IT-Security 1

Chapter 3: Integrity

Prof. Dr.-Ing. Ulrike Meyer

WS 15/16

Chapter Overview

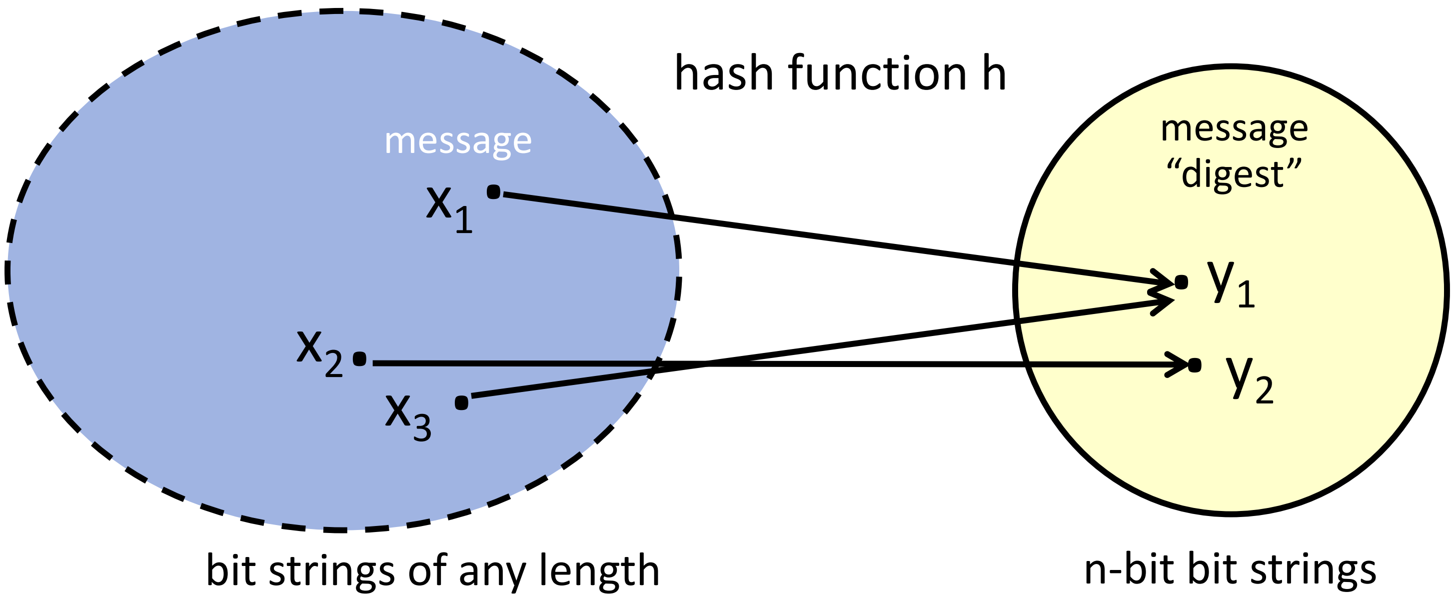
- Hash Functions in Cryptography
- Message Authentication Codes
  - From cryptographic hash functions
  - From block ciphers
- The replay problem
- Other applications of cryptographic hash functions

Definition of a Hash Function

- A hash function is a function h which has the following two properties
  - compression — h maps an input x of arbitrary finite bit-length, to an output h(x) of fixed bit-length n.
  - ease of computation — given h and an input x, h(x) is easy to compute

- A collision of a hash function is a pair of inputs x_1, x_2 that hash to the same value $h(x_1) = h(x_2)$

