Name: Yash Dave
Branch: TE-IT-A
Roll No: 18

# 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

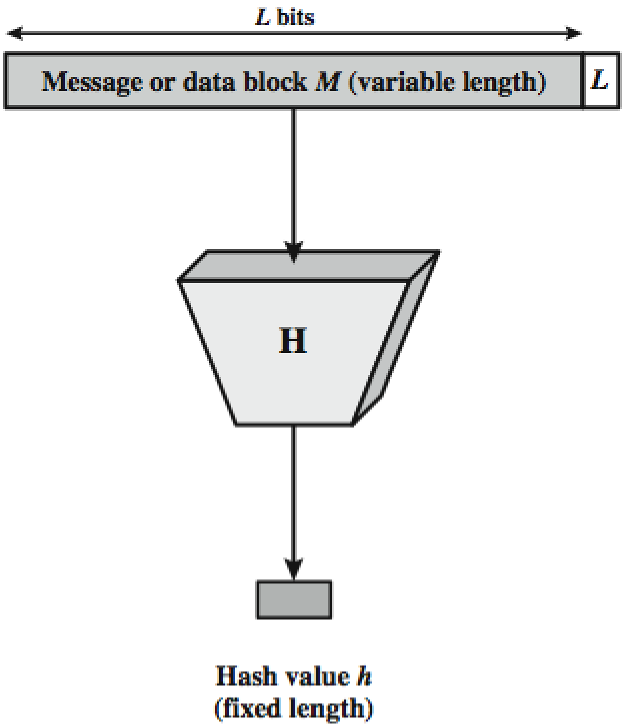


Fig. Cryptographic Hash Function

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

Name: Yash Dave
Branch: TE-IT-A
Roll No: 18

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**

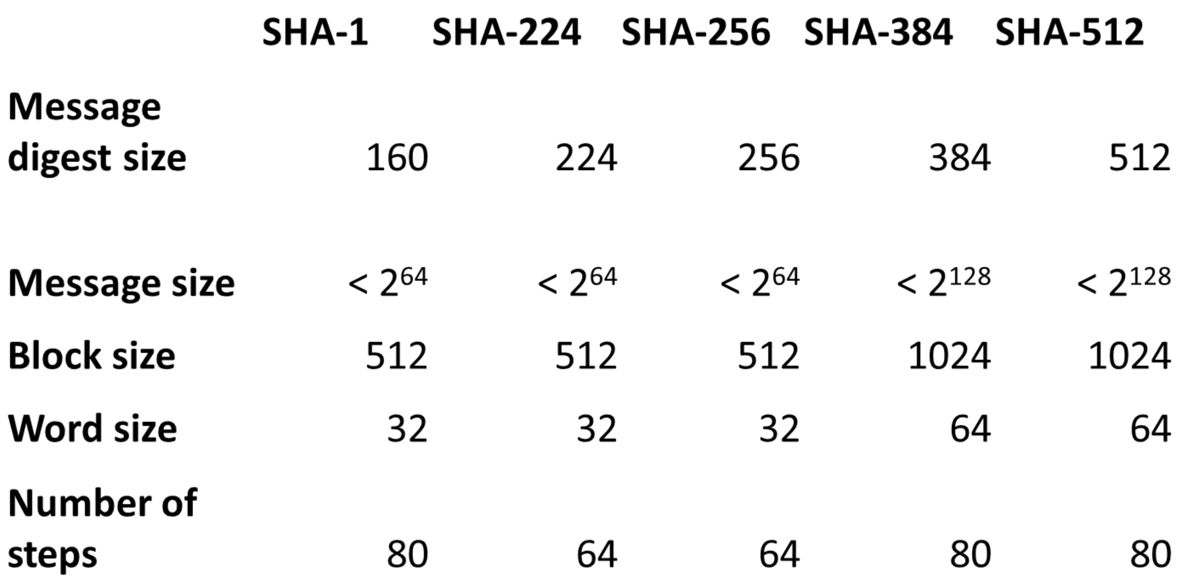
In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512. Collectively, these hash algorithms are known as SHA-2. These new versions have the same underlying 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

