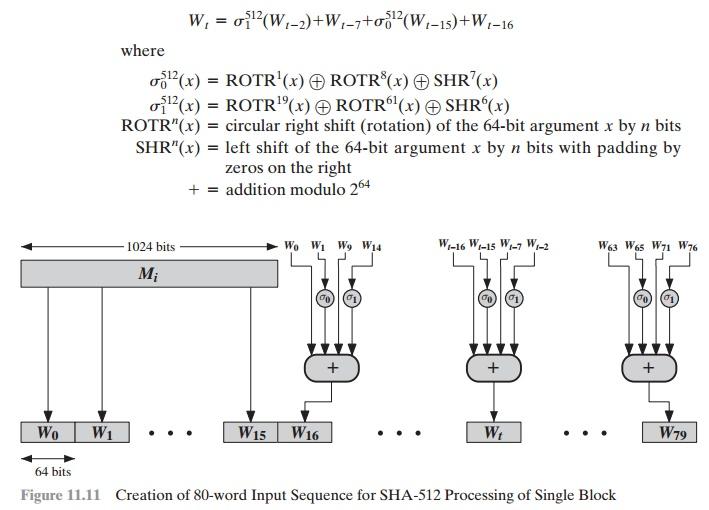
* heart of the algorithm
* processing message in 1024-bit blocks
* consists of 80 rounds

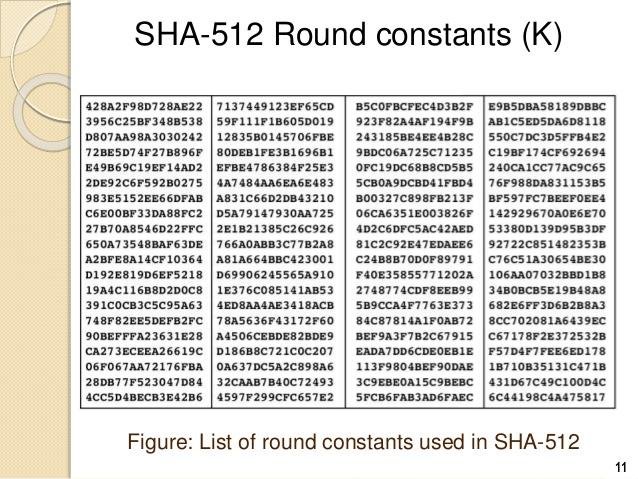
-updating a 512-bit buffer

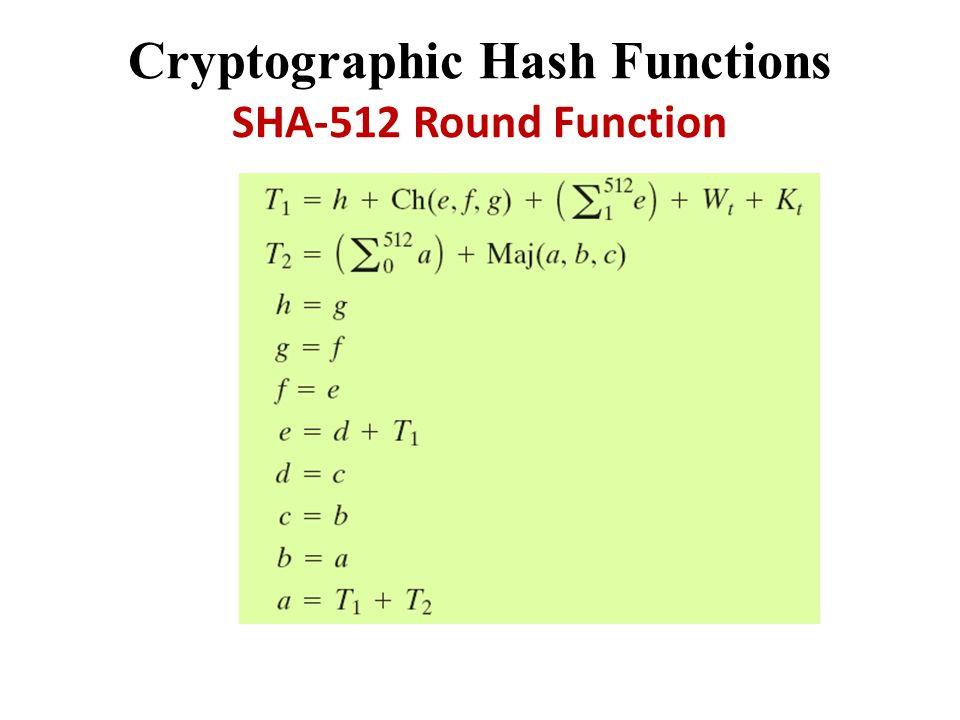
-using a 64-bit value Wt derived from the current message block and a round constant based on cube root of first 80 prime numbers

