Message Authentication

Dr. Demetrios Glinos
University of Central Florida

CIS3360 - Security in Computing

Readings

- "Computer Security: Principles and Practice", 3rd Edition, by William Stallings and Lawrie Brown
    - Section 21.2
    - Appendix E

Outline

- Message Authentication Requirements
- Message Authentication Model
- Using Hash Functions for MACs
    - HMAC Architecture
    - Security of HMAC
    - Birthday Paradox and Attack
    - Is HMAC-MD5 secure?
    - Combining MAC with Encryption
- Using Block Ciphers for MACs
    - Cipher-based MAC (CMAC)
    - CTR with Cipher Block Chaining MAC (CCM)

Message Authentication Requirements

- Message authentication must provide these assurances
  1. the message has not been altered
  2. the message came from the alleged sender

- Consider this scenario:
  - Alice wishes to send an authenticated message to Bob
  - Alice computes the SHA-512 hash code for the email and sends it to Bob along with the original message
  - Eve (our attacker) intercepts the message, changes it, computes a fresh SHA-512 value, and forwards it to Bob instead of Alice's original transmission
  - Bob receives the message and hash value that Eve sent, computes his own SHA-512 value for the message, and believes he has authenticated a message from Alice

Q: How can we fix this (and provide both assurances above)?

A:

Message Authentication Requirements

- Message authentication must provide these assurances
  1. the message has not been altered
  2. the message came from the alleged sender

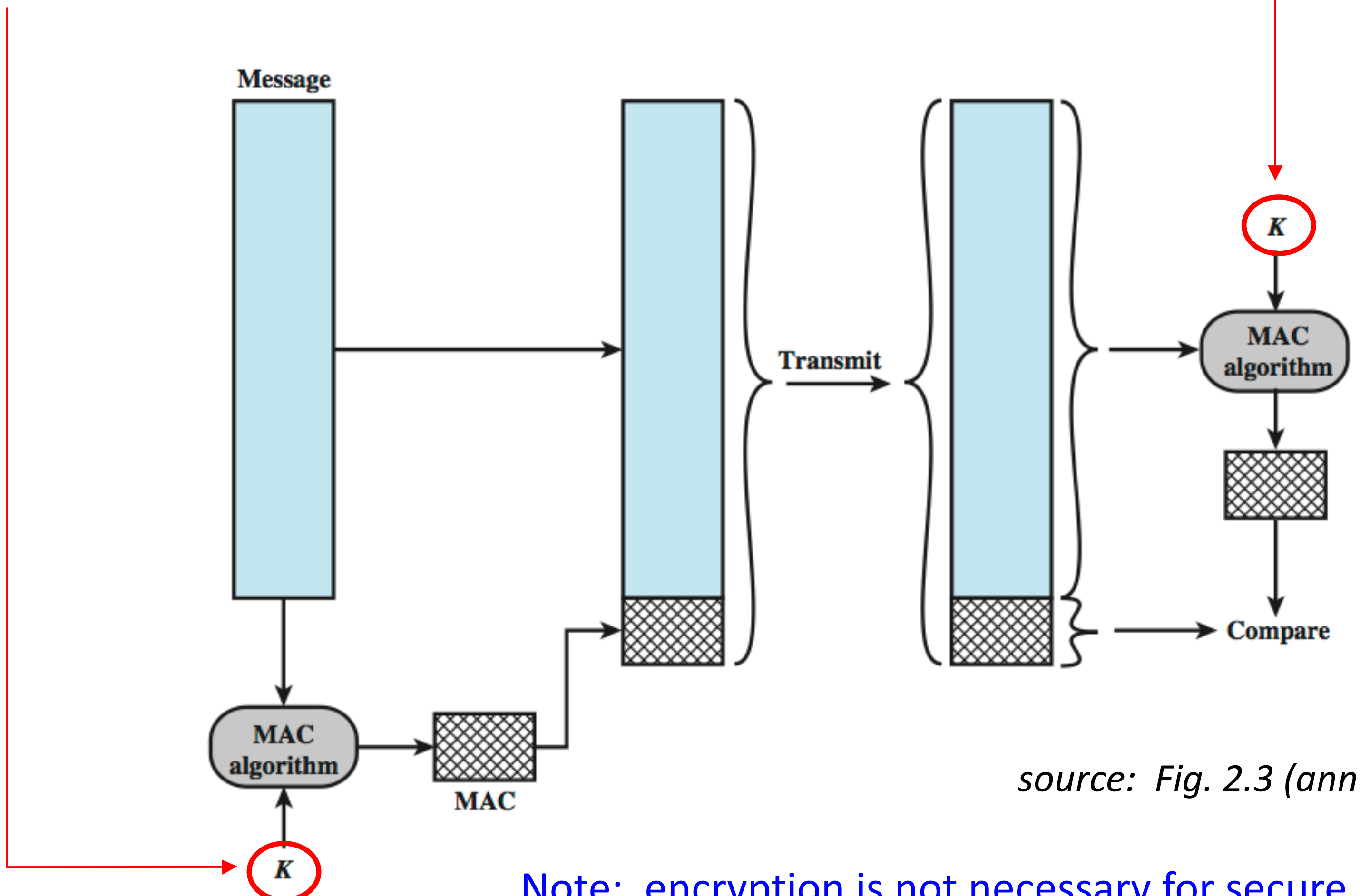
- Consider this scenario:
  - Alice wishes to send an authenticated message to Bob
  - Alice computes the SHA-512 hash code for the email and sends it to Bob along with the original message
  - Eve (our attacker) intercepts the message, changes it, computes a fresh SHA-512 value, and forwards it to Bob instead of Alice's original transmission
  - Bob receives the message and hash value that Eve sent, computes his own SHA-512 value for the message, and believes and believes he has authenticated a message from Alice