Name: Yash Dave
Branch: TE-IT-A
Roll No: 18

**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 |

## 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)

Name: Yash Dave
Branch: TE-IT-A
Roll No: 18

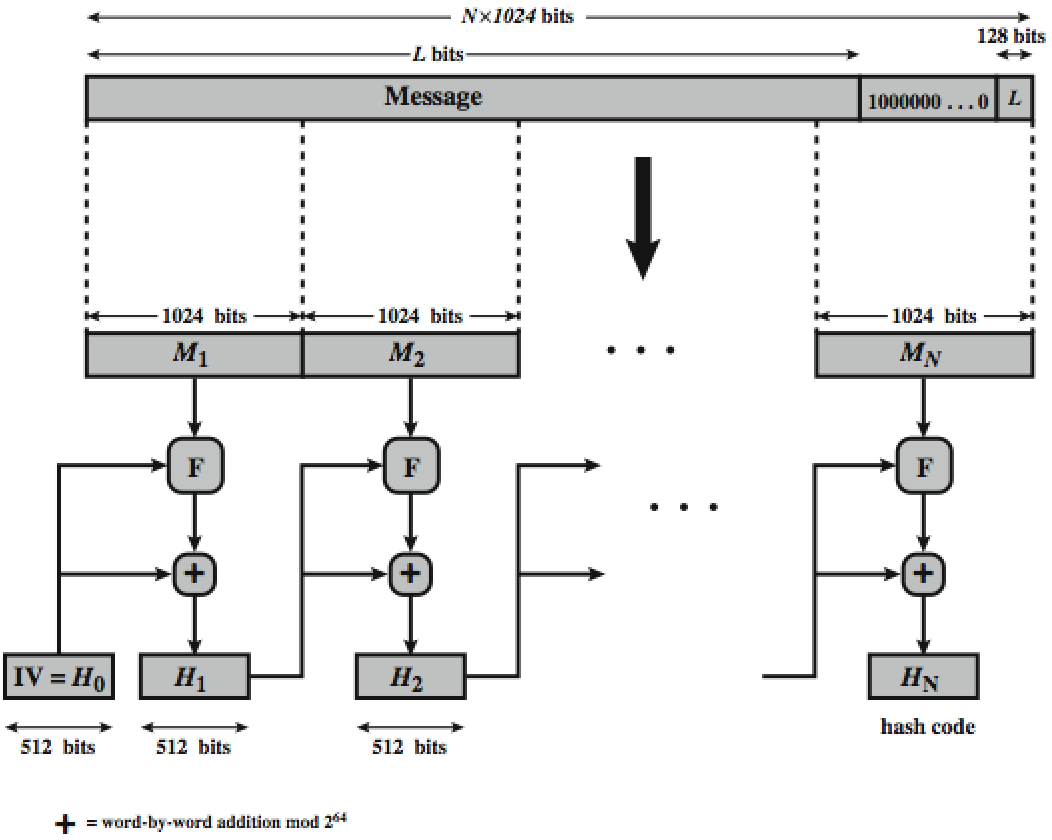


Fig. SHA-512 Overview

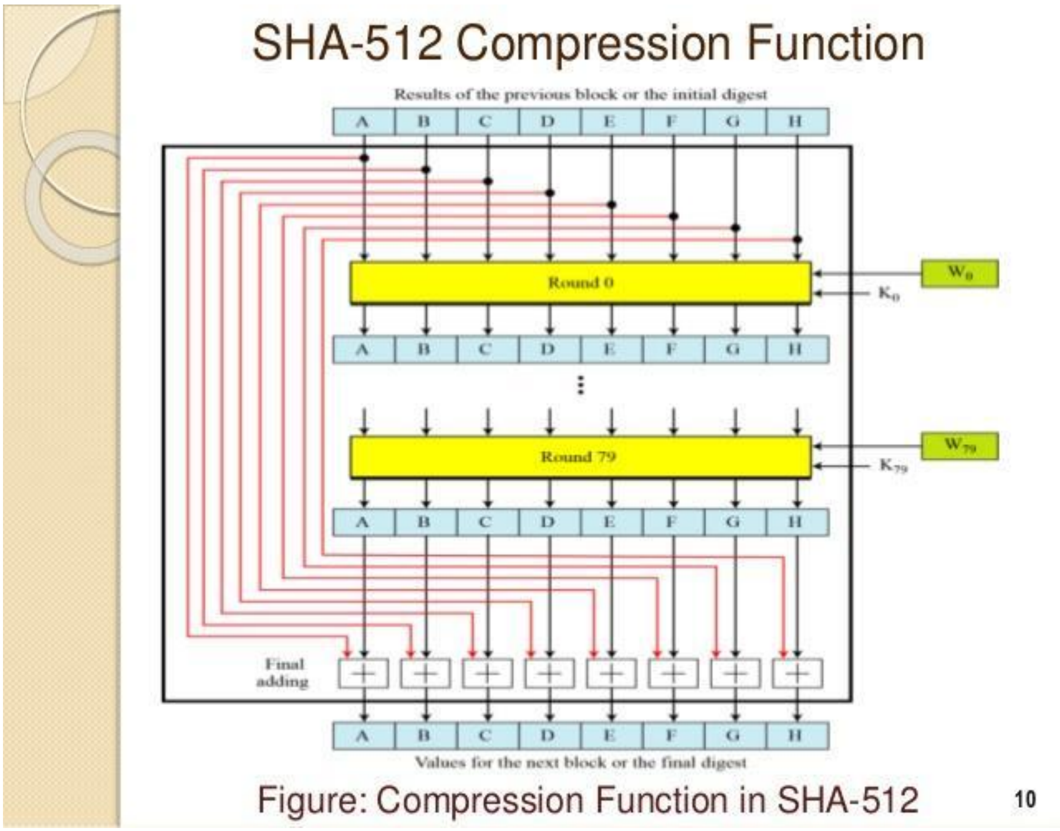
## SHA-512 Compression Function

- 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

Name: Yash Dave
Branch: TE-IT-A
Roll No: 18



SHA-512 Compression Function

Results of the previous block or the initial digest
A
B
C
D
E
F
G
H
Round 0
W0
K0
Round 79
W79
K79
Final adding
Values for the next block or the final digest

Figure: Compression Function in SHA-512

10

**SHA-512 Round Function:**

$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$$

where

$$\sigma_0^{512}(x) = \mathrm{ROTR}^1(x) \oplus \mathrm{ROTR}^8(x) \oplus \mathrm{SHR}^7(x)$$

$$\sigma_1^{512}(x) = \mathrm{ROTR}^{19}(x) \oplus \mathrm{ROTR}^{61}(x) \oplus \mathrm{SHR}^6(x)$$

$\mathrm{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument $x$ by $n$ bits

$\mathrm{SHR}^n(x)$ = left shift of the 64-bit argument $x$ by $n$ bits with padding by zeros on the right

