Message Authentication Codes

Chapter 12 in Stallings book (7th Ed)

↳ In the context of communications across a network, following attacks may happen -

1. Disclosure
2. Traffic Analysis

} Breaks confidentiality. So, encryption is required.

3. Masquerade / Spoofing
4. Content modification
5. Sequence modification
6. Timing modification

} Breaks Integrity. So, message authentication is required.

7. Source repudiation
8. Destination repudiation

} Breaks non-repudiation. So, Digital Signature is required.

↳ In brief, message authentication is a procedure to verify that received message come from the alleged source and have not been altered. It may also verify sequencing and timeliness.

↳ Any message authentication mechanism has two levels of functionality.

↳ At the lower level, there must be some function that produces an authenticator - a value to be used to authenticate a message.

↳ A higher level authentication protocol that enables a receiver to verify the authenticity of a message.

Types of functions that may be used to produce an authenticator.

I → Hash Function: maps a message of any length into a fixed-length hash value.

II → Message Encryption: Ciphertext could be used as authenticator

III → Message Authentication Codes (MAC): A function of the message and a secret key that produces a fixed-length value that serves as authenticator.

≈ Cryptographic Hash Function

(I)

Hash function-based Message Authentication

↳ It is already discussed in previous chapter
chapter 11 in Stallings's Book (7th Ed).

(II)

Message Encryption-based Message Authentication

(1) Symmetric Encryption

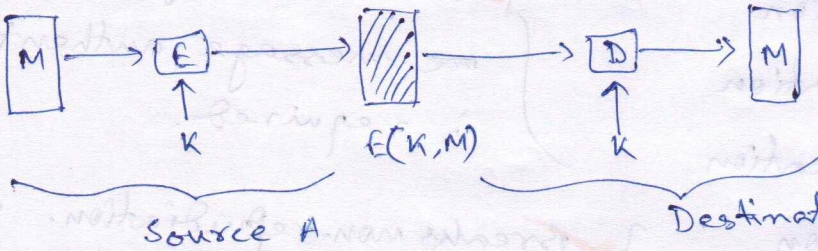


Fig: Confidentiality and Message Authentication

↳ Except B, no other party can recover the plaintext.
↳ so, confidentiality

↳ B is assured that the message was generated by A because A knows the secret key K.
So, message authentication.

Note:
In general, the message M can be any arbitrary bit pattern.
Now, source A sends $x = E(K, M)$ to destination B.
Now, destination B recives a bit pattern, say x, and do

