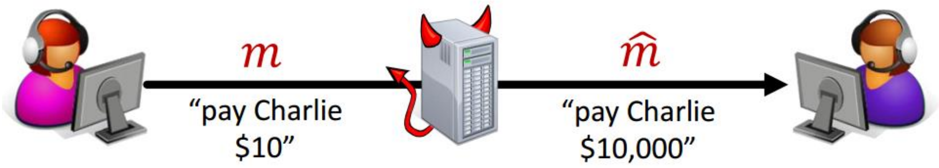
Chapter 5- Message authentication codes and hash functions

Message authentication is a procedure to verify that received messages come from the *alleged* source (authentication) and have *not been altered (integrity)*.



- 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 METHODS

All message authentication methods must produces an *authenticator*: a value to be used to authenticate a message.

There are three methods to perform authentication.

1- **Hash function:** A function that maps a message of any length into a fixed length hash value, which serves as the authenticator

2- **Message encryption:** The ciphertext of the entire message serves as its authenticator

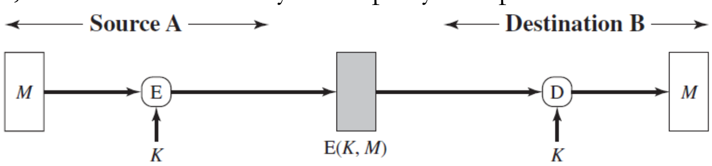
3- **Message authentication code (MAC):** A function of the message and a secret key that produces a fixed-length value that serves as the authenticator

Message Encryption

Symmetric encryption

Consider the straightforward use of symmetric encryption. A message transmitted from source A to destination B is encrypted using K a secret key shared by A and B.

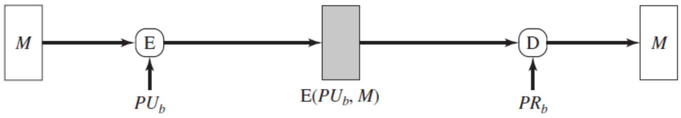
- B is assured that the message was generated by A. Why? The message must have come from A, because A is the only other party that possesses K.



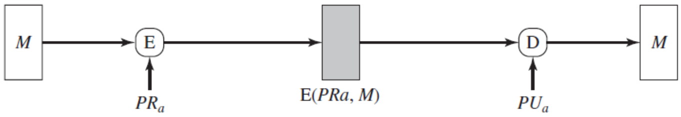
(a) Symmetric encryption: confidentiality and authentication

Public key encryption

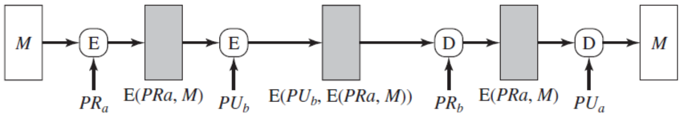
Public key encryption provides *confidentiality*, *authentication (digital signature)* and *combination* of them.



(b) Public-key encryption: confidentiality



(c) Public-key encryption: authentication and signature



(d) Public-key encryption: confidentiality, authentication, and signature

Message Authentication Code

This method 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.

$$\text{MAC} = \text{MAC}(K, M)$$

where

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

- Receiver performs same computation on message and checks it matches the MAC.
- A MAC function is similar to encryption but it need not to be reversible.
- 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
  - potentially many messages have same MAC,
    - make sure finding **collisions** is very difficult
- MAC can be used as follows:

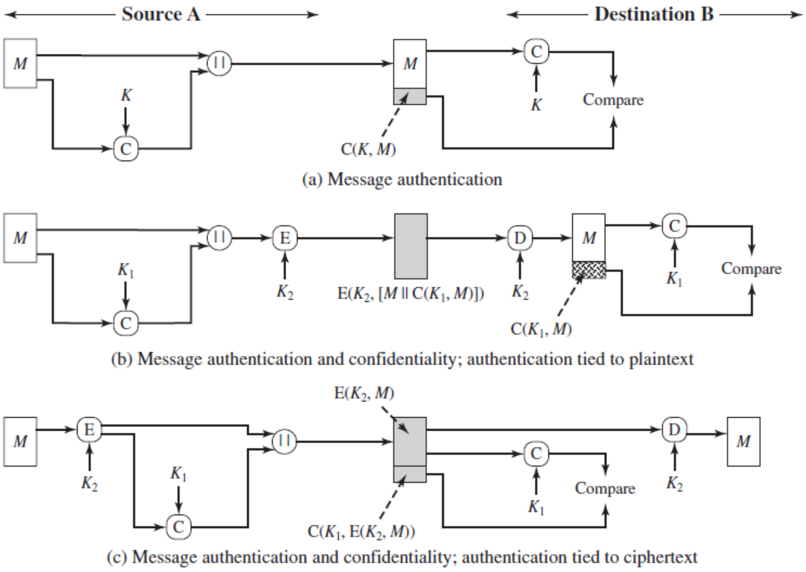


Figure 12.4 Basic Uses of Message Authentication code (MAC)