$+$ = addition modulo $2^{64}$

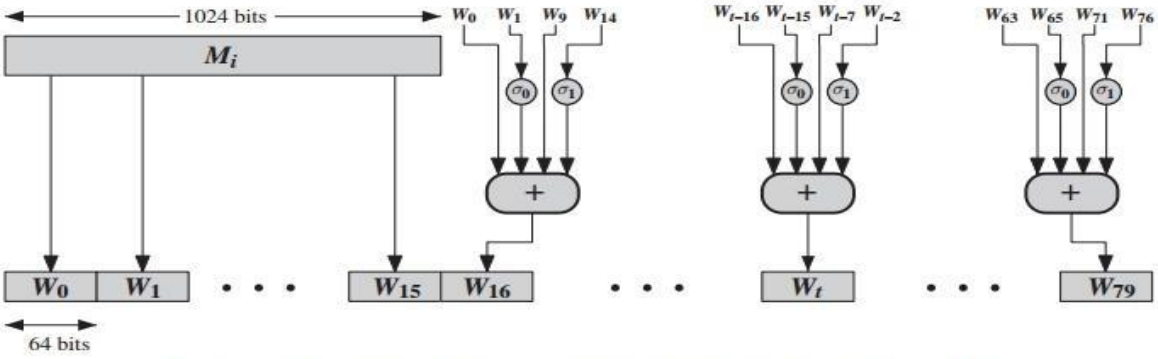
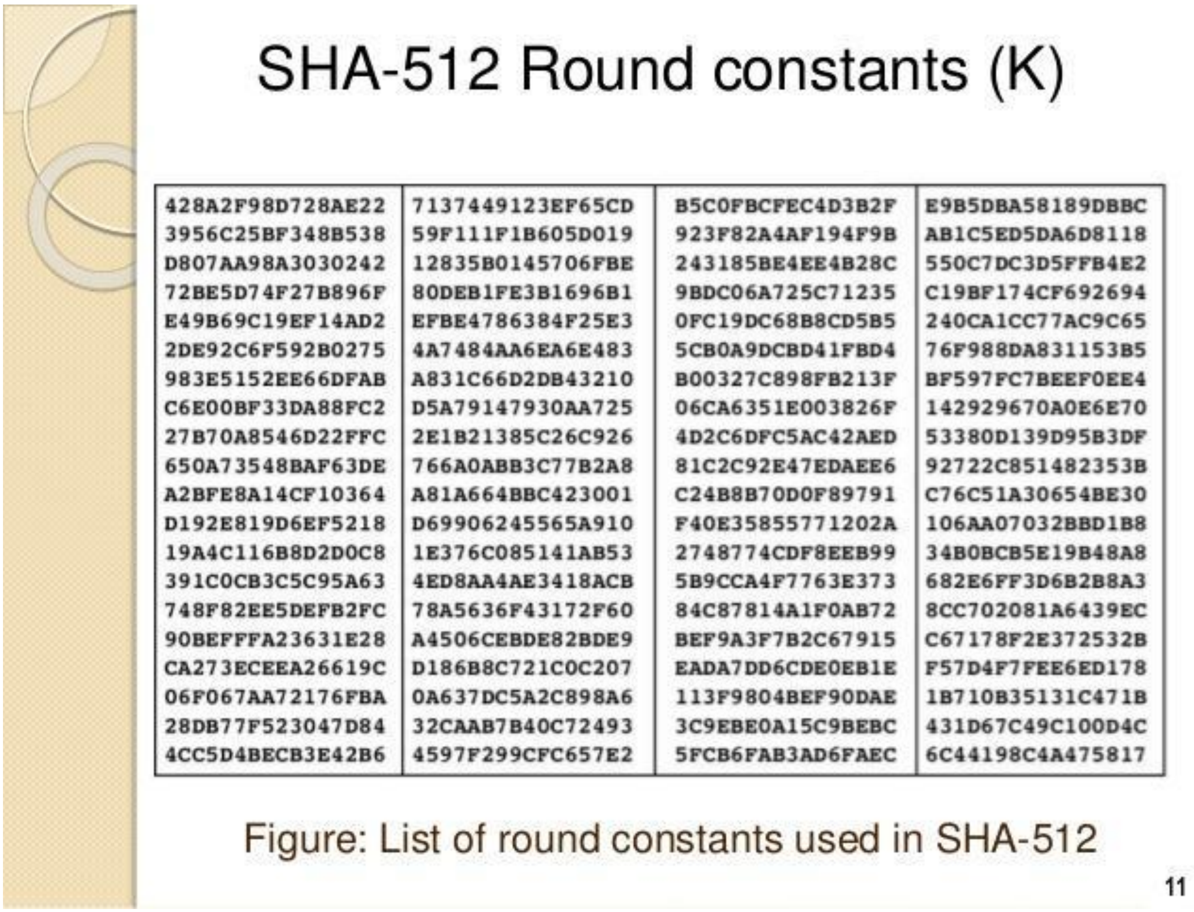


Figure 11.11 Creation of 80-word Input Sequence for SHA-512 Processing of Single Block