Q: How can we fix this (and provide both assurances above)?

A: Use a MAC algorithm that requires a secret key known only to Alice and Bob

Message Authentication Model

Secret key K is known only to sender and recipient



source: Fig. 2.3 (annotated)

Note: encryption is not necessary for secure authentication, although it may be desirable

Using Hash Functions for MACs

- Why use hash functions? (After all, one could also use encryption)
  - Hash algorithms run faster than encryption (in software)
  - Library code for hash functions is widely available
  - Can easily replace one hash function with another in a modular MAC

- Design problem: How to use keys with secure hash functions

- Solution: **HMAC (Hash-based Message Authentication)**
  - invented by Bellare, Canetti, and Krawczyk (1996)
  - can use any existing secure hash algorithm as a replaceable module
  - block size of HMAC is block size of the embedded hash function
  - security of HMAC is provably related to security of embedded hash function
  - Issued as RFC 2104: mandatory for IPSec, also used in SSL/TLS

HMAC Architecture

K^+ = shared secret key, padded to match the hash algorithm block size (or, if larger, then its hash value)

ipad = 00110110 (36_{16}) repeated b/8 times

opad = 01011100 ($5C_{16}$) repeated b/8 times

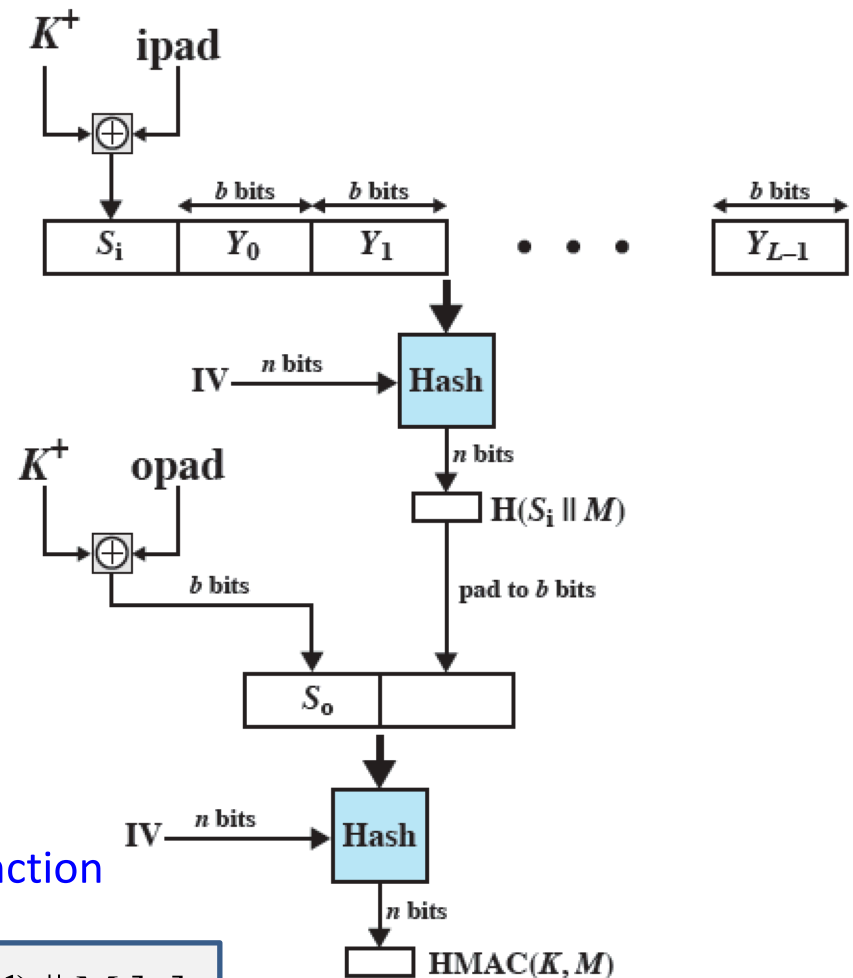
S_i = K^+ XOR'd with ipad

S_o = K^+ XOR'd with opad

Y_i = ith block of Message M, $0 \leq i \leq (L-1)$

IV = fixed initialization vector used by hash function

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



source: *Fig. 22.4*

Security of HMAC

- The quantitative measure of security that is used
  - the probability of successful forgery with a given amount of time spent by the forger and a given number of message-MAC pairs created with the same key
- HMAC security has been proved to be equivalent to either

