# Message Authentication Code

Gianluca Dini
Dept. of Ingegneria dell'Informazione
University of Pisa
Emai: gianluca.dini@unipi.it
Version: 2021-04-24

Message Authentication Codes (MACs)

# PRINCIPLES OF MACS

# Message Authentication Code

- Synonims
  - Cryptographic checksum
  - Keyed hash function
- Similarly to digital signatures, MACs provide message authentication and integrity
- Unlike digital signatures, MACs are symmetric schemes and do not provide nonrepudiation
- MACs are much faster than digital signatures

SE NON FIDANO, USANO LA FIRMA DIGITALE

# Communication model

INTEGRITY AND AUTHENTICATION


BOB NON PROVERA' A CREARE COLLISIONI

SI FIDANO UNO DELL'ALTRO!
[MUTUAL TRUST]

k ← Alice and Bob share a secret key k → k

① ASSUMO CHE L'ALTRO NON DIVULGHI SEGRETI

② L'ALTRO HA LA STESSA COMPETENZA PER MANTENERE IL SEGRETO

MSG KEY

**MAC Generation** $S$: $\{0,1\}^* \times \{0,1\}^k \to \{0,1\}^n$ ➔ $t \leftarrow S(k, x)$

**MAC Verification** $V$: $\{0,1\}^* \times \{0,1\}^k \times \{0,1\}^n \to$ Boolean ➔ $\{true, false\} \leftarrow V(x, k, t)$

**MAC generation**

$t = S(k, x)$

**MAC verification**

$t' = MAC(k, x)$

**return** $(t == t')$

Alice and Bob want to be assured that any manipulations of a message m in transit are detected

apr. '21 Hash functions 4

# Properties od MACs

- Cryptographic checksum
  - A MAC generates a cryptographically secure authentication tag for a given message.
- Symmetric
  - MACs are based on secret symmetric keys. The signing and verifying parties must share a secret key.
- Arbitrary message size
  - MACs accept messages of arbitrary length.
- Fixed output length
  - MACs generate fixed-size authentication tags.
- Message integrity
  - MACs provide message integrity: Any manipulations of a message during transit will be detected by the receiver.
- Message authentication
  - The receiving party is assured of the origin of the message.
- No nonrepudiation
  - Since MACs are based on symmetric principles, they do not provide nonrepudiation

# Security

- Computation-resistance (chosen message attack)
  - For each key k, given zero o more ($x_i$, $t_i$) pairs, where $t_i$ = S(k, $x_i$), it is computationally infeasible to compute (x, t), s.t. t = S(k, x), for any new input x ≠ xi  (including possible t = ti for some i)
    - Adaptive chosen-message attack
    - Existential forgery

SE E' GARANTITA, NON DEVE ESSERE QUAL IN
GRADO DI CALCOLARE LA CHIAVE

apr. '21 Hash functions 6

## Types of forgery

- Selective forgery
  - Attacks whereby an adversary is able to produce a new text-MAC pair for a text of his choice (or perhaps partially under his control)
    - Note that here the selected value is the text for which a MAC is forged, whereas in a chosen-text attack the chosen value is the text of a text-MAC pair used for analytical purposes (e.g., to forge a MAC on a distinct text).
- Existential forgery
  - Attacks whereby an adversary is able to produce a new text-MAC pair, but with no control over the value of that text.

# Implications of a secure MAC

- FACT 1 - Computation resistance ➔ key non-recovery (but not vice versa)
  - It must be computationally infeasible to compute k from $(x_i, t_i)$s
  - However, it may be possible to forge a tag without knowing the key

## Implications of a secure MAC

- FACT 2 **-** Attacker cannot produce a valid tag for any new message
  - Given (x, t), attacker cannot even produce (x, t’) –a collision– for t’ ≠ t

# Implications of a secure MAC

- FACT 3 - For an adversary not knowing k
  - S must be 2nd-preimage and collision resistant;
  - S must be preimage resistant w.r.t. a chosen-text attack;
- FACT 4 - Secure MAC definition says nothing about preimage and 2nd-preimage for parties knowing k
  - Mutual trust model

# How to use MACs in practice

- In combination with encryption
  - x: PT message; x': transmitted message; e: encryption key; a: MAC key
  - Option 1 (SSL)
    - t = S(a, x); c = E(e, x || t), x' = c
  - Option 2 (IpSec)
    - c = E(e, x); t = S(a, c); x' = c || t
  - Option 3 (SSH)
    - c = E(e, x); t = S(a, x); x' = c || t

Message Authentication Codes (MACs)

# HOW TO BUILD A MAC

# How to build a MAC

- From Block Ciphers
  - CBC-MAC
  - NMAC
  - PMAC
- From a hash functions
  - HMAC

# HMAC

How to build a MAC from ah hash function

- Insecure constructions
  - Secret prefix scheme
    - S(k, x) = H(k||x), H hash function
  - Secret suffix scheme
    - S(k, x) = H(x || k), H hash function
  - Forgery is possible in both cases
  - HMAC construction is necessary



S(k, m) = H(k||m) is insecure. The proof is simple. Given H(k||m||PB) it is possible to compute H(k||m||PB||w) (extension attack) so obtaining existential forgery.

