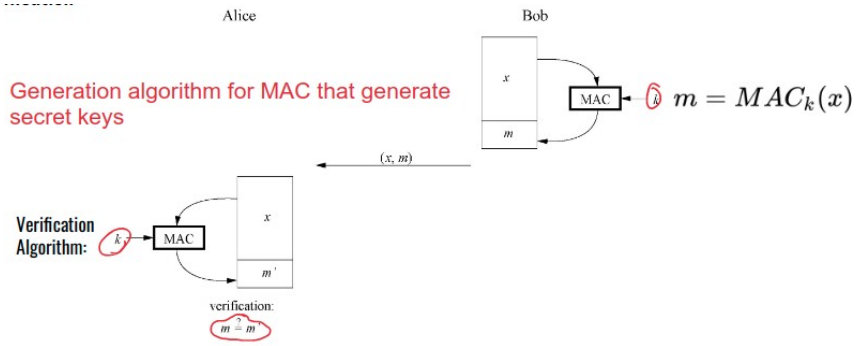
# 9. Message Authentication Code

Short recap:
- Confidentiality → ciphers
- Integrity → hash functions
- No repudiation → Symmetric ciphers

## 9.1 Message Authentication

It is a property that a message has not been modified while in transit and that the receiving party can verify the source of the message. (does not imply not repudiation)




### 9.1.1 Properties

- Cryptographic checksum: MAC generates output tag (authentication tag)

- Symmetric: MACs are based on secret symmetric keys.

- Arbitrary message size

- Fixed output length

- Message integrity (hash function)

- Message authentication

- No non repudiation

### 9.1.2 Security requirements

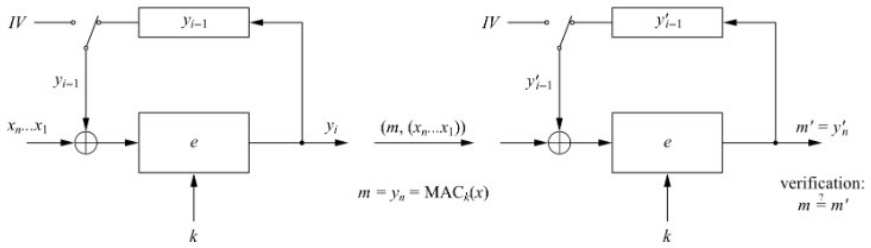
An adversary should not be able to construct (forge) a new legal (valid) pair (x, MACk(x)) even after seeing valid pairs from previous communication sessions.

**Goal of attacked:** Find a new legal pair (y, MACk(y)) efficiently and with non negligible probabilty. The attack is successful even when y is meaningless (in general defined when a message is meaningful is not trivial and requires other rules).

# 9.2 Implementations

## 9.2.1 AES in CBC mode (slow)

CBC-MAC:




IV=0 in most scenarios

**MAC Generation**

•Divide the message x into blocks xi

•Compute first iteration y1 = ek(x1⊕IV)

•Compute yi = ek(xi⊕yi−1) for the next blocks

•Final block is the MAC value: m = MACk(x) = yn

**MAC Verification**

•Repeat MAC computation (m')

•Compare results: in case m' = m, the message is verified as correct

•In case m' ≠ m, the message and/or the MAC value m have been altered during transmission

If ek is a pseudo random function, then the fixed-length CBC MAC is resilient to forgery.

The variable-length CBC MAC is insecure.

$$
\begin{aligned}
y''_1 &= E_k(0 \oplus x_1) = y_1 \\
y''_2 &= E_k(y''_1 \oplus x_2) = E_k(y_1 \oplus x_2) = y_2 \\
&\cdots \\
y''_L &= E_k(y''_{L-1} \oplus x_L) = E_k(y_{L-1} \oplus x_L) = y_L = t \\
y''_{L+1} &= E_k(y''_L \oplus (x'_1 \oplus t)) = E_k(t \oplus (x'_1 \oplus t)) = E_k((t \oplus t) \oplus x'_1) = E_k(0 \oplus x'_1) = y'_1 \\
y''_{L+2} &= E_k(y''_{L+1} \oplus x'_2) = E_k(y'_1 \oplus x'_2) = y'_2 \\
&\cdots \\
y''_{L+L'} &= E_k(y''_{L+L'-1} \oplus x'_{L'}) = E_k(y'_{L'-1} \oplus x'_{L'}) = y'_{L'} = t'
\end{aligned}
$$

We know a priori.