Hash Functions

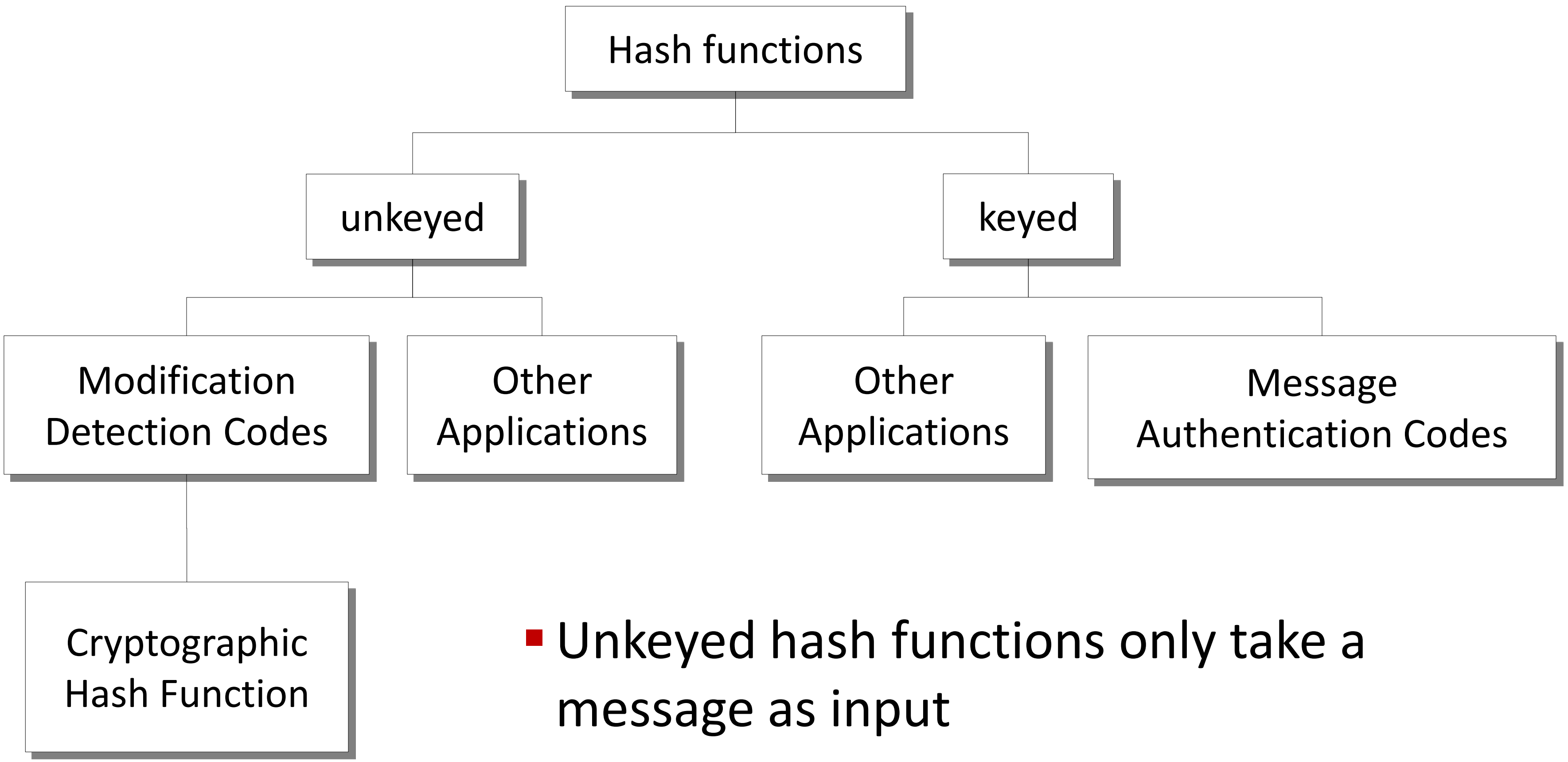


- The output of a hash function is called hash value, message digest, or fingerprint
- Note: a hash function cannot be injective
- Every hash function has collisions

Pigeonhole Principle

- Simple case:
  - If **n** pigeonholes are occupied by **n+1** pigeons, then at least one hole is occupied with more than one pigeons
- Generalization:
  - If **n** pigeonholes are occupied by **kn+1** pigeons, then at least one pigeonhole is occupied with more than **k** pigeons
- This gives a feeling for the number of collisions of a hash function
  - If a hash function maps 6-bit messages on 4-bit digests, than 64 messages are mapped on 16 possible digests
  - At least one digest corresponds to four or more messages

Types of Hash Functions



- Unkeyed hash functions only take a message as input
- Keyed hash functions take a message and a secret key as input

Potential Properties of Hash Functions

- Preimage resistant
  - Given y = h(x) but not x it is computationally infeasible to find any pre-image x’ with h(x’) = y
- Second preimage resistant
  - Given x, h(x) it is computationally infeasible to find a second pre-image x’ different from x with h(x’) = h(x)
- Collision resistant
  - It is computationally infeasible to find any two different inputs x, x’ that hash to the same value, i.e. such that h(x) = h(x’)
- A hash function with these three properties is also called cryptographic hash function

Relations Between the Terms

- Collision resistance $\Rightarrow$ 2nd pre-image resistance
- 2nd pre-image resistance $\not\Rightarrow$ collision resistance
- Collision resistance $\not\Rightarrow$ pre-image resistance
- 2nd pre-image resistance $\not\Rightarrow$ pre-image resistance
- Pre-image resistance $\not\Rightarrow$ 2nd pre-image resistance
- Pre-image resistance $\not\Rightarrow$ collision resistance

Example proofs

- Collision resistance $\Rightarrow$ 2nd pre-image resistance
  - Proof by contradiction
    - Assume there is a hash function **h** that is collision resistant but not 2nd pre-image resistant.
    - Then for some **x**, **h(x)** you can find a second pre-image **x'**. The pair (**x,x'**) is a collision. This contradicts the assumption
- Collision resistance $\not\Rightarrow$ pre-image resistance
  - Constructive proof
    - Assume **g** is a collision resistant **n**-bit hash function
    - Define $$\mathbf{h(x)} = \begin{cases} \mathbf{1 \| x} \text{ if the bitlength of } \mathbf{x} \text{ is } \leq \mathbf{n} \\ \mathbf{0 \| g(x)} \text{ if the bitlength of } \mathbf{x} \text{ is } > \mathbf{n} \end{cases}$$
    - Then **h(x)** is collision resistant but not pre-image resistant
  - This also proofs 2nd pre-image resistance $\not\Rightarrow$ pre-image resistance

Clarification of Terms

- One-way hash function:
  - = preimage resistant hash function
- Cryptographic hash function:
  - = preimage resistant + (second preimage resistant) + collision resistant
- Secure hash function:
  - = cryptographic hash function
- Second preimage resistant:
  - = weak collision resistant
- Collision resistant:
  - = sometimes called strong collision resistant

Random Oracle Model

- Mathematical model for an ideal cryptographic hash function
  - Upon receipt of a new message of any length the oracle randomly chooses a fixed-length message digest, records the message and the digest, and returns the digest
  - Upon receipt of a message for which a digest has already been recorded by the oracle the oracle returns the digest in the record
  - The digest for a new message is chosen independently at random from any previously chosen digest

$\xrightarrow{m_1}$

$\xleftarrow{h(m_1)}$

$\vdots$

$\xrightarrow{m_5}$

$\xleftarrow{h(m_5)}$

$\xrightarrow{m_2}$

$\xleftarrow{h(m_2)}$

Oracle chooses hash values randomly and independently

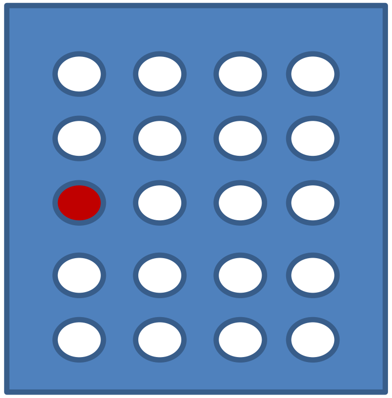
Use of Random Oracle Model

- The ideal hash function generated by the oracle is as...
  - preimage resistant
  - 2nd-preimage resistant
  - collision resistant
- … as possible
- So we can use it to determine how resistant a hash function can be at most

