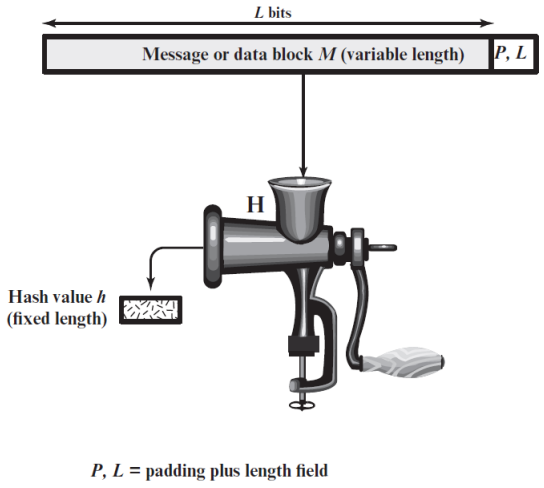
UNIT 5

Cryptographic Hash Functions:

Applications of Cryptographic Hash Functions, Two Simple Hash Functions, Message Authentication Requirements, Message Authentication Functions, MACs based on Hash functions: HMAC

Cryptographic Hash Functions:

- A **hash function** H accepts a variable-length block of data *M* as input and produces a fixed-size hash value *h* = H(*M*).
- A "good" hash function has the property that the results of applying the function to a large set of inputs will produce outputs that are evenly distributed and apparently random. In general terms, the principal object of a hash function is data integrity. A change to any bit or bits in *M* results, with high
- probability, in a change to the hash value.
- The kind of hash function needed for security applications is referred to as a **cryptographic hash function**.
- A cryptographic hash function is an algorithm for which it is computationally infeasible (because no attack is significantly more efficient than brute force) to find either
    - a data object that maps to a pre-specified hash result (the one-way property) or
    - two data objects that map to the same hash result (the collision-free property).
- Because of these characteristics, hash functions are often used to determine whether or not data has changed.



Cryptographic Hash function h=H(M)

The above figure depicts the general operation of a cryptographic hash function.

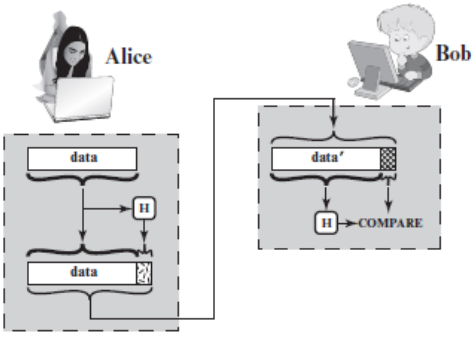
- Typically, the input is padded out to an integer multiple of some fixed length (e.g., 1024 bits), and the padding includes the value of the length of the original message in bits.
- The length field is a security measure to increase the difficulty for an attacker to produce an alternative message with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

Applications of Cryptographic Hash Functions:

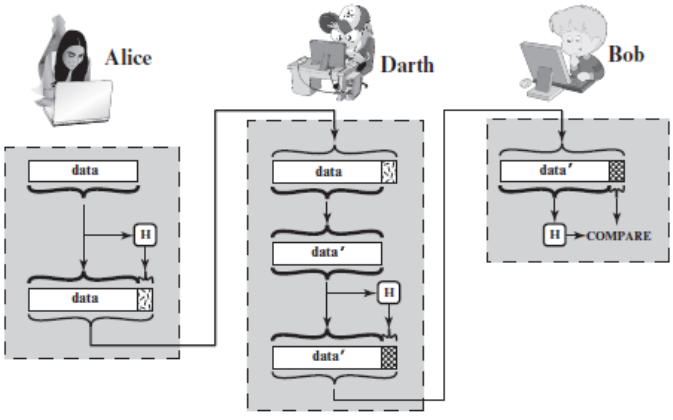
The most versatile cryptographic algorithm is the cryptographic hash function. It is used in a wide variety of security applications and Internet protocols. The following are various applications where it is employed.

