Cryptography and Network Security Chapter 12

Fifth Edition

by William Stallings

Lecture slides by Lawrie Brown

(with edits by RHB)

Chapter 12 – Message Authentication Codes

- *At cats' green on the Sunday he took the message from the inside of the pillar and added Peter Moran's name to the two names already printed there in the "Brontosaur" code. The message now read: "Leviathan to Dragon: Martin Hillman, Trevor Allan, Peter Moran: observe and tail." What was the good of it John hardly knew. He felt better, he felt that at last he had made an attack on Peter Moran instead of waiting passively and effecting no retaliation. Besides, what was the use of being in possession of the key to the codes if he never took advantage of it?*
- **—*Talking to Strange Men,* Ruth Rendell**

Outline

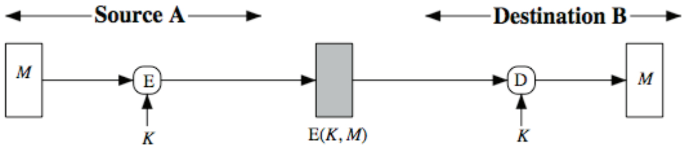
- we will consider:
  - message authentication requirements
  - message authentication using encryption
  - MACs
  - HMAC authentication using a hash function
  - DAA
  - CMAC authentication using a block cipher and CCM
  - GCM authentication using a block cipher
  - PRNG using Hash Functions and MACs

Message Authentication

- message authentication is concerned with:
  - protecting the integrity of a message
  - validating identity of originator
  - non-repudiation of origin (dispute resolution)
- three alternative approaches used:
  - hash functions (see Ch 11)
  - message encryption
  - message authentication codes (MACs)

Symmetric Message Encryption

- encryption can also provides authentication
- if symmetric encryption is used then:
  - receiver knows sender must have created it
  - since only sender and receiver know key used
  - know content cannot have been altered …
  - *if message has suitable structure*, redundancy or a checksum can detect any changes …



(a) Symmetric encryption: confidentiality and authentication

Message Authentication Code (MAC)

- generated by an algorithm that creates a small fixed-sized block
  - depending on both message and some **key**
  - like encryption, but **need not be reversible**
- appended to message as a **digest** / **tag**
- receiver performs same computation on message and checks it matches the MAC
- provides assurance that message is unaltered and comes from sender
- esp. useful if message can be **any bitpattern**

Message Authentication Codes

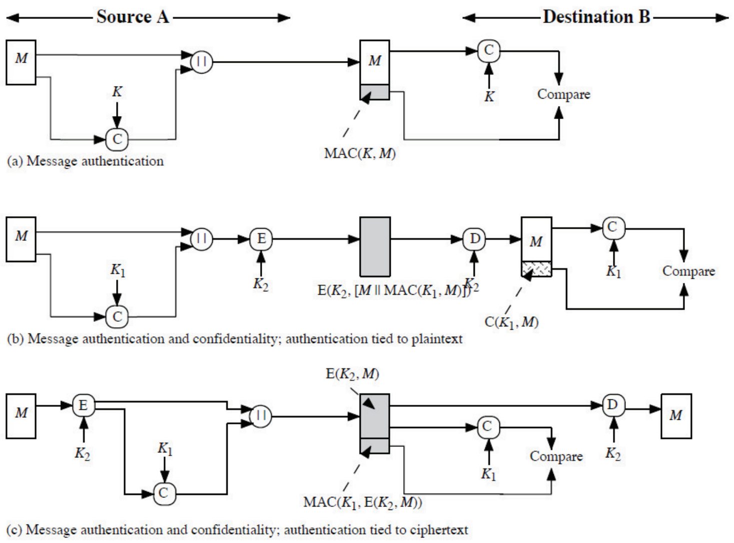
- a MAC is a cryptographic checksum

  $\text{MAC} = \text{C}_\text{K}(\text{M})$
  - condenses a variable-length message `M`
  - using a secret key `K`
  - to a fixed-sized authenticator
- is a many-to-one function
  - many messages will have same MAC
  - but finding these needs to be very difficult

