

DI-FCT-UNL

Segurança de Redes e Sistemas de Computadores  
*Network and Computer Systems Security*

Mestrado Integrado em Engenharia Informática  
MSc Course: Informatics Engineering  
2020/2021, 1st Sem.

# Secure Hashing and MACs Message Authentication Codes

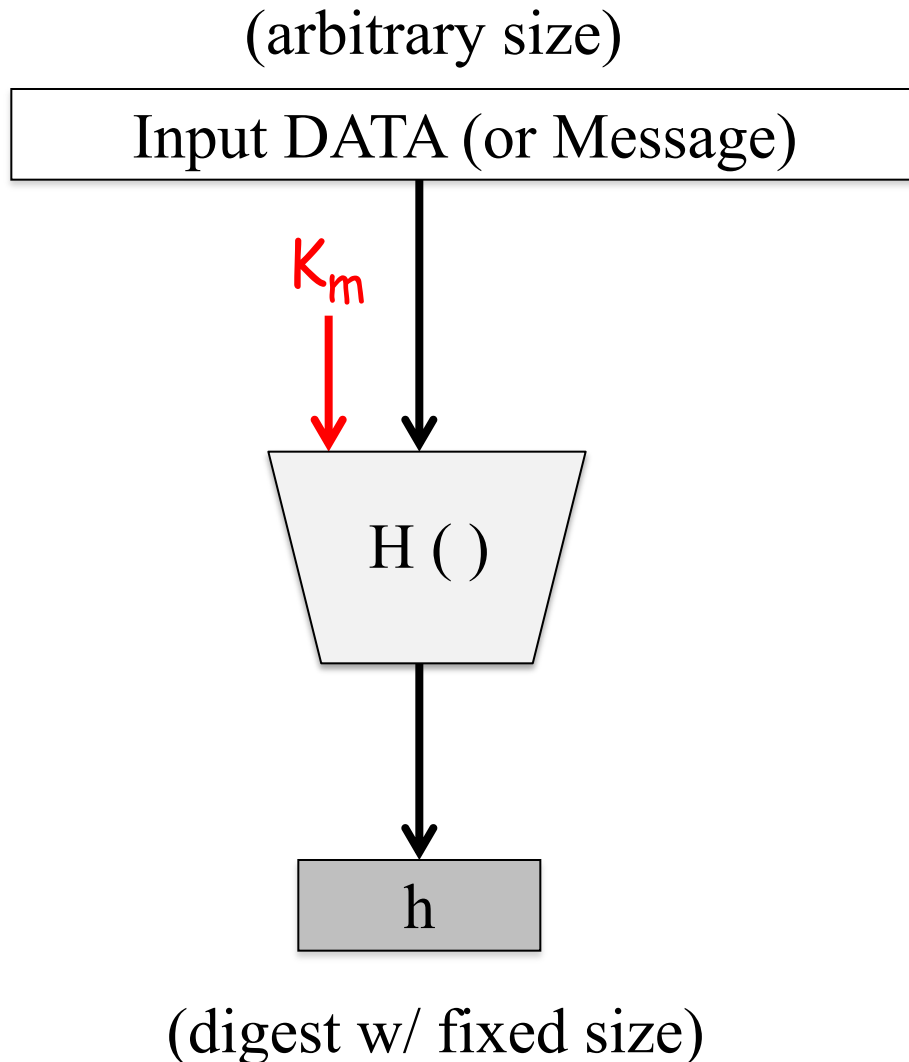
# Outline

- **Secure Hash Functions and Message Authentication**
  - Secure Hash Functions and Their Use
  - Security Properties of Secure Hash Functions
  - Secure Hash Algorithms
  - SHA-2-512 Case Study: Internal structure
  - Performance of Secure Hash Functions
  - Message authentication codes (MACs)
  - Confidentiality with Integrity using MACs
  - HMAC schemes
  - CMAC schemes

# Outline

- **Secure Hash Functions and Message Authentication**
  - Secure Hash Functions and Their Use
  - Security Properties of Secure Hash Functions
  - Secure Hash Algorithms
  - SHA-2-512 Case Study: Internal structure
  - Performance of Secure Hash Functions
  - Message authentication codes (MACs)
  - Confidentiality with Integrity using MACs
  - HMAC schemes
  - CMAC schemes

# Secure Hash Functions



Secure Digesting  
of arbitrary input data  
to fixed size output

$$h = H(M)$$

Also known as:  
Secure hash code  
or Secure Message Digest

Can use as Not-Keyed  
Hash

Or **Keyed-Hash => MAC**  
 $h = H_k(M)$

# Use of Secure Hash Functions (not Keyed)

## Used as integrity proofs of input data or messages

Detection of changes to validate the integrity in different situations, ex.:

- File Integrity Checks (or File Tampering Detection)
- Message Integrity Detection (or Message Tampering Detection)
- DB Integrity Control (Tables, Attributes in Entity Sets, ...)

Ex: Principle of File System Integrity Control in a HIDS:

- Files hashed and hash values stored a "secure place" (ex., read-only storage device)
  - If you compute the hash values of your supposed unchanged files, comparing with those hash values, you can detect anomalous changes (integrity breaks)

# Use of Keyed Secure Hash Functions

Used as fast authentication and integrity proofs of input data or messages

- File Integrity with Authenticated Checks
- Message Authentication and Integrity
- DB Authenticated Integrity Controls

Ex: Principle of File System Integrity in a HIDS:

- Authenticated Hashing (HMACs) of files in a computer stored in a "secure place" (ex., read-only storage device)
  - Can use for an authenticated attestation of integrity

>> DEMO

Integrity Checks w/ Secure Hashing and MACs using  
openssl

# Examples of Secure Hash Functions and MACs

See for ex: openssl, Java/JCE-JCE Labs

## Secure Hash Functions (Not Keyed Hash)

- MD2, MD4, MD5
- MDGOST-94
- SHA-1 (or SHA-160)
- RIPEMD-160
- Streebog256
- Streebog512
- SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512)
- Whirlpool

## MAC Constructions (Keyed Hash)

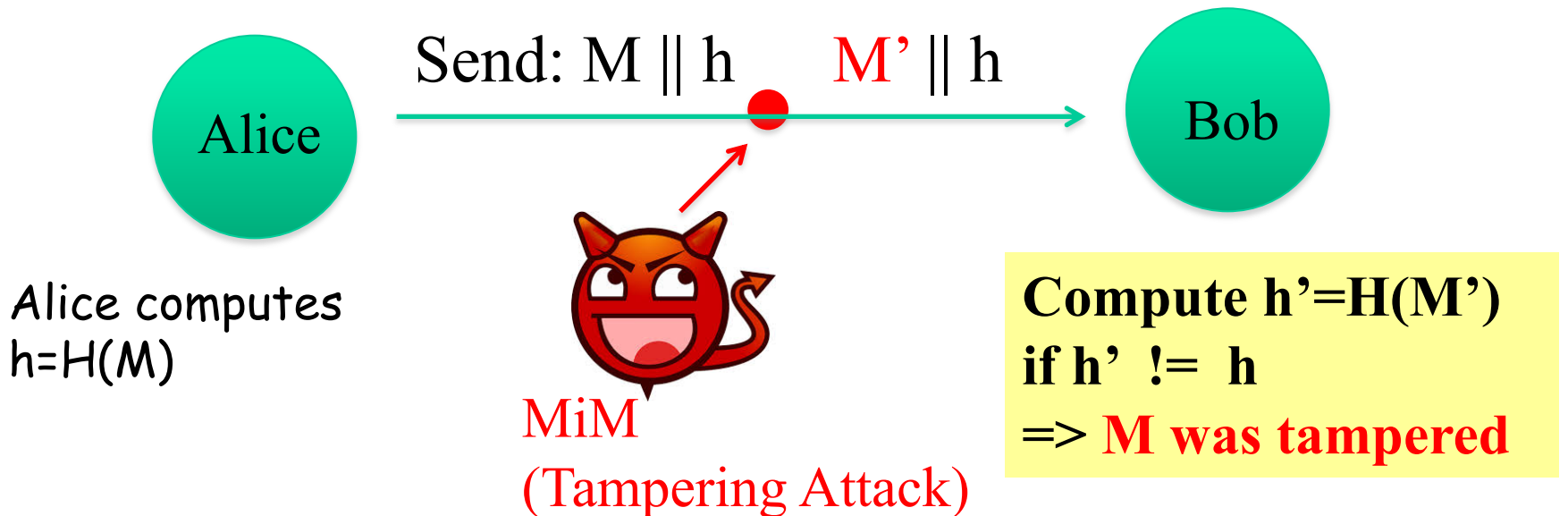
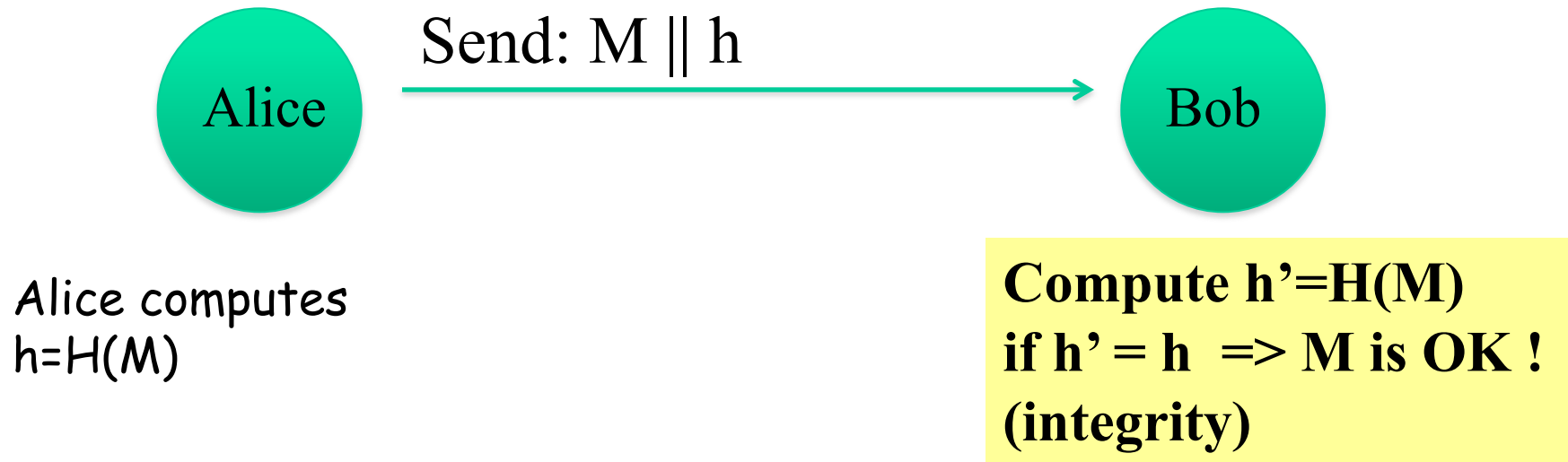
- GOST-MAC
- HMAC variants ...
- etc

See also Secure Hash Functions and MAC Functions on your Crypto Providers: (Lab 1 Materials)