The Birthday Problems

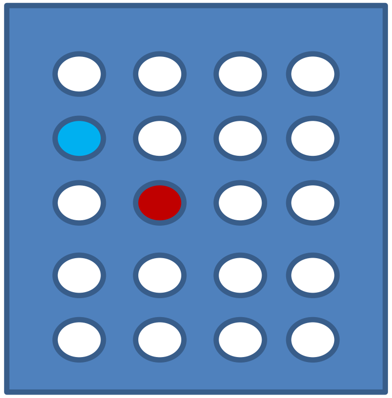
- Question: what is the minimal number k of students we need to have in a room to solve the following problems with a probability P

1st problem



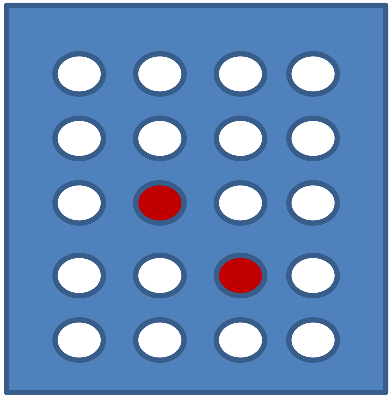
Fix a birthday
find a student
that is born
on this day

2nd problem



Select a student
and find a second
one with the same
birthday

3rd problem



Find any two
students with the
same birthday

1^{st} Birthday Problem

Problem:

- What is the minimum number **k** of students in a classroom such that with probability P at least one student has a predefined day as his/her birthday
  - N=365 uniformly distributed random values

Solution:

Probability	k	k for P = 1/2	# of students N = 365, P=1/2
$P \sim 1 - e^{-k/N}$	$k \sim \ln[1/(1-P)]\ N$	$k \sim 0.69\ N$	253

1^{st} Birthday Problem - Proof

Problem:

- What is the minimum number **k** of students in a classroom such that with probability P at least one student has a predefined day as his/her birthday
  - N=365 uniformly distributed random values

Proof:

- If we chose one student randomly, than the probability that he does not have the fixed birthday is $1 – 1/N$
- The probability that NO student has the fixed birthday is $(1-1/N)^k$
- The probability that at least one student has the fixed birthday is $1 – (1-1/N)^k$
- Using the approximation $1-x \sim e^{-x}$ (for $x << 1$) gives us the probability
  - $P \sim 1- e^{-k/N}$

2^{nd} and 3^{rd} Birthday Problem

Problem 2:

What is the minimum number k of students such that with probability P at least one student has the same birthday as one particular pre-selected student selected by the professor?

Probability	k	k for P = 1/2	# of students N = 365, P=1/2
$P \sim 1 - e^{-(k-1)/N}$	$k \sim \ln[1/(1-P)]\ N + 1$	$k \sim 0.69\ N + 1$	254

Problem 3: What is the minimum number k of students in a classroom such that with a probability of P at least two students have the same birthday?

Probability	k	k for P = 1/2	# of students N = 365, P=1/2
$P \sim 1 - e^{-(k-1)/2N}$	$k \sim (2 \ln[1/(1-P)]\ N)^{1/2}$	$k \sim 1.18\ N^{1/2}$	23

How Preimage Resistant Can a Hash Function be?

- Brute force attack to find pre-image to a given digest y
  - Attacker randomly selects x', hashes it and compares the result to y until he finds a pre-image
- How many hashes does the attacker have to compute to be successful with probability ½ ?
- Using the first birthday problem
  - For an n-bit hash function the attacker has to calculate $0.69 * 2^n$ hashes

- O(2^n) hash computations necessary to find a pre-image with probability ½ on an ideal hash function

How 2^{nd} preimage Resistant Can a Hash Function Be

- Brute force attack to find a 2^{nd} preimage for a pair x, h(x)
  - Attacker randomly selects message x', hashes it and compares the result to h(x)
  - Attacker returns x' if h(x') = h(x)
- How many hashes does the attacker have to compute to find a 2^{nd} preimage with probability of ½
- Using the 2^{nd} birthday problem
  - $0.69 * 2^{n} + 1$ hashes need to be calculated to brute force a 2^{nd} preimage of an n-bit hash function

➢ $O(2^{n})$ hash computations required to find a 2^{nd} preimage on an ideal hash function with probability ½

How Collision Resistant Can a Hash Function be?

- Brute forcing a collision
  - Attacker tries to find two messages x, x' that hash to the same value by randomly create messages x
  - Compute the hash of x and store the hashes in a list
  - Compare a new hash with each hash already stored in the list
  - Return the messages if the hashes coincide
- How many hashes does the attacker have to compute to find a collision with probability ½?
- Using the third birthday problem the number of hashes needs to be $1.18 * 2^{n/2}$

➢ $O(2^{n/2})$ hash computations required to find a collision for the ideal hash function with probability ½