Message Authentication Codes

- as shown, the MAC provides authentication
- can also use encryption for secrecy
  - generally use separate keys for each
  - can compute MAC either before or after encryption
  - is generally regarded as better done before
- why use a MAC?
  - sometimes only authentication is needed
  - sometimes need authentication to persist longer than the encryption (eg. archival use)
  - protection for arbitrary bitpattern plaintexts
- a MAC is not a digital signature (repudiation)

Uses of MAC



(a) Message authentication

(b) Message authentication and confidentiality; authentication tied to plaintext

(c) Message authentication and confidentiality; authentication tied to ciphertext

Authenticated Encryption

- simultaneously protect confidentiality and authenticity of communications
  - often required but usually separate
- approaches
  - Hash-then-encrypt: E(K, (M || H(M))
  - MAC-then-encrypt: E(K2, (M || MAC(K1, M))
  - Encrypt-then-MAC: (C=E(K2, M), T=MAC(K1, C))
  - Encrypt-and-MAC: (C=E(K2, M), T=MAC(K1, M))
- decryption / verification straightforward
- vulnerabilities with all, without good design

Requirements for MACs

- taking into account the types of attacks
- need the MAC to satisfy the following:
  1. knowing a message and MAC, is infeasible to find another message with same MAC
  2. MACs should be uniformly distributed
  3. MAC should depend equally on all bits of the message
- similar to hash function properties (preimage resistance, second preimage resistance, collision resistance)

Security of MACs

- like block ciphers we have:
- **brute-force** attacks exploiting
  - strong collision resistance — a hash has strength $2^{m/2}$
    - 128-bit hash looks vulnerable, 160-bits better
  - MACs with known message-MAC pairs
    - can either attack keyspace (cf key search) or MAC
    - at least 128-bit MAC is needed for security

Security of MACs

- like block ciphers we have:
- **cryptanalytic attacks** exploit structure
  - like block ciphers want brute-force attacks to be the best alternative
- greater variety of MACs so harder to generalize about cryptanalysis

Keyed Hash Functions as MACs

- want a MAC based on a hash function
  - because hash functions are generally faster
  - crypto hash function code is widely available
- hash includes a key along with message
- original proposal:

  `KeyedHash = Hash(Key|Message)`
  - some weaknesses were found with this
- eventually led to development of HMAC

HMAC Design Objectives

- use, without modifications, hash functions
- allow for easy replacement of embedded hash function
- preserve original performance of hash function without significant degradation
- use and handle keys in a simple way.
- have well understood cryptographic analysis of authentication mechanism strength

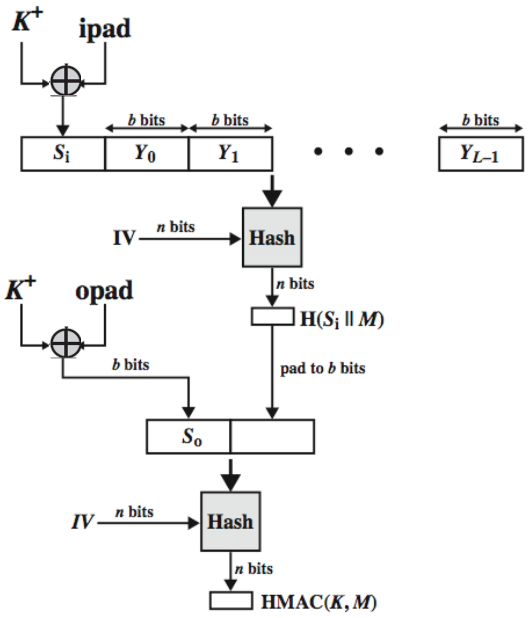
HMAC

- specified as Internet standard RFC2104
- uses hash function on the message:

  $\mathrm{HMAC}_K(M) = \mathrm{Hash}[(K^+ \text{ XOR opad}) \,||\, \mathrm{Hash}[(K^+ \text{ XOR ipad}) \,||\, M)]]$
  - where K^+ is the key, **zero**-padded out to size
  - `opad`, `ipad` are specified padding constants (50% bits in common), **repeated** to pad out to size
- overhead is just 3 more hash calculations than the message needs alone
- any hash function can be used
  - eg. MD5, SHA-1, RIPEMD-160, Whirlpool, etc.