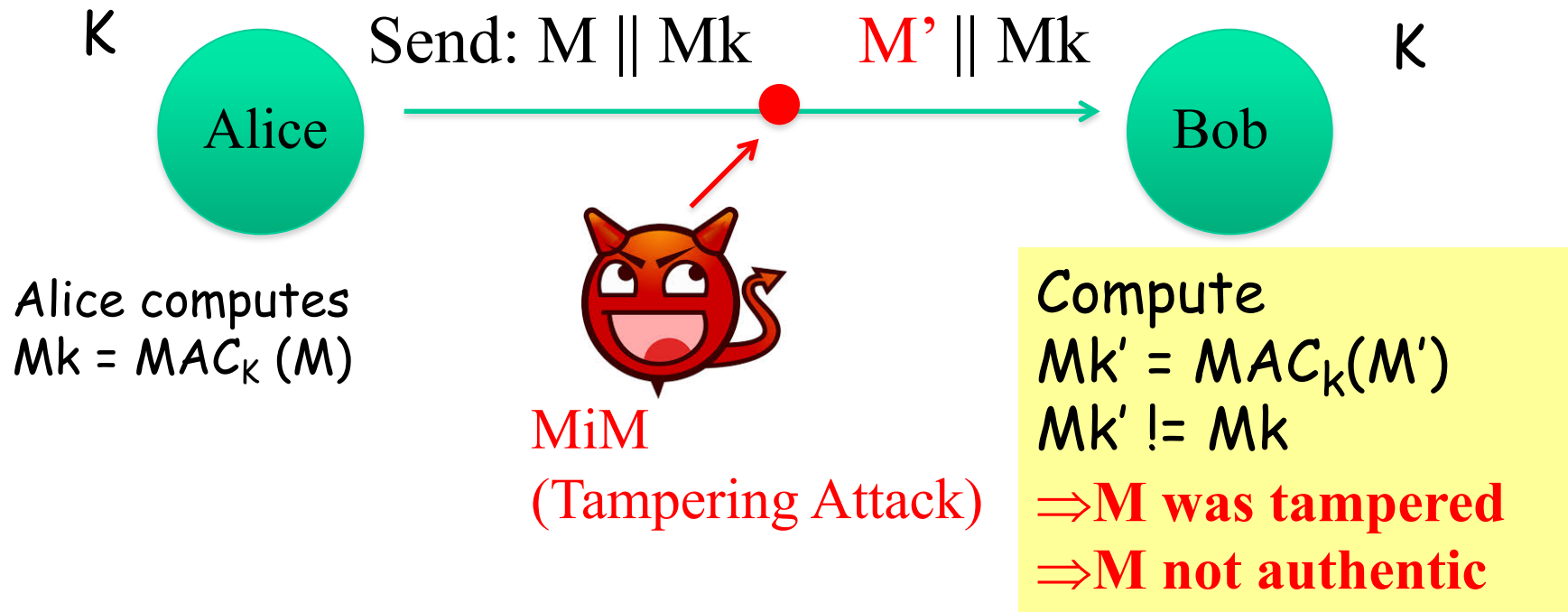
We will see also how to program with Secure Hash Functions and MACs (with detailed security observations) in Lab (Lab2 Materials)



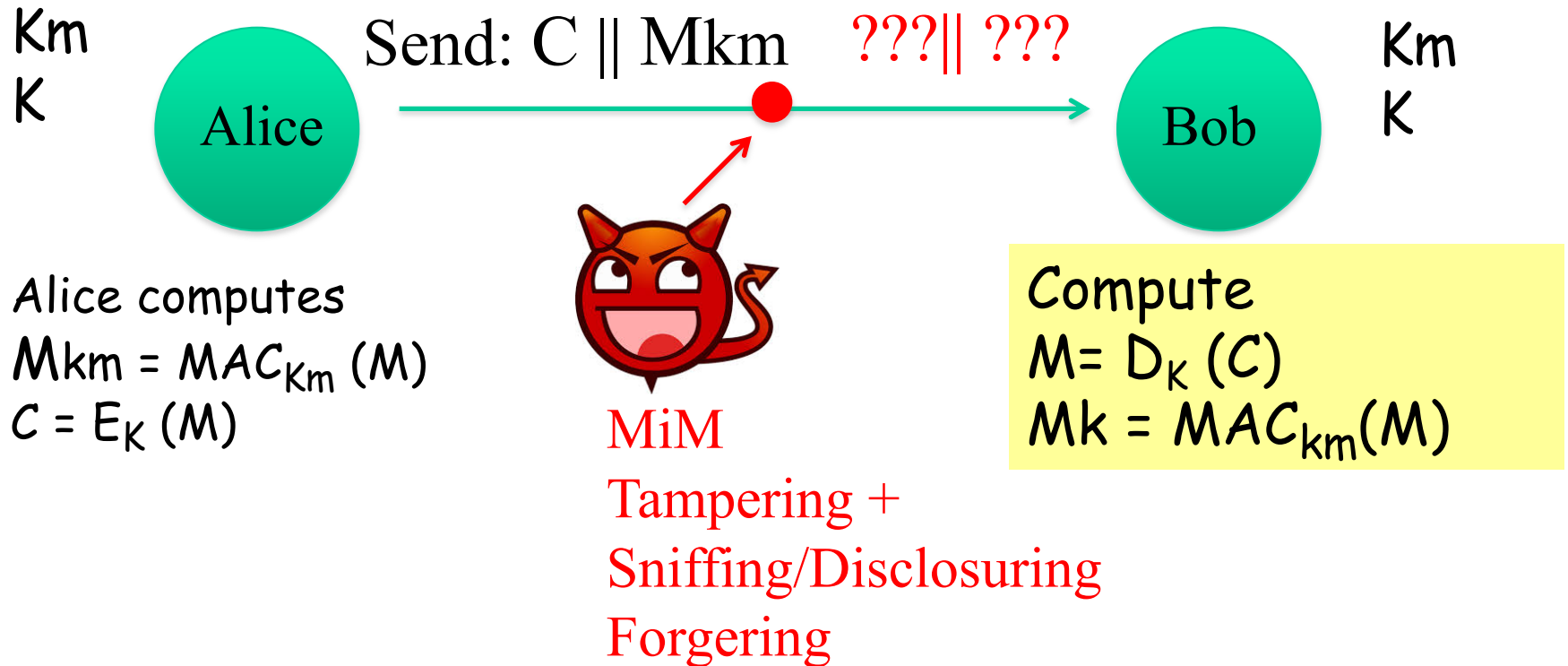
# Secure Hash Functions in Secure Communication



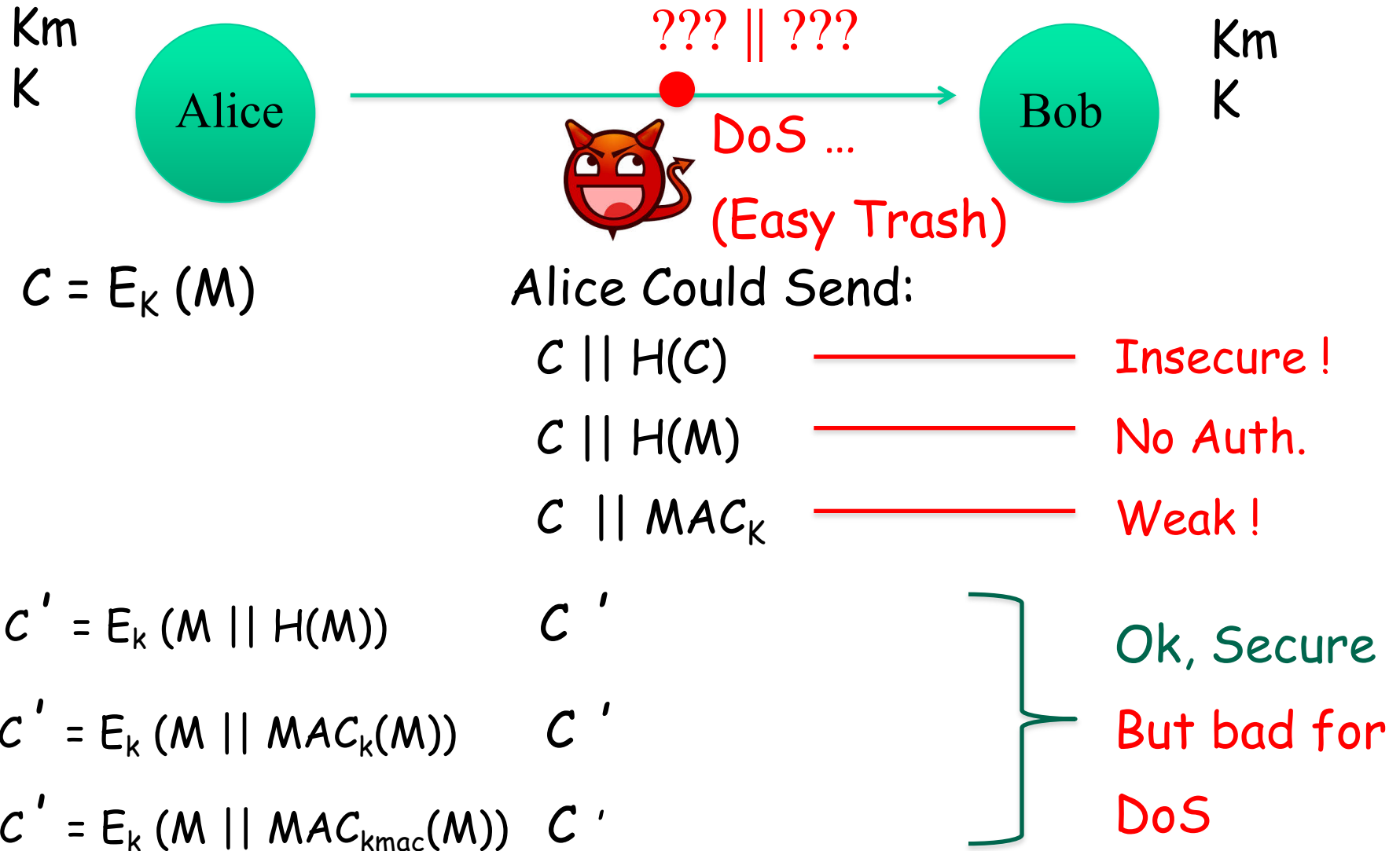
# MACs in Secure Communication



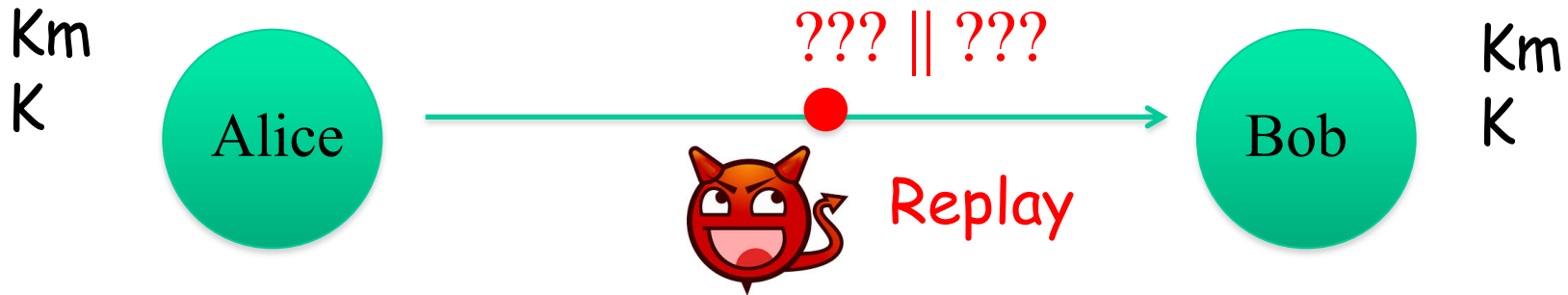
# Message Confidentiality + Integrity + Authenticity



# MSG Confidentiality + Integrity + Authenticity: What is better ? What is wrong ?



# What in case of Message Replaying ?



$$C = E_K(M)$$

Alice Could Send:

$$C || H(C)$$

$$C || H(M)$$

$$C || MAC_K$$

$$C' = E_K(M || H(M))$$

$C'$

$$C' = E_K(M || MAC_K(M))$$

$C'$

$$C' = E_K(M || MAC_{K_{mac}}(M))$$

$C'$

No  
Protection  
How to  
Protect ?

# Outline

- **Secure Hash Functions and Message Authentication**

- 
- Secure Hash Functions and Their Use
  - Security Properties of Secure Hash Functions
  - Secure Hash Algorithms
  - SHA-2-512 Case Study: Internal structure
  - Performance of Secure Hash Functions
  - Message authentication codes (MACs)
  - Confidentiality with Integrity using MACs
  - HMAC schemes
  - CMAC schemes

# Properties of Secure HASH Functions

- 1) **Arbitrary Input:**  $H( )$  can be applied to input blocks of data with any size
- 2) **Fixed Output:**  $H( )$  produces a fixed length output (hash value with fixed size)
- 3) **Performance:**  $H(x)$  is easy (low computational work, fast) to compute  $H(x)$  for any given  $x$ .