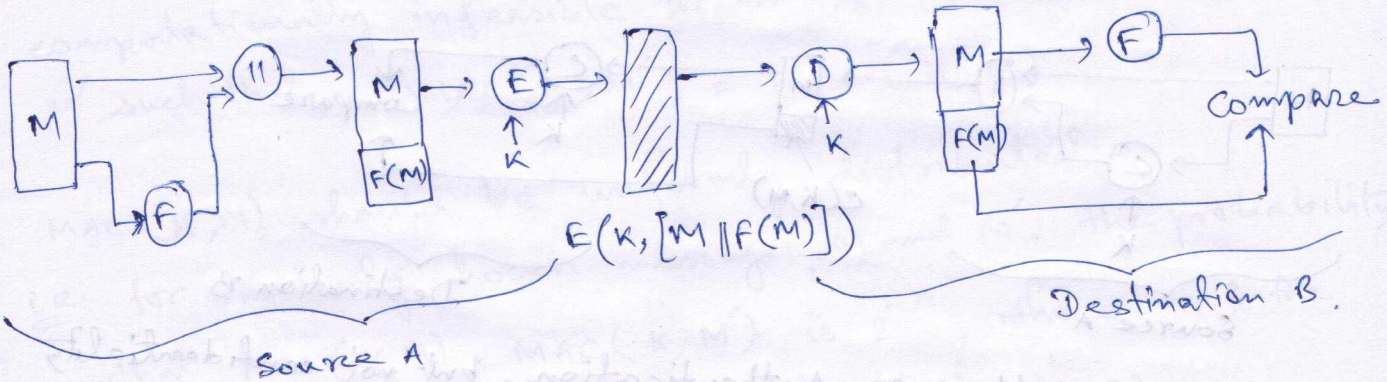
$$Y = D(K, X).$$

Here y is also an arbitrary bit pattern to B.
So, how can B ensure y is M ?
↳ not possible automatically. So, attacker can forward x' instead of x to destination B, and still the authentication will pass!
Thus, in general, a small subset of all possible bit patterns be considered legitimate plaintext for message authentication. So, any spurious ciphertext is unlikely to produce legitimate plaintext.
↳ Problem: That small subset does not appear always. How to provide authentication then ?

One solution:

↳ Force the plaintext to have some structure that is easily recognized, but cannot be replicated without recourse to the encryption function.

e.g. Frame Check Sequence (FCS) or checksum or any such kind of error detection code can serve the purpose. See the below Diagram.



(2) Public-Key Encryption

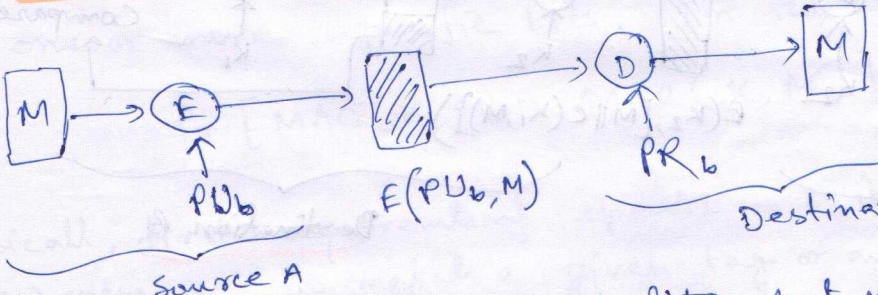


fig: Confidentiality but not Message Authentication

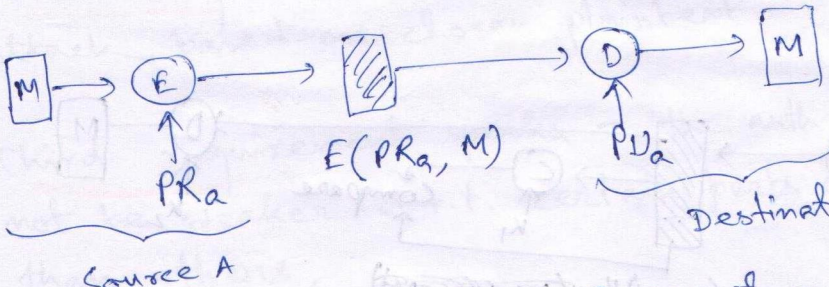


fig: Authentication and signature, but not confidentiality.

But this is not proper way.
So, we have separate Digital Signature Algo.

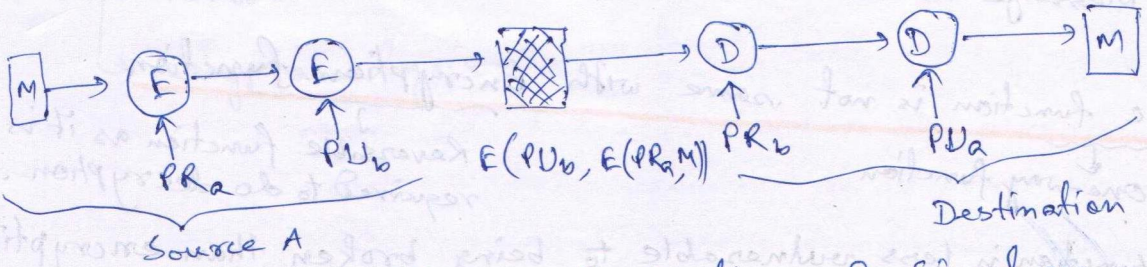


fig: Confidentiality, Authentication, and Signature.