Examples for Hash functions

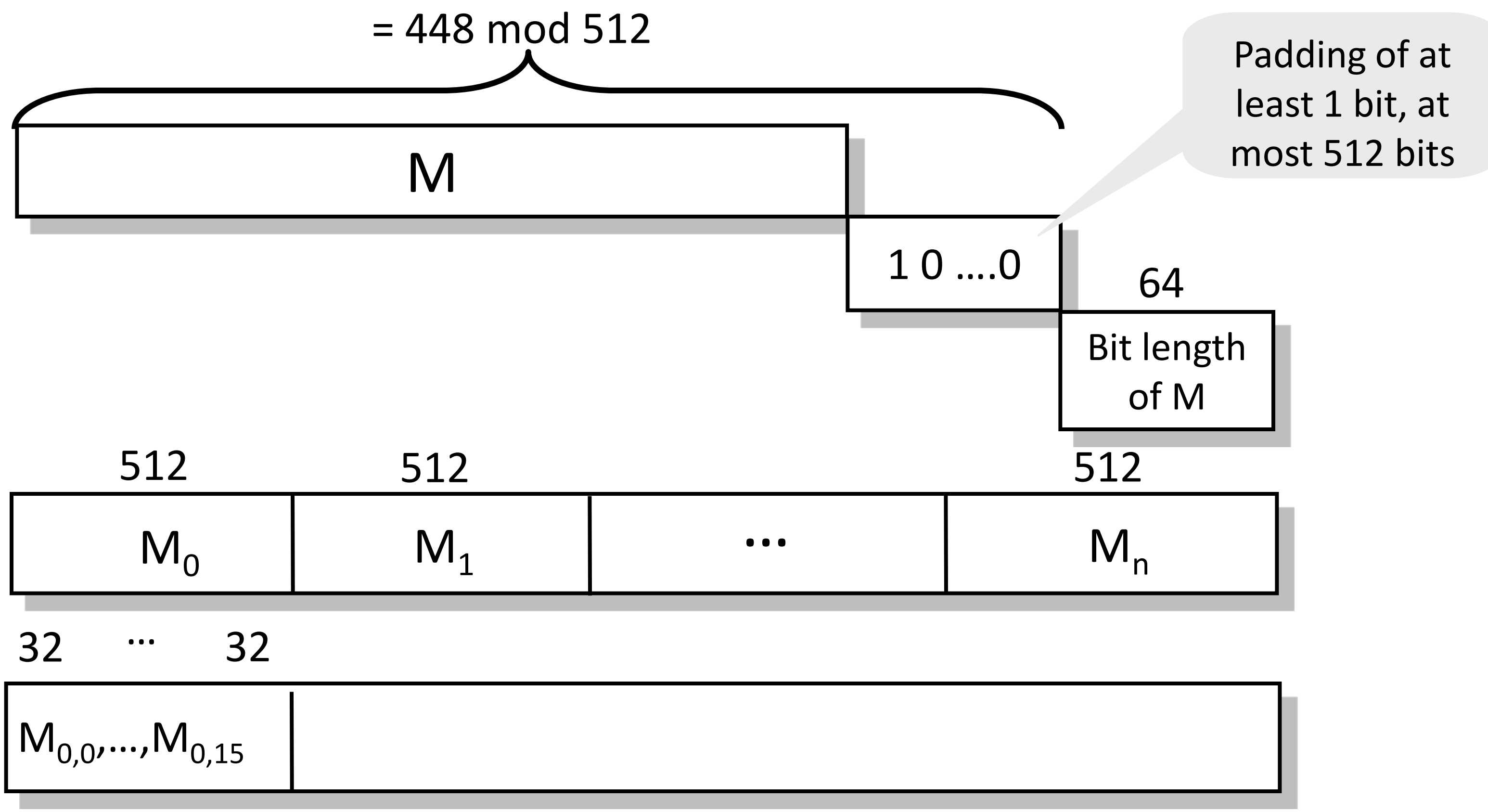
- MD5
  - Designed by Ronald Rivest (the R in RSA) in 1991
  - Described in RFC 1321
  - Produces a hash of 128 bit
- SHA / SHA-1
  - Designed by the NSA in 1993
  - Updated to SHA-1 in 1995
  - Defined in FIPS PUP 180-2
  - SHA-1 produces a hash of 160 bit
- SHA-2
  - Computations of longer hashes possible with SHA-256, SHA-384, SHA-512
- SHA3

MD-5 Overview

- Pad the message such that bit-length is a multiple of 512
- Initialize four word buffers
- Initialize a 64 elements table T[1],.... ,T[64]
- Shuffle each 512 bit block in 4 rounds and use the output as input to the next round

Padding and Splitting into Blocks

- Let M be the message to be hashed



Initialization and Auxiliary Functions

Initialization of four word buffers

- Word A: 01 23 45 67
- Word B: 89 ab cd ef
- Word C: fe dc ba 98
- Word D: 76 54 32 10

Auxiliary functions operating on 32 bit words

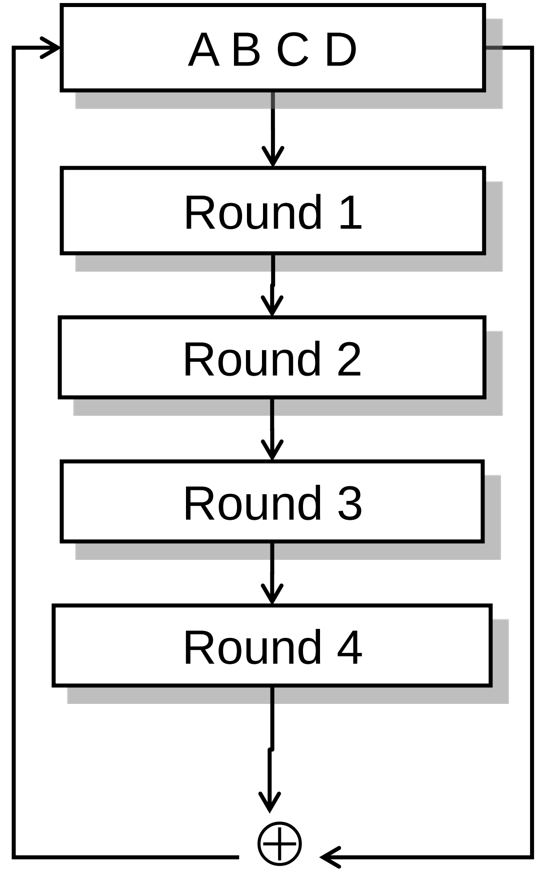
- F(X,Y,Z) = XY v not(X) Z (bitwise “if X then Y else Z”)
- G(X,Y,Z) = XZ v Y not(Z)
- H(X,Y,Z) = X $\oplus$ Y $\oplus$ Z
- I(X,Y,Z) = Y $\oplus$ (X v not(Z))

