K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: A3 Roll No.: 16010121045

Experiment No.: 9

Title: Cryptographic Hash Functions and Applications

Objective:
Our main objective is to understand the need, design and applications of collision resistant hash functions.

Expected Outcome of Experiment:

CO	Outcome
3	Comprehend cryptographic hash functions, Message Authentication Codes and Digital Signatures for Authentication

Books/ Journals/ Websites referred:

1. http://cse29-iiith.vlabs.ac.in/
2. https://en.wikipedia.org/wiki/SHA-1
3. https://en.wikipedia.org/wiki/HMAC

Abstract: - Some key terms used:

- Plaintext – Original message
- Encryption algorithm – Performs substitution and transformations on the plaintext.
- key – An input to the algorithm that makes it produce a different output depending on the key.
- Ciphertext – Unintelligible message created using the encryption algorithm.
- Decryption algorithm – Encryption algorithm applied in reverse and produces the plaintext using the key and ciphertext.
- Hash Function: A hash function is a mathematical function that converts a numeric al input into another compressed numerical value.
- SHA- Stands for Secure Hashing Algorithm. It is a one-way function which converts a data string into a numerical string output of fixed length.

Hash Function:

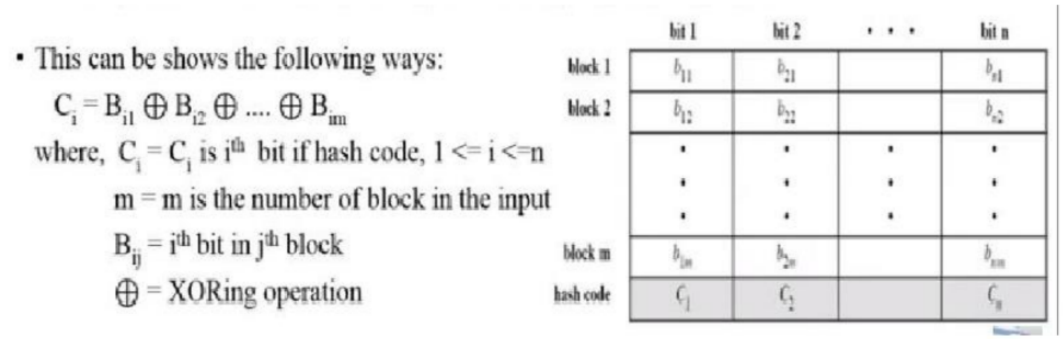
A hash function is a mathematical function that converts a numerical input value to another compressed numerical value.
The input to the hash function is of arbitrary length but output is always of fixed length.
Values returned by a hash function are called message digestors simply hash values.
Cryptographic Hash Function: Hash functions used for Security applications are known as Cryptographic Hash Function.

Properties:

- It is computationally infeasible to find either
  - A data object that maps to a pre-specified hash result (the one-way property)
  - Two data objects that map to the same hash result (the collision-free property)

Simple Hash Function: There are two simple hash function, all hash functions are operating using the same principle:

1. The message file is like a simple input it opens a sequence on n-bit blocks
2. When input is processed only one block at the given time in iterative fashion to generate an n-bit hash function.
3. The simple hash function is the bit-by-bit XORing done of every block.



Hash Algorithm:

- A hash algorithm is a one-way function that converts a data string into a numeric string output of fixed length. The output string is generally much smaller than the original data. Therefore, it is also called message digest or message compression algorithm.
- Hash algorithms are designed to be collision-resistant, meaning that there is a very low probability that the same string would be created for different data. Two of the most common hash algorithms are the MD5 (Message-Digest algorithm 5) and the SHA-1 (Secure Hash Algorithm). MD5 Message Digest checksums are commonly used to validate data integrity when digital files are transferred or stored.

Hash Function Family:

- MD (Message Digest) - Designed by Ron Rivest

Family: MD2, MD4, MD5

- SHA (Secure Hash Algorithm) - Designed by NIST

Family: SHA-O, SHA-1, and SHA-2: SHA-224, SHA-256, SHA-384, SHA-512
SHA 3: New standard in competition

- RIPEMD (Race Integrity Primitive Evaluation Message Digest) - Developed by Katholieke University Leuven Team

Family: RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320 SHA-1

- SHA1: Secure Hash Algorithm 1 - Designed by the United States National Security Agency
  - Produces hash value known as Message Digest
  - Works for any input message that is less than 264 bits

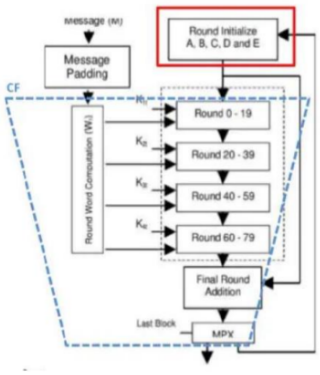
- Produces 160 bits length message digest
- Infeasible to retain the original message from the message digest
- Same message digest to be produced from both sender and receiver
- Purpose: Authentication, not Encryption
- Widely used in security applications and protocols, including TLS,
- SSL, PGP, SSH, IPsec and S/MIME