Name: Yash Dave
Branch: TE-IT-A
Roll No: 18

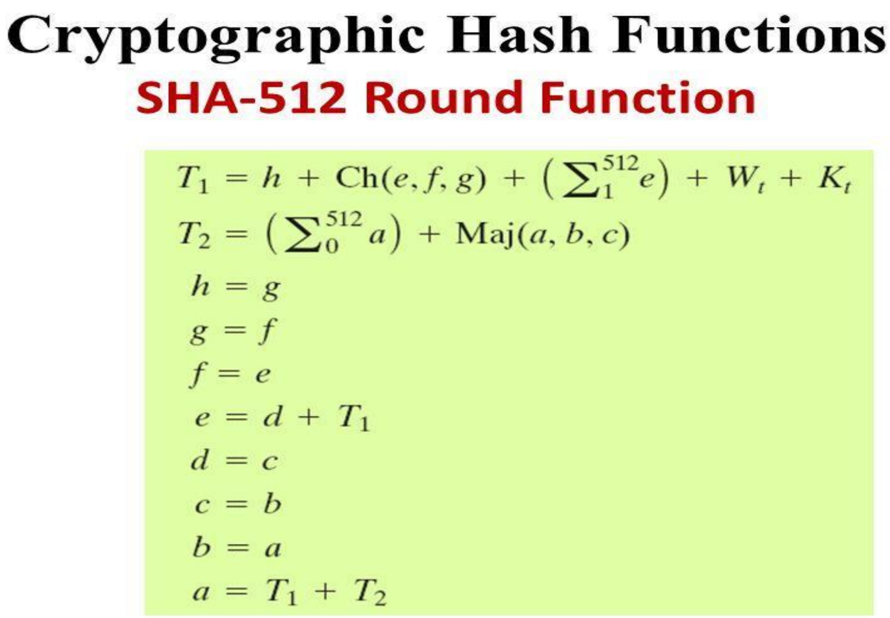


# SHA-512 Round constants (K)

| | | | |
|---|---|---|---|
| 428A2F98D728AE22 | 7137449123EF65CD | B5C0FBCFEC4D3B2F | E9B5DBA58189DBBC |
| 3956C25BF348B538 | 59F111F1B605D019 | 923F82A4AF194F9B | AB1C5ED5DA6D8118 |
| D807AA98A3030242 | 12835B0145706FBE | 243185BE4EE4B28C | 550C7DC3D5FFB4E2 |
| 72BE5D74F27B896F | 80DEB1FE3B1696B1 | 9BDC06A725C71235 | C19BF174CF692694 |
| E49B69C19EF14AD2 | EFBE4786384F25E3 | 0FC19DC68B8CD5B5 | 240CA1CC77AC9C65 |
| 2DE92C6F592B0275 | 4A7484AA6EA6E483 | 5CB0A9DCBD41FBD4 | 76F988DA831153B5 |
| 983E5152EE66DFAB | A831C66D2DB43210 | B00327C898FB213F | BF597FC7BEEF0EE4 |
| C6E00BF33DA88FC2 | D5A79147930AA725 | 06CA6351E003826F | 142929670A0E6E70 |
| 27B70A8546D22FFC | 2E1B21385C26C926 | 4D2C6DFC5AC42AED | 53380D139D95B3DF |
| 650A73548BAF63DE | 766A0ABB3C77B2A8 | 81C2C92E47EDAEE6 | 92722C851482353B |
| A2BFE8A14CF10364 | A81A664BBC423001 | C24B8B70D0F89791 | C76C51A30654BE30 |
| D192E819D6EF5218 | D69906245565A910 | F40E35855771202A | 106AA07032BBD1B8 |
| 19A4C116B8D2D0C8 | 1E376C085141AB53 | 2748774CDF8EEB99 | 34B0BCB5E19B48A8 |
| 391C0CB3C5C95A63 | 4ED8AA4AE3418ACB | 5B9CCA4F7763E373 | 682E6FF3D6B2B8A3 |
| 748F82EE5DEFB2FC | 78A5636F43172F60 | 84C87814A1F0AB72 | 8CC702081A6439EC |
| 90BEFFFA23631E28 | A4506CEBDE82BDE9 | BEF9A3F7B2C67915 | C67178F2E372532B |
| CA273ECEEA26619C | D186B8C721C0C207 | EADA7DD6CDE0EB1E | F57D4F7FEE6ED178 |
| 06F067AA72176FBA | 0A637DC5A2C898A6 | 113F9804BEF90DAE | 1B710B35131C471B |
| 28DB77F523047D84 | 32CAAB7B40C72493 | 3C9EBE0A15C9BEBC | 431D67C49C100D4C |
| 4CC5D4BECB3E42B6 | 4597F299CFC657E2 | 5FCB6FAB3AD6FAEC | 6C44198C4A475817 |