Message Authentication:

- Message authentication is a mechanism or service used to verify the integrity of a message.
- Message authentication assures that data received are exactly as sent (i.e., there is no modification, insertion, deletion, or replay).
- When a hash function is used to provide message authentication, the hash function value is often referred to as a **message digest**.
- The essence of the use of a hash function for message integrity is as follows.
  - The sender computes a hash value as a function of the bits in the message and transmits both the hash value and the message.
  - The receiver performs the same hash calculation on the message bits and compares this value with the incoming hash value.
  - If there is a mismatch, the receiver knows that the message (or possibly the hash value) has been altered (Figure a).
  - The hash value must be transmitted in a secure fashion. That is, the hash value must be protected so that if an adversary alters or replaces the message, it is not feasible for adversary to also alter the hash value to fool the receiver. This type of attack is shown in Figure b.



(a) Use of hash function to check data integrity



(b) Man-in-the-middle attack

Attach against Hash function

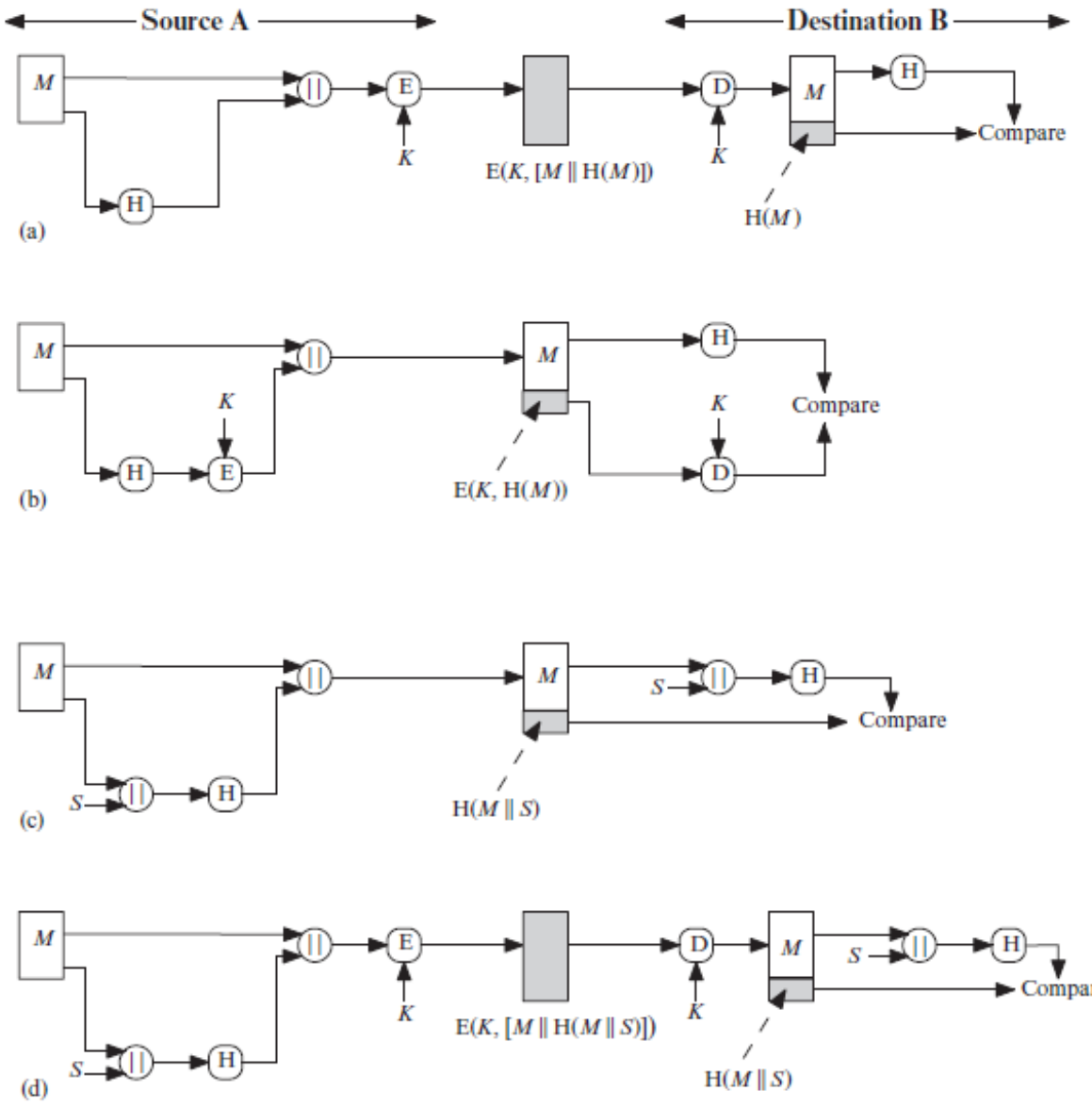
The following are a variety of ways in which a hash code can be used to provide message authentication.