- 4) **Irreversibility (One-Way, Pre-Image Resistance):** for any given block  $h$ , it is computationally infeasible to find  $x$  such that  $H(x)=h$
- 5) **Weak collision resistance:** for any given block  $x$ , it is computationally infeasible to find  $y$ , with  $H(y) = H(x)$ .

Also known as **Second Pre-Image Resistance**

- 6) **Strong collision resistance:** it is computationally infeasible to find any pair  $(x, y)$ , with  $H(x) = H(y)$

- 7) **Uniform distribution of  $H(x)$  for any  $x$**

# Irreversibility and Collision Resistance

In “real practical secure hash algorithms” these are probabilistic properties meaning: impossibility to break in useful computational time (or computational effort)

**Note: “strong” vs. “weak” collision resistance:**

- For security protocols, in general, we are interested for maximum security in secure hash algorithms with Strong Collision Resistance
  - Subsumes Pre-Image and Second Pre-Image Resistance

... Think on it by the **Birthday Paradox**

- The chance that in a group of people two will share the same birthday: with 23 persons,  $p > 0.5$
- Generalization: matching pair from any two sets:  $2^{m/2}$  in each set to get a matching  $m$ -bit hash



# A Simple and Naïve Hash Function (LRC MIC)

	bit 1	bit 2	• • •	bit $n$
Block 1	$b_{11}$	$b_{21}$		$b_{n1}$
Block 2	$b_{12}$	$b_{22}$		$b_{n2}$
	•	•	•	•
	•	•	•	•
	•	•	•	•
Block $m$	$b_{1m}$	$b_{2m}$		$b_{nm}$
Hash code	$C_1$	$C_2$		$C_n$

Simple: Bit by Bit Exclusive OR  
MIC - Longitudinal Redundancy Check  
(MIC means Message Integrity Check)

Is it Secure ?

# Not secure ☹️ ... No Collision Resistance

- Suppose  $C_i = B_{i1} \text{ xor } B_{i2} \text{ xor } \dots \text{ xor } B_{im}$   
for  $1 \leq i \leq n$
- Good (acceptable) as a MIC ? probability of an error will result in an unchanged hash value is  $1/2^n$
- Is it secure ? Ex., Predictable input as text with 128 bit hash, reduces P to  $1/2^{112}$
- Simple ... don't means "Secure"
  - How to do it better ?

Need to provoke an effect of randomizing the input more completely and overcoming any regularities

MICs (No Secure Message Integrity Codes): used for FEC codes ... very different from Secure MACs or Secure Hash Functions

# An "improved" Hash Function. Is it Better ?

## Use of RXOR (Rotated XOR)

- Ex., one-bit circular shift on the hash value after each block is processed would improve
- But is it secure ? Useful for integrity ?
- No ! See the demonstration (bibliography)

---

See also:

$2^{37}$  variations of a letter

[Dav 89], Stallings, Brown, Computer Security Principles and Practice

# Collision-Attacks

- For a given "secure hash function" it takes a certain computational effort (or energy).
- Is it simple ?

>> DEMO

Breaking MD5 Collision Resistance

# Security of Secure Hash Functions

- Resistance against attacks to break the security properties
- Attacks (and studies) conducted (as in the case of symmetric encryption algorithms) by:

BRUTE FORCE

CRYPTANALYSIS

# Brute Force on Hash Functions

Strong collision resistance:

Hash cost with complexity  $O(2^{m/2})$

- Proposal for HW MD5 cracker (see the book)
- 128-bit hash looks vulnerable, 160-bits better
- More bits => Even better

MACs (pre-keyed Hashes) with known message-MAC pairs

- Can either attack key space (key search) or MAC
- Cost in this case:  $O(\min 2^k, 2^n)$  which is similar to symmetric encryption algorithms
- At least 128-bit MAC is needed for security today
  - Similar discussion on Symmetric Encryption Algorithms

# Collision Resistance on Secure Hash Functions

- **Cryptanalytic attacks** exploit the internal structure
  - Like block ciphers want brute-force attacks to be the best alternative
- Have a number of analytic attacks on iterated hash functions
  - $CV_i = f[CV_{i-1}, M_i]; H(M) = CV_N$
  - Typically focus on collisions in function  $f$
  - Like block ciphers is often composed of rounds
    - Many rounds are ok (to improve the randomization: confusion and diffusion effect)
  - Attacks exploit properties of core-round functions

# Collision Resistance Metrics for Secure Hash Functions with Hash Size N

Security Property	Metrics	Probability of Attack	Example (SHA 256)
Pre-image Resistance	$2^N$	$1/2^N$	$1/2^{256}$
Second Preimage Resistance	$2^N$	$1/2^N$	$1/2^{256}$
(Strong) Collision Resistance	$2^{N/2}$	$1/2^{N/2}$	$1/2^{128}$



# Collision Resistance Metrics for Secure Hash Functions with Hash Size N

Security Property	Example (SHA 256)	My Computer (openssl lib.)	Time to Attack
Pre-image Resistance	$2^{256} \sim 1.2 \times 10^{77}$	$22 \times 10^6 \text{ h/s}$	$\sim 54 \times 10^{68} \text{ s}$
Second Preimage Resistance	$2^{256} \sim 1.2 \times 10^{77}$	$22 \times 10^6 \text{ h/s}$	$\sim 54 \times 10^{68} \text{ s}$
(Strong) Collision Resistance	$2^{128} \sim 3.4 \times 10^{38}$	$22 \times 10^6 \text{ h/s}$	$154 \times 10^{29} \text{ s}$

# Collision Resistance Metrics for Secure Hash Functions with Hash Size N

Security Property	Time to Attack	My Computer (openssl lib.)	ASIC Dragomint 16T $\sim 10^{12} \text{ h/s}$
Pre-image Resistance	$\sim 54 \times 10^{68} \text{ s}$	$\sim 1.7 \times 10^{62} \text{ years}$	$\sim 10^{50} \text{ years}$
Second Preimage Resistance	$\sim 54 \times 10^{68} \text{ s}$	$\sim 1.7 \times 10^{62} \text{ years}$	$\sim 10^{50} \text{ years}$
(Strong) Collision Resistance	$154 \times 10^{29} \text{ s}$	$\sim 4,9 \times 10^{23} \text{ years}$	$\sim 10^{11} \text{ years}$

<https://www.buybitcoinworldwide.com/mining/hardware/dragonmint-16t/>

# Birthday Attacks ... and Proofs of Work (PoW)

Might think a 64-bit hash is secure ? 128 is better ?  
what about ... 224, 256, 384, 512 ... Etc ?

## The Security vs. Performance Tradeoff

### PoW - Proof of Work (using Secure Hash-Functions)

- The required computational work or energy :- ( to find a collision, given a certain challenge

- Ex: Find a collision in such a way that

$$\text{SHA-2-256}(X \parallel \text{NONCE}) = 0000 \dots\dots\dots$$

$$\Rightarrow H(X) < 2^{224}$$

Challenge

Fixed: 32 bits

(any): 224 bits

# Some examples of Proof of Work uses

## Countermeasures for DoS or DDoS Attacks

Ex:

---

**Henrique Domingos, J. Martins, Ricardo Martins, EIP – Preventing DDoS with Ephemeral IP Identifiers Cryptographically Generated, Proc. of SecNets / Cornell University arXiv 1612.07065, Dec 2016**

A “competitive” form of Probabilistic Consensus in Bitcoin Mineration (ex. Bitcoin/Blockchain PoW strategy)

- Challenge regulated for required computation and energy to close a block of transactions each ~10 minutes (or ~2016 blocks each 2 weeks)
- What does it means ?

---

**See more, ex: <https://en.bitcoin.it/wiki/Bitcoin>**

# Outline

- **Secure Hash Functions and Message Authentication**
  - Secure Hash Functions and Their Use
  - Security Properties of Secure Hash Functions
  - Secure Hash Algorithms
  - SHA-2-512 Case Study: Internal structure
  - Performance of Secure Hash Functions
  - Message authentication codes (MACs)
  - Confidentiality with Integrity using MACs
  - HMAC schemes
  - CMAC schemes

# Well known (classic) Secure HASH algorithms: So far ... so weak ? ... !

	MD5	RIPEMD-160	SHA-1
Output Digest length	128 bits	160 bits	160 bits
Basic unit of processing	512 bits	512 bits	512 bits
Number of steps	64 (4 rounds of 16)	60 (5 paired rounds of 16)	80 (4 rounds of 20)
Maximum input message size	$2^{64}-1$ bits	$2^{64}-1$ bits	$2^{64}-1$ bits
Cryptanalysis (metrics) published Collision Resistance	$1/2^{24}$ to $1/2^{39}$ PIR: $1/2^{123}$	$1/2^{70}$ to $1/2^{75}$ PIR: $1/2^{148}$ to $1/2^{158}$	$1/2^{52}$ to $1/2^{63}$ PRI: $1/2^{150}$ to $1/2^{159}$

Considered  
Weak Today

Possible  
Weaknesses

Possible  
Weaknesses

# SHA-2

NIST rev FIPS PUB 180-2 (2002), FIPS PUB 180-3 (2008)

SHA-2 (Families) now also in IETF RFC 6234

SHA-1 Phaseout (NIST): 2005 to 2010

SHA-1 Attacks (Collision Resistance w/  $1/2^{63}$  )

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
<b>Message Digest Size</b>	160	224	256	384	512
<b>Message Size</b>	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
<b>Block Size</b>	512	512	512	1024	1024
<b>Word Size</b>	32	32	32	64	64
<b>Number of Steps</b>	80	64	64	80	80
<b>Security</b>	80	112	128	192	256

Notes: 1. All sizes are measured in bits.


2. Security refers to the fact that a birthday attack on a message digest of size  $n$  produces a collision with a workfactor of approximately  $2^{n/2}$ .



# SHA-2 Family and Revisions

NIST issued revisions FIPS 180-2 (2002) and 180-3 (2008)

- Designed for compatibility with increased security as also provided by the AES cipher
- However, structure & detail is similar to SHA-1
- Hence analysis should be similar
- But security levels are rather higher (esp. SHA-256, SHA-384, SHA-512)



NIST announced in 2007 a competition for the SHA-3 next gen NIST hash function

Goal to have in place by 2012 but not fixed

Only initially standardized in Oct/2015 after a big debate



# Standardized (SHA-3)

Also known as “Keccak” family of secure hash functions

- The new kids in town !
- Hash outputs with variable size:
  - 128, 224, 256, 384, 512
- New structure (Sponge constructions)
- SHA-3 family (and variants) are evolving, and some IETF RFC Drafts on going very recently (Feb, Mar 2019): you can expect very dynamic work in the short-medium term ...