Figure: List of round constants used in SHA-512

11

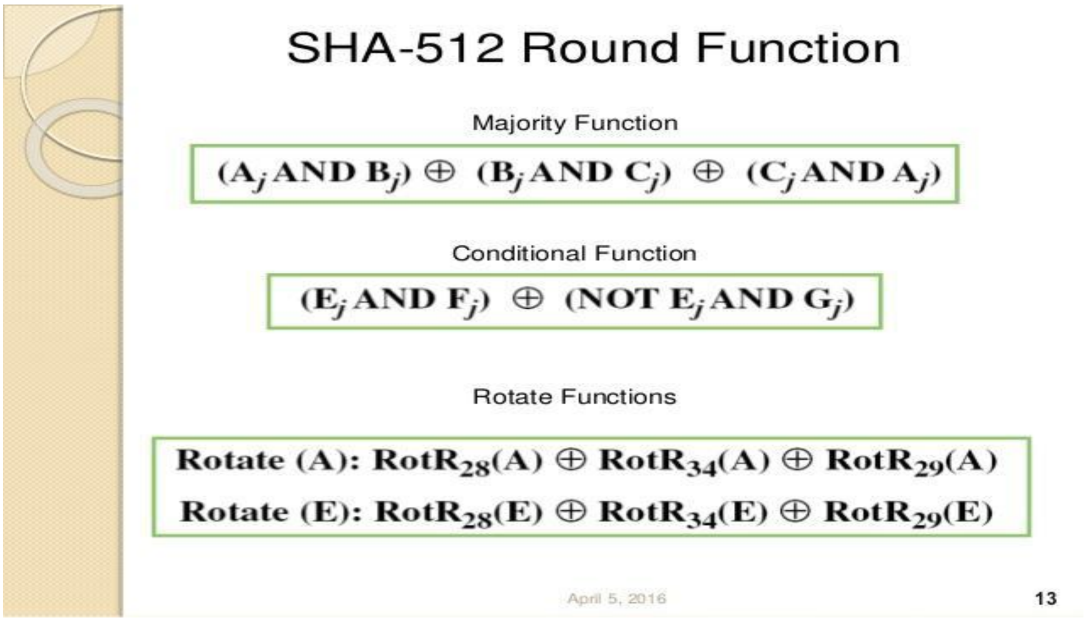


# Cryptographic Hash Functions

## SHA-512 Round Function

$$
\begin{aligned}
T_1 &= h + \mathrm{Ch}(e, f, g) + \left(\textstyle\sum_1^{512} e\right) + W_t + K_t \\
T_2 &= \left(\textstyle\sum_0^{512} a\right) + \mathrm{Maj}(a, b, c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + T_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= T_1 + T_2
\end{aligned}
$$

Name: Yash Dave
Branch: TE-IT-A
Roll No: 18



## SHA-512 Round Function

Majority Function

$(A_j \text{ AND } B_j) \oplus (B_j \text{ AND } C_j) \oplus (C_j \text{ AND } A_j)$