a. The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.

b. Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.

c. It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value *S*. A computes the hash value over the concatenation of *M* and *S* and appends the resulting hash value to *M*. Because B possesses *S*, it can re-compute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

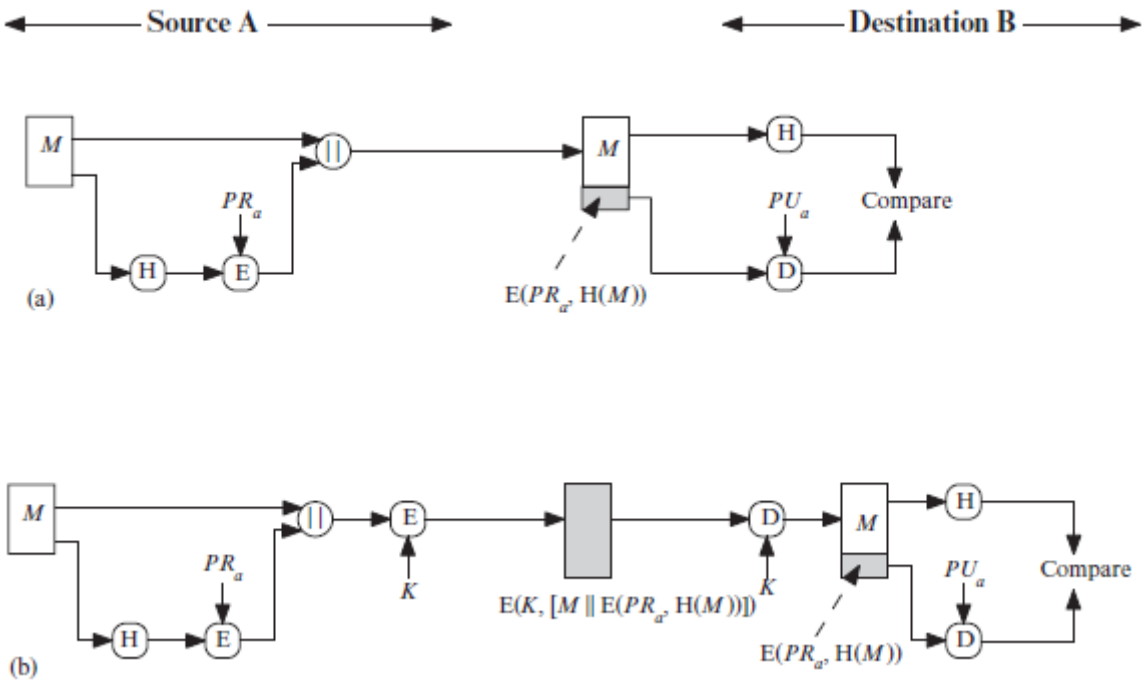
d. Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code.



- More commonly, message authentication is achieved using a **message authentication code (MAC)**, also known as a **keyed hash function**.
- Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties.
- A MAC function takes as input a secret key and a data block and produces a hash value, referred to as the MAC, which is associated with the protected message.
- If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the associated MAC value.
- An attacker who alters the message will be unable to alter the associated MAC value without knowledge of the secret key.

Digital Signatures:

- Another important application, which is similar to the message authentication application, is the **digital signature**.
- The operation of the digital signature is similar to that of the MAC.
- In the case of the digital signature, the hash value of a message is encrypted with a user's private key.
- Anyone who knows the user's public key can verify the integrity of the message that is associated with the digital signature.
- In this case, an attacker who wishes to alter the message would need to know the user's private key.
- Following figures illustrates, in a simplified fashion, how a hash code is used to provide a digital signature.
  a. The hash code is encrypted, using public-key encryption with the sender's private key. As with Figure b, this provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.
  b. If confidentiality as well as a digital signature is desired, then the message plus the private-key-encrypted hash code can be encrypted using a symmetric secret key. This is a common technique.



Other Applications:

- Hash functions are commonly used to create a **one-way password file**.
- Hash functions can be used for **intrusion detection** and **virus detection**.
- A cryptographic hash function can be used to construct a **pseudorandom function (PRF)** or a **pseudorandom number generator (PRNG)**.