**IMPROVEMENTS:**

- Input-length key separation: generate a new key k' = Ek(l) to use in CBC-MAC where l is the message length.

- Length-prepending: include message length in the first block, i.e., x' = l || x, where l is the message length.

- Encrypt last block. ECBC-MAC: $E_{k2}(\text{CBC-MAK}_{k1}(x))$

## appending the length at the end is not safe

One may think that appending the length at the end would be safe. However, given:

$$(x = (x_1, x_2, \ldots, x_l), m_x)$$
$$(x' = (x'_1, x'_2, \ldots, x'_l), m'_x)$$
$$(x'' = (x_1, x_2, \ldots, x_l, l, x''_1, x''_2, \ldots, x''_l), m_{x''})$$

Then we can forge a valid pair as:

$$(x''' = (x'_1, x'_2, \ldots, x'_l, l, x''_1 \oplus m_x \oplus m'_x, x''_2, \ldots, x''_l), m_{x''})$$

## Non constant IV

Using a variable IV may expose to attacks. Given:

$$(x = (x_1, x_2, \ldots, x_l), m_x, IV)$$

The attacker can forge:

$$(x' = (x'_1, x_2, \ldots, x_l), m_x, IV')$$

choosing an IV' such that

$$x'_1 \oplus IV' = x_1 \oplus IV$$

## 9.2.2 MACs from Hash Functions

MAC is realized with cryptographic hash functions (e.g., SHA-1). In particular, when we combine the message and key and then perform hashing we called the process keyed hashing.

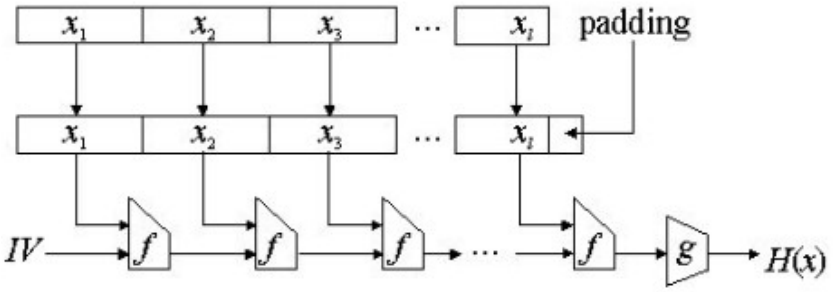
Idea: key is hashed together with the message,

e.g., secret prefix MAC: m = MACk(x) = h(k || x)

secret suffix MAC: m = MACk(x) = h(x || k)

### 9.2.3 Merkle-Damgård construction

Several cryptographic hash function takes as input a fixed block size (e.g., 256 bits). To fulfill the requirement on arbitrary input length, we can use the Merkle-Damgård construction:




### 9.2.4 Secret prefix MAC: attack

$$m_x = h(k||x)$$

Given ($x = (x_1, x_2, \ldots, x_n)$, $m_x$), an attacker can easily forge ($x' = (x_1, x_2, \ldots, x_n, x_{n+1})$, $m'_x$) without knowing the secret key as:

$$m'_x = h(x_{n+1})$$

Using $IV = m_x$ during Merkle-Damgård construction.

### 9.2.5 Secret suffix MAC: attack

$$m_x = h(x||k)$$

Assume adversary can find a collision h(x) = h(x') then also h(x||k) = h(x'||k) = $m_x$

- brute force attack: if |k|=128 then 2128 attempts to forge MAC
- collision attack: if output of h() is 160 bits then 2160/2=80 attempts to forge MAC

## 9.3 HMAC

Idea: use 2 nested secret prefix MACs, e.g., h(k || h(k || x))

HMAC is an implementation of this idea with some additional details. It is extremely popular and used in several protocols.