Initialisation of T-table

- For i = 1,…,64: T[i] = integer part of (2^{32} |sin(i)|),
  where |sin(i)| is the absolute value of sin(i)

MD 5 – Word Processing Rounds

- The word procession takes place for each message block $M_1, \ldots, M_n$
- In each round all sub blocks of the current message block are used
- The sub blocks are denoted by X[i]



X[i]= $\mathbf{M_{i,j}}$, where j = 0,…,15

Round 1

- [abcd **k** **s** i] is the operation
  - a = b + ((a + **F(b,c,d)** + X[**k**] + T[i]) <<< **s**)
- Do the following 16 operations.
  - [ABCD **0** **7** 1] [DABC **1** **12** 2] [CDAB **2** **17** 3]
  - [BCDA **3** **22** 4] [ABCD **4** **7** 5] [DABC **5** **12** 6]
  - [CDAB **6** **17** 7] [BCDA **7** **22** 8] [ABCD **8** 7 9]
  - [DABC **9** **12** 10] [CDAB **10** **17** 11] [BCDA **11** **22** 12]
  - [ABCD **12** **7** 13] [DABC **13** **12** 14] [CDAB **14** **17** 15]
  - [BCDA **15** **22** 16]

Round 2

- Let [abcd k s i] denote the operation
  - a = b + ((a + **G(b,c,d)** + X[k] + T[i]) <<< s)
- Do the following 16 operations
  - [ABCD 1 5 17] [DABC 6 9 18] [CDAB 11 14 19]
  - [BCDA 0 20 20] [ABCD 5 5 21] [DABC 10 9 22]
  - [CDAB 15 14 23] [BCDA 4 20 24] [ABCD 9 5 25]
  - [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]
  - [ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31]
  - [BCDA 12 20 32]

Round 3

- Let [abcd **k** **s** **t**] denote the operation
  - a = b + ((a + **H(b,c,d)** + X[**k**] + T[**i**]) <<< **s**)
- Do the following 16 operations
  - [ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35]
  - [BCDA 14 23 36] [ABCD 1 4 37] [DABC 4 11 38]
  - [CDAB 7 16 39] [BCDA 10 23 40] [ABCD 13 4 41]
  - [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]
  - [ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47]
  - [BCDA 2 23 48]

Round 4

- Let [abcd **k** **s** i] denote the operation
  - a = b + ((a + **I(b,c,d)** + X[**k**] + T[i]) <<< **s**)
- Do the following 16 operations
  - [ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51]
  - [BCDA 5 21 52] [ABCD 12 6 53] [DABC 3 10 54]
  - [CDAB 10 15 55] [BCDA 1 21 56] [ABCD 8 6 57]
  - [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]
  - [ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63]
  - [BCDA 9 21 64]

How Secure is MD5?

- 1993: Collision found by Boer and Bosselaers
- 1996: Attack published that found a collision in a modified version of MD5 in which the words A, B, C, and D were initialized differently
- 2004: Wang et al. found collisions in MD5 and other hash functions
- 2005: Further enhanced to make collision finding feasible on a notebook (8 hours to find a collision)
- 2006: Black et al. implemented a toolkit for collisions in MD5
  - http://www.cs.colorado.edu/~jrblack/md5toolkit.tar.gz
- 2007: Stevens et al. find collisions in less than 10 seconds on a on a 2.6Ghz Pentium 4
- 2007, 2009: MD5 attacks successfully used to fake certificates
- IETF recommendation of March 2011: MD5 should not be used any more where collision resistance is needed

Security of SHA-1

- 2004: 2^{nd} preimage attack on SHA-1 in 2^{106}
- 2005: Attack found by Wang et al. that finds a collision with 2^{69} hash operations
- 2013: Attack by Stevens et al. finds identical prefix collision in 2^{61} and chosen prefix collision in $2^{77.1}$
- 2015: Attack by Stevens et al. that finds a Free-Start Collision on 76-step SHA-1 in 2^{50} hash operations
- Very serious situation, SHA-1 will be phased out in 2016 by all major browsers
- However:
  - No feasible preimage attack or 2^{nd} preimage attack yet
  - No feasible attacks on the other SHA versions yet

How Bad is That?