---

For on-going SHA-3: see (if you want the details):

<https://en.wikipedia.org/wiki/SHA-3>

NIST Standardization: <https://www.nist.gov/news-events/news/2015/08/nist-releases-sha-3-cryptographic-hash-standard>

See also ... About SPONGE Constructions (in the SHA-3 Core)

<http://keccak.noekeon.org>

# SHA-3 Variants

Instance	Definition
SHA3-224( $M$ )	Keccak[448]( $M \parallel 01$ , 224)
SHA3-256( $M$ )	Keccak[512]( $M \parallel 01$ , 256)
SHA3-384( $M$ )	Keccak[768]( $M \parallel 01$ , 384)
SHA3-512( $M$ )	Keccak[1024]( $M \parallel 01$ , 512)
SHAKE128( $M$ , $d$ )	Keccak[256]( $M \parallel 1111$ , $d$ )
SHAKE256( $M$ , $d$ )	Keccak[512]( $M \parallel 1111$ , $d$ )

We can use these implementations in crypto-providers available for JAVA JCE

# SHA-3 Requirements and On-Going Research

- It must replace SHA-2 with SHA-3 in any use
  - So use same hash sizes
- Preserve the online nature of SHA-2
  - So must process small blocks (512 / 1024 bits)
- Evaluation criteria (over the table)
  - Security close to theoretical max for hash sizes
  - Cost in time & memory (efficiency)
  - Tradeoff Characteristics: security, performance, flexibility & simplicity

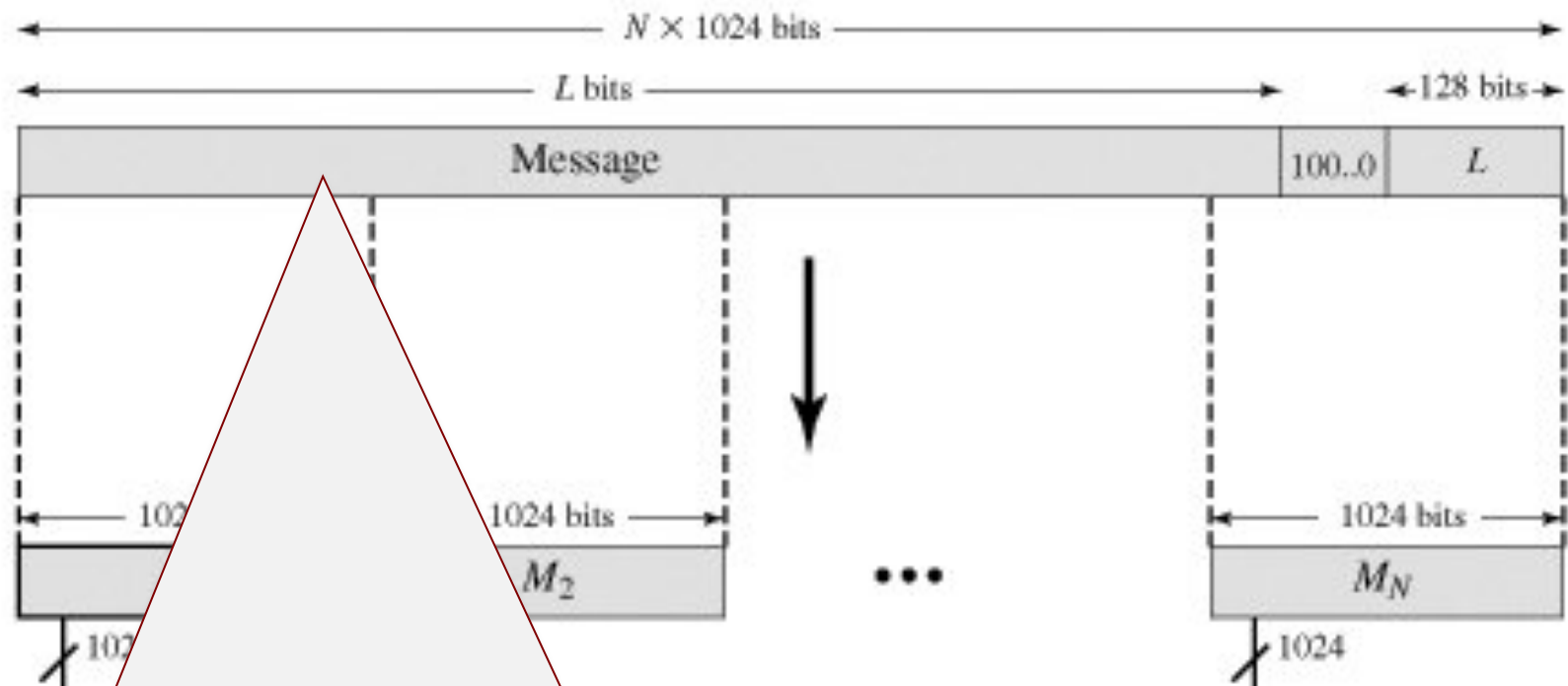
# Outline

- **Secure Hash Functions and Message Authentication**
  - Secure Hash Functions and Their Use
  - Security Properties of Secure Hash Functions
  - Secure Hash Algorithms
  - **SHA-2-512 Case Study: Internal structure**
  - Performance of Secure Hash Functions
  - Message authentication codes (MACs)
  - Confidentiality with Integrity using MACs
  - HMAC schemes
  - CMAC schemes

# Internal Structure for SHA-2 512

## How SHA-2 (SHA-512) Works ...

(1)



Input MSG: must be smaller than  $2^{128}$  bits representation

**Step 1:** Append of Padding bits: 100000...0000, for congruency w/  
 $896 \bmod 1024$

**Step 2:** Append 128 bits (Total Length, Multiple of 1024 bits)

# Internal Structure for SHA-2 512

(1)

## How SHA-2 (SHA-512) Works ...

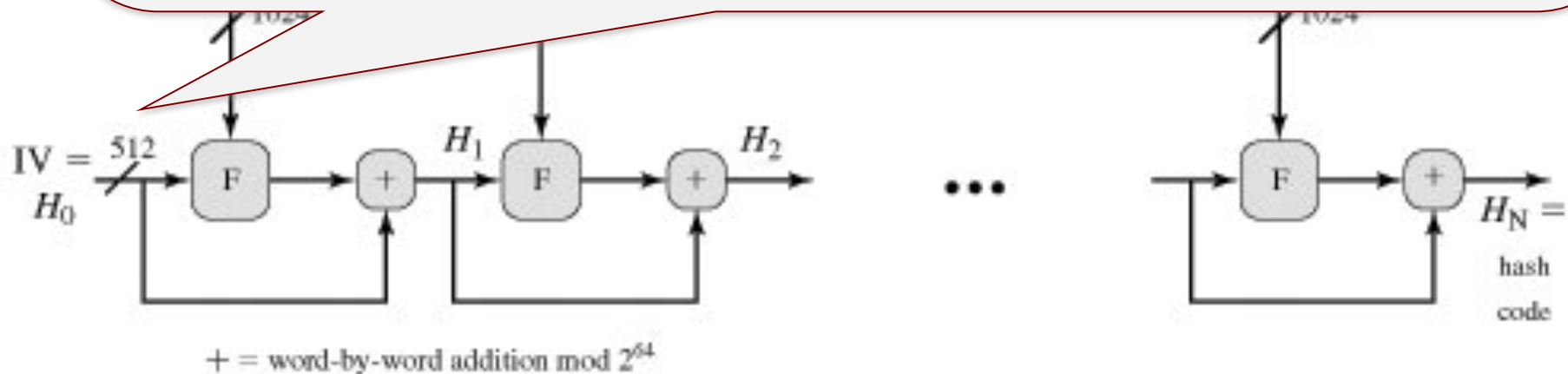
Now we have N words of 2014 bits

**Step 3:** Initialization of Hash Buffers w/ Pre-Defined Constants:

512 bits = 32 bit x 8 value-letters

The 64 MSBs of fractional part of Root Squares of the first 8 Prime Numbers

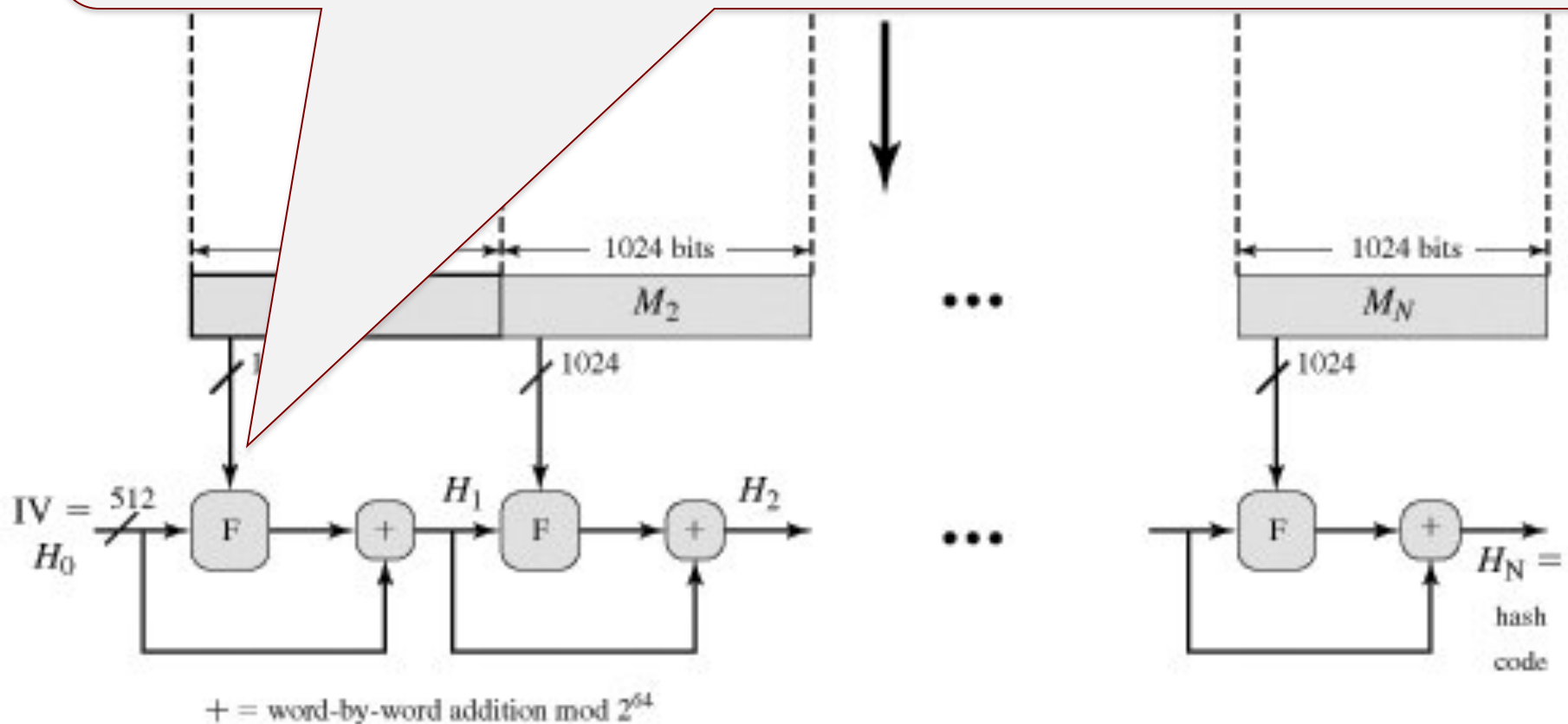
a = 6A09 E667 F3BC C908;	b = BB67 AE85 84CA A73B
c = 3C6E F372 FE94 F82B;	d = A54F F53A 5F1D 36F1
e = 510E 527F ADE6 82D1;	f = 9B05 688C 2B3E 6C1F
g = 1F83 D9AB FB41 BD6B;	h = 5BE0 CD19 137E 2179



# Internal Structure for SHA-2 512 (1)

## How SHA-2 (SHA-512) Works ...

**Step 4:** Message is processed in 1024 bit ( 64 bits per round) blocks with the SHA-512 Core Function, executed in 80 Steps:



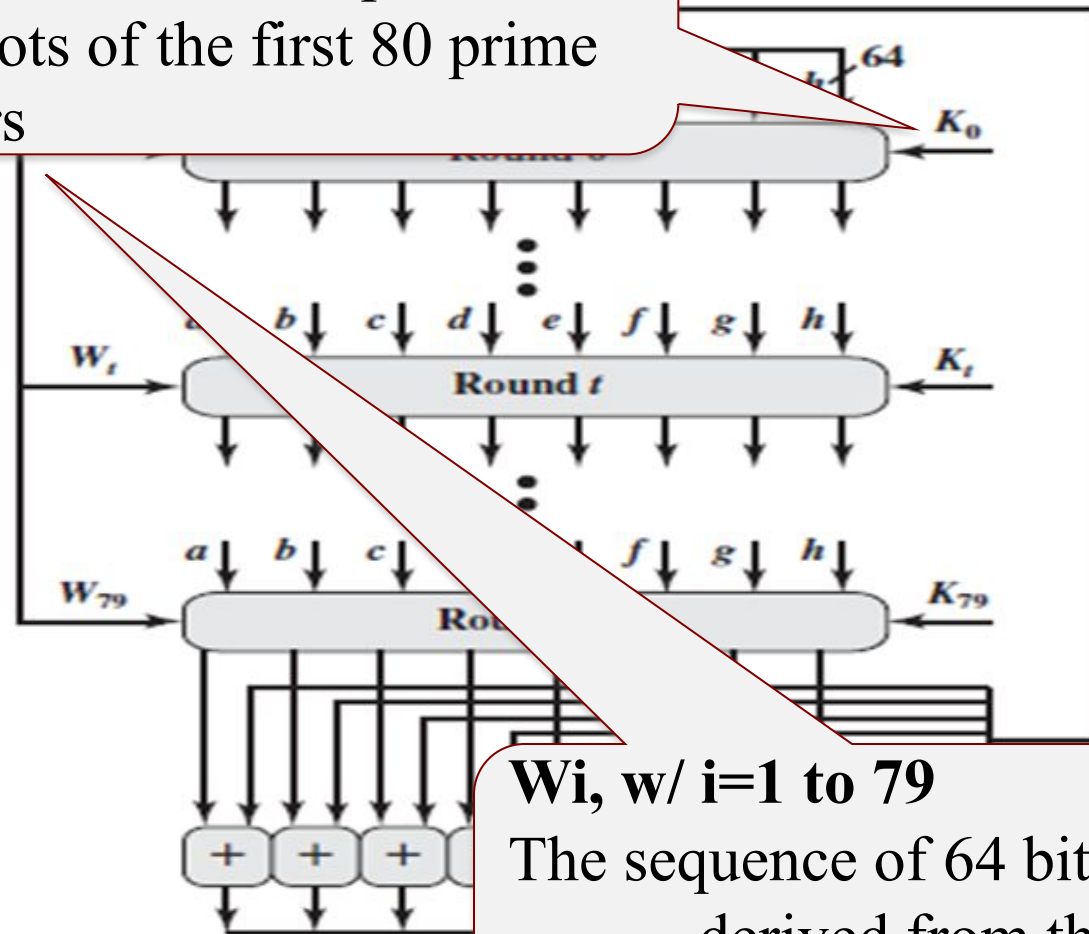
# Internal Structure for SHA-2 512

(2)

## How SHA-2 (SHA-512) Works ...

**$K_i$ , w/  $i=1$  to 79**

First 64 bits of the fractional part of the cube roots of the first 80 prime numbers



**$W_i$ , w/  $i=1$  to 79**

The sequence of 64 bit words, derived from the current 1024 bit block

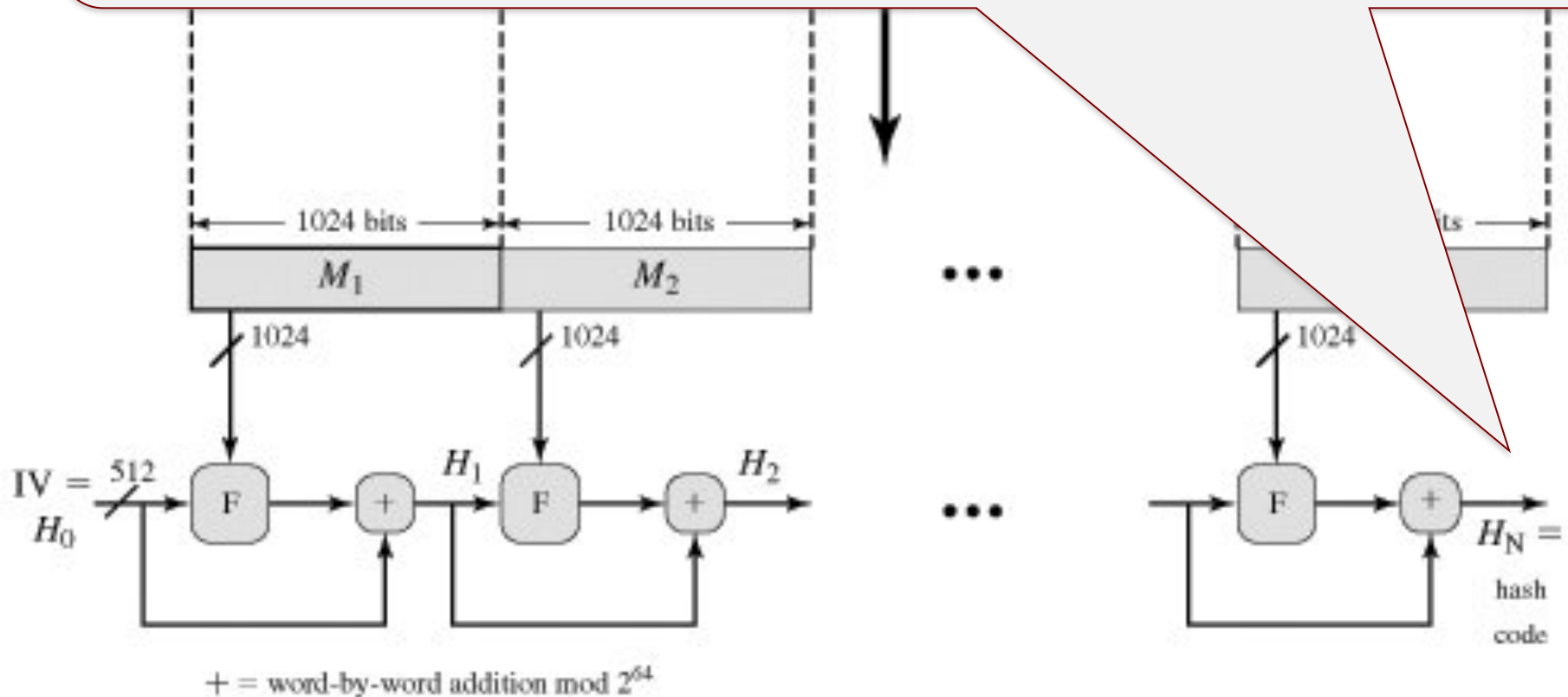


# Internal Structure for SHA-2 512 (1)

## How SHA-2 (SHA-512) Works ...

### Step 5: Output

After all  $N$  1024 bit blocks have been processed, the output from the  $N$ th stage will be the 512 bit output hash value



# Outline

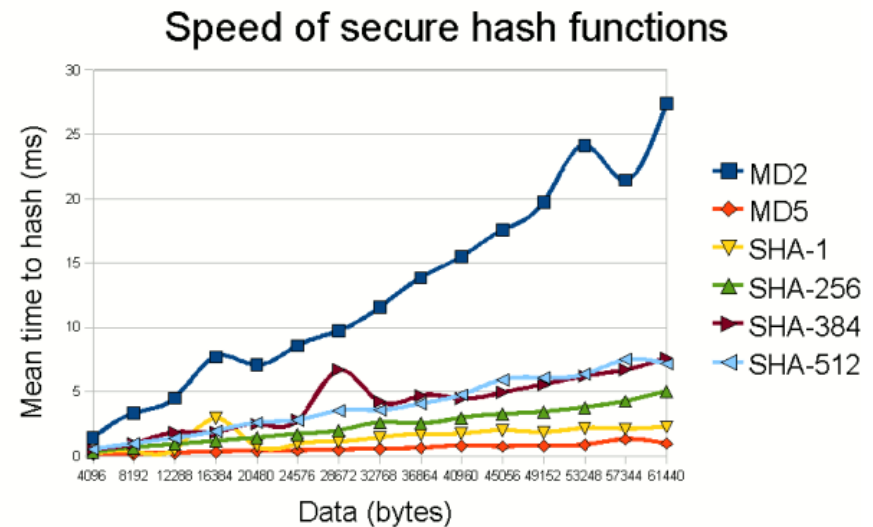
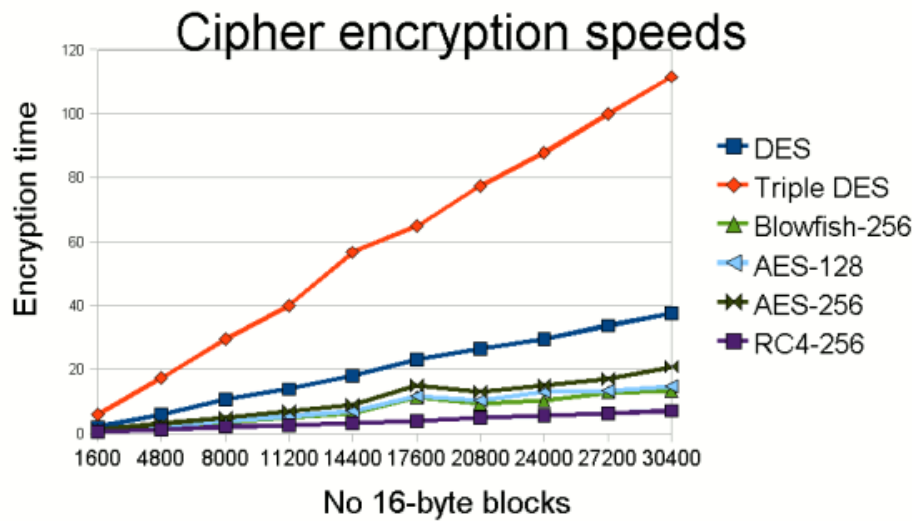
- **Secure Hash Functions and Message Authentication**
  - Secure Hash Functions and Their Use
  - Security Properties of Secure Hash Functions
  - Secure Hash Algorithms
  - SHA-2-512 Case Study: Internal structure
  - Performance of Secure Hash Functions
  - Message authentication codes (MACs)
  - Confidentiality with Integrity using MACs
  - HMAC schemes
  - CMAC schemes

# Performance of Secure Hash Algorithms

Comparisons in Java (standard JDK and Sun Crypto-Provider)

<http://www.javamex.com/tutorials/cryptography/ciphers.shtml>

[http://www.javamex.com/tutorials/cryptography/hash\\_functions\\_algorithms.shtml](http://www.javamex.com/tutorials/cryptography/hash_functions_algorithms.shtml)



as ref: hash (ex., SHA-512) ~100 x faster than symmetric encryption (ex., 3DES)  
hash (ex., SHA-512) ~ 4x faster than symmetric encryption (ex., AES-256)  
hash (ex., SHA-1) ~ 18x faster than symmetric encryption (ex., AES-128)

See also:

Openssl speed benchmark (speed test library performance)

<http://wikis.sun.com/display/CryptoPerf/UltraSPARC+cryptographic+performance>

# Performance analysis:

## Secure Hashing vs. Symmetric Encryption

Try to make these performance evaluations in your computer using JAVA JCE (cryptography) and the materials we provide for LABs !

Or use openssl (linux. mac, windows) to see the performance in processing input blocks of different sizes with different algorithms ... Examples:

```
# openssl speed sha512
```

...

Bytes/Second:

type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
sha512	23798.27k	93818.06k	169995.47k	261958.66k	301029.51k

```
# openssl speed aes-256-cbc
```

...