  1. Attacker is able to compute an output of the hash function even with an IV that is random, secret, and unknown to the attacker

  2. The attacker finds collisions in the hash function even when the IV is random and secret

- Analysis shows:
  - Brute force effort is $O(b^n)$, where n is the size of the IV
  - "Birthday attack" reduces this to O(b n/2)

➔ This is why the security of a hash function is considered roughly half the digest size, e.g. 80 for SHA-1, 256 for SHA-512, etc.

Birthday Paradox (1)

- Surprisingly, a brute-force attack against a cryptographic hash function **can** succeed if enough attempts are made to find a collision
  - The attacker does not need to try (on average) half of the possible choices, as we would expect for cracking a password, for example
  - The number of choices to try is much less
  - Reason is the "**birthday paradox**" (aka "birthday problem")
  - see http://betterexplained.com/articles/understanding-the-birthday-paradox/

- The paradox in a nutshell:
  - Suppose we have a room with 23 strangers in it
  - What is the likelihood that 2 of them have the same birthday?
  - Intuitive answer is that this is not likely, since there are 365 days in a year and we have only 23 people (~6.3% of them)

Birthday Paradox (2)

- Birthday paradox analysis:
  - Consider a room with just 23 people in it.
  - The probability that Person B does ***not*** have same birthday as person A is 364/365
  - Now, there are 253 such pairs of comparisons possible in our group of 23 since the first person must check against 22 others, second against 21 others, etc.
  - So, number of pairs is 22 + 21 + 20 + … + 1 = 253
  - Therefore, the probability that ***no pair is a match*** is $(364/365)^{253} = .9972^{253} = .4995$
  - Therefore, the probability that 2 people ***have*** the same birthday is 1 - .4995 = .5005

→ **Thus, we have shown that the chances are greater than 50% that 2 people out of a group of 23 people have the same birthday!**