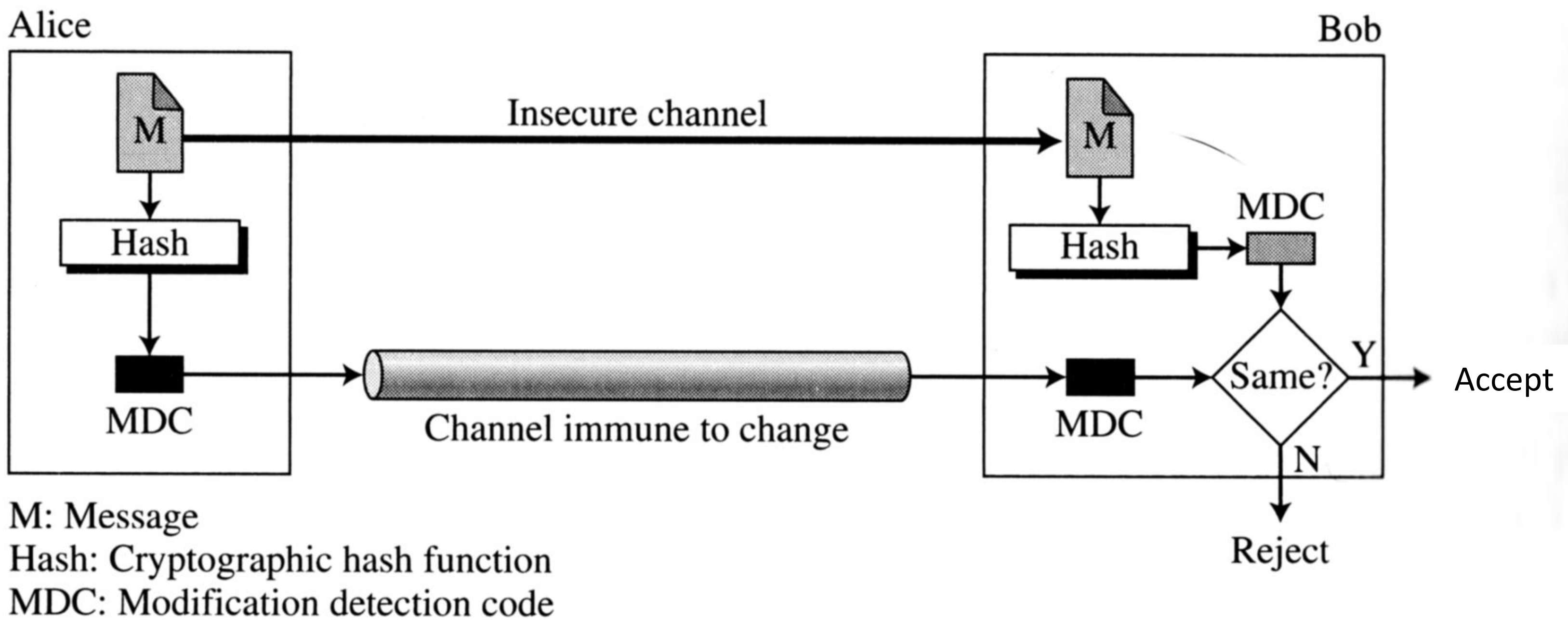
- As we will see later on hash functions are used for
  - Digital signatures on messages and certificates
    - Messages are hashed before they are signed
  - Constructing message authentication codes
  - Constructing modification detection codes
  - Key derivation functions

- Only in the context of digital signatures collision resistance is of importance
  - For the other applications preimage and 2nd preimage resistance are more interesting
  - “Its time to walk, but not run, to the fire exit” Jon Callas PGP's CTO

NIST Call for SHA-3

- November 2007: Call for a new hash function
- October 2008: Deadline for submissions
  - 64 submissions, 51 selected for the first round
- July 2009: Fourteen second round candidates announced
- December 2010: Five candidates for the third and final round determined: BLAKE, Grøstl, JH, Keccak, and Skein
- Final decision announced on October 2nd 2012
  - Keccak was slelected as SHA-3
- See http://keccak.noekeon.org/
- Since early 2015, Keccak is standardized in FIPS 202
  - http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