(11) # Message Authentication Code (MAC) - based Authentication

↳ Use 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.

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

↑ MAC function

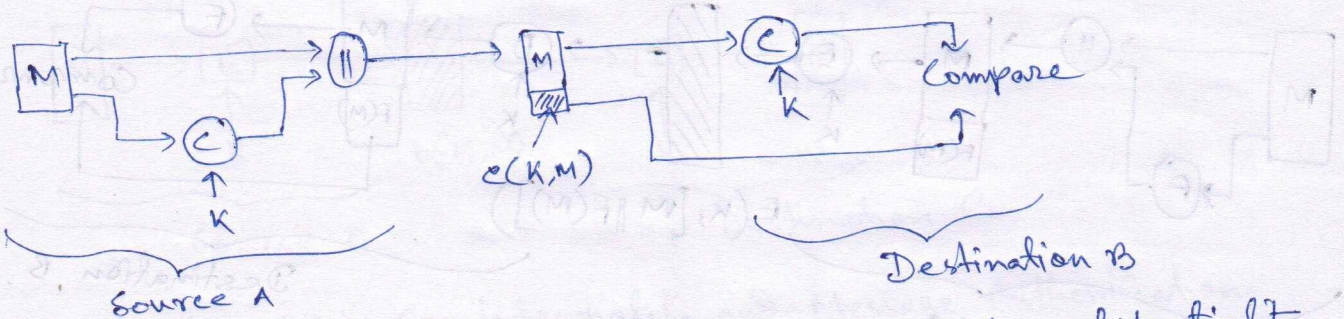


Fig: Message Authentication, but not confidentiality

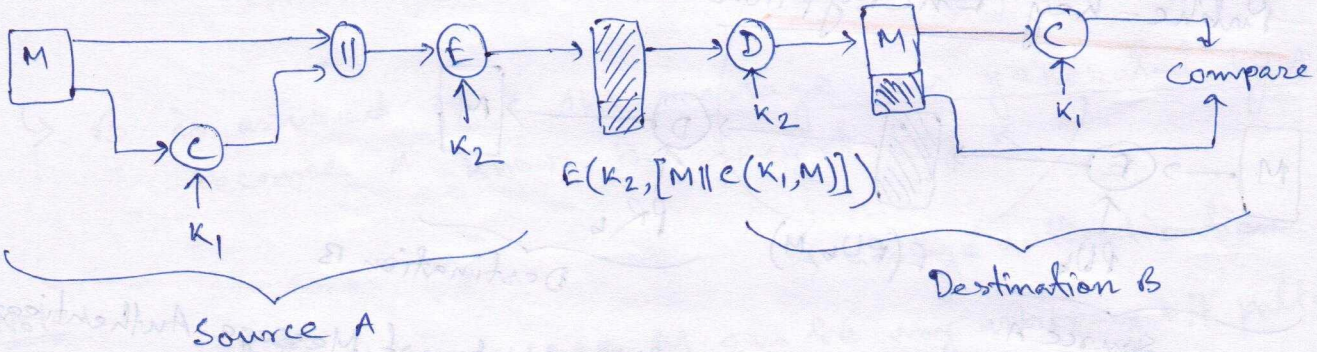


Fig: Message Authentication and Confidentiality

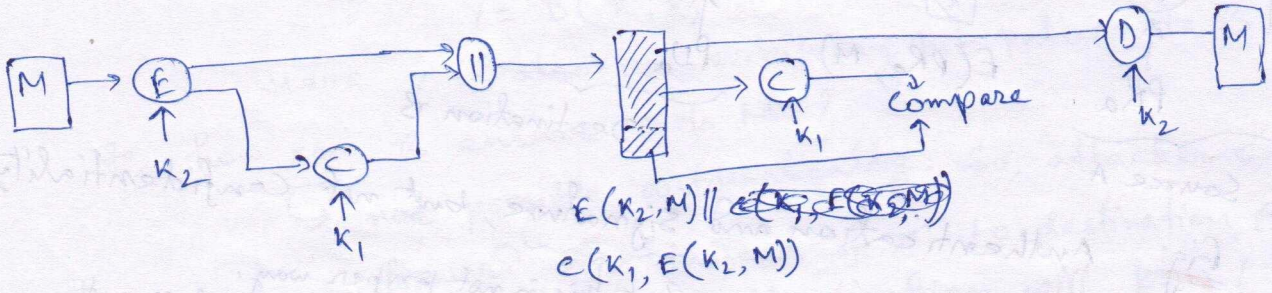


Fig: Message authentication and confidentiality.

Note: MAC function is not same with Encryption function

↓ generally one-way function

↓ Reversible function as it is required to do decryption.

↳ MAC function is less vulnerable to being broken than encryption.

Note: MAC does not provide a digital signature, because both sender and receiver share the same key.

Requirements for MAC w.r.t. security

Assume that an opponent knows the MAC function but does not know the authentication key K.

(1) If an opponent observes M and MAC(K, M), it should be computationally infeasible for the opponent to find a message M' such that MAC(K, M') = MAC(K, M)

(2) MAC(K, M) should be uniformly distributed
i.e. for randomly chosen messages, M and M', the probability that MAC(K, M) = MAC(K, M') is 2^{-n}, where n is the number of bits in tag or authenticator.

(3) Let M' = f(M). for example, f may involve inverting one or more specific bits. In this case also,