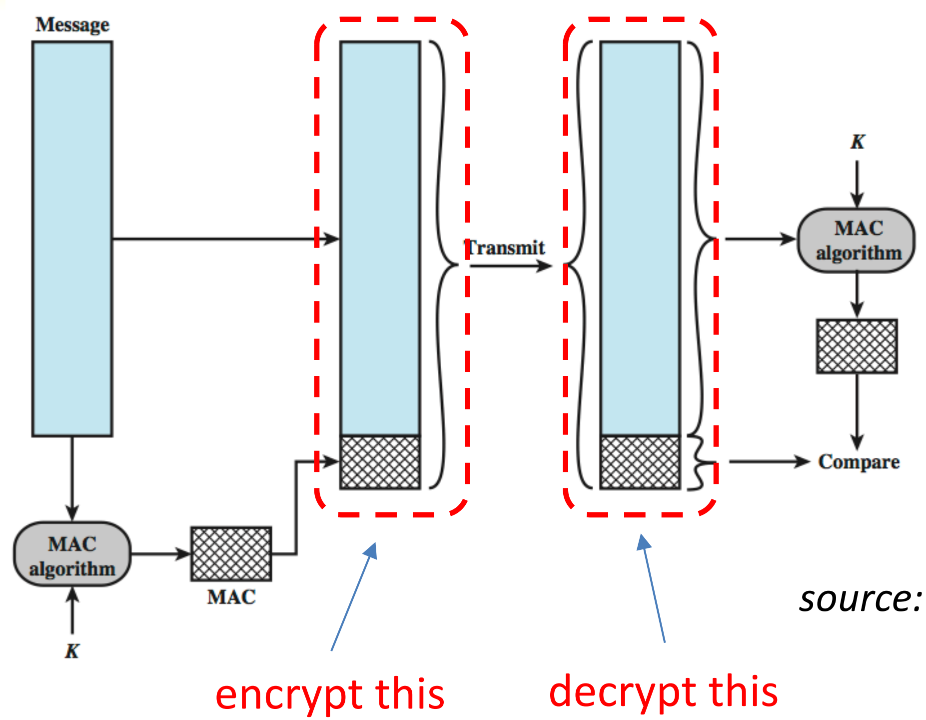
Birthday Attack

- How the birthday paradox is used as an attack against a cryptographic hash function:
  - Suppose the cryptographic hash function produces a number with **b bits**
  - Then, the number of possible hash values is $\mathbf{2^b}$; let's call this value **m**
  - Eve, our attacker, generates a **large number** of random messages and computes their hash values.
  - Because of the birthday paradox, it turns out that Eve needs to generate only about $\mathbf{2^{b/2}}$ random messages to find two messages with the same hash value
    - For $b = 16$, $2^b = 2^{16} = 65{,}536$ but Eve only needs to check $2^{b/2} = 2^8 = 256$
  - For this reason, the security of a cryptographic hash function is generally considered to be half the size of its output
    - **Example:** SHA-256, which has 256-bit output, is considered to have only 128-bit security, etc.

Is HMAC-MD5 secure?

- MD5 hash algorithm
  - widely used for integrity checking of software downloads
  - similar, but slightly weaker than SHA-1, invented by Ron Rivest in 1992
  - uses a 512-bit block and produces a 128-bit digest
  - now considered to be "cryptographically broken" when used directly
    - most US government applications now require SHA-2 family
- Direct attack
  - since digest is only 128 bits, need only 2^{64} effort (per birthday attack) to find a collision
  - feasible to do with current technology since attacker can choose any set of messages and work offline to find collisions
- Attack on HMAC-MD5
  - still not feasible, since attacker cannot work offline (doesn't know secret key)
  - attacker must observe 2^{64} blocks computed with same key
    - on 1 Gbps link, would need to observe continuous stream of messages with no change in key for about 150,000 years to get enough data

Combining MAC with Encryption



source: Fig. 2.3 (annotated)

- Note:
  - In the basic MAC model, both the message and the MAC are sent in the clear
  - But sender can encrypt both before transmission; the recipient first decrypts, then computes and compares MACs

➔Combining encryption with MAC assures confidentiality as well as integrity

Using Block Ciphers for MACs

- We consider two methods for using block ciphers for message authentication
  - **Cipher-based Message Authentication Code (CMAC)**
    - specified in NIST Special Pub. 800-38B
    - considered a mode of operation
    - intended for use with AES and triple DES
    - uses a single key for both the encryption and MAC algorithms
  - **Counter with Cipher Block Chaining MAC (CCM)**
    - specified in NIST Special Pub. 800-38C
    - referred to as an "**authenticated encryption**" mode of operation
      - achieves both authentication and encryption
    - uses AES, CTR, and CMAC
    - uses a single key for both the encryption and MAC algorithms

CMAC Operation

- Given
  - Plaintext message divided into blocks $M_1, M_2, \ldots, M_n$
    - pad (if needed) with a 1 bit followed by as many 0 bits as needed
  - Encryption key K

- Operation
  - Generate 2 constants (subkeys):