Scheme consists of an inner and outer hash

•k+ is expanded key k

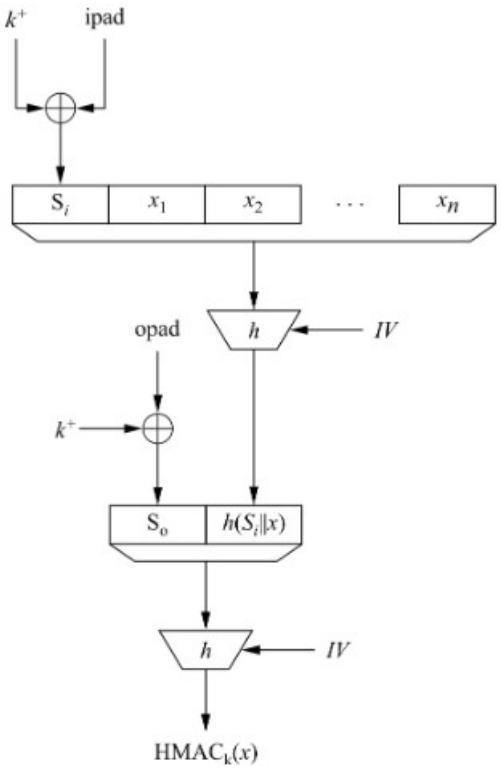
•expanded key k+ is XORed with the inner pad

•ipad = 00110110,00110110, . . .,00110110

•opad = 01011100,01011100, . . .,01011100

•HMACk(x) = h( (k+⊕opad) || h( (k+⊕ipad) || x ) )

ipad and opad are constants defined to have a large Hamming distance from each other and so the inner and outer keys will have fewer bits in common.




### versus birthday paradox

an attacker cannot generate MAC by himself with HMAC because he does not know the key k

●to exploit a collision he needs to find it: when output space is 160 bit, 280 different HMACs using the same key k to have a collision probability of 0.5.

●hence, HMAC is vulnerable to a birthday attack (this is true for any hash function!), but performing it is quite impractical.

## 9.4 Authenticated Encryption

Authenticated encryption (AE) and Authenticated Encryption with Associated Data (AEAD) are forms of encryption which simultaneously assure the confidentiality and authenticity of data.

It can provide security against chosen ciphertext attack, it can recognize improperly-constructed ciphertexts and refuse to decrypt them.

### 9.4.1 AE API

Encryption

●input: plaintext, key, and optionally a header covered by authenticity protection.

●output: ciphertext and authentication tag (message authentication code).

Decryption

●input: ciphertext, key, authentication tag, and optionally a header.

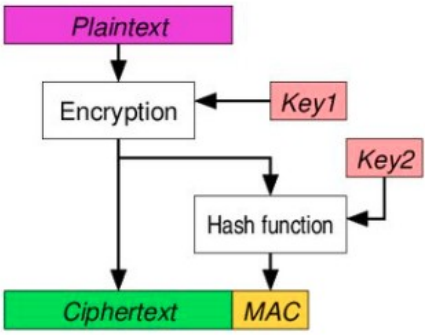
●output: plaintext, or an error .The header part is intended to provide authenticity and integrity protection when authenticity is desired.

## 9.4.2 AE Approaches

### 1. Encrypt-then-MAC

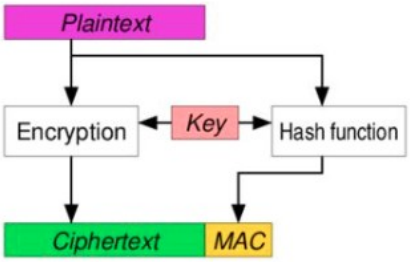
This is the only approach which can reach the highest definition of security in AE.

(used in IPSec) It can only be achieved when the MAC used is "strongly unforgeable".



