Message Authentication Code

Computer and Network Security

Emilio Coppa

Message Authentication

Confidentiality - ciphers
Integrity - hash function
No repudiation - asymmetric/symmetric ciphers

Message Authentication (or Data Origin Authentication) is a property that a message has not been modified while in transit (data integrity) and that the receiving party can verify the source of the message.

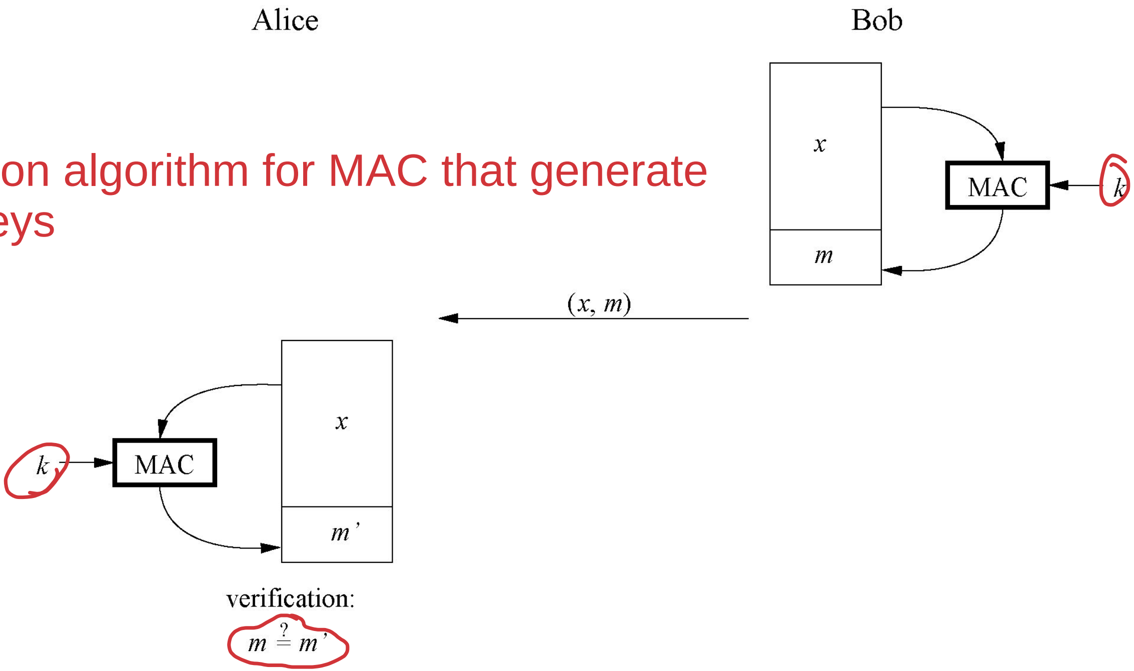
Implemented through:

- Message Authentication Code (MAC)
- Authenticated Encryption (AE)
- Digital Signature

Remark. Message authentication does not always imply non repudiation (which is offered by, e.g., Digital Signature schemes)

Message Authentication Code (MAC)

Often referred as cryptographic checksums. Based on a symmetric key for generation and verification:



Generation algorithm for MAC that generate secret keys

$$m = MAC_k(x)$$

Verification Algorithm:

Chapter 12 of Understanding Cryptography by Christof Paar and Jan Pelzl

Properties of Message Authentication Codes

cryptographic checksum (output of MAC ALG.)

- **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, i.e., any manipulations of a message during transit will be detected by the receiver.

HASH FUNCTION

- **Message authentication:** The receiving party is assured of the origin of the message.
- **No non repudiation:** MAC does not provide non repudiation.

Ensure that sent by who generated the key.

Chapter 12 of Understanding Cryptography by Christof Paar and Jan Pelzl

Security Requirement

A verification algorithm for a MAC should return “accept” or “reject” based on key k, message x, $\mathrm{MAC}_k(x)$. Message space is extremely large. $\mathrm{MAC}_k(x)$ is also called authentication tag of x. MAC function is not a 1-to-1 mapping.

An adversary should not be able to construct (forge) a new legal (valid) pair (x, $\mathrm{MAC}_k(x)$) even after seeing valid pairs from previous communication sessions.