Steps:

1. Append Padding bits- padding bits are added to the original message to make the original message equal to a value divisible by 512
2. Append length. The rest two words are preserved for the original message length.
    a. These 512 bits input to the compression function
    b. The message divided into 16 words.
    c. Each word consists of 32 bits.
    d. 512/32 =16 words
3. Initialize the hash buffer- Initial values of H0 are predefined and stored in registers ABCDE

H	Hex
H0(A)	01234567
H0(B)	89ABCDEF
H0(C)	FEDCBA98
H0(D)	76543210
H0(E)	C3D2E1F0

These initial values are used in Round 0



4. SHA processing- Word assigning to rounds:
    a. SHA1 has 80 rounds defined.
    b. The Message Scheduler Algorithm schedules each word to rounds as:
        Wo → Round 0
        W → Round 1
        W 15 → Round 15
        W 16 → Round 16
        .
        .
        .
        .
        W79 → Round 79
    c. Each round = 20 iterations
    d. Total iterations = 80
    e. Word assigning to other rounds:
    f. For others (i.e., round 16- 79)
    g. W[t] = S'(W[t-16] XOR W[t-14] XOR W[t-8] XOR W[t-3])

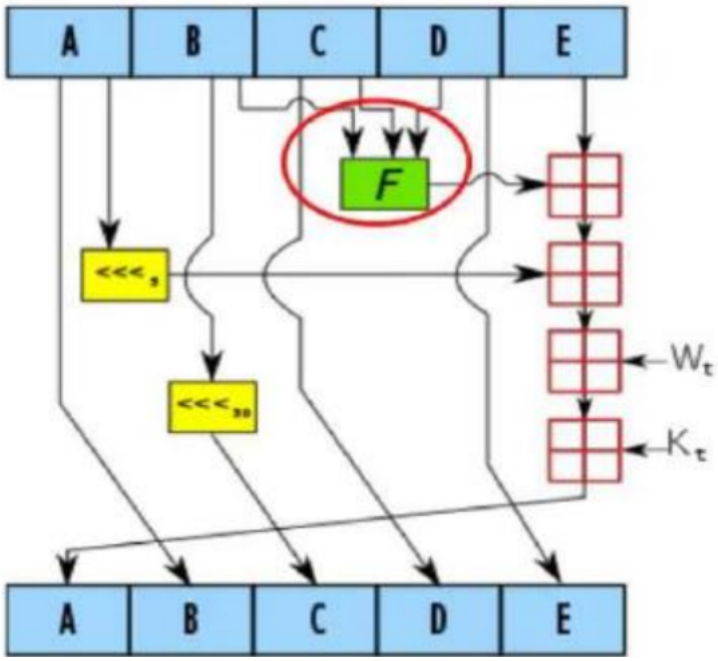
Division of stages: **Each round has 20 rounds**

Stages	Round
t=1	0 – 19
t=2	20 – 39
t=3	40 – 59
t=4	60 – 79

Stages	Round
t1	K1 = 0X5A827999
t2	K2 = 0X6ED9EBA1
t3	K3= 0X8F1BBCDC
t4	K4 = 0XCA62C1D6

Process in each round: Each round takes 3 inputs:

- 32-bit word form 512-bit block (i.e., Wt.)
- The values from register ABCDE
- Constant K_t



The F_t is different at each stage

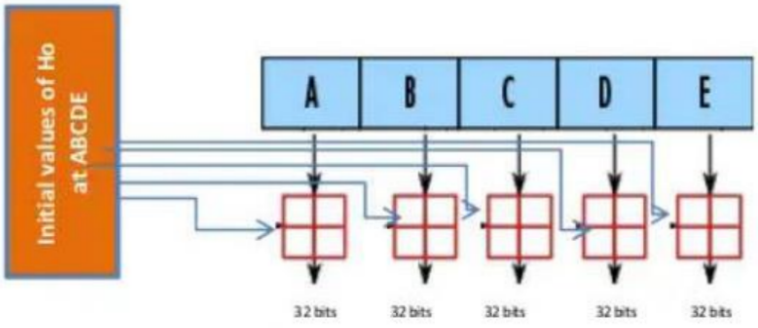
Stage	F_t
t_1	F_t (B, C, D) = (B AND C) OR ((NOT B) AND D)
t_2	F_t (B, C, D) = B XOR C XOR D
t_3	F_t (B, C, D) = (B AND C) OR (B AND D) OR (C AND D)
t_4	F_t (B, C, D) = B XOR C XOR D