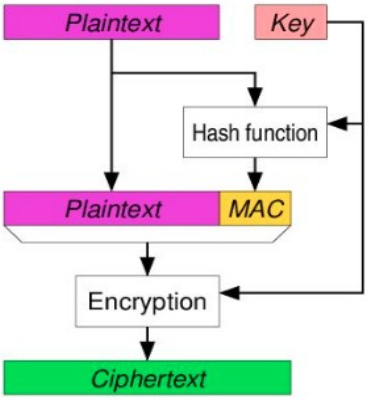

### 2. Encrypt-and-MAC

Used in, e.g., SSH. Even though the E&M approach has not been proved to be strongly unforgeable in itself, it is possible to apply some minor modifications to SSH to make it strongly unforgeable despite the approach.




### 3. MAC-then Encrypt

(Used in SSL/TLS). Even though the MtE approach has not been proven to be strongly unforgeable in itself, the SSL/TLS implementation has been proven to be strongly unforgeable due to the encoding used alongside the MtE mechanism.




# 9.5 Authenticated encryption with associated data (AEAD)

It’s variant of AE that allows a recipient to check the integrity of both the encrypted and unencrypted information in a message. AEAD binds associated data (AD) to the ciphertext and to the context where it is supposed to appear so that attempts to "cut-and-paste" a valid ciphertext into a different context are detected and rejected.

It is required, for example, by network packets or frames where the header needs visibility, the payload needs confidentiality, and both need integrity and authenticity

# 9.6 Galois Counter Mode (GCM) (Extension of CTR)

GCM is an operation mode for a block cipher that provides both encryption and message authentication.

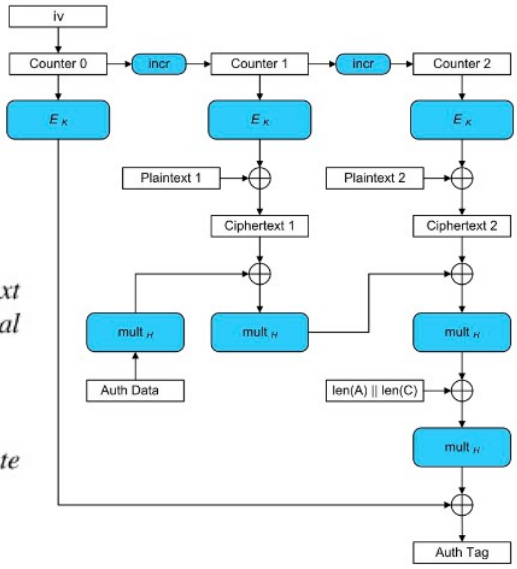
it can guarantee integrity of the encrypted message and also of another piece of information.

The main idea is to perform polynomial multiplications over Galois Fields to compute the MAC. For instance, in AES, since the block size is 128 bits, GCM uses GF($2^{128}$).

- Similarly to CTR mode

- Perform authentication as a multiplication by H = EK(0) in GF($2^{128}$)

- Send: (Ciphertext, Auth Data, Auth Tag)




**Definition 5.1.6** Basic Galois Counter mode (GCM)
*Let $e()$ be a block cipher of block size 128 bit; let $x$ be the plaintext consisting of the blocks $x_1, \ldots, x_n$; and let AAD be the additional authenticated data.*

1. ***Encryption***
   a. *Derive a counter value $CTR_0$ from the IV and compute $CTR_1 = CTR_0 + 1$.*
   b. *Compute ciphertext: $y_i = e_k(CTR_i) \oplus x_i, \quad i \geq 1$*

2. ***Authentication***
   a. *Generate authentication subkey $H = e_k(0)$*
   b. *Compute $g_0 = AAD \times H$ (Galois field multiplication)*
   c. *Compute $g_i = (g_{i-1} \oplus y_i) \times H, \quad 1 \leq i \leq n$ (Galois field multiplication)*
   d. *Final authentication tag: $T = (g_n \times H) \oplus e_k(CTR_0)$*