Adversarial Model

Assumptions:

- the MAC function is known
- known valid past pairs: $(x_1, MAC_k(x_1))$, $(x_2, MAC_k(x_2))$, $(x_3, MAC_k(x_3))$, ...
- the adversary can request $MAC_k(x)$ for a given x

Goal: Find a new legal pair $(y, MAC_k(y))$ 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).

Implementing Message Authentication Codes

MAC can be implemented based on:

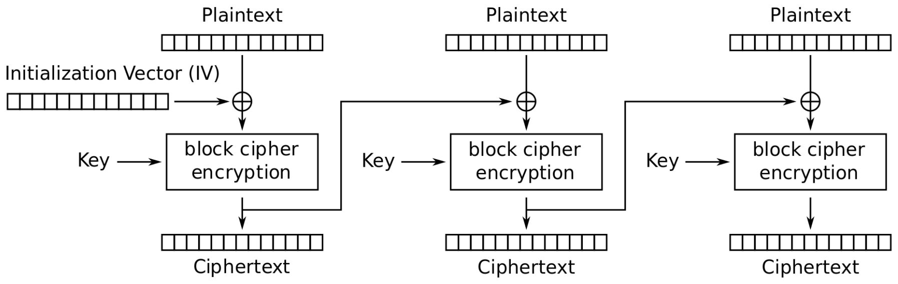
1. Block Ciphers, e.g., AES in CBC mode. It can be slow.

2. Cryptographic Hash Functions

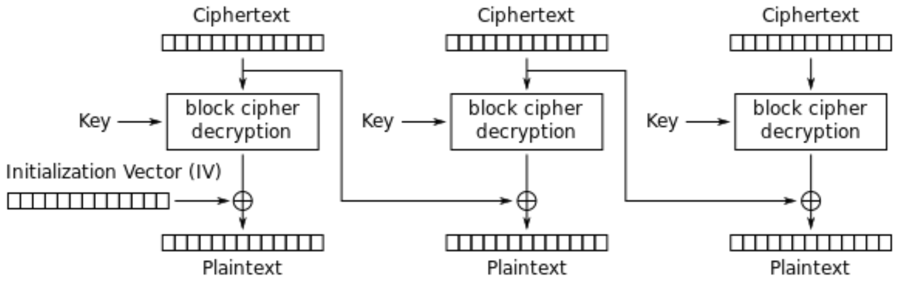
SHA-1
SHA-2
⋮

Recap: Cipher Block Chaining (CBC)

Encryption:



Decryption:



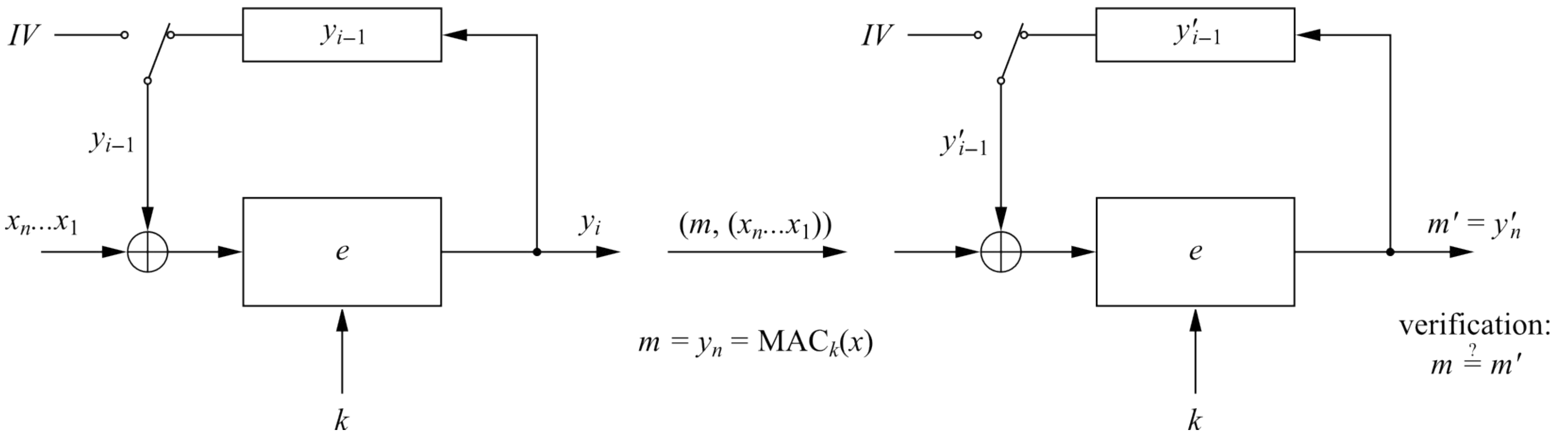
- block y_i depends on y_{i-1}
- encryption is randomized using an IV
- encryption: no parallelization
- decryption: parallelizable