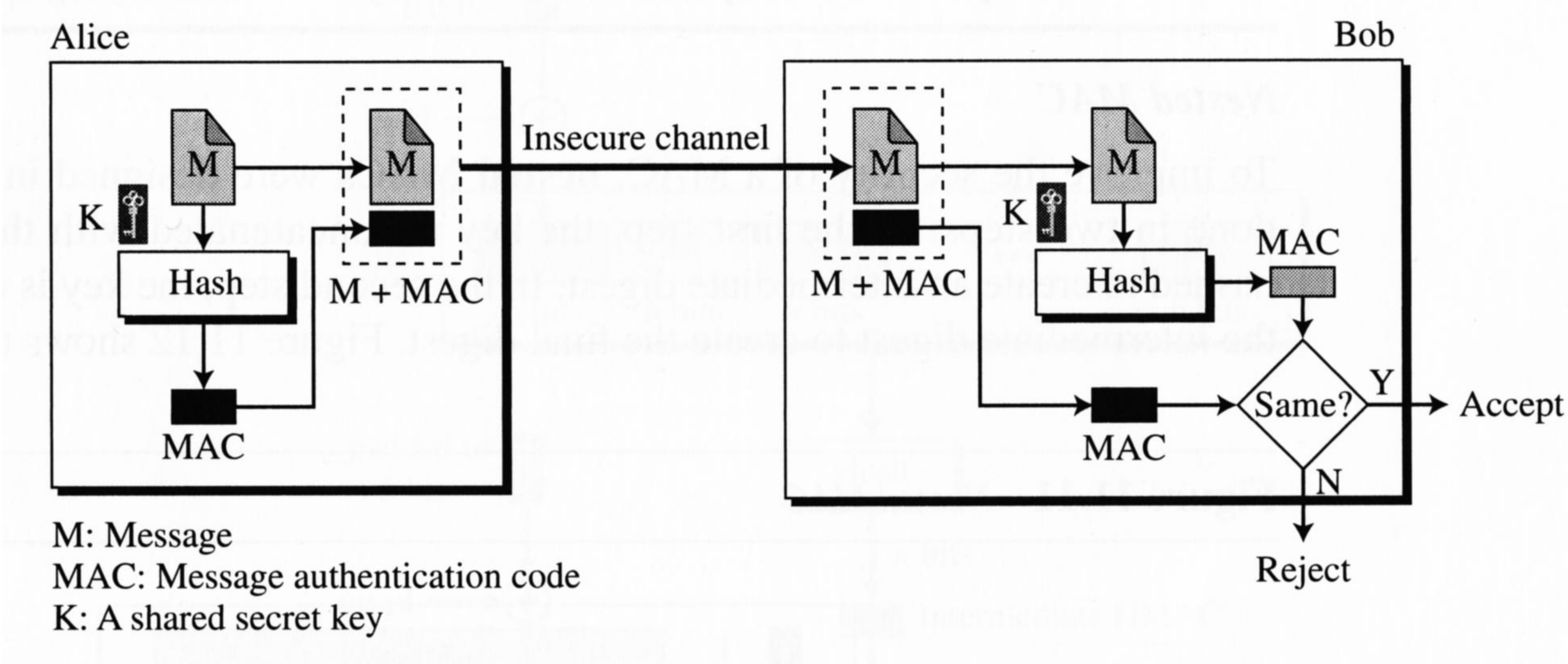
Modification Detection Code



Definition of Message Authentication Codes

- A Message Authentication Code (MAC) is a family of functions $\mathbf{h_k}$ parameterized by a secret key **k** with the following properties
  - Ease of computation – given **k** and **x**, $\mathbf{h_k(x)}$ is easy to compute.
  - Compression – $\mathbf{h_k}$ maps an input **x** of arbitrary finite bit-length to an output $\mathbf{h_k(x)}$ of fixed bit-length **n**
  - Computation resistance – for every **k** and any given amount of pairs $\mathbf{(x_i, h_k(x_i))}$ it is computationally infeasible to compute any pair $\mathbf{(x, h_k(x))}$ with **x** different from all $\mathbf{x_i}$ without knowledge of **k**

Message Authentication Codes from Hash Functions



- Hash functions are not the only way to construct MACs!

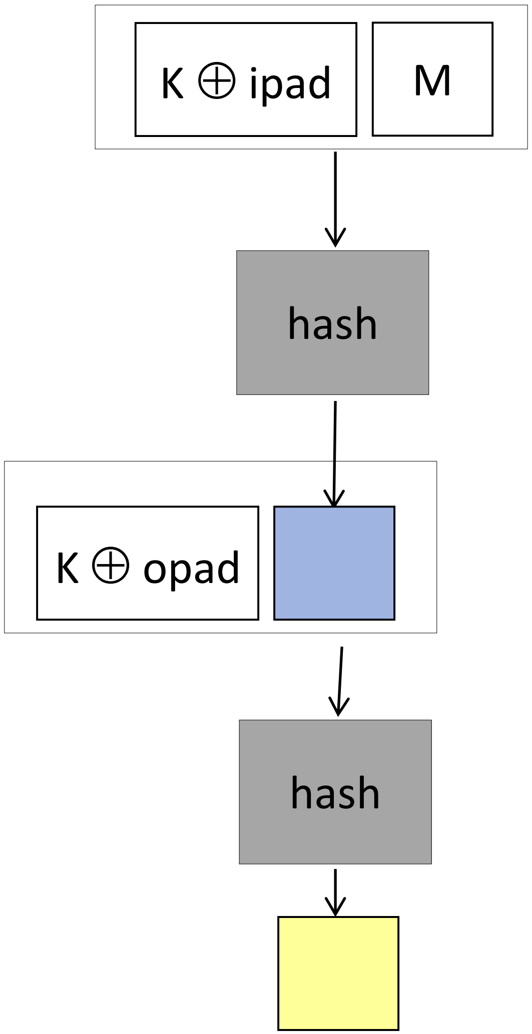
How Should a Hash Function be Keyed?

- For some hash functions simple constructions lead to insecure MACs, e.g.
  - h(k||m) leads to insecure MACs for certain hash functions
  - E.g. insecure with MD5!
- For other constructions it is unclear whether they are secure or not
- Idea: find a construction such that the security of the MAC entirely depends on the properties of the hash function
  - Regardless of the hash function used
- HMAC meets this goal

HMAC

- Construct MAC by applying a cryptographic hash function to message and key
  - Can also use encryption to construct MACs (see CMAC below) but…
  - Hashing is faster than encryption in software
  - Library code for hash functions widely available
  - Can easily replace one hash function with another
  - There used to be US export restrictions on encryption
- Invented by Bellare, Canetti, and Krawczyk (1996)
- E.g. mandatory for IP security, also used in SSL/TLS

HMAC: High-level view



- HMAC is constructed from a cryptographic hash function h
- HMAC can be proven to be as secure as the underlying hash function, i.e. HMAC is computation resistant if the underlying hash function is a cryptographic hash function
- $HMAC(M) = h(K \oplus opad \mid\mid h(K \oplus ipad \mid\mid M))$ with constant values for opad and ipad
- Rational for applying the hash twice
  - Attacker cannot control input to the outer hash function
- Rational for the constants
  - Ensure that different keys are used for the inner and outer hash computation, ipad and opad chosen such that their Hamming distance is maximal