Two-Simple Hash Functions:

- To get the understanding of security considerations involved in cryptographic hash functions, we present two simple, insecure hash functions in this section.
- All hash functions operate using the following general principles.
  - The input (message, file, etc.) is viewed as a sequence of *n* -bit blocks.
  - The input is processed one block at a time in an iterative fashion to produce an *n*-bit hash function.
- One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as:

$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

where

$C_i = i^{th}$ bit of the hash code, 1 … i … n

m = number of n-bit blocks in the input

$b_{ij} = i^{th}$ bit in j^{th} block

$\oplus$ = XOR operation

- This operation produces a simple parity bit for each bit position and is known as a longitudinal redundancy check.
- It is reasonably effective for random data as a data integrity check. Each n-bit hash value is equally likely.
- Thus, the probability that a data error will result in an unchanged hash value is 2-n.
- With more predictably formatted data, the function is less effective.
- For example, in most normal text files, the high-order bit of each octet is always zero.
- So if a 128-bit hash value is used, instead of an effectiveness of 2-128, the hash function on this type of data has an effectiveness of 2-112.
- A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows.
    1. Initially set the n-bit hash value to zero.
    2. Process each successive n-bit block of data as follows:
        a. Rotate the current hash value to the left by one bit.
        b. XOR the block into the hash value.
- This has the effect of "randomizing" the input more completely and overcoming any regularities that appear in the input.
- Although the second procedure provides a good measure of data integrity, it is virtually useless for data security when an encrypted hash code is used with a plaintext message.
- Although a simple XOR or rotated XOR (RXOR) is insufficient if only the hash code is encrypted, you may still feel that such a simple function could be useful when the message together with the hash code is encrypted

Message Authentication Requirements:

In the context of communications across a network, the following attacks can beidentified.

1. **Disclosure:** Release of message contents to any person or process not possessing
2. the appropriate cryptographic key.
3. **Traffic analysis:** Discovery of the pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined. In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined.
4. **Masquerade:** Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an opponent that are purported to come from an authorized entity. Also included are fraudulent acknowledgments of message receipt or nonreceipt by someone other than the message recipient.
5. **Content modification:** Changes to the contents of a message, including insertion, deletion, transposition, and modification.
6. **Sequence modification:** Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.

7. **Timing modification:** Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual message (e.g., datagram) could be delayed or replayed.
8. **Source repudiation:** Denial of transmission of message by source.
9. **Destination repudiation:** Denial of receipt of message by destination.

- Measures to deal with the first two attacks are in the realm of message confidentiality and are dealt with in Encryption techniques.
- Measures to deal with items (3) through (6) in the foregoing list are generally regarded as message authentication.
- Mechanisms for dealing specifically with item (7) come under the heading of digital signatures.
- Generally, a digital signature technique will also counter some or all of the attacks listed under items (3) through (6). Dealing with item (8) may require a combination of the use of digital signatures and a protocol designed to counter this attack.
- In summary, message authentication is a procedure to verify that received messages come from the alleged source and have not been altered.
- Message authentication may also verify sequencing and timeliness.
- A digital signature is an authentication technique that also includes measures to counter repudiation by the source.

Message Authentication Functions:

- Any message authentication or digital signature mechanism has two levels of functionality.
- At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message.
- This lower-level function is then used as a primitive in a higher-level authentication protocol that enables
- a receiver to verify the authenticity of a message.
- We are concerned with the types of functions that may be used to produce an authenticator. These may be grouped into three classes.
    - **Hash function:** A function that maps a message of any length into a fixed-length hash value, which serves as the authenticator
    - **Message encryption:** The ciphertext of the entire message serves as its authenticator
    - **Message authentication code (MAC):** A function of the message and a secret key that produces a fixed-length value that serves as the authenticator