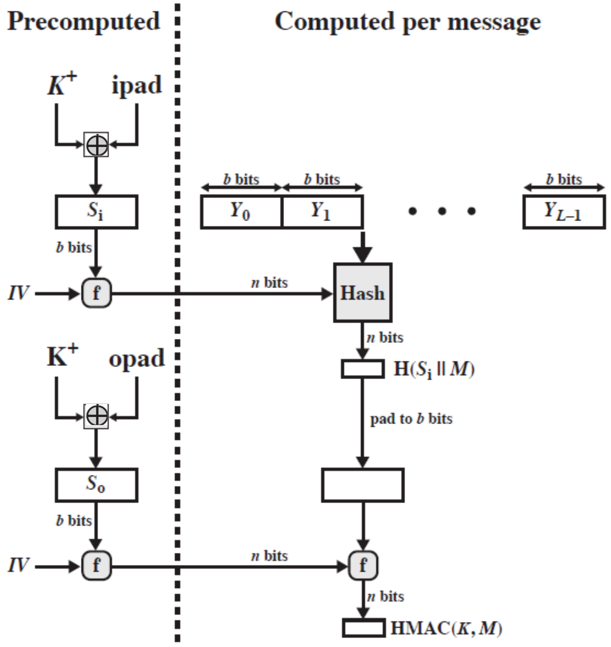
HMAC Overview

ipad = (00110110)*



opad = (01011100)*

HMAC Precomputation



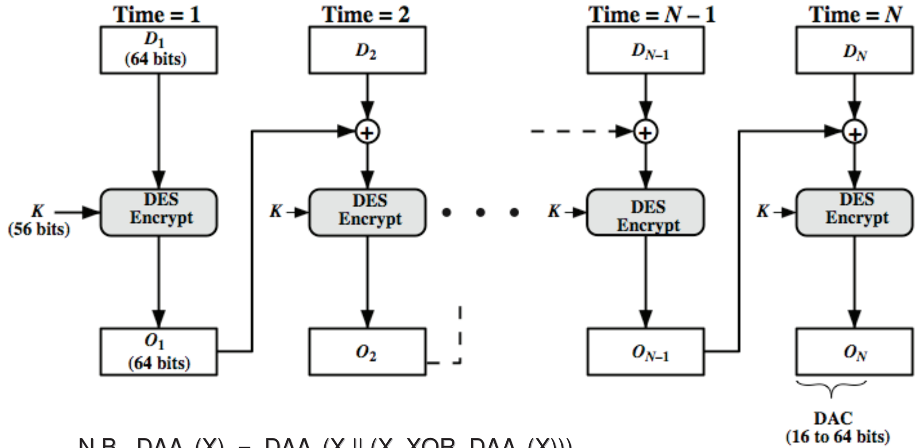
HMAC Security

- proved 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)
- choose hash function used based on speed versus security constraints

Using Symmetric Ciphers for MACs

- 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
  - using IV=0 and zero-pad of final block
  - encrypt message using DES in CBC mode
  - and send just the final block as the MAC
    - or the leftmost M bits (16≤M≤64) of final block
- but final MAC is now too small (64 bits) for security