# Insecurity of prefix scheme

- Let $x = x_1, x_2, x_3, \ldots, x_n$
- Let t = S(k, x) = H(k || x1,x2, …, xn)
- Construct t' of $x' = x_1, x_2, \ldots x_n, x_{n+1}$ without knowing k ($x_{n+1}$: additional block)
  - Consider the Merkle-Damgard scheme ➔
  - $t' = h(x_{n+1}, t)$, h compression function
    - The MAC of $x_{n+1}$ only needs the previous hash output t but not k

## Insecurity of the suffix scheme

- Let t = S(k, x) = H(x || k)
- Construct t’ of a x’ without knowing the key k
  - Consider Merkle-Damgard scheme
  - Assume the adversary is able to find a collision H(x) = H(x’) then t = h(H(x), k) = h(H(x’), k), thus t’ = t, h compression function

AGISCE SEMPRE SU UN NUMERO DI BIT FISSATO!

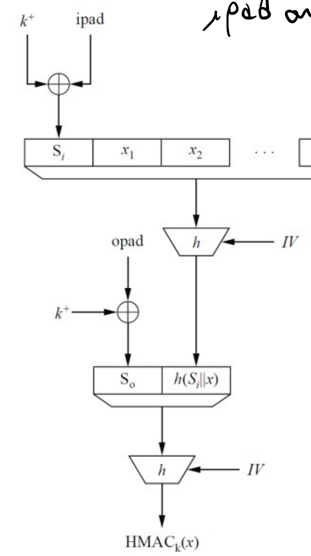
# HMAC

STANDARD

$$H\Big(\underbrace{k \oplus opad \;\Big\|\; \overbrace{H\big(\underbrace{k \oplus ipad \;\|\; k}_{128}\big)}^{\text{COSTOSO}}}_{256}\Big)$$