At each round:

- Output of Ft and E are added
- Value in register A is 5 bit circular-left shifted
- This then added to previous sum
- Wt. is added
- Kt introduced
- B is circular left shifted by 30 bits
- New values for next round

5. The Output- After the Final round:

- The 160-bit output from the final round is modulo added to the initial predefined values of H0 at registers ABCDE
- Output obtained this is a 160-bit hash code



H(X)
160 bits

Hash based Message Authentication Code (HMAC)

- Hash-based message authentication code (HMAC) is a mechanism for calculating a message authentication code involving a hash function in combination with a secret key. This can be used to verify the integrity and authenticity of a message.
- HMACs are almost similar to digital signatures. They both enforce integrity and authenticity. They both use cryptography keys. And they both employ hash functions.
- The main difference is that digital signatures use asymmetric keys, while HMACs use symmetric keys (no public key).

HMAC Structure

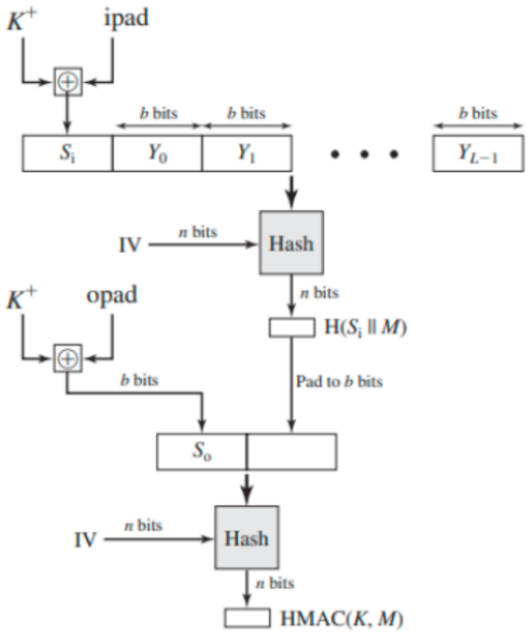


Figure 21.4 **HMAC Structure**

The figure illustrates the overall operation of HMAC.

Define the following terms:

- II = embedded hash function (e.g., SI IA)
- M = message input to HMAC (including the padding specified in the embedded hash function)
- Yi = ith block of M, 0<i <(L — 1)
- L = number of blocks in M
- b = number of bits in a block
- n = length of hash code produced by embedded hash function
- K = secret key; if key length is greater than b, the key is input to the hash function to produce an n-bit key; recommended length is > n
- K+ = K padded with zeros on the left so that the result is b bits in length
- ipad = 00110110 (36 in hexadecimal) repeated b/8 times
- opad = 01011100 (5C in hexadecimal) repeated b/8 times
- Then HMAC can be expressed as follows:
  HMAC (K, M) = H [(K+ ⊕ opad) = H (K+ ⊕ipad) || M]]

In words,

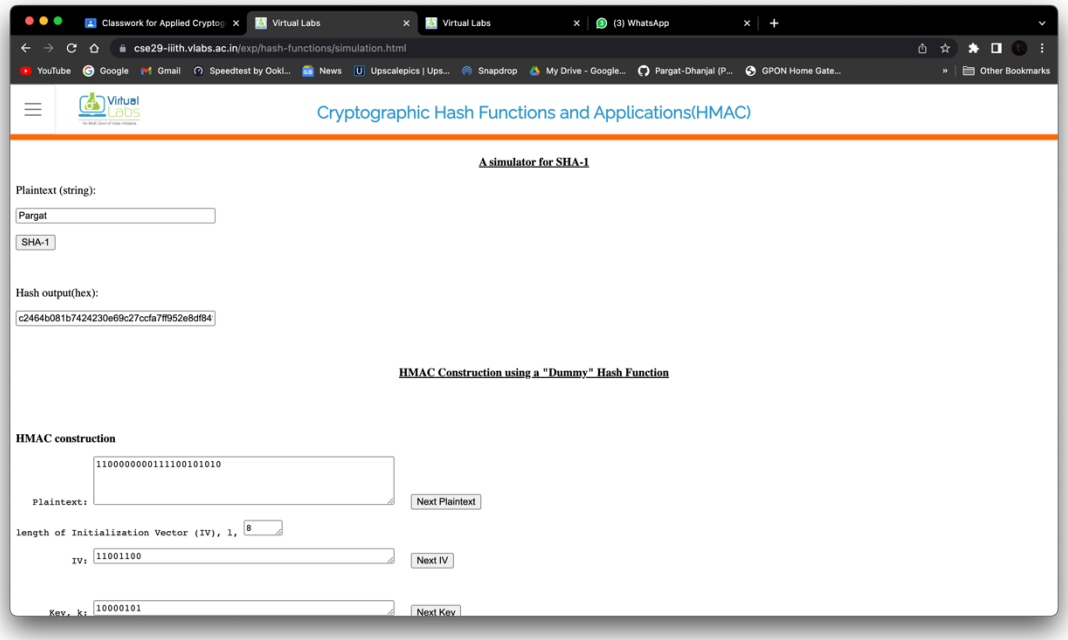
1. Append zeros to the left end of K to create a b-bit string K+ (e.g., if K is of length 160 bits and b = 512, then K will be appended with 44 zero bytes Ox00).
2. XOR (bitwise exclusive-OR) K+ with ipad to produce the b-bit block Si.
3. Append M to Si.
4. Apply H to the stream generated in step 3.
5. XOR K+ with opad to produce the b-bit block S0.
6. Append the hash result from step 4 to S0.
7. Apply H to the stream generated in step 6 and output the result.