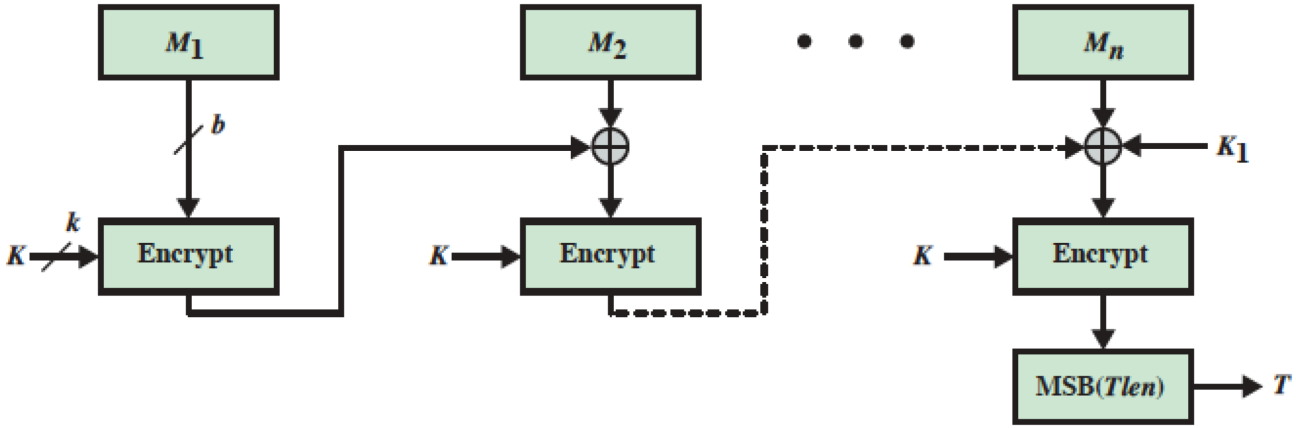
    K_1 = E(block of zeroes), then left shift 1 bit and XOR with a constant

    K_2 = E(K_1), then left shift 1 bit and XOR with a constant
  - Thereafter

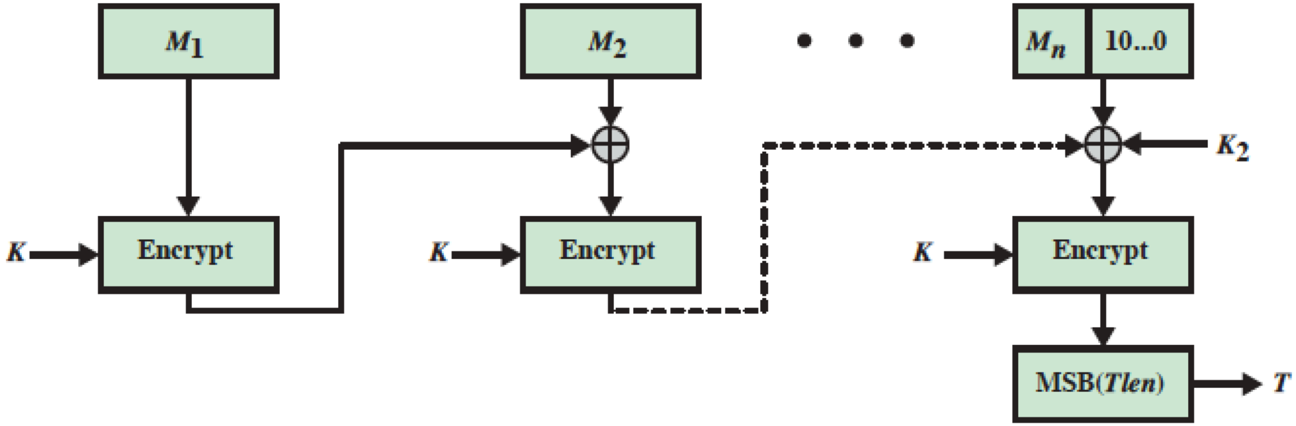
$$C_1 = E(K, M_1)$$
$$C_2 = E(K, [M_2 \oplus C_1])$$
$$C_3 = E(K, [M_3 \oplus C_2])$$
$$\bullet\bullet\bullet$$
$$C_n = E(K, [M_n \oplus C_{n-1} \oplus (K_2 \text{ if } M_n \text{ padded, else } K_1)])$$
$$T = MSB_{Tlen}(C_n)$$