Bytes/Second:

type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
des cbc	72212.05k	74651.61k	74132.22k	74070.99k	73933.64k

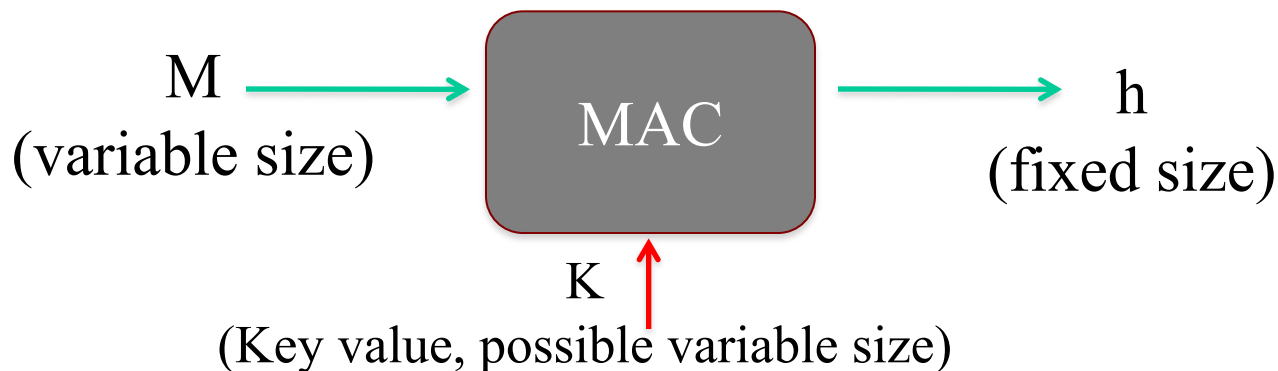
# Outline

- **Secure Hash Functions and Message Authentication**
  - Secure Hash Functions and Their Use
  - Security Properties of Secure Hash Functions
  - Secure Hash Algorithms
  - SHA-2-512 Case Study: Internal structure
  - Performance of Secure Hash Functions
  - Message authentication codes (MACs)
  - Confidentiality with Integrity using MACs
  - HMAC schemes
  - CMAC schemes

# MACs as Keyed Secure Hash Functions

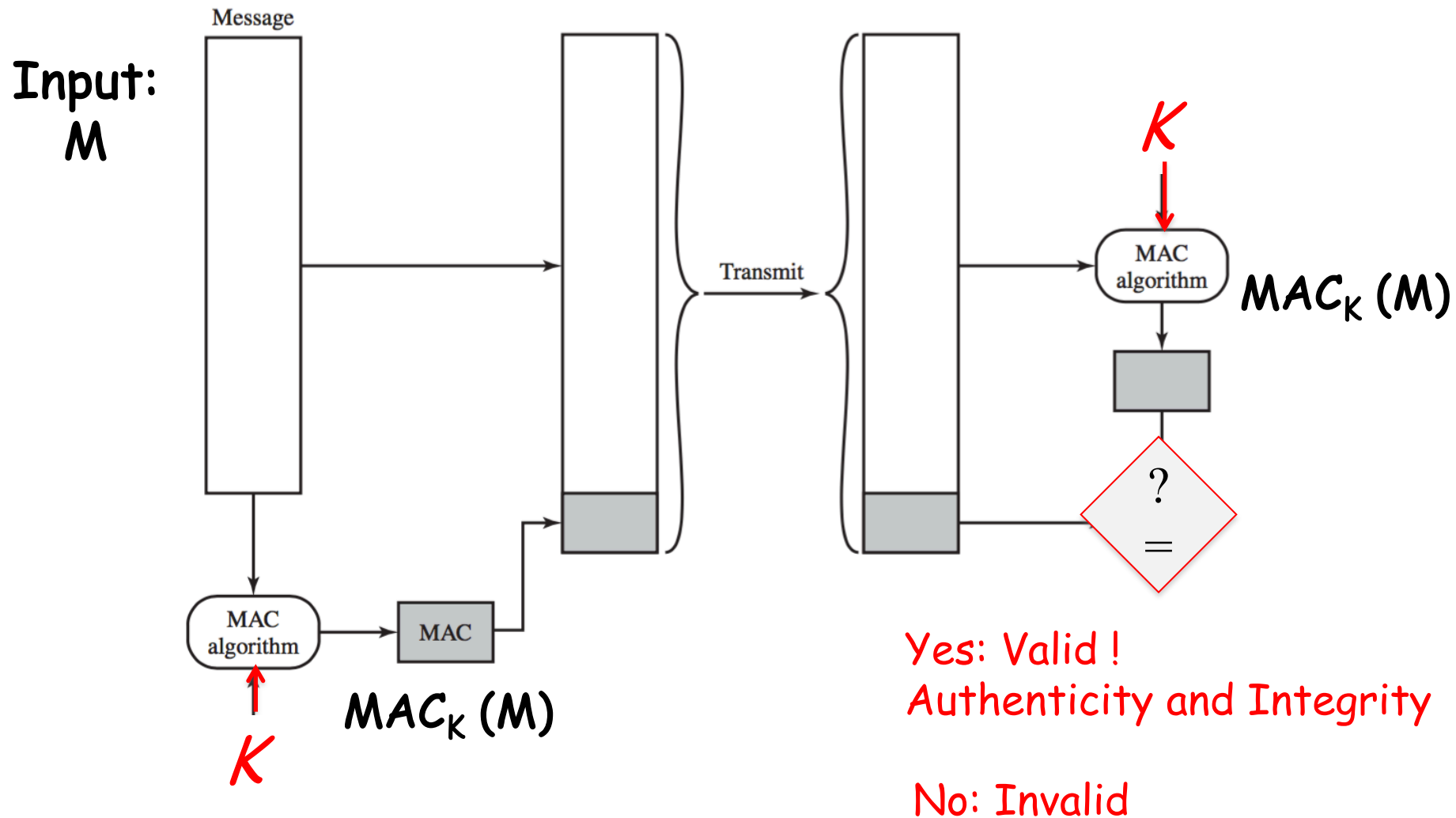
- Used as a **Message Authentication Code**

$MAC_{M,K} = H(f(M,K))$  : hash code or message digest



- Can use in various ways with message
  - Examples :
    - $MAC_{M,K} = H(M || K)$  // Append  $M$  with  $K$
    - $MAC_{M,K} = H(M \text{ xor } K)$  //  $M$  and  $K$  "xored"
    - $MAC_{M,K} = H(F(M,K))$  // with a known  $F()$
- Most often to create a "*fast or light-weight message signature*", for a previously shared  $K$

# Use of MACs (MAC Algorithm Constructions)



# Message Authentication

- Message authentication: proof of origin, confirming the sender (identity of originator)
  - Protecting (implicitly) the message integrity
    - Contents have not been altered (tampering detection)
    - Proof that, it was sent at a certain time or sequence (message replay detection or non-sequence detection)
  - Validating message authenticity as coming from the expected originator
    - Message came from an origin sharing the MAC key
- In some cases... message authentication is arguably more important than secrecy (ex., E-Commerce)



# Message Authentication Codes

- As shown, MAC provides authentication and implicit integrity (but no confidentiality)
- Why use a MAC with no confidentiality warranties?
  - Sometimes only authentication is needed
  - Sometimes need authentication and integrity to persist longer than the encryption (eg. archival use)  
(Integrity checking may be supported with other schemes (MICs, other checksum functions but these are weak solutions !)
- But can be combined with encryption for secrecy (combining authentication + integrity + confidentiality)
  - Generally use separate keys for each
  - Can compute MAC either before or after encryption
  - Is generally regarded as better done before
  - ... But must consider also balance requirements for authentication, confidentiality, integrity vs. DoS

# MAC Properties (is not a Digital Signature)

- A MAC works as a cryptographic secure checksum

$$\text{MAC} = C_K(M) \text{ or } \text{MAC}_K(M) \text{ or } \text{MAC}(K, M)$$

- Note that a MAC is not a digital signature (authentication of principals) in the sense of digital signatures for principals
  - A MAC requires shared keys
  - Used like "fast or light-weight message signature scheme" based on a NDA of the KEY. Why ?

- Is it a "many-to-one" function ?
  - potentially many messages have same MAC. Why ?
    - Not warranted collision free
  - but finding these needs to be very difficult
    - With appropriate security properties of MAC cryptographic functions. **Which properties ?**

# Birthday Attacks on MACs

Birthday attack works thus:

- opponent generates  $2^{m/2}$  variations of a valid message all with essentially the same meaning
- opponent also generates  $2^{m/2}$  variations of a desired fraudulent message (to get a matching m-bit hash)
- two sets of messages are compared to find pair with same hash (probability  $> 0.5$  by birthday paradox)
- have user sign the valid message, then substitute the forgery which will have a valid signatures
- Conclusion: need to use larger MAC/hash values

# Outline

- **Secure Hash Functions and Message Authentication**
  - Secure Hash Functions and Their Use
  - Security Properties of Secure Hash Functions
  - Secure Hash Algorithms
  - SHA-2-512 Case Study: Internal structure
  - Performance of Secure Hash Functions
  - Message authentication codes (MACs)
  - Confidentiality with Integrity using MACs
  - HMAC schemes
  - CMAC schemes

# Security properties of MACs

Same as Keyed Hash Functions:

- **Irreversibility (one way, pre-image resistance)**  
If  $Y = \text{MAC}(X1)$   
it is not possible to find  $X1$  knowing  $Y$
- **Collision resistance:** knowing a message and MAC, is infeasible to find another message with same MAC
  - **Weak collision resistance**  
Given  $Y = \text{MAC}(X1)$  and  $X1$  : no other  $X2$  with  $Y = \text{MAC}(X2)$
  - **Strong collision resistance**  
Given a value  $y$ , it is not possible to find any  $Z1$  and  $Z2$ , in a way that  $\text{MAC}(Z1) = \text{MAC}(Z2) = Y$
- **Uniform distribution**
  - MACs should be uniformly distributed, independently of the message input
  - MAC should depend equally on all bits of the message

# Examples (typical uses): hands-on discussion on LABs

- Ex. 1:

Header | {M, TS, NS, ... }<sub>k1</sub> | MAC<sub>k2</sub> (M, TS, NS, ...)

- Ex. 2:

Header | {M, TS, SN, ... , MAC<sub>kS</sub> (M, TS, SN, ... ) }<sub>kS</sub>

- Ex. 3:

Content-Type | M version | m version | Compressed length | {  
compress [ M, TS, SN,... MAC<sub>k<sub>m</sub></sub> (M, TS, N,...) ] }<sub>k<sub>S</sub></sub>

- Ex. 4: (DoS minimization tradeoff)

Content-Type | M version | m version | Compressed length | {  
compress [M,TS,SN, ...] }<sub>k<sub>S</sub></sub> | MAC<sub>k<sub>m</sub></sub> (M,TS,NS, ...)

> Note: you must take in account other issues for security  
assumptions: Block modes of operation, key-distribution,  
attacks to weaknesses of MAC properties

# Outline

- **Secure Hash Functions and Message Authentication**
  - Secure Hash Functions and Their Use
  - Security Properties of Secure Hash Functions
  - Secure Hash Algorithms
  - SHA-2-512 Case Study: Internal structure
  - Performance of Secure Hash Functions
  - Message authentication codes (MACs)
  - Confidentiality with Integrity using MACs
  - HMAC schemes
  - CMAC schemes

# HMAC (RFC 2104)

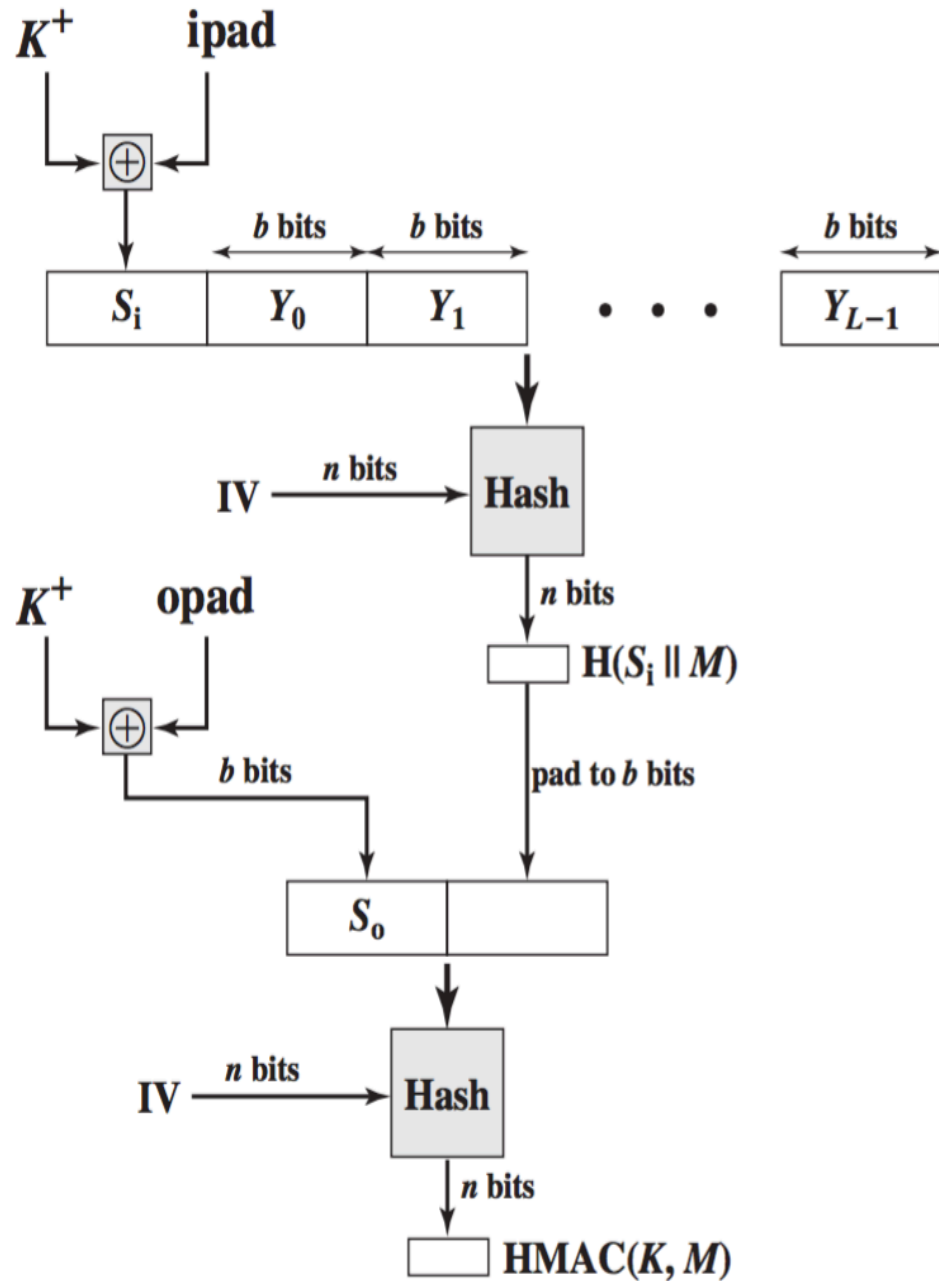
- It was born from the idea of a “keyed hash”
- Uses hash function on the message:  
$$HMAC_K = \text{Hash}[(K^+ \text{ XOR opad}) || \text{Hash}[(K^+ \text{ XOR ipad}) || M]]$$
- Where  $K^+$  is the key padded out to size; opad and ipad are specified padding constants
- Overhead is just 3 more hash calculations than the message needs alone
- Flexibility and Security: Any hash function can be composed