$$Pr[MAC(K, M) = MAC(K, M')] = 2^{-n}$$

↳ Basically, first requirement speaks — is it possible to construct a new message to match a given tag or authenticator.

↳ Second requirement speaks — need to thwart a brute-force attack based on chosen plaintext.

↳ Third requirement speaks — the authentication algorithm should not be weaker w.r.t. certain parts or bits of the message than others.

Attack types on MAC

↳ Brute-force Attack
↳ Cryptanalysis

MACs based on Hash functions: HMAC

↳ Before the development of AES, the main motivation was to use MD5 and SHA for designing MAC as DES is not faster design.

↳ SHA cannot be directly used as a MAC because it does not rely on a secret key.

↳ Many proposals came to incorporate a secret key into an existing hash function.

↳ widely accepted one is HMAC.

RFC 2104

↳ NIST also accepted HMAC as a standard. (FIPS 198)

↳ HMAC is used in IPsec, SSL, etc.

HMAC Design objectives

↳ To use, without modifications, available hash functions.

↳ To allow for easy replaceability of the embedded hash function if required

↳ To preserve the original performance of the hash function

↳ To use and handle key 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

$$HMAC(K, M) = H[(K^{+} \oplus opad) \,||\, H[(K^{+} \oplus ipad) \,||\, M]]$$

Here, H ⇒ embedded hash function (eg. MD5, SHA-1, ~~SHA-256~~, ...)