- if one bit flipped in x_i then all subsequent blocks are affected
- if one bit is flipped in y_{i-1} then xi is affected an unpredictable manner, while x_i in a predictable manner. This could be exploited by an attacker. Hence use CRC/etc.

MACs from Block Ciphers

Popular: AES in CBC mode

CBC-MAC:



IV=0 in most scenarios

Chapter 12 of Understanding Cryptography by Christof Paar and Jan Pelzl

CBC-MAC

Take the message, split in blocks, compute output for the current block, goon till the end, consider last block as authentication tag

MAC Generation

- Divide the message x into blocks x_i
- Compute first iteration $y_1 = e_k(x_1 \oplus IV)$
- Compute $y_i = e_k(x_i \oplus y_{i-1})$ for the next blocks
- Final block is the MAC value: $m = MAC_k(x) = y_n$

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

Chapter 12 of Understanding Cryptography by Christof Paar and Jan Pelzl

Security of fixed-length CBC-MAC

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

Proof. Assume CBC MAC can be forged efficiently. Transform the forging algorithm into an algorithm distinguishing e_k from random function efficiently.

Security of variable-length CBC-MAC

Claim. The variable-length CBC MAC is insecure.

Proof. If attacker knows correct message-tag pairs (x, t) and (x’, t'), where x has L blocks and x’ has L’ blocks, then he can generate a third (longer) message x" whose tag will also be t':

$$x'' = x_1 \,||\ldots\,||x_L\,||\,(x'_1 \oplus t)\,||x'_2\,||..\,||x'_{L'}$$

XOR first block of x' with t and then concatenate x with this modified x'. The resulting (x”, t’) is valid pair. This works because the xor operation with t “cancel out” the contribution from x.

Security of variable-length CBC-MAC (2)

$$y''_1 = E_k(0 \oplus x_{\overset{1}{\cancel{2}}}) = y_1$$

$$y''_2 = E_k(y''_1 \oplus x_2) = E_k(y_1 \oplus x_2) = y_2$$

$$\dots$$

$$y''_L = E_k(y''_{L-1} \oplus x_L) = E_k(y_{L-1} \oplus x_L) = y_L = \boxed{t} \rightarrow \text{We know a priori.}$$

$$y''_{L+1} = E_k(y''_L \oplus (x'_1 \oplus t)) = E_k(t \oplus (x'_1 \oplus t)) = E_k((t \oplus t) \oplus x'_1) = E_k(0 \oplus x'_1) = y'_1$$

$$y''_{L+2} = E_k(y''_{L+1} \oplus x'_2) = E_k(y'_1 \oplus x'_2) = y'_2$$

$$\dots$$

$$y''_{L+L'} = E_k(y''_{L+L'-1} \oplus x'_{L'}) = E_k(y'_{L'-1} \oplus x'_{L'}) = y'_{L'} = t'$$

Improvements to CBC-MAC

Possible improvements to address issues with original CBC-MAC:

- **Input-length key separation:** generate a new key k' = E_k(l) to use in CBC-MAC where l is the message length. Hence, messages with different lengths use different keys.

→ concatenation

- **Length-prepending:** include message length in the first block, i.e., x' = l || x, where l is the message length. Problem: when processing a stream we may not know the length ad the beginning.
- **Encrypt last block.** ECBC-MAC: $E_{k2}(\text{CBC-MAC}_{k1}(x))$
  Problem: we need to use two different keys.