where
Tlen = desired digest size
$MSB_s(X)$ = the leftmost s bits of X

CMAC Diagram



(a) Message length is integer multiple of block size



(b) Message length is not integer multiple of block size

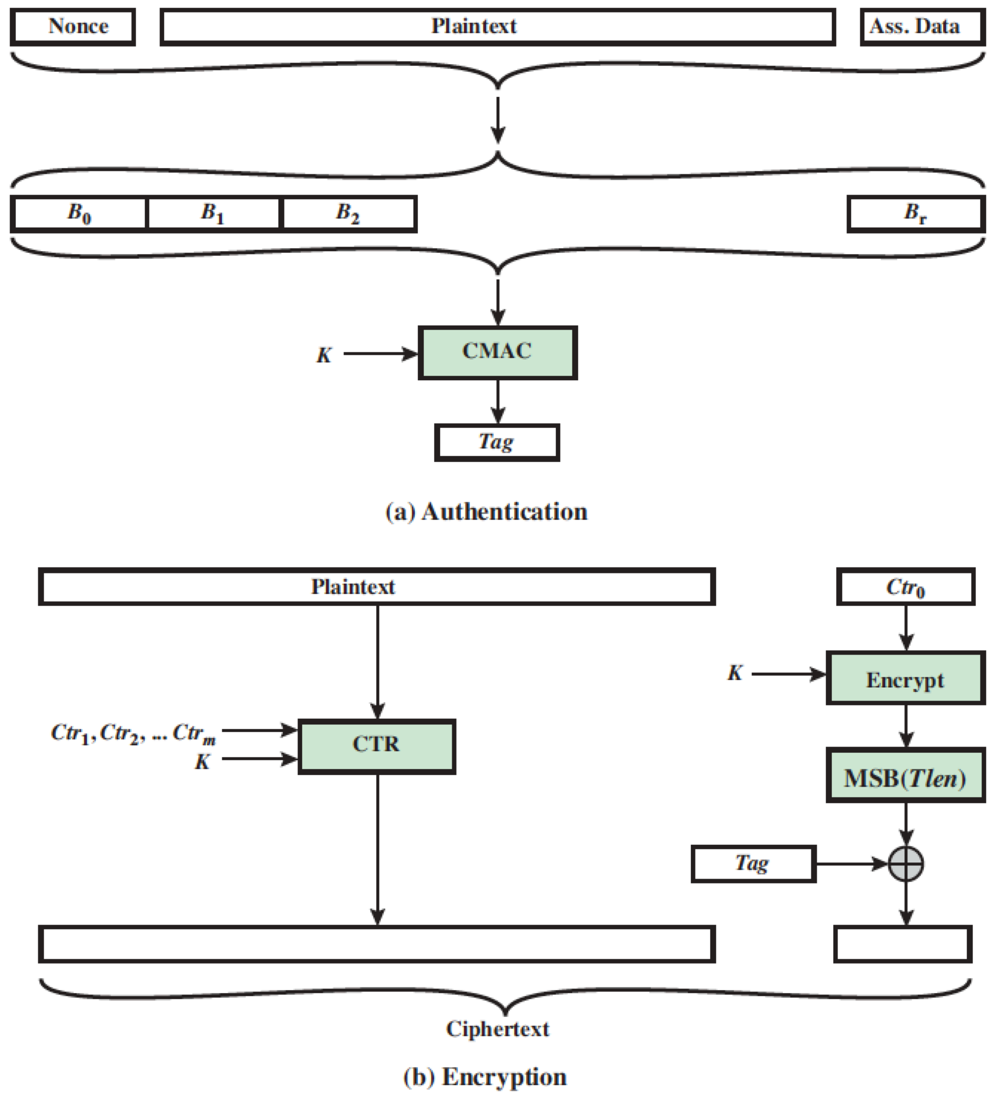
source: Fig. E.1

CCM Operation

- Given
  - Plaintext message divided into blocks $B_0, B_1, \ldots, B_r$
    - pad (if needed) with a 1 bit followed by as many 0 bits as needed
  - Associated data A that will be authenticated, but not encrypted
    - e.g., protocol header
  - A "**nonce**" N
    - a unique value assigned to every transmission, to prevent replay attacks
  - Encryption key K

- Operation
  - use CMAC with key K to generate the MAC (also called "Tag" in this context)
  - generate a sequence of counters $CTR_0, CTR_1, \ldots,$ in the usual way (with seed S)
  - encrypt counters $CTR_1, \ldots, CTR_m$ and XOR with plaintext blocks to build up the ciphertext
  - encrypt CTR_0, XOR its most significant bits with Tag, and append to encryption of plaintext
  - transmit the combined CTR-encrypted ciphertext + MAC

CCM Diagram



source: Fig. E.2