REQUIREMENTS FOR MESSAGE AUTHENTICATION CODES

1. If an opponent observes M and $\text{MAC}(K, M)$, it should be computationally infeasible for the opponent to construct a message M' such that

$$\text{MAC}(K, M') = \text{MAC}(K, M)$$

2. $\text{MAC}(K, M)$ should be uniformly distributed in the sense that for randomly chosen messages, M and M', the probability that $\text{MAC}(K, M) = \text{MAC}(K, M')$ is 2^{-n}, where n is the number of bits in the tag.
3. Let M' be equal to some known transformation on M. That is, $M' = f(M)$. For example, f may involve inverting one or more specific bits. In that case,

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

SECURITY OF MACS

- MAC algorithm must have the following security property: *knowing a message and MAC, it is infeasible to find another message with same MAC.*

Computation resistance: Given one or more text-MAC pairs $[x_i, \text{MAC}(K, x_i)]$, it is computationally infeasible to compute any text-MAC pair $[x, \text{MAC}(K, x)]$ for any new input $x \neq x_i$.

MACS BASED ON BLOCK CIPHERS: DAA

- ❖ Can use any block cipher chaining mode and use final block as a MAC
- ❖ Data Authentication Algorithm (DAA) is a widely used MAC based on DES-CBC
  - o using IV=0 and zero-pad of final block
  - o encrypt message using DES in CBC mode
  - o and send just the final block as the MAC
- ❖ But final MAC is now too small for security

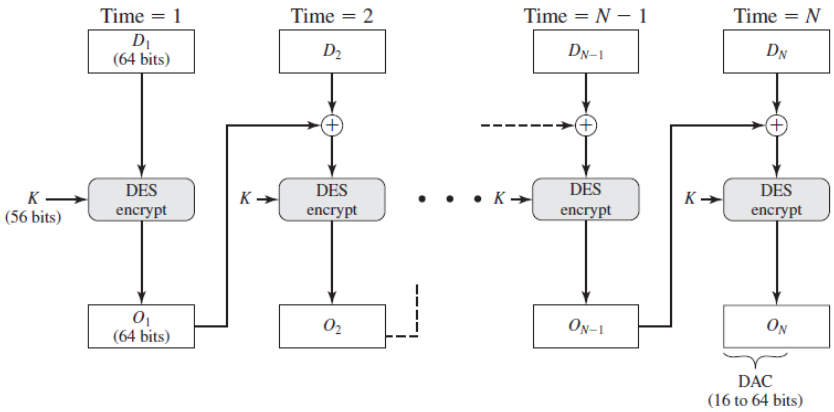


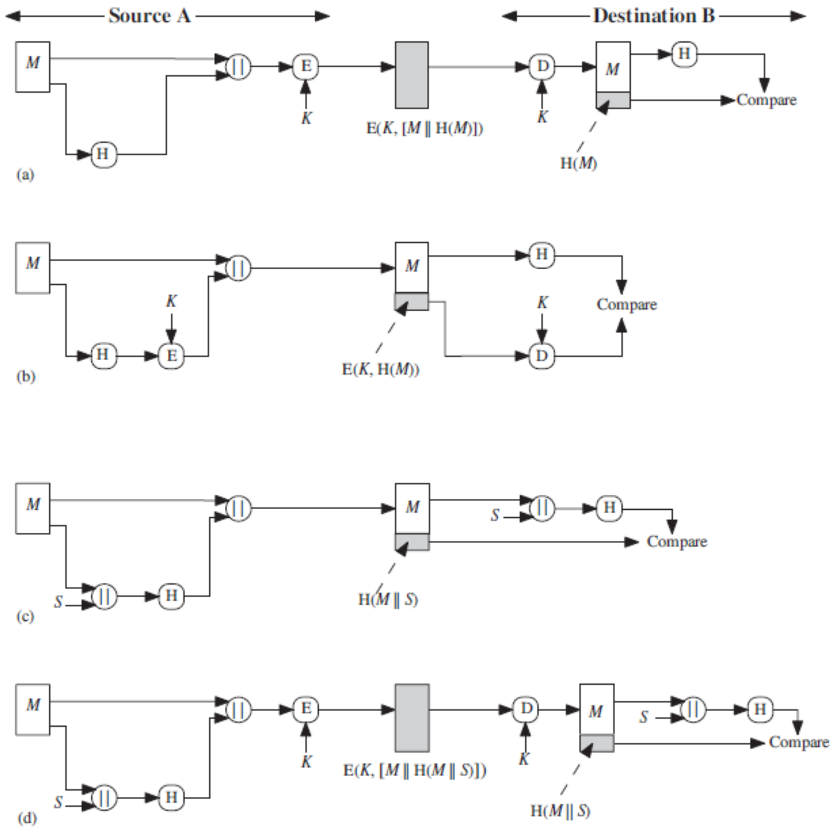
Figure 12.7 Data Authentication Algorithm (FIPS PUB 113)