CBC-MAC: 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:

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

CBC-MAC: 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

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

MACs from Hash Functions

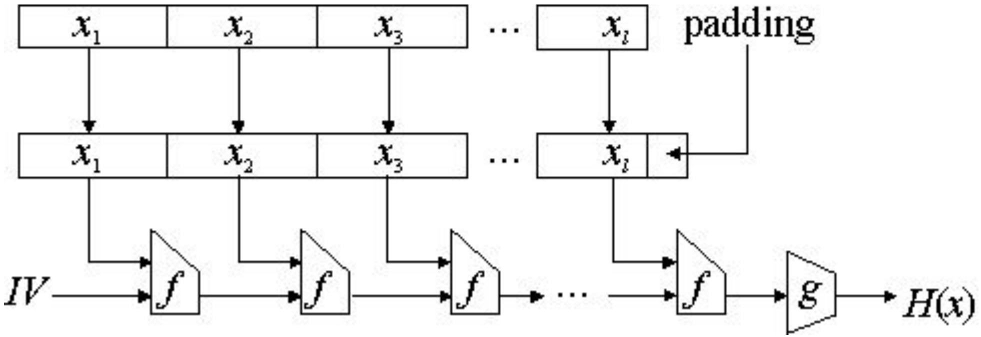
MAC is realized with cryptographic hash functions (e.g., SHA-1). In particular, when we combine the message and key and then perform hashing we called the process keyed hashing.

Idea: key is hashed together with the message, e.g.,

- secret prefix MAC: $m = MAC_k(x) = h(k \,||\, x)$
- secret suffix MAC: $m = MAC_k(x) = h(x \,||\, k)$

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:



- IV is usually constant
- if hash function f is collision resistant, then its MDC extension is collision resistant

Secret prefix MAC: attack

$$m_x = h(k \underset{\text{append.}}{||} x)$$

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

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

Using IV=m_x during Merkle-Damgård construction.

Secret suffix MAC: attack

After the whole message

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

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

Notice that this attack is interesting even if it requires to find a collision as it lower the attack complexity with respect to a brute force attack wrt the key space:

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

Sooner or later we'll get a collision(birthday paradox)

CONCL: HAVE AN ALG. HARDER TO BROKE THAN W/ THE BRUTE FORCE.

HMAC

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

message will be hashed 2 times, so more secure than PREFIX MAC.

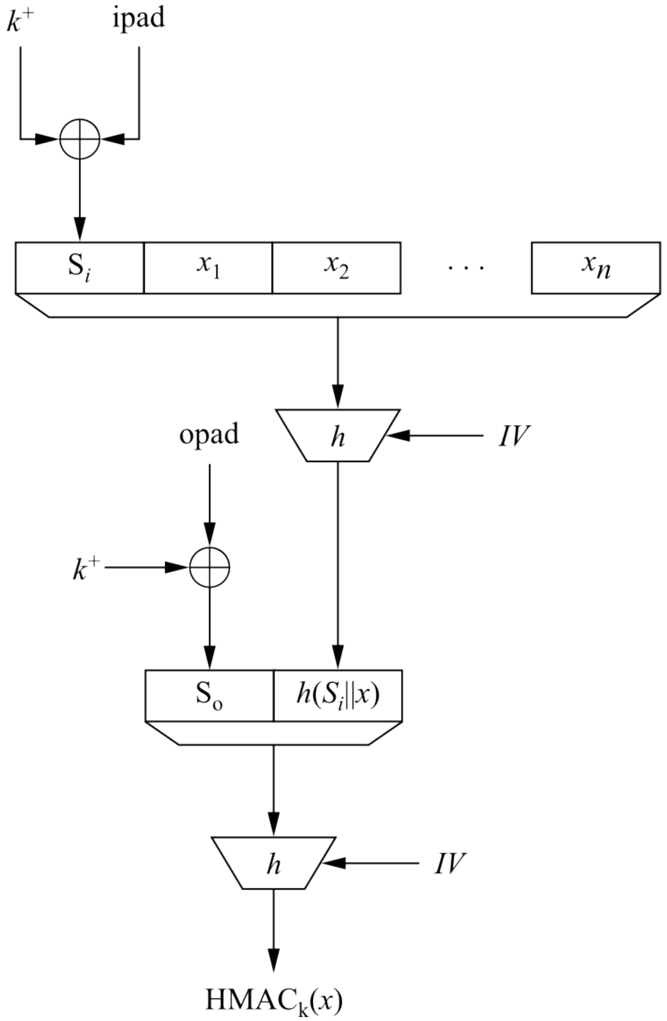
HMAC is an implementation of this idea with some additional details. It is extremely popular and used in several protocols.