ipad = 00110110 (36 in hex) repeated b/8 times

opad = 01011100 (5C in hex) repeated b/8 times

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

Y_i = i-th block of M, $0 \le i \le L-1$

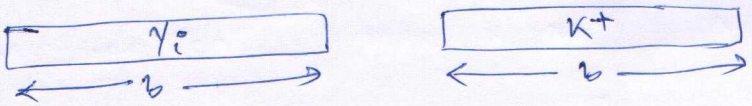
L = number of blocks in M

b = number of bits in a block

n = length of hash code in bits

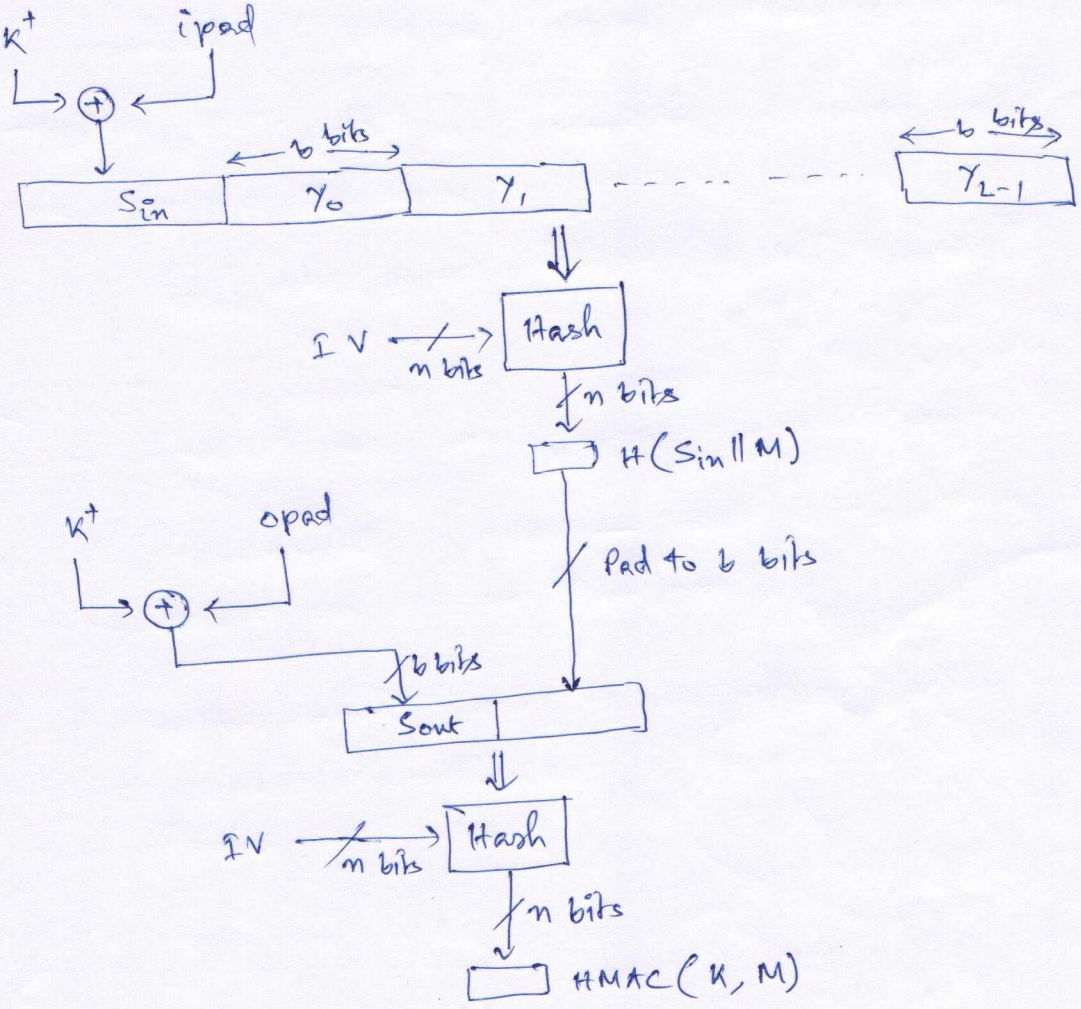
K = secret key ; recommended length is $\geq n$.

K^+ = K padded with zeros on the left so that the result is b bits in length.



if key length is > b, then key is input to the hash function to produce an n-bit key.

HMAC Structure



Note: HMAC adds three executions of the hash compression function f (for S_{in}, S_{out} and the block produced from inner hash).

→ $f(IV, (K^+ \oplus ipad))$

→ $f(IV, (K^+ \oplus opad))$

→ $f(f(IV, (K^+ \oplus opad)), H(S_{in} \| M)$ with Pad)