# HMAC Structure (RFC 2018)

## Initial Motivations:

- Cryptographic hash functions executes faster in software than encryption algorithms such as DES
- Library code for cryptographic hash functions is widely available
- No export restrictions from the US



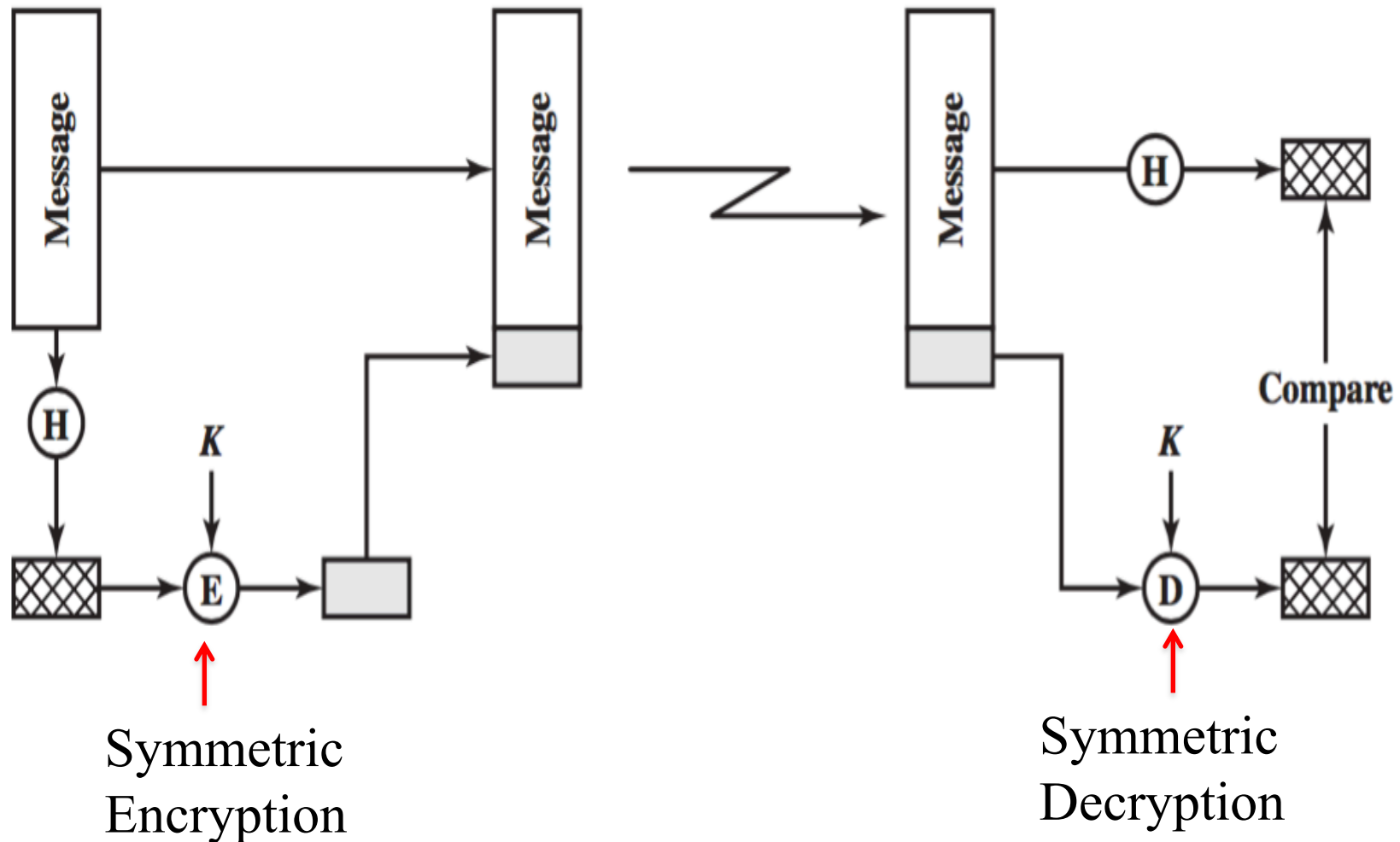
# HMAC Security vs. Speed

- 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 verses security constraints (Parameterized Constructions)
- HMAC very used for "fast (or light-weight) signatures" (comparing with CMACs or PubKey Signatures)
- But...
  - CMAC with AES 256 can run faster than HMAC with SHA-3
  - AES implementations also provided in Hardware (cryptographic modules)
  - And in some processors (ex., INTEL, Advanced Encryption Standard New Instructions; AES-NI)
  - [https://en.wikipedia.org/wiki/AES\\_instruction\\_set](https://en.wikipedia.org/wiki/AES_instruction_set)

# Outline

- **Secure Hash Functions and Message Authentication**
  - Secure Hash Functions and Their Use
  - Security Properties of Secure Hash Functions
  - Secure Hash Algorithms
  - SHA-2-512 Case Study: Internal structure
  - Performance of Secure Hash Functions
  - Message authentication codes (MACs)
  - Confidentiality with Integrity using MACs
  - HMAC schemes
  - CMAC schemes

# CMAC: MACs w/ Symmetric Encryption Alg.



# Block Ciphers used as Hash Functions

- Without secret keys: instead using the message blocks as "keys"
- can use block ciphers as hash functions
  - using  $H_0=0$  and zero-pad of final block
  - compute:  $H_i = E_{M_i} [H_{i-1}]$
  - and use final block as the hash value
  - similar to CBC but "without" a key
- Problems ?
- resulting hash too small depending on the crypto algorithm used (ex., 64-bit using DES)
  - both due to direct birthday attack
  - and to "meet-in-the-middle" attack
- Other variants ... also susceptible to attack

# We must be careful [Jueneman, Matyas and Meyer]

Message Authentication, IEEE Communications, Sep 1988]

## Ex., Operation with CBC mode

- $M$  = set of words of (ex., 64 bits):  $X_1, X_2, \dots, X_n$  with some standard (known) padding
- $H(M) = X_{n+1} = X_1 \text{ xor } X_2 \text{ xor } X_3, \dots, X_{n-1} \text{ xor } X_n$
- $C = E(k, (M || H(M)))$ , using CBC
- $C = Y_1, Y_2, Y_3, \dots, Y_n$

From the CBC mode we know that:

$$X_1 = IV \text{ xor } D(k, Y_1), \quad X_i = Y_{i-1} \text{ xor } D(k, Y_i)$$

$$H(M) = X_{n+1} = Y_n \text{ xor } D(k, Y_{n+1})$$

$$= X_1 \text{ xor } X_2 \text{ xor } \dots \text{ xor } X_n$$

$$= [IV \text{ xor } D(k, Y_1)] \text{ xor } [Y_1 \text{ xor } D(k, Y_2)] \text{ xor } \dots \\ \text{ xor } [Y_{n-1} \text{ xor } D(k, Y_n)]$$

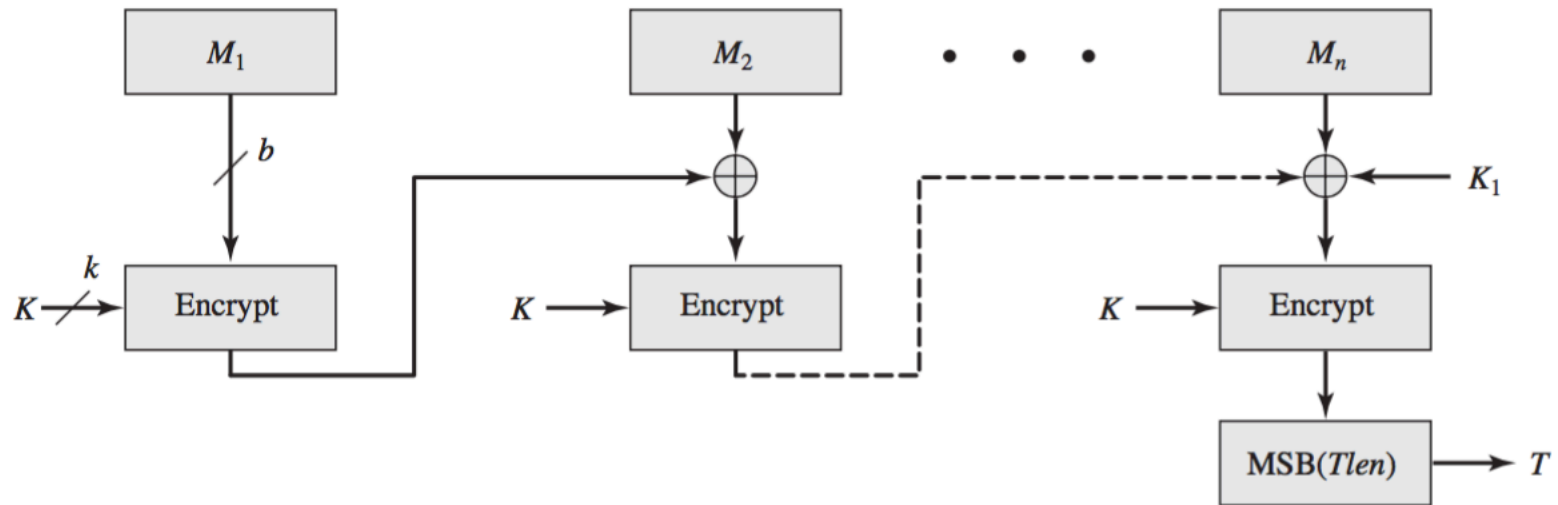
So we need  
**SECURE  
HASH  
FUNCTIONS**

Problem ? The hash is the same if you permute the cipher text blocks (non collision resistance) !!!