HMAC is provable secure which means (informally speaking) that is secure if the hash function is secure.

HMAC (2)

- Proposed by Mihir Bellare, Ran Canetti and Hugo Krawczyk in 1996

- Scheme consists of an inner and outer hash
  - k^+ is expanded key k → No need to know.
  - expanded key k^+ is XORed with the inner pad
  - ipad = 00110110,00110110, . . .,00110110
  - opad = 01011100,01011100, . . .,01011100
  - $\mathrm{HMAC}_k(x) = h(\ \underline{(k^+ \oplus \mathrm{opad})}\ ||\ h(\ \underline{(k^+ \oplus \mathrm{ipad})}\ ||\ x\ )\ )$



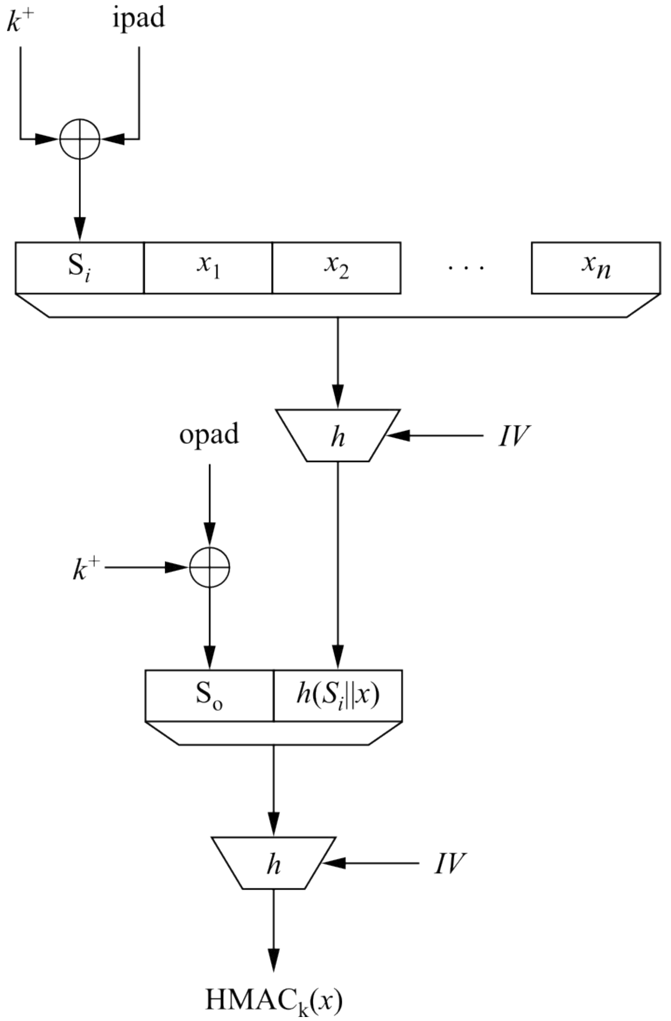
Chapter 12 of *Understanding Cryptography* by Christof Paar and Jan Pelzl

HMAC (3)

Notice that HMAC does not mean that we have to perform 2n hashes instead of n hashes as in simple keyed hashing.

Indeed, the inner hash is computer n times, but the outer hash only once! The message x is only processed by the inner hash.

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.



Chapter 12 of *Understanding Cryptography* by Christof Paar and Jan Pelzl

HMAC vs Birthday Paradox

- an attacker cannot generate MAC by himself with HMAC because he does not know the key k (he can still brute force k but then the number of attempts depends on the key space)

- to exploit a collision he needs to find it: he need to intercept (as it cannot generate them) on average, when output space is 160 bit, 2^{80} 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.