Hash functions

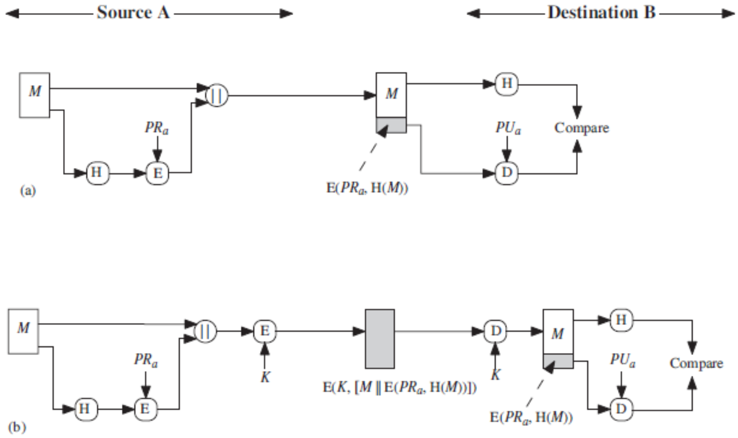
A **hash function** H accepts a variable-length block of data as input and produces a fixed-size hash value **h=H(M).**

Applications of hash functions

1- **Message authentication**



2- Digital signature



3- Store the one-way password file.

4- hash function can be used to construct a **pseudorandom function (PRF)** or a **pseudorandom number generator (PRNG)**.

Requirements for Hash Functions

1. Can be applied to any sized message M
2. Produce fixed-length output h
3. Easy to compute h=H(M) for any message M
4. **one-way property**: given h is infeasible to find x s.t. H(x)=h
5. **weak collision resistance**: given x is infeasible to find y s.t. H(y)=H(x)
6. **strong collision resistance**: infeasible to find any x,y s.t. H(y)=H(x)

Birthday Attacks

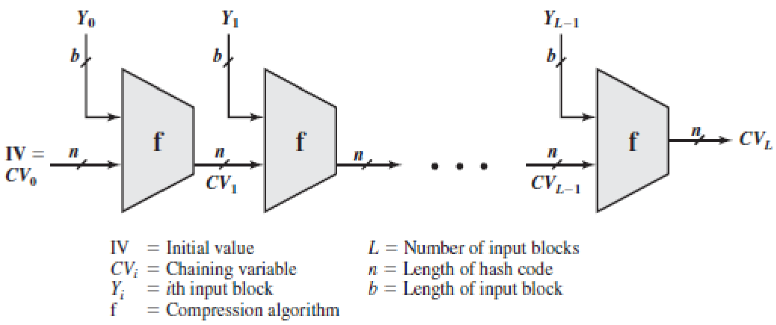
- If there are 23 people in a room, the probability that at least two people have the same birthday is slightly more than 50%.
- If there are 30, the probability is around 70%.
    - Finding collisions of a hash function using Birthday Paradox.
    - randomly chooses k messages, $x_1, x_2, \ldots, x_k$
    - search if there is a pair of messages, say x_i and x_j such that $h(x_i) = h(x_j)$.
        - If so, one collision is found.
    - This birthday attack imposes a lower bound on the size of message digests.
    - If n = 64, the probability of finding one collision will be higher than half after slightly more than 2^{32} random hashes being tried.