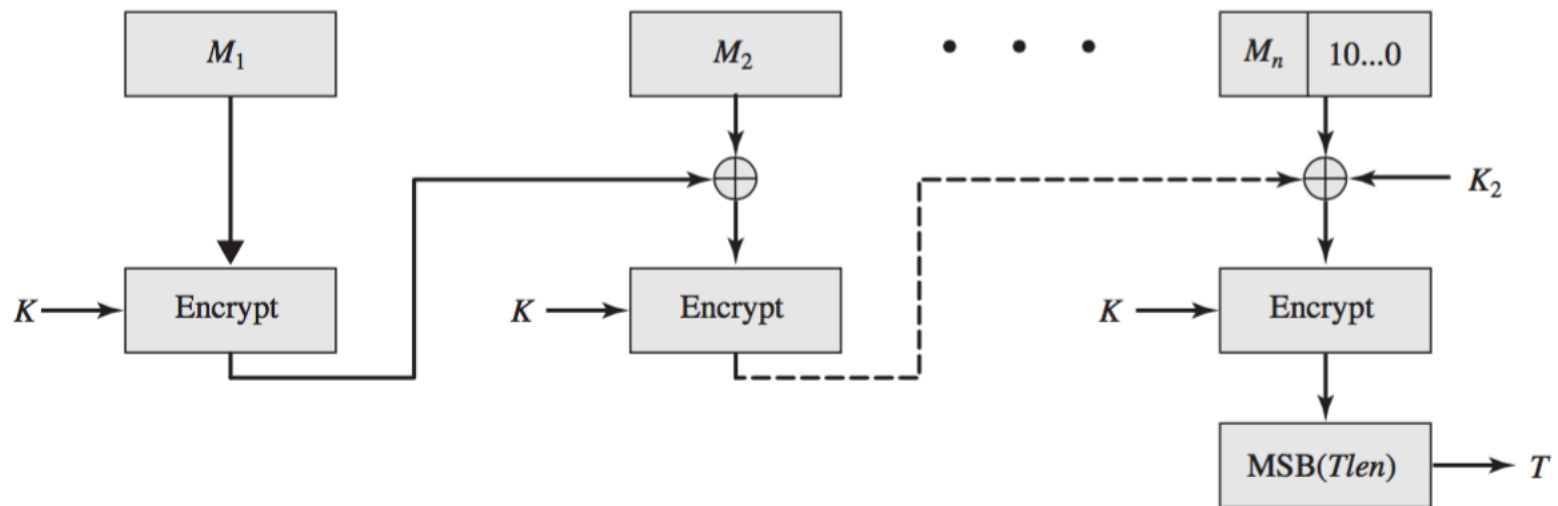
# CMAC based schemes

- Can use any block cipher chaining mode and use final block as a MAC
- **Data Authentication Algorithm (DAA)** was a widely used MAC based on DES-CBC (scheme known as CMAC with DES)
  - 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 ( $16 \leq M \leq 64$ ) of final block

# CMAC computations



(a) Message length is integer multiple of block size



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



# DAA - Data Authentication Algorithm

FIPS PUB 113 / ANSI X.917 MAC (with DES-CBC)

... final MAC with DEC CBC is today considered too small for security

In general, is possible to use CMACs with other symmetric algorithms to have more strong CMACs

# CMAC (NIST improvement 800-38B)

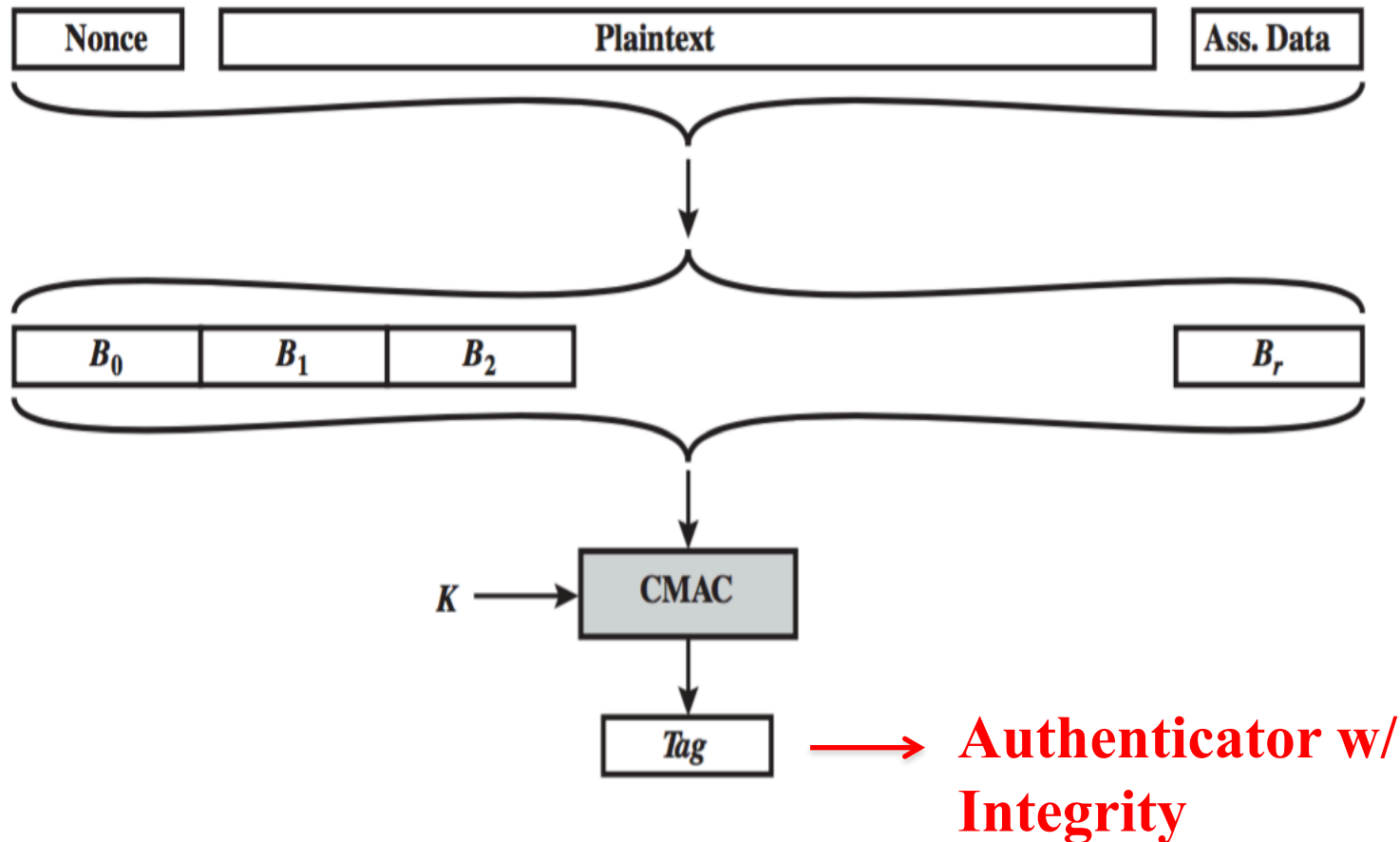
- Previously saw the DAA (CBC-MAC), widely used in govt & industry
  - But has message size limitation (DES and DES blocks)
  - AdHoc practices with other algorithms
- NIST standardization
- Can overcome using 2 keys & padding (use of Triple DES with two keys), thus forming a more secure Cipher-based Message Authentication Code (CMAC)
  - adopted by NIST SP800-38B for use with AES and Triple DES

# Counter with CBC MAC (NIST SP 800-38C)

- It is known as: *CCM Mode*
- A more recent scheme used for the combination of encryption (confidentiality) and message authentication (authentication + integrity)

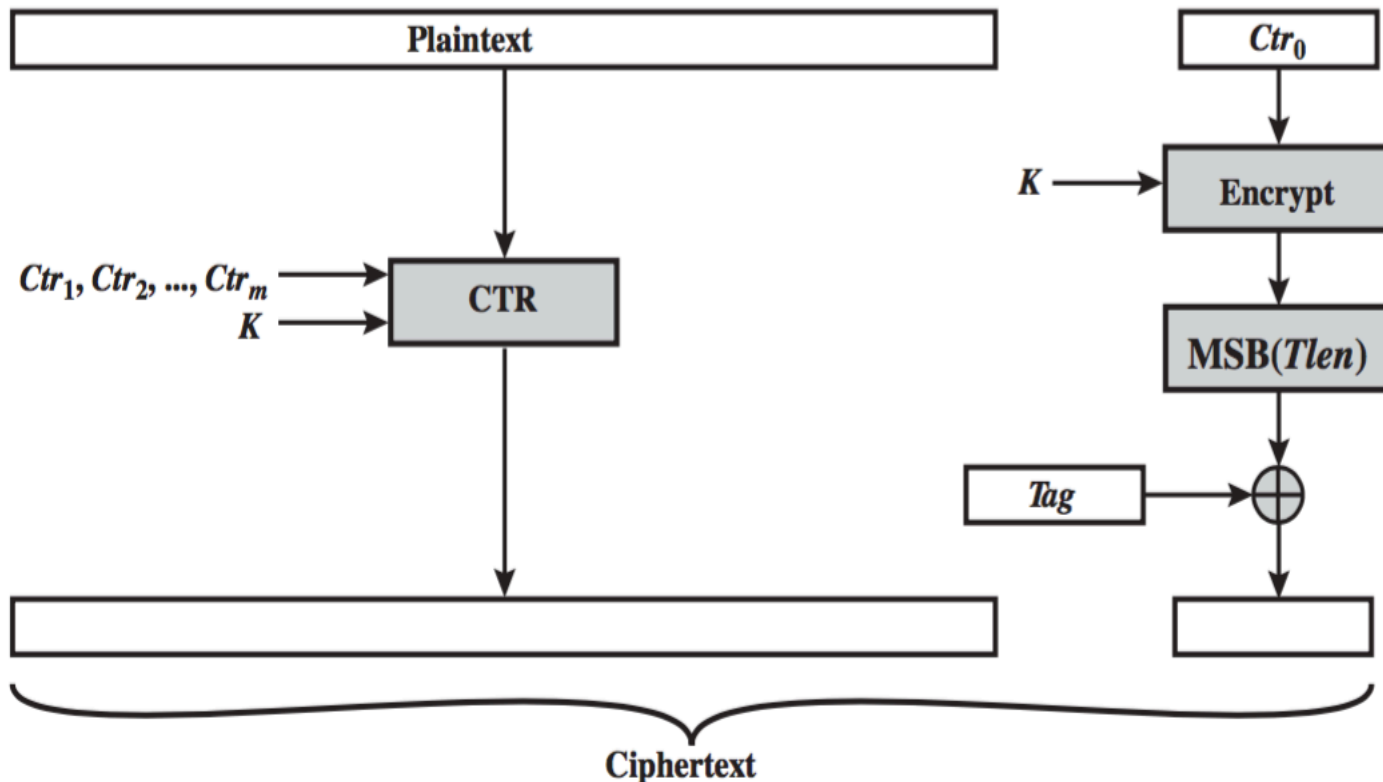
# Counter with CBC MAC (NIST SP 800-38C)

- For Authentication + Integrity



# Counter with CBC MAC (NIST SP 800-38C)

- Encryption with CCM



**Confidentiality (with internal CMAC proof)**

# GCM

- Galois Mode is also a CMAC Construction combined together with encryption
- Remember the class of Symmetric Encryption
  - See Cipher Modes of Operation

# So ... In summary !

- We need secure hash-functions, cryptographically SAFE !
- Secure Hash Functions: Specific Family of Cryptographic Algorithms, for this specific purpose : secure hashing ! (not-keyed)
- Implementing the security properties as initially stated !
- Keyed-Hash functions, equivalent to Message Authentictipn Codes (MACs), can use both:
  - Secure Hash Functions: HMACs
  - Symmetric Encryption Algorithms: CMACs

# We discussed all this outline ...

- **Secure Hash Functions and Message Authentication**
  - Secure Hash Functions and Their Use
  - Security Properties of Secure Hash Functions
  - Secure Hash Algorithms
  - SHA-2-512 Case Study: Internal structure
  - Performance of Secure Hash Functions
  - Message authentication codes (MACs)
  - Confidentiality with Integrity using MACs
  - HMAC schemes
  - CMAC schemes



# Revision: Suggested Readings



## Suggested Readings:

W. Stallings, Network Security Essentials - Applications and Standards, Chap 3., sections 3.1, 3.2

# Optional Complementary Readings



Optional / Complementary Readings:  
(Internals of Secure Hash Algorithms), more detail for  
those who are interested :

W. Stallings, Cryptography and Network Security - Principles  
and Practices, Pearson - Prentice Hall, Chap. 11 and 12