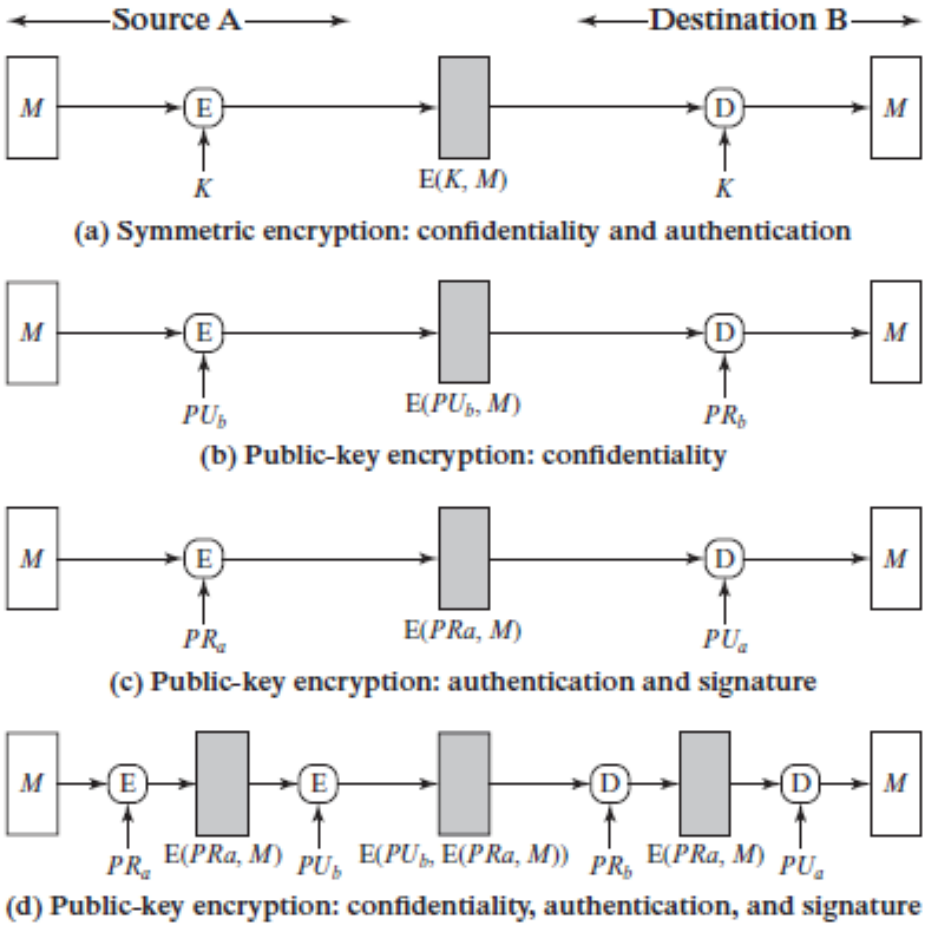
Message Encryption:

Message encryption by itself can provide a measure of authentication. The analysis differs for symmetric and public-key encryption schemes.

Symmetric Encryption:

Consider the straightforward use of symmetric encryption (Figure a).

- A message *M* transmitted from source A to destination B is encrypted using a secret key *K* shared by A and B. If no other party knows the key, then confidentiality is provided: No other party can recover the plaintext of the message.



Public-Key Encryption:

- The straightforward use of public-key encryption (Figure b) provides confidentiality but not authentication.
- The source (A) uses the public key PU_b of the destination (B) to encrypt M. Because only B has the corresponding private key PR_b, only B can decrypt the message. This scheme provides no authentication, because any opponent could also use B's public key to encrypt a message and claim to be A.
- To provide authentication, A uses its private key to encrypt the message, and B uses A's public key to decrypt (Figure c). This provides authentication using the same type of reasoning as in the symmetric encryption case: The message must have come from A because A is the only party that possesses PR_a and therefore the only party with the information necessary to construct ciphertext that can be decrypted with PU_a.
- There must be some internal structure to the plaintext so that the receiver can distinguish between
- well-formed plaintext and random bits.
- Assuming there is such structure, then the scheme of Figure c does provide authentication. It also provides what is known as digital signature.
- Only A could have constructed the ciphertext because only A possesses PR_a. Not even B, the recipient, could have constructed the ciphertext. Therefore, if B is in possession of the ciphertext, B has the means to prove that the message must have come from A.
- In effect, A has "signed" the message by using its private key to encrypt.
- Note that this scheme does not provide confidentiality. Anyone in possession of A's public key can decrypt the ciphertext.

- To provide both confidentiality and authentication, A can encrypt *M* first using its private key, which provides the digital signature, and then using B's public key, which provides confidentiality (Figure d).
- The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.