Block Ciphers as Hash Functions

- Can use block ciphers as hash functions
    - use H_0=0 and zero-pad of final block
    - compute $H_i = E_{Mi} [H_{i-1}]$
    - use final block as the hash value
    - similar to CBC but without a key
- Resulting hash is too small (64-bit) both due to direct birthday attack

The hash function structure

The hash function takes an input message and partitions it into *L* fixed-sized blocks *b* of bits each. If necessary, the final block is padded to bits. The final block also includes the value of the total length of the input to the hash function.



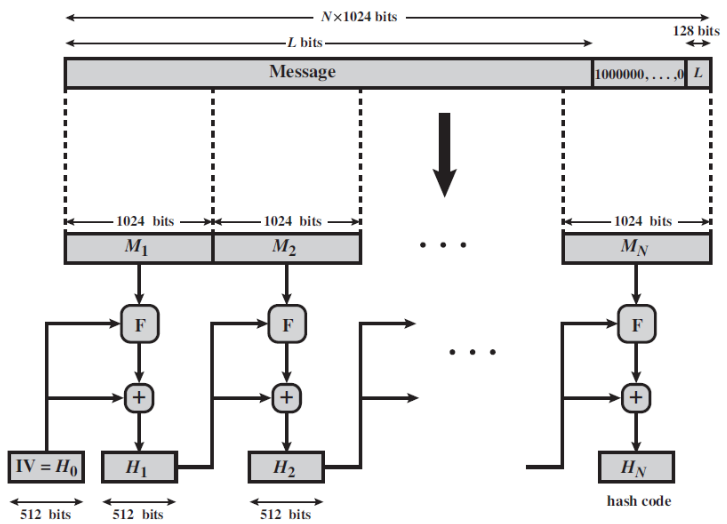
- Cryptanalysis of hash functions focuses on the internal structure of f and is based on attempts to find efficient techniques for producing collisions for a single execution of *f*.
- Typically, as with symmetric block ciphers, f consists of a series of rounds of processing.
- Two popular hash functions are MD5 and SHA-1.
- Both MD5 and SHA-1
- The output length of MD5 is 128 bits and that of SHA-1 is 160 bits. The longer output length of SHA-1 makes the generic "birthday attack" more difficult: for MD5, a birthday attack requires about $2^{128/2} = 2^{64}$ hash computations, while for SHA-1 such an attack requires about $2^{160/2} = 2^{80}$hash computations.

SECURE HASH ALGORITHM (SHA)

- SHA was developed by the National Institute of Standards and Technology (NIST) and published in 1993. The first version is called AHA-0.
- SHA-1, produced in 1995, produces a hash value of 160 bits.
- In 2002, NIST produced a revised version 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, respectively. These hash algorithms are known as **SHA-2**.

SHA-512 Logic

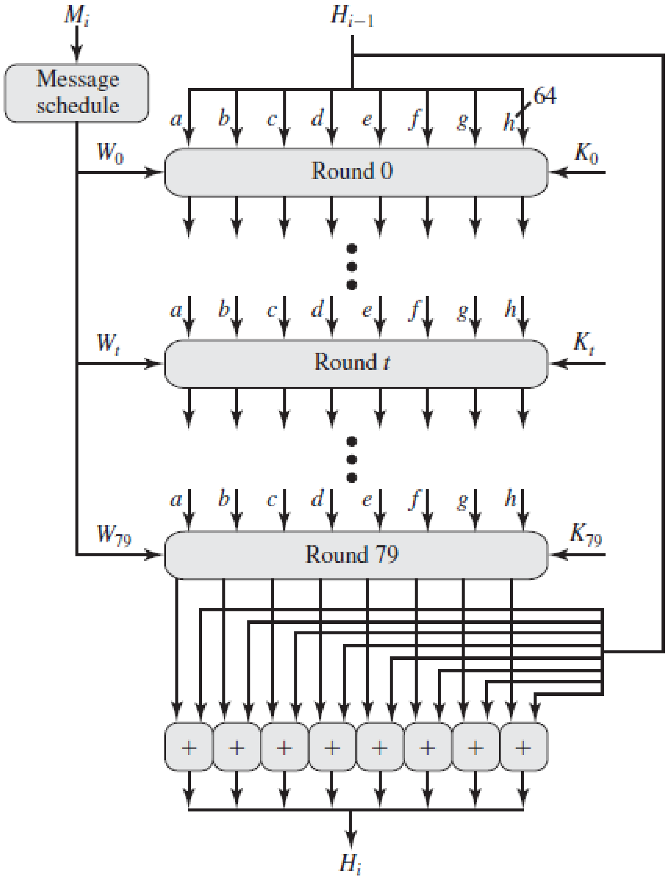
The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks.