Data Authentication Algorithm

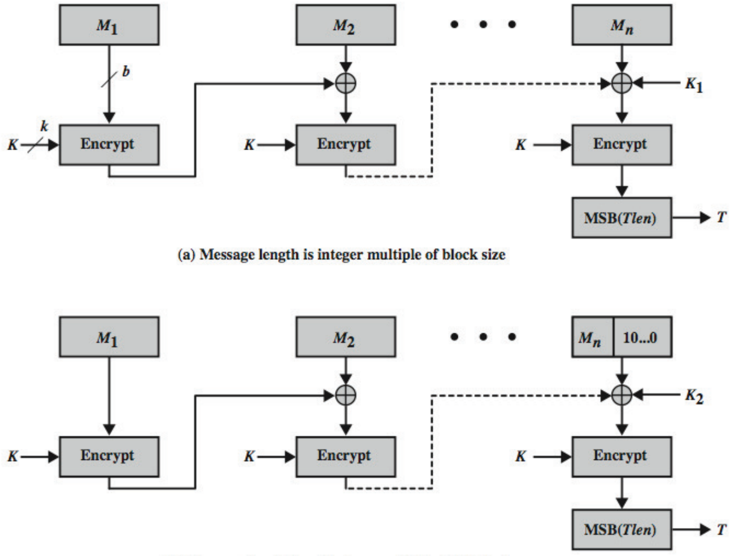


N.B. $\text{DAA}_K(X) = \text{DAA}_K(X \,||\, (X \text{ XOR } \text{DAA}_K(X)))$

CMAC

- previously saw the DAA (CBC-MAC)
- widely used in government and industry
- but has message size limitation
- can overcome using 2 keys and padding
- thus forming the Cipher-based Message Authentication Code (CMAC)
- adopted by NIST SP800-38B

CMAC Overview



(a) Message length is integer multiple of block size

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

$K_1 = L \bullet x$

$L = E\,(K, 0^n)$

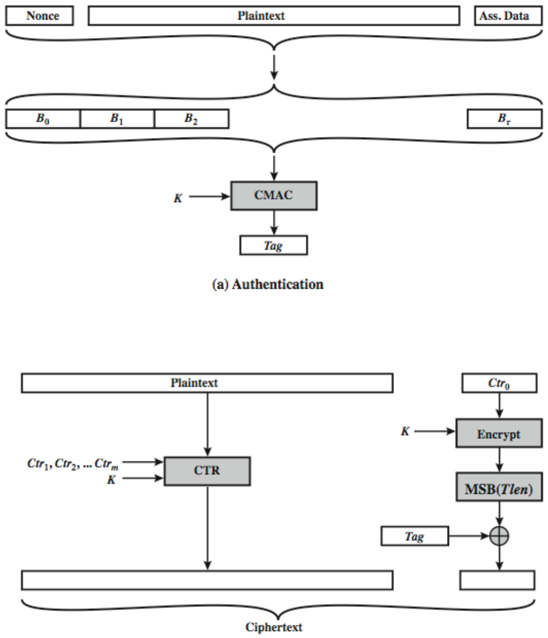
use $GF(2^{\text{blocksize}})$

$K_2 = L \bullet x^2$

Counter with Cipher Block Chaining-Message Authentication Code (CCM)

- NIST standard SP 800-38C for WiFi
- variation of encrypt-and-MAC approach
- algorithmic ingredients
  - – AES encryption algorithm
  - – CTR mode of operation
  - – CMAC authentication algorithm
- single key used for both encryption & MAC

CCM Operation



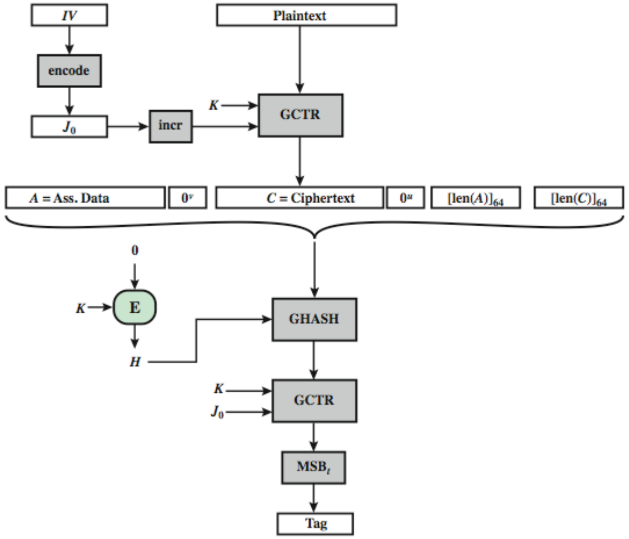
(a) Authentication

(b) Encryption

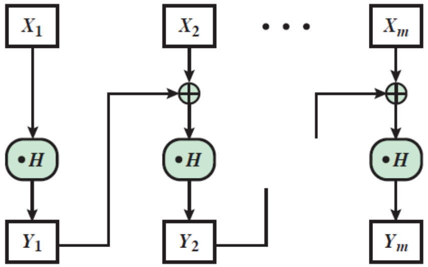
Galois/Counter Mode (GCM)