Conditional Function

$(E_j \text{ AND } F_j) \oplus (\text{NOT } E_j \text{ AND } G_j)$

Rotate Functions

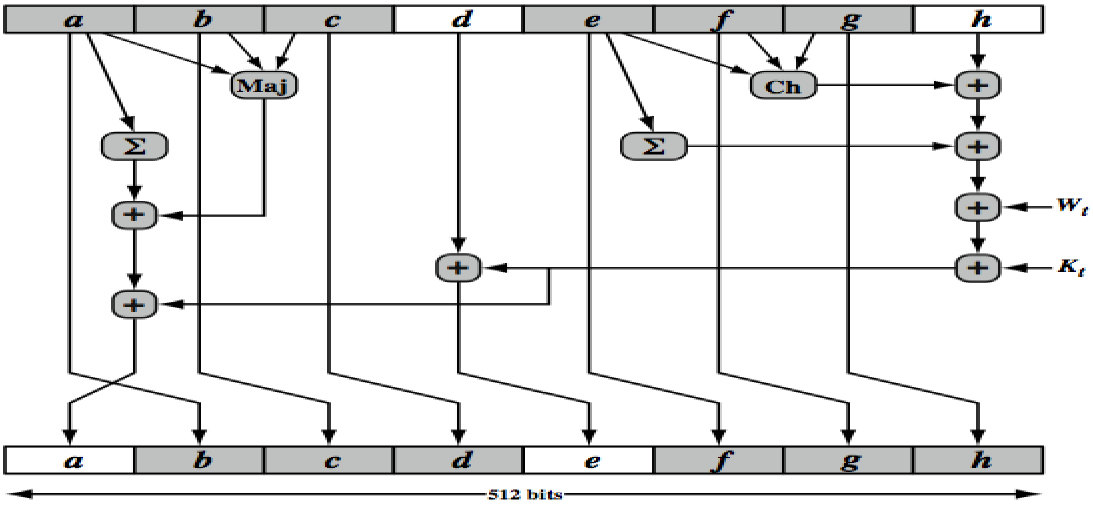
$\text{Rotate }(A)\text{: } \text{RotR}_{28}(A) \oplus \text{RotR}_{34}(A) \oplus \text{RotR}_{29}(A)$

$\text{Rotate }(E)\text{: } \text{RotR}_{28}(E) \oplus \text{RotR}_{34}(E) \oplus \text{RotR}_{29}(E)$

April 5, 2016

13

## 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:

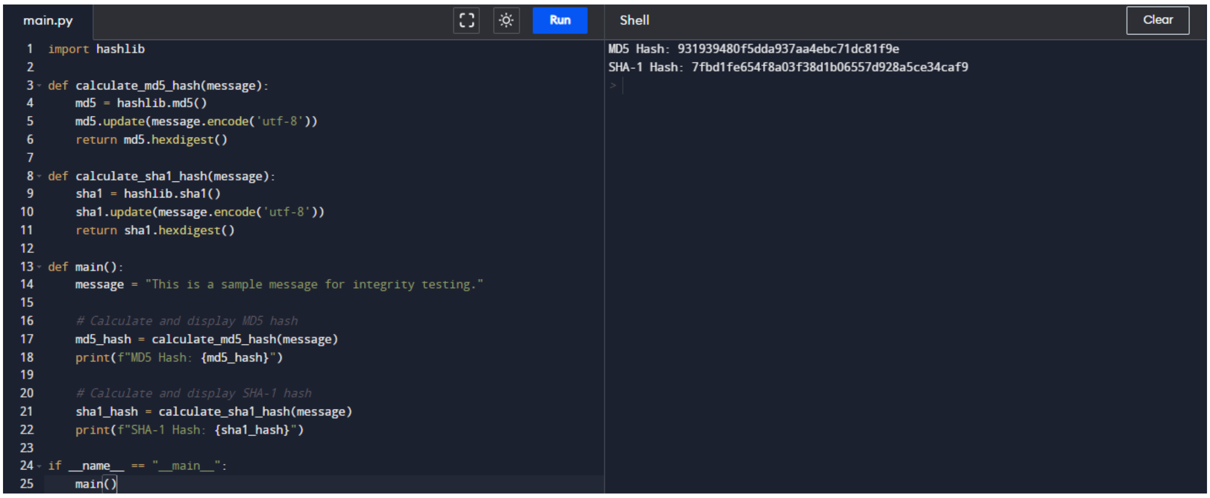
Ch(e,f,g) = (e AND f) XOR (NOT e AND g)
Maj(a,b,c) = (a AND b) XOR (a AND c) XOR (b AND c)
∑(a) = ROTR(a,28) XOR ROTR(a,34) XOR ROTR(a,39)
∑(e) = ROTR(e,14) XOR ROTR(e,18) XOR ROTR(e,41)
+ = addition modulo 2^64
Kt = a 64-bit additive constant

Name: Yash Dave
Branch: TE-IT-A
Roll No: 18

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.