- A 512-bit buffer is used to hold intermediate and final results of the hash function.The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h).These registers are initialized to the following 64-bit integers (hexadecimal values):

```
a = 6A09E667F3BCC908   e = 510E527FADE682D1
b = BB67AE8584CAA73B   f = 9B05688C2B3E6C1F
c = 3C6EF372FE94F82B   g = 1F83D9ABFB41BD6B
d = A54FF53A5F1D36F1   h = 5BE0CD19137E2179
```

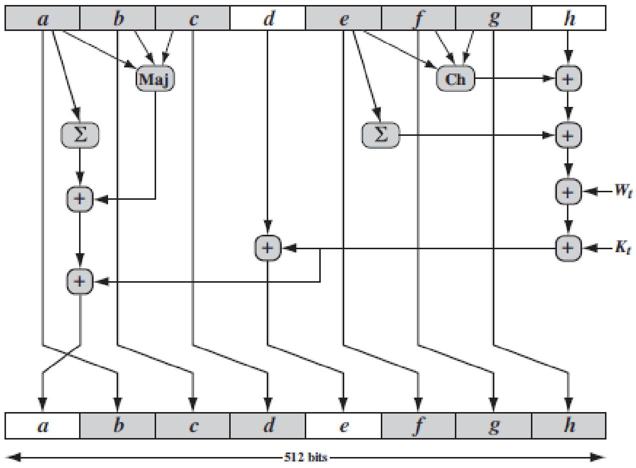
- The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F.



- Each round makes use of a 64-bit value *wi*, derived from the current 1024-bit block being processed (*Mi*). Each round also makes use of an additive constant *Kt* where t=0…79, indicates one of the 80 rounds.

SHA-512 Round Function

Each round is defined by the following set of equations:



Where,

$\mathrm{Maj}(a, b, c) = (a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$

the function is true only of the majority (two or three) of the arguments are true

$\mathrm{Ch}(e, f, g) = (e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g)$

the conditional function: If e then f else g

$$\left(\sum\nolimits_0^{512} a\right) = \mathrm{ROTR}^{28}(a) \oplus \mathrm{ROTR}^{34}(a) \oplus \mathrm{ROTR}^{39}(a)$$

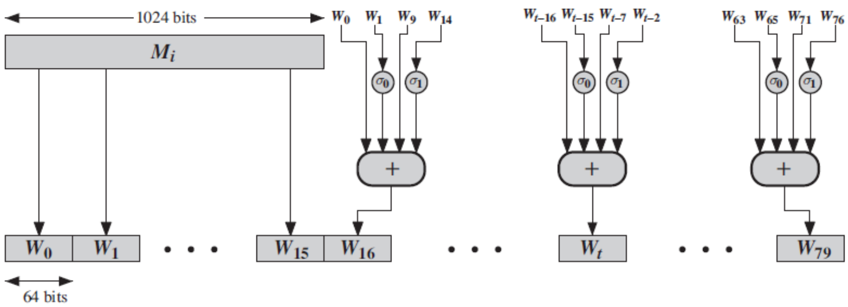
$$\left(\sum\nolimits_1^{512} e\right) = \mathrm{ROTR}^{14}(e) \oplus \mathrm{ROTR}^{18}(e) \oplus \mathrm{ROTR}^{41}(e)$$

- It remains to indicate how the 64-bit word values are derived from the 1024-bit message.
- The first 16 values of are taken directly from the 16 words of the current block. The remaining values are defined as

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

where

$$\sigma_0^{512}(x) = \mathrm{ROTR}^1(x) \oplus \mathrm{ROTR}^8(x) \oplus \mathrm{SHR}^7(x)$$
$$\sigma_1^{512}(x) = \mathrm{ROTR}^{19}(x) \oplus \mathrm{ROTR}^{61}(x) \oplus \mathrm{SHR}^6(x)$$