Authenticated Encryption (AE)

(DATA INTEGRITY, CONF.

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

Payload should be confidential
Parameters public: addresses, id of receiver etc.

Additionally, authenticated encryption can provide security against chosen ciphertext attack. In these attacks, an adversary attempts to gain an advantage against a cryptosystem (e.g., information about the secret decryption key) by submitting carefully chosen ciphertexts to some "decryption oracle" and analyzing the decrypted results. Authenticated encryption schemes can recognize improperly-constructed ciphertexts and refuse to decrypt them.

AE API

Encryption

- input: plaintext, key, and optionally a header in plaintext that will not be encrypted, but will be covered by authenticity protection.
- output: ciphertext and authentication tag (message authentication code).

Decryption

(opposite)

- input: ciphertext, key, authentication tag, and optionally a header (if used during the encryption).
- output: plaintext, or an error if the authentication tag does not match the supplied ciphertext or header.

The header part is intended to provide authenticity and integrity protection for networking or storage metadata for which confidentiality is unnecessary, but authenticity is desired.

AE approaches

1. Encrypt-then-MAC (EtM)

2. Encrypt-and-MAC (E&M)

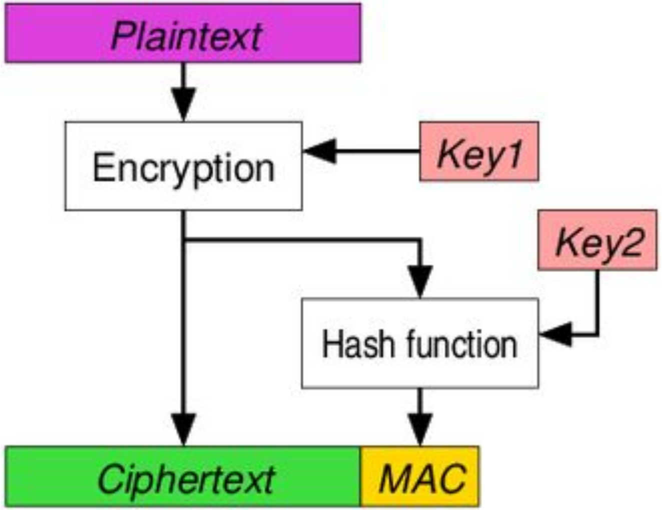
3. MAC-then-Encrypt (MtE)

Encrypt-then-MAC (EtM)

The plaintext is first encrypted, then a MAC is produced based on the resulting ciphertext. The ciphertext and its MAC are sent together.

Used in, e.g., IPsec. This is the only approach which can reach the highest definition of security in AE accorting to ISO/IEC 19772:2009, but this can only be achieved when the MAC used is "strongly unforgeable".

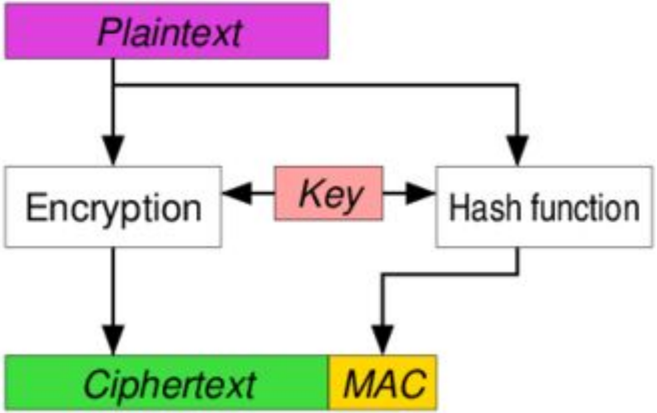
Various EtM ciphersuites exist for SSHv2 and TLS.



Encrypt-and-MAC (E&M)

A MAC is produced based on the plaintext, and the plaintext is encrypted without the MAC. The plaintext's MAC and the ciphertext are sent together.