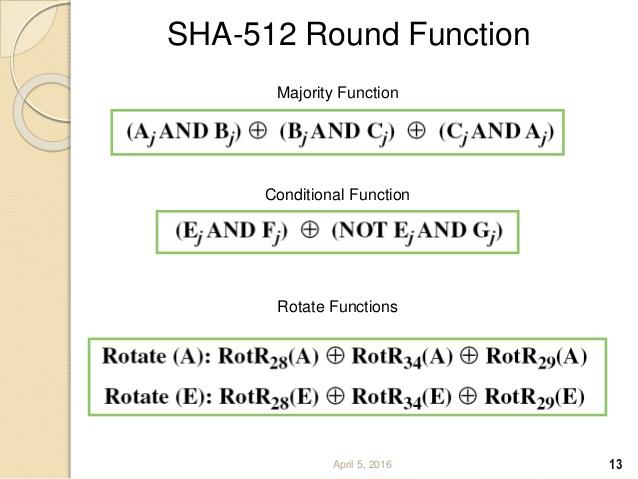


**SHA-512 Round Function:**

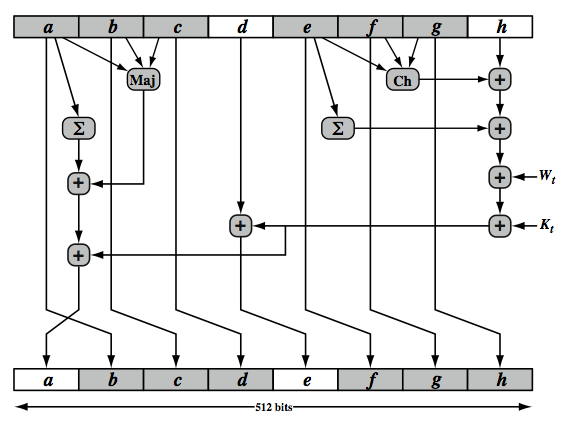
****







**SHA-512 Round Function:**



The structure of each of the 80 rounds is shown in Stallings Figure 11.10. Each 64-bit word is shuffled along one place, and in some cases manipulated using a series of simple logical functions (ANDs, NOTs, ORs, XORs, ROTates), in order to provide the avalanche & completeness properties of the hash function.

The elements are:

Ch(e,f,g) = (e AND f) XOR (NOT e AND g)

Maj(a,b,c) = (a AND b) XOR (a AND c) XOR (b AND c)

∑(a) = ROTR(a,28) XOR ROTR(a,34) XOR ROTR(a,39)

∑(e) = ROTR(e,14) XOR ROTR(e,18) XOR ROTR(e,41)

+ = addition modulo 2^64

Kt  = a 64-bit additive constant

Wt = a 64-bit word derived from the current 512-bit input block.  
 Six of the eight words of the output of the round function involve simply permutation (*b, c, d, f, g, h*) by means of rotation. This is indicated by shading in Figure 11.10. Only two of the output words (*a, e)* are generated by substitution. Word e is a function of input variables *d, e, f, g, h,* as well as the round word W t and the constant Kt. Word a is a function of all of the input variables, as well as the round word W t and the constant Kt.

**Implementation:**

import hashlib

input = "This is the message to be encoded”

md5 = hashlib.md5(input.encode())

sha1 = hashlib.sha1(input.encode())

print(f"Hash of the input string: {md5.hexdigest()}")

print(f"Hash of the input string: {sha1.hexdigest()}")

**Output:**