- NIST standard SP 800-38D, parallelizable
- message is encrypted in variant of CTR
- ciphertext multiplied with key `H` and length over GF(2^{128}) to generate authenticator
- have GMAC MAC-only mode also
- uses two functions:
  - GHASH - a keyed hash function
  - GCTR - CTR mode with incremented counter

GCM Mode Overview



GCM Functions — GHASH



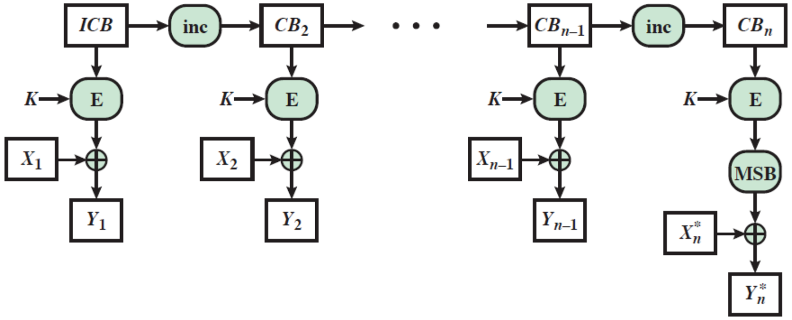
(a) $\text{GHASH}_H(X_1 \parallel X_2 \parallel \ldots \parallel X_m) = Y_m$

GHASH details

- GHASH is based entirely on GF(2^{128})
- XOR is addition, • is multiplication, distributive law works

$$
\begin{aligned}
Y_1 &= X_1 \bullet H \\
Y_2 &= (X_2+Y_1) \bullet H = (X_2+X_1 \bullet H) \bullet H \\
&= X_2 \bullet H + X_1 \bullet H^2 \\
&\dots \dots \\
Y_m &= (X_m+Y_{m-1}) \bullet H \\
&= X_m \bullet H + X_{m-1} \bullet H^2 + \dots + X_1 \bullet H^m
\end{aligned}
$$

GCM Functions — GCTR



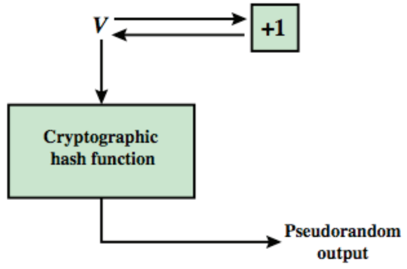
(b) $\mathrm{GCTR}_K(ICB, X_1 \| X_2 \| \dots \| X_n^*) = Y_n^*$

Pseudorandom Number Generation (PRNG) Using Hash Functions and MACs

- essential elements of PRNG are
  - seed value
  - deterministic algorithm
- seed must be known only as needed
- can base PRNG on
  - encryption algorithm (Chs 7 & 10)
  - hash function (ISO18031 & NIST SP 800-90)
  - MAC (NIST SP 800-90)

PRNG using a Hash Function

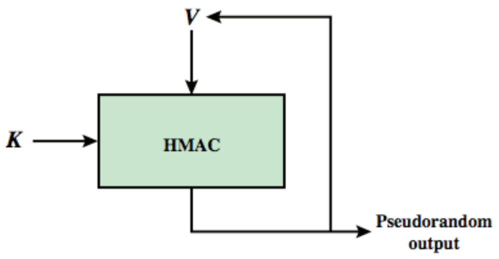
- hash PRNG from SP800-90 and ISO18031
  - take seed V
  - repeatedly add 1
  - hash V
  - use n-bits of hash as random value
- secure if good hash used



(a) PRNG using cryptographic hash function

PRNG using a MAC

- MAC PRNGs in SP800-90, IEEE 802.11i, TLS
  - use key
  - input based on last hash in various ways



(b) PRNG using HMAC