ipad and opad = FIXED QUANTITY

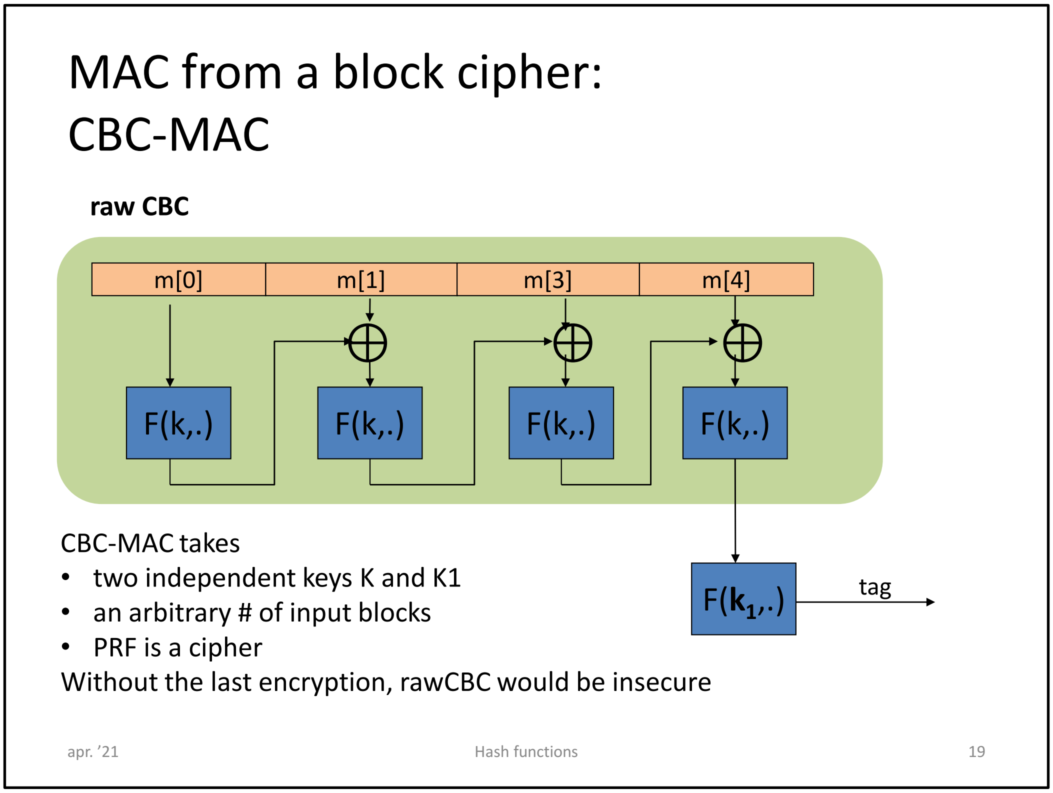
K+: padded key




apr. '21 Hash functions 17

## HMAC

- Computational efficiency
  - The message is hashed in the inner hash
  - The outer hash only hashes two blocks
- Security
  - There exists a proof of security in HMAC
  - THM - If an attacker can break HMAC then (s)he can break H




**It is important to notice that without the last encryption the MAC would be insecure. Notice that many product and standards implement CBC-MAC incorrectly, i.e., without the last encryption, and thus they are insecure.**

In order to get convinced of this let us consider the following attack.

1. The adversary chooses a one-block message p
2. The adversary requests t = MAC(k, p). However, by assumption, MAC = rawCBC
3. The adversary outputs t as MAC forgery of the two-block message m' = m, $(t \oplus m)$

**Proof**. For brevity let me call H the function rawCBC.

Thus $H(k, (m, (t \oplus m)) = F(F(k, m) \oplus (t \oplus m)) = F(k, t \oplus (t \oplus m)) = F(k, m) = t$. Therefore we have just shown that t is the tag of m, $(t \oplus m)$. CVD

# On the security of CBC-MAC ($\rightarrow$)

- Normally CBC-MAC does not use the last encryption, so it is insecure
- The attack
  - The adversary chooses a one-block message x
  - The adversary requests t = rawCBC(k, x)
  - The adversary outputs t' = t as MAC forgery of the two-block message x' = x, (t $\oplus$ x)

# On the security of CBC-MAC (↓)

- Proof (for brevity rawCBC = H)
  - Let t' = H(k, (x, (t $\oplus$ x)) =
    E(k, (E(k, x) $\oplus$ (t $\oplus$ x))) = E(k, t $\oplus$ (t $\oplus$ x)) = E(k, x) = t,
    where E is the cipher
    Q.E.D

MAC

# AUTHENTICATED ENCRYPTION

# Encrypt and authenticate

- Alice and Bob want to achieve confidentiality and integrity

**Alice** (k1, k2)
x
$y = E_{k1}(x)$
$t = MAC_{K2}(x)$

**Bob** (k1, k2)

---------- [y, t] --------- >

$x = D_{k1}(y)$
return V(x, k2, t)

## Problems

- The tag t might leak information about x
  - Nothing in the definition of security for a MAC implies that it hides information about x
- If the MAC is deterministic (e.g., CBC-MAC and HMAC), then it leaks whether the same message is encrypted twice

# Different approaches

- Three different approachs
  - Encrypt then MAC (EtM) 🡐 always correct
    - Ipsec
  - Encrypt and MAC (E&M) 🡐 discouraged
    - SSH
  - MAC then Encrypt (MtE)
    - TLS/SSL
- Security
  - EtM is always correct
  - MtE's correctness depends on E-MAC combinations

# Encrypt then MAC (EtM)

- Alice and Bob want to achieve confidentiality and integrity

**Alice** (k1, k2) NON E' IL MIGLIOR SCHEMA **Bob** (k1, k2)

x

$y = E_{k1}(x)$

$t = MAC_{K2}(y)$

---------- [y, t] --------- >

if (V(y, k2, t)) return ($x = D_{k1}(y)$)

else return error

E in CBC MODE

## Properties

- Given ciphertexts corresponding to (chosen) plaintexts $x_1, \dots, x_m$, it is infeasible for the attacker to generate any new valid ciphertext
- The adversary cannot trick Bob into outputting any message that was not sent by Alice

# Standards and associated data

- NIST
  - CCM: CBC-MAC then CTR mode encryption
    - 802.11i
  - GCM: CTR mode encryption then MAC
    - Very efficient
- IETF
  - EAX: CTR mode encryption ~~than~~ then OMAC
- All support authenticated encryption with associated data (AEAD)
  - E.g. the header of a packet is just authenticated

| associated data | encrypted data |
|---|---|

# Galois Counter Mode (GCM)

- GCM is an encryption mode which also computes a MAC
  - Confidentiality and authenticity
- GCM protects
  - Confidentiality of a plaintext x
  - Authenticity of plaintext x and
  - Authenticity of additional authenticated data (AAD) which is left in the clear
    - ADD might include addresses and parameters in network protocols