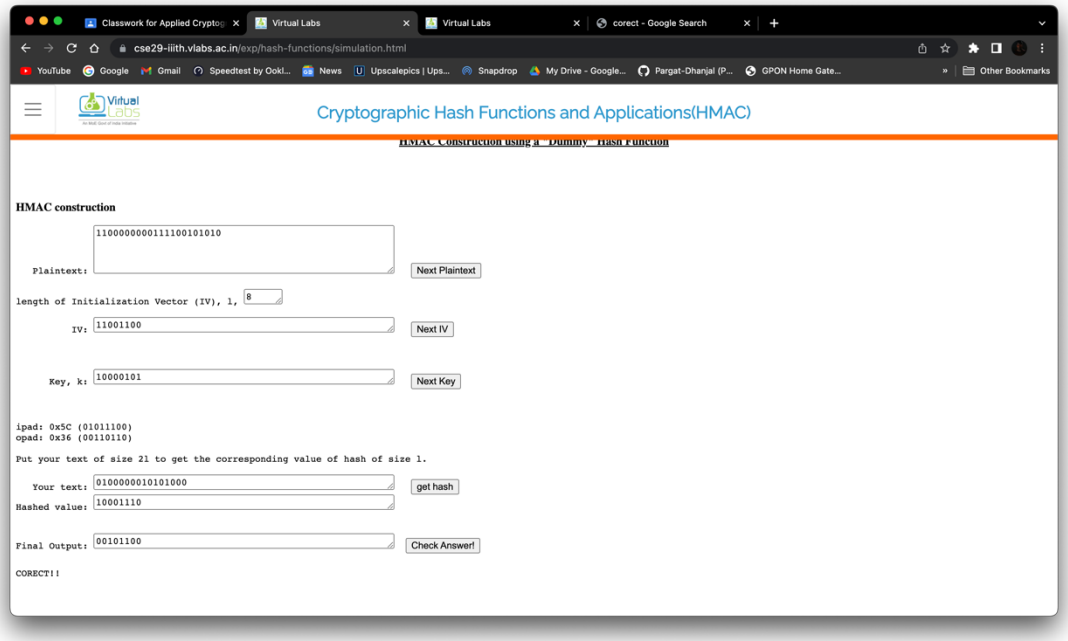
Procedure:

1. Familiarize yourself with the working of SHA-1. Though we would be using a dummy hash in the sequel for simplicity, in general, you could be using SHA-1 instead
2. Select a plaintext for which the HMAC tag is to be computed (by clicking on Next Plain text Button)
3. For simplicity fix l=8 which is default, but it should be l < (length of plaintext)/4.]
4. Select an Initialization Vector, IV of length l.by clicking on "Next IV" button)
5. Use the ipad and opad as described in theory part to compute the ciphertext with the help of the hash function provided to you.
6. Divide generated plaintext 'm' into say 'k' chunks of 8 bits and kth chunk will have bits less than 8, to make it 8-bits by padding zeros at end
7. Compute z0="IV|| (k XOR ipad)" manually where || implies concatenation and enter z0 in "Your text" field to get z1
8. Compute z1="z0||m1" manually where || implies concatenation and enter z1 in "Your text" field to get z2
9. Repeat above step and finally compute z(k+1)="zk||L" where L=|m|, make L 8-bits by padding zeros to left of it
10. Compute p="IV|| (k XOR opad)" manually where || implies concatenation and enter p in "Your text" field to get q
11. Compute r="q||z (k+1)" manually where || implies concatenation and enter 'r' in "Your text" field to get final HMAC tag 't'

12. Notice that z0, z1, z2, ...zk, z (k+1), p, r is all of size '2l' (=16 in our case as l=8).
13. Write the final cipher text 't' in 'Final Output' field and check your answer

Virtual labs assignment screenshots:





Conclusion: Successfully conducted the given Hash based experiment.