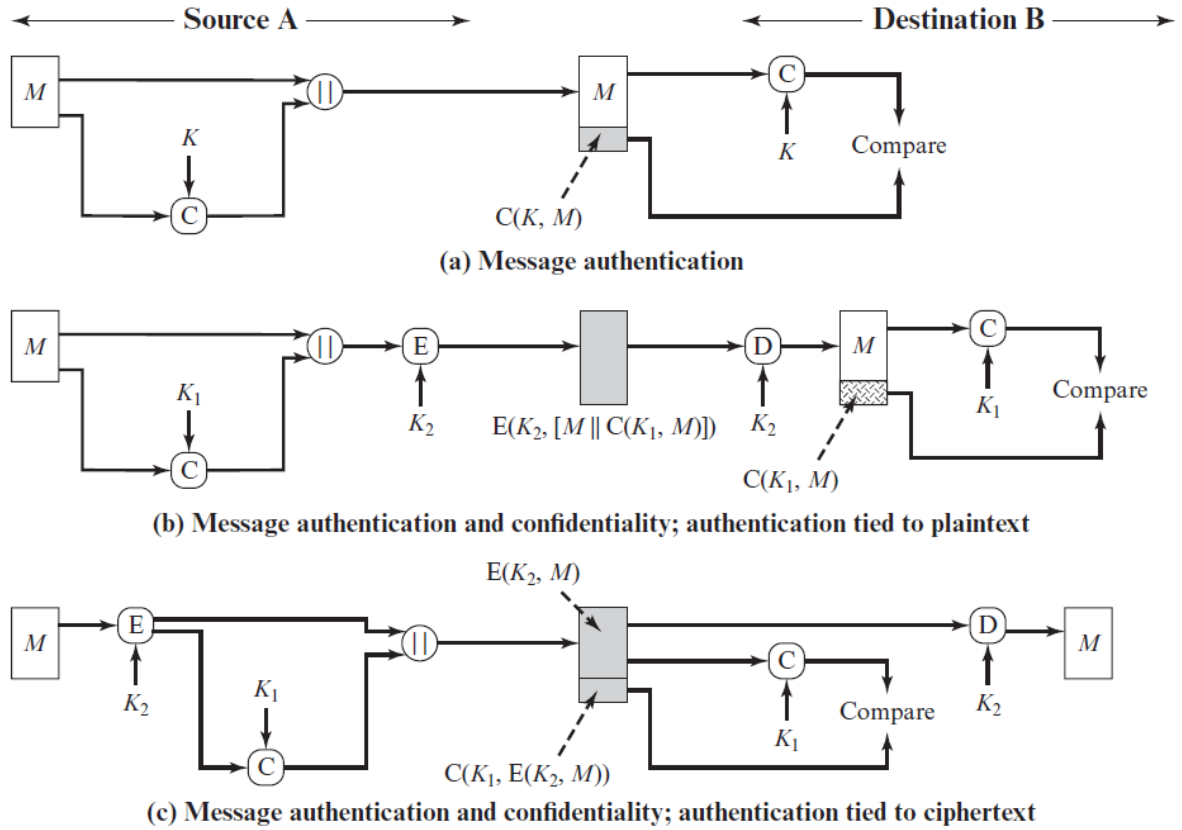
Message Authentication Code

- An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a **cryptographic checksum** or MAC, that is appended to the message.
- This technique assumes that two communicating parties, say A and B, share a common secret key *K*.
- When A has a message to send to B, it calculates the MAC as a function of the message and the key:

$$MAC = C(K, M)$$

where
M = input message
C = MAC function
K = shared secret key
MAC = message authentication code

- The message plus MAC are transmitted to the intended recipient.
- The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC.
- The received MAC is compared to the calculated MAC (Figure a). If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then
  1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC. Because the attacker is assumed not to know the secret key, the attacker cannot alter the MAC to correspond to the alterations in the message.
  2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper MAC.
  3. If the message includes a sequence number (such as is used with HDLC, X.25, and TCP), then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.
- A MAC function is similar to encryption. One difference is that the MAC algorithm need not be reversible, as it must be for decryption.
- In general, the MAC function is a many-to-one function.
- The domain of the function consists of messages of some arbitrary length, whereas the range consists of all possible MACs and all possible keys.
- If an *n*-bit MAC is used, then there are 2*n* possible MACs, whereas there are *N* possible messages with $N >> 2n$. Furthermore, with a *k*-bit key, there are 2*k* possible keys.
- For example, suppose that we are using 100-bit messages and a 10-bit MAC. Then, there are a total of 2100 different messages but only 210 different MACs. So, on average, each MAC value is generated by a total of 2100/210 = 290 different messages. If a 5-bit key is used, then there are 25 = 32 different mappings from the set of messages to the set of MAC values.