## Main components

- Cipher in the Counter Mode (CTR)
  - Confidentiality
  - Block size: 128 bit (e.g. AES-128)
- Galois field multiplication
  - Authentication
  - Multiplication in $GF(2^{128})$ with irreducible polynomial $P(x) = x^{128} + x^7 + x^2 + x + 1$

# Encryption

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

# Authentication

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

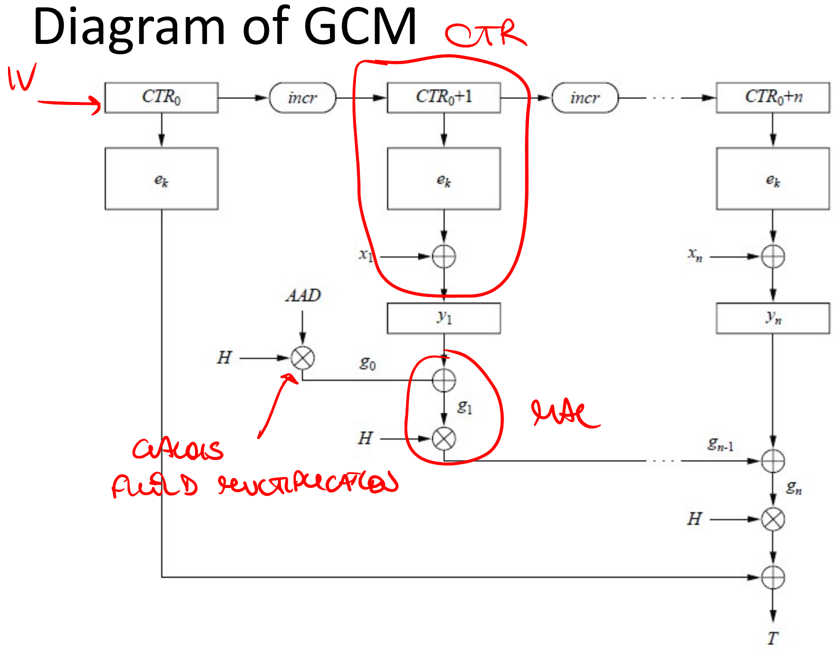
## GF($2^m$) - elements

- Elements are represented as polynomials with coefficient in GF(2)
- Polynomials have maximum degree of m – 1
- Example: GF($2^8$)
  - Element $A \in GF(2^8)$ is represented as $A = a_7 \cdot x^7 + \ldots + a_1 \cdot x + a_0$, $a_i \in GF(2)$
  - Element A can be simply stored as $(a_7, a_6, \ldots, a_1, a_0)$

# GF($2^m$) – operations

- Addition and subtraction
  - $C(x) = A(x) + B(x)$
  - Addition/subtraction modulo 2 of coefficients
- Multiplication
  - $C(x) = A(x) \times B(x)$
    - Order greater than m – 1, thus has to be *reduced* ➔
    - The operation becomes $C(x) \equiv A(x) \times B(x) \bmod P(x)$
      - P(x) is an *irreducible* polynomial
- Inversion
  - $A(x) \times A^{-1}(x) \equiv 1 \bmod P(x)$
    - P(x) is an *irreducible* polynomial

# Diagram of GCM



# The protocol

- Sender
  - Computes ($y_1$, $y_2$,…, $y_n$) and T
  - Sends [IV, ($y_1$, $y_2$,…, $y_n$), T, ADD]
- Receiver
  - Receives [IV, ($y_1$, $y_2$,…, $y_n$), T, ADD]
  - Decrypts ($y_1$, $y_2$,…, $y_n$) by applying CTR with IV
  - Computes T’ from ($y_1$, $y_2$,…, $y_n$) and ADD
  - Checks whether T == T’
    - If so, ciphertext and ADD were not manipulated in transit and only the sender could have generated the message

Message Authentication Code (MAC)

# PADDING

# MAC Padding

- Pad by zeroes ⇒ insecure
  - pad(m) and pad(m||0) have the same MAC
- Padding must be an invertible function
  - m0 ≠ m1 ⇒ pad(m0) ≠ pad(m1)
- Standard padding (ISO)
  - Append "100…00" as needed
    - Scan right to left
    - "1" determines the beginning of the pad
  - Add a dummy block if necessary
    - When the message is a multiple of the block
    - The dummy block is necessary or existential forgery arises

apr. '21 Hash functions 38

For the moment we have considered messages that are multiple block long. If a message is not multiple block long, then we need padding.