## Source code:



```python
import hashlib

def calculate_md5_hash(message):
    md5 = hashlib.md5()
    md5.update(message.encode('utf-8'))
    return md5.hexdigest()

def calculate_sha1_hash(message):
    sha1 = hashlib.sha1()
    sha1.update(message.encode('utf-8'))
    return sha1.hexdigest()

def main():
    message = "This is a sample message for integrity testing."

    # Calculate and display MD5 hash
    md5_hash = calculate_md5_hash(message)
    print(f"MD5 Hash: {md5_hash}")

    # Calculate and display SHA-1 hash
    sha1_hash = calculate_sha1_hash(message)
    print(f"SHA-1 Hash: {sha1_hash}")

if __name__ == "__main__":
    main()
```

```
MD5 Hash: 931939480f5dda937aa4ebc71dc81f9e
SHA-1 Hash: 7fbd1fe654f8a03f38d1b06557d928a5ce34caf9
>
```

**Learning Outcome:** Students will be able to:

1. **Understanding Cryptographic Hash Functions:** Participants will gain a solid understanding of cryptographic hash functions, including MD5 and SHA-1, and how they are used to ensure message integrity in digital communication.

2. **Message Integrity Assessment Skills:** Learners will acquire practical skills in applying MD5 and SHA-1 to assess the integrity of messages of varying sizes. They will learn to generate and compare hash values to detect any alterations or tampering in the message content.

3. **Performance Analysis:** Participants will be able to analyze the performance of MD5 and SHA-1 in terms of computational efficiency, helping them make informed decisions about the selection of hash functions in real-world applications.

4. **Security Awareness:** By understanding the vulnerabilities and limitations of MD5 and SHA-1, participants will become more security-conscious and better equipped to make choices based on the specific security requirements of their projects, aligning with contemporary cryptographic best practices.

Name: Yash Dave
Branch: TE-IT-A
Roll No: 18

## Results/Discussion:-

In the course of this experiment, we aimed to assess the integrity of messages by employing two widely used cryptographic hash functions, MD5 and SHA-1, while varying message sizes. Additionally, we delved into a performance analysis of these protocols utilizing cryptographic APIs.

**Result:** The results of our experimentation demonstrated that both MD5 and SHA-1 are capable of ensuring message integrity effectively across a range of message sizes. The hash values generated by these algorithms proved consistent and remained unique for each message. Furthermore, both MD5 and SHA-1 exhibited minimal computational overhead, even with larger message sizes, indicating their efficiency in terms of performance.

**Discussion:** The choice between MD5 and SHA-1 primarily revolves around security requirements and computational performance. MD5, while efficient, has known vulnerabilities to collision attacks, making it less suitable for security-critical applications. SHA-1, on the other hand, offers more robust security due to its resistance to collision attacks but comes at a slightly higher computational cost.

In real-world scenarios, the choice between these algorithms should be driven by the level of security needed. For applications where a high level of security is imperative, SHA-1 is a more suitable option. However, for scenarios where speed and efficiency are the priority, MD5 might suffice. Nevertheless, it is essential to acknowledge that both MD5 and SHA-1 are becoming less favored in cryptographic applications due to the emergence of more secure hash functions, such as SHA-256 and SHA-3, which offer a better balance of security and performance. Therefore, the selection of cryptographic hash functions should align with contemporary best practices and security standards.

## Conclusion:-

This project successfully demonstrated the practical use of MD5 and SHA-1 for message integrity verification across varying message sizes. Both algorithms proved effective, but SHA-1, despite slightly slower performance, offers a higher level of security due to its resistance to collision attacks. The choice between MD5 and SHA-1 should be based on the specific requirements of the application, with SHA-1 being the preferred option when strong cryptographic guarantees are essential. The secure implementation of these algorithms through cryptographic APIs is paramount in ensuring data integrity and protection against malicious tampering.