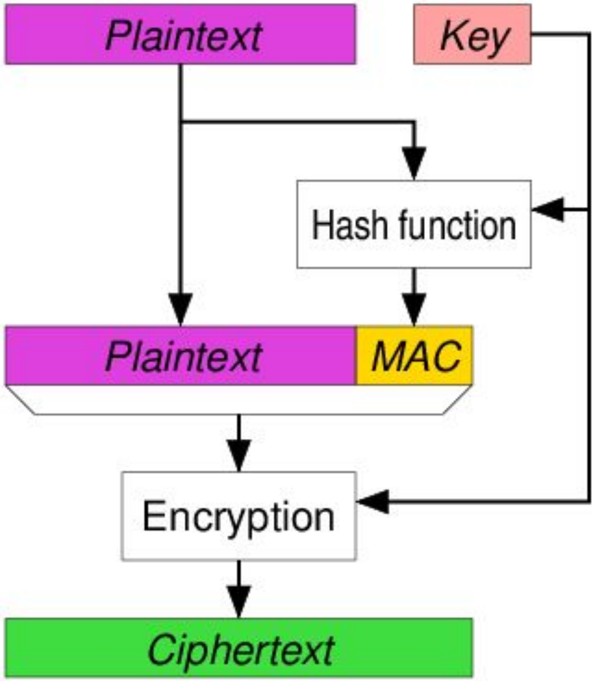
Used in, e.g., SSH. Even though the E&M approach has not been proved to be strongly unforgeable in itself, it is possible to apply some minor modifications to SSH to make it strongly
unforgeable despite the approach.



MAC-then-Encrypt (MtE)

A MAC is produced based on the plaintext, then the plaintext and MAC are together encrypted to produce a ciphertext based on both. The ciphertext (containing an encrypted MAC) is sent.

Used in, e.g., 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.



Authenticated encryption with associated data (AEAD)

AEAD is a 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

Galois Counter Mode (GCM)

Extension of ctr

GCM is an operation mode for a block cipher that provides both encryption and message authentication. In particular, it can guarantee integrity of the encrypted message and also of another piece of information, called “authenticated data”, that we send unencrypted (e.g., parameters of the protocol).

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}).

Galois Counter Mode (2)

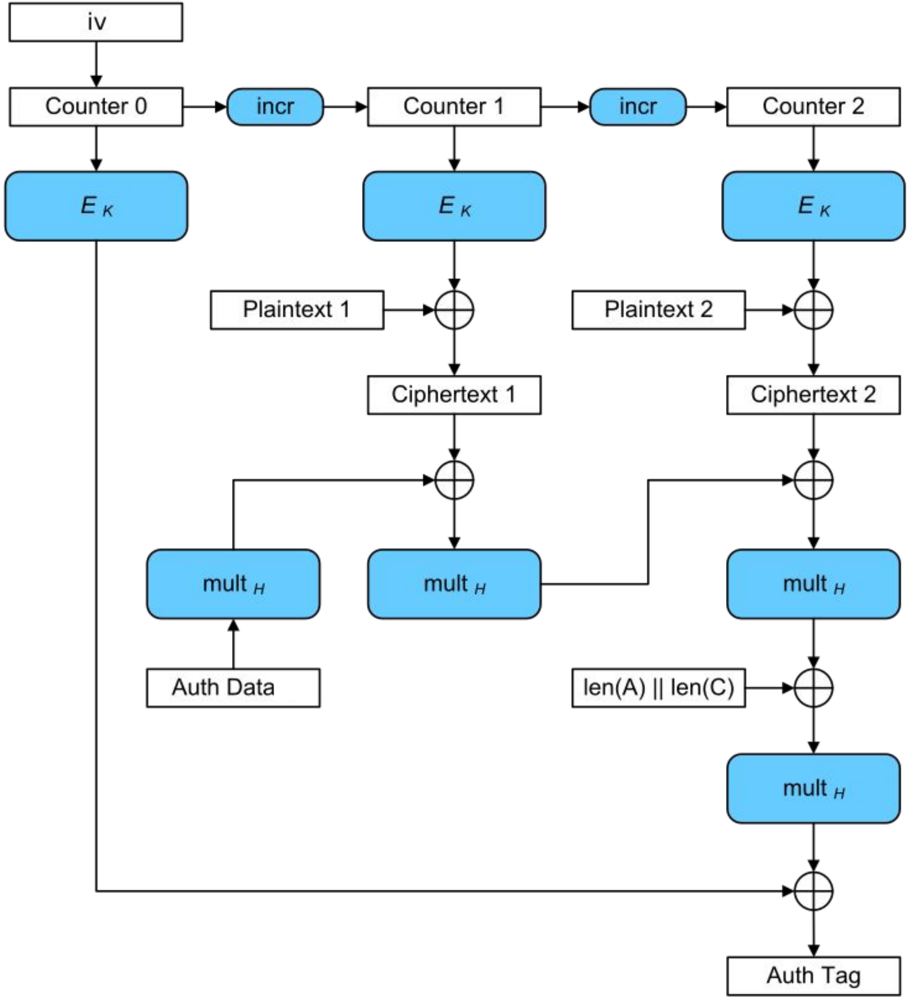
Idea:

- perform encryption similarly to CTR mode (use a different IV for each message!)

- Auth Data is a piece of (unencrypted) information that we want to authenticate

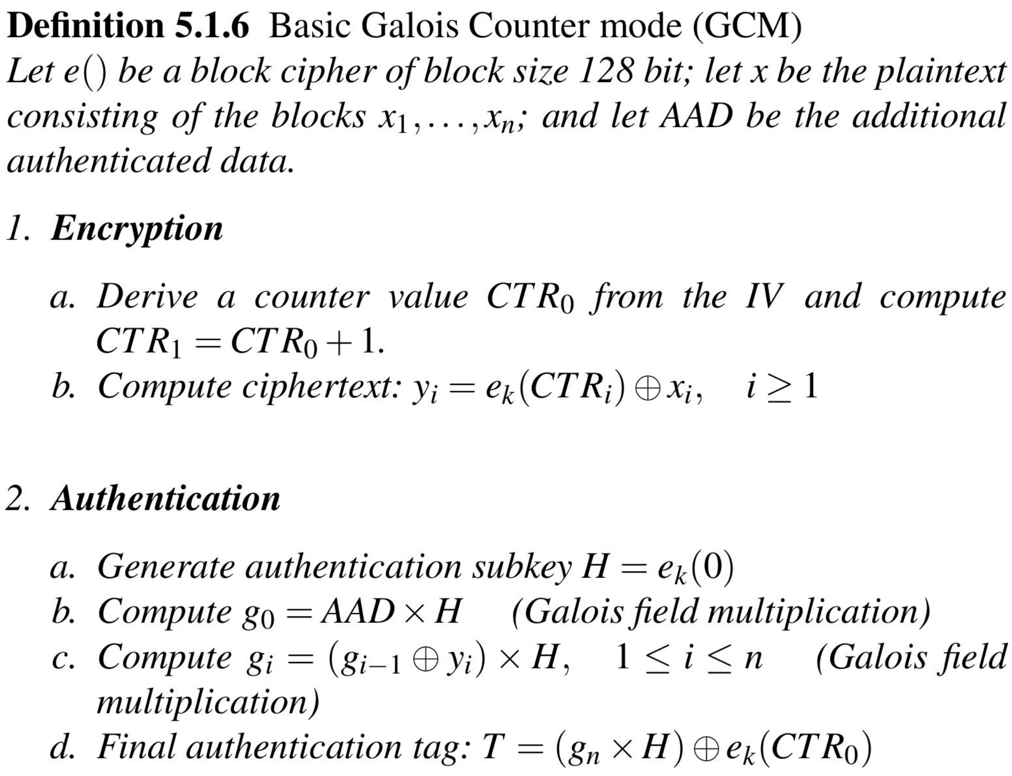
- perform authentication as a multiplication by $H = E_K(0)$ in $GF(2^{128})$

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



Galois Counter Mode (3)

PSEUDOCODE.



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: $T = (g_n \times H) \oplus e_k(CTR_0)$*

Chapter 5 of Understanding Cryptography by Christof Paar and Jan Pelzl

Credits

These slides are based on material from:

- Slides of Prof. D’Amore from CNS 2019-2020

- Christof Paar and Jan Pelzl. Understanding Cryptography: A Textbook for Students and Practitioners. Springer. http://www.crypto-textbook.com/

- Wikipedia (english version)