## Padding by 0es is a bad idea

- Proof
  - Let x = $x_1$, $x_2$, $x_3$ where $m_3$ is shorter than a block
  - Let's pad $m_3$ as follows $m_3||000$ (for example)
  - Let t be the tag outputted.
  - Consider know a message m' = m||0.
    - x' would be composed of three blocks $x'_1 = x_1$, $x'_2 = x_2$, and $x'_3 = x_3||0$.
    - $x'_3$ needs padding and becomes $x'_3 = x_3||0||00 = x_3||000$.
    - So x and x' after padding are equal and thus have the same tag.

Q.E.D.

apr. '21 Hash functions 39

The first idea it could come into mind is to **pad by means of zeroes**. This **is a bad idea**. The resulting system is insecure!

**Proof.** Let us suppose that m is composed of $m_1$, $m_2$, $m_3$. Let us suppose that m3 is shorter than a block and thus we need padding, e.g., $m_3||000$. Let t be the tag outputted.

Let us consider know a message m' = m||0. Then m' would be composed of three blocks $m'_1 = m_1$, $m'_2 = m_2$, and $m'_3 = m_3||0$. $m'_3$ needs padding. After padding $m'_3$ becomes $m_3||0||00 = m_3||000$. If follows that m and m' after padding are equal and thus have the same tag. **CVD**

## On dummy block

- Without dummy block, existential forgery arises
- Proof
  - Let x = x1, x2 which needs padding
  - For example x* = x1, x2||100, where x* is the padded message
  - Let x' = x1, x2||100
    - Since x' is a multiple of the block we don't pad it
  - It follows that x' = x* and thus x ad x' have the same tag

Q.E.D.



The **dummy block** is necessary for security or existential forgery arises.

**Proof**. Let us assume that we don't pad message that are multiple of the block in order to save bandwidth.
Let us consider a message m = m[0], m[1] which needs three bits padding. The message becomes m[0], m[1]||100. Let t be the corresponding tag. Let us know consider a message m' which is exactly 2-block long and structured as follows m[0], m[1]100. Since it is a multiple of the block we don't add the pad. It follows that m' has the same tag t as m. So we have built an existential forgery. **CVD**

Dummy blocks can be avoided by means of CMAC (NIST standard)

# TIMING ATTACK

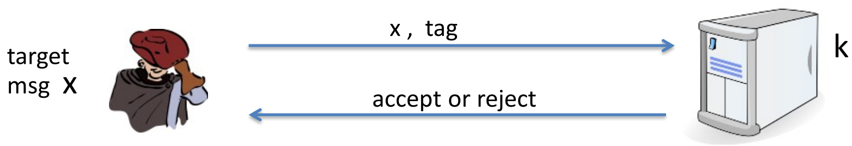
## Timing Attack

- Example: Keyczar crypto library (Python) [simplified]

```
def Verify(key, msg, tag):
        return HMAC(key, msg) == tag
```

- The problem: operator '==' is implemented as a byte-by-byte comparison
  - It returns false when first inequality found

# Timing attack




Timing attack: to compute tag for target message do:

Step 1: Query server with random tag

Step 2: Loop over all possible first bytes of tag and query server.
Stop when verification takes a little longer than in step 1

Step 3: Repeat for all tag bytes until valid tag found

| | | | | | |
|---|---|---|---|---|---|
| 3 | 47 | * | * | * | * |

# Defense #1

- Make string comparator always take same time
- Solution 1:

```
return false if  tag  has wrong length
result = 0
for x, y in zip( HMAC(key,msg) , tag):
    result |= ord(x) ^ ord(y)
return result == 0
```

- Can be difficult to ensure due to optimizing compiler

apr. '21 Hash functions 44

We compare the MACs in a way that takes a constant time.

Python bitwise operators
| OR
| AND
^ XOR

This function zip returns a list of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables. The returned list is truncated in length to the length of the shortest argument sequence. When there are multiple arguments which are all of the same length, zip() is similar to map() with an initial argument of None. With a single sequence argument, it returns a list of 1-tuples. With no arguments, it returns an empty list.

## Defense #2

- Make string comparator always take same time
- Solution 2

```
def Verify(key, msg, tag):
    mac = HMAC(key, msg)
    return HMAC(key, mac) == HMAC(key, tag)
```

- Attacker doesn’t know values being compared



Instead of comparing two macs, the received and the computed, we compare the macs of the macs.