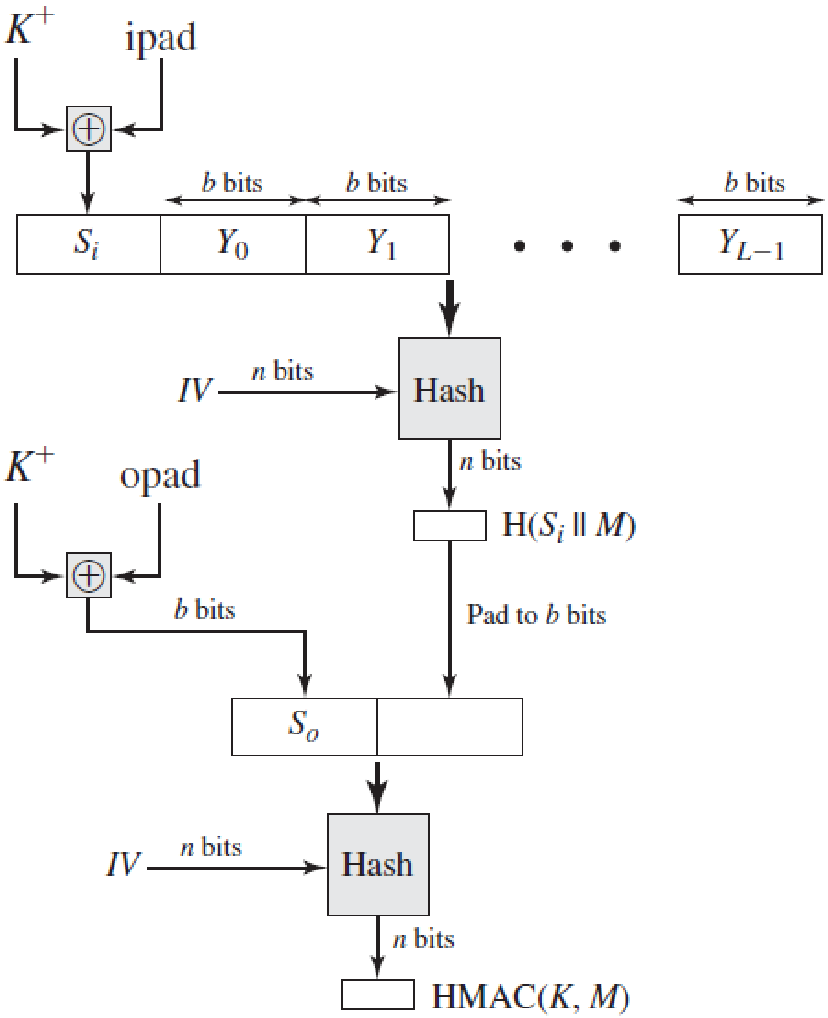


MACS BASED ON HASH FUNCTIONS: HMAC

- MAC can be generated by use hash functions.
- This is because hash functions such as MD5 or SHA are faster than symmetric block ciphers like DES.

- A hash function such as SHA was not designed for use as a MAC and cannot be used directly for that purpose, because it does not rely on a secret key.
- There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm.
- The approach that has received the most support is HMAC
- HMAC has been issued as RFC 2104, has been chosen as the mandatory-to-implement MAC for IP security, and is used in other Internet protocols, such as SSL.

HMAC Algorithm



Where,

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)
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 produced by embedded hash function
K = secret key; recommended length is $\ge n$; 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

Then HMAC can be expressed as

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

Security of HMAC

- Security of HMAC relates to that of the underlying hash algorithm
- Attacking HMAC requires either:
  - brute force attack on key used
  - birthday attack (but since keyed would need to observe a very large number of messages)

Obtaining Privacy and Message Authentication

Sometimes we actually need both *privacy* and *authentication*. There are three common approaches to combining encryption and message authentication.

1. *Encrypt-and-authenticate*: In this method, encryption and message authentication are computed and sent separately. That is, given a message *m*, the final message is the pair (*c, t*) where:

$$c = Enc_{k1}(m) \text{ and } t = Mac_{k2}(m)$$

2. *Authenticate-then-encrypt*: Here a MAC tag *t* is first computed, and then the message and tag are encrypted together. That is, the message is *c*, where:

$$c = Enc_{k1}(m, t) \text{ and } t = Mac_{k2}(m)$$

Note that *t* is not sent separately to *c*, but is rather incorporated into the plaintext.

3. *Encrypt-then-authenticate*: In this case, the message *m* is first encrypted and then a MAC tag is computed over the encrypted message. That is, the message is the pair (*c, t*) where:

$$c = Enc_{k1}(m) \text{ and } t = Mac_{k2}(c)$$

- Both 1 and 2 approaches have been proven to be *not secure*.
- Approach 3 is secure.
- If an encryption scheme and a message authentication scheme are both needed, then *independent* keys should be used for each one.

الفاتحة على روح المرحوم الاستاذ الدكتور اياد ابراهيم مسبوقة بالصلاة على محمد و ال محمد