CMAC: Computing MAC from Block Cipher

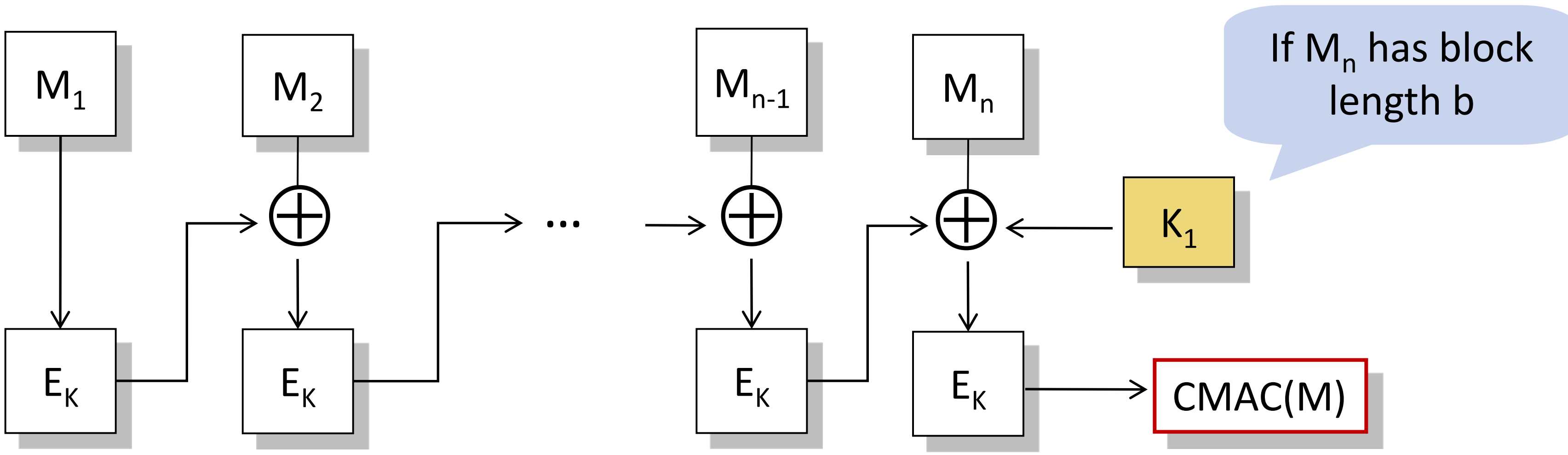
- CMAC uses a block cipher E_K of block length b = 64 or b = 128
- A message M is split into n blocks of length b:

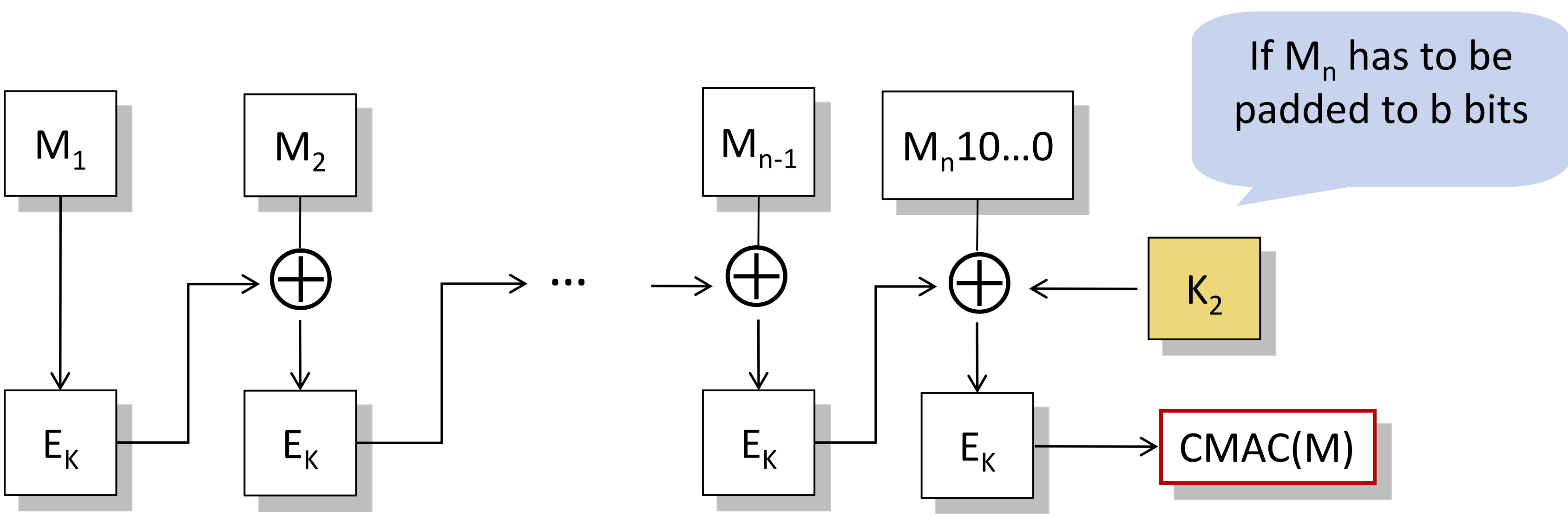
  $M = M_1 \,||\, M_2 \,||\, ... \,||\, M_n$
- If the last block M_n is not of length b it is padded with 10…0 until it is b bits long
- CMAC computation is equivalent to applying CBC Mode to the message except that the last block is additionally masked with a sub-key K1 if M_n is of bit length b and with a sub-key K2 if M_n was padded to be of full bit length b

CMAC Sub-key Computation

- Let $L = E_K(0^b)$, where 0^b is the bit string of b zeros
- Let $R_{128} = 0^{120}10000111$, $R_{64} = 0^{59}11011$
- Then K1 is computed by
  - If $MSB_1(L) = 0$, $K1 = L<<1$
  - Else $K1 = L \oplus R_b$
- K2 is computed by
  - If $MSB1(K1) = 0$, $K2 = K1<<1$
  - Else $K2 = (K1<<1) \oplus R_b$

CMAC computation





Why the Masking?

- Using a “pure” CBC-MAC allows for forgery in some specific settings
  - i.e. without the masking by K1 or K2 in the last step and an IV of 0^b
- For example, let M and P be two one-block messages, then a pure CBC-MAC would lead to:
  - $MAC(M) = E_K(M)$
  - $MAC(P) = E_K(P)$
  - $MAC(M \,||\, (P \oplus MAC(M))) = E_K\,(E_K(M \oplus 0^b) \oplus P \oplus MAC(M)) = E_K(P \oplus MAC(M) \oplus MAC(M)) = E_K(P)$
- I.e. it is possible to forge MACs on longer messages from MACs of shorter messages observed
- The masking solves this problem

The Replay Problem