- The process depicted in Figure (a) provides authentication but not confidentiality, because the message as a whole is transmitted in the clear.
- Confidentiality can be provided by performing message encryption either after (Figure b) or before (Figure c) the MAC algorithm.
- In both these cases, two separate keys are needed, each of which is shared by the sender and the receiver. In the first case, the MAC is calculated with the message as input and is then concatenated to the message.
- The entire block is then encrypted. In the second case, the message is encrypted first.
- Then the MAC is calculated using the resulting ciphertext and is concatenated to the ciphertext to form the transmitted block.
- Typically, it is preferable to tie the authentication directly to the plaintext, so the method of Figure b is used.

MACs based on hash functions: HMAC

HMAC Design Objectives

RFC 2104 lists the following design objectives for HMAC.

- To use, without modifications, available hash functions. In particular, to use hash functions that perform well in software and for which code is freely and widely available.
- To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required.
- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function.

HMAC Algorithm

H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)

IV = initial value input to hash function

M = message input to HMAC (including the padding specified in the embedded hash function)

Yi _ i th block of M, $0 \leq i \leq (L-1)$

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; recommended length is $\geq n$; if key length is greater than b, the key is input to the hash function to produce an n-bit key

$K+$ _ 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

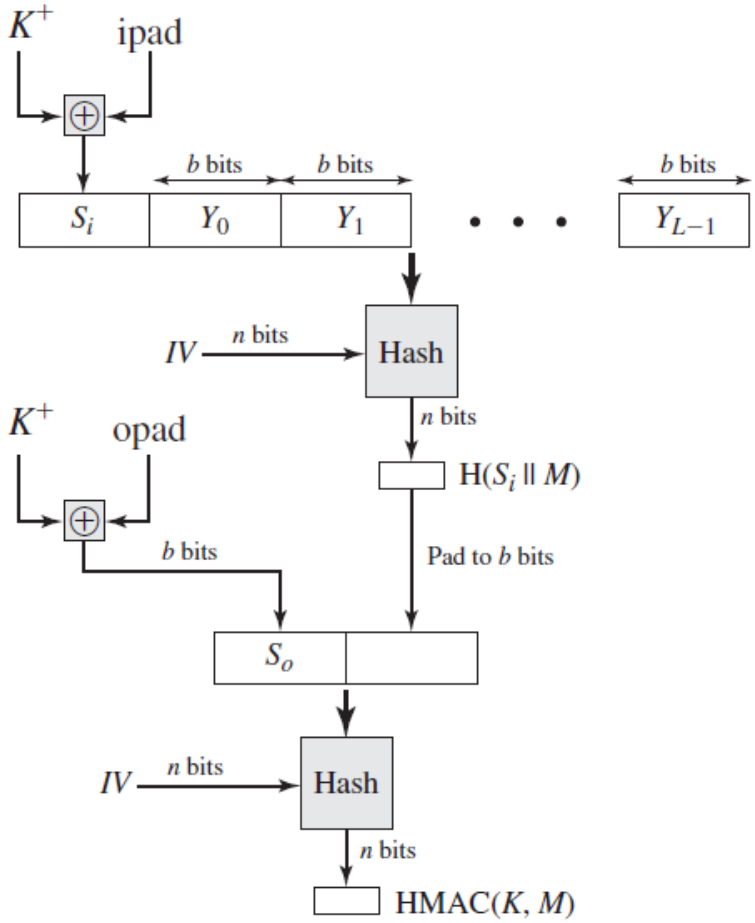


fig: HMAC Structure

HMAC can be expressed as: $\text{HMAC}(K, M) = \text{H}[(K+ \oplus \text{opad}) \| \text{H}[(K+ \oplus \text{ipad}) \| M]]$

The algorithm is as follows:

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 , then will be appended with 44 zeroes).

2. XOR (bitwise exclusive-OR) with ipad to produce the b-bit block S_i.

3. Append M to S_i.

4. Apply H to the stream generated in step 3.

5. XOR K^+ with opad to produce the b-bit block S_o

6. Append the hash result from step 4 to S_o.

7. Apply H to the stream generated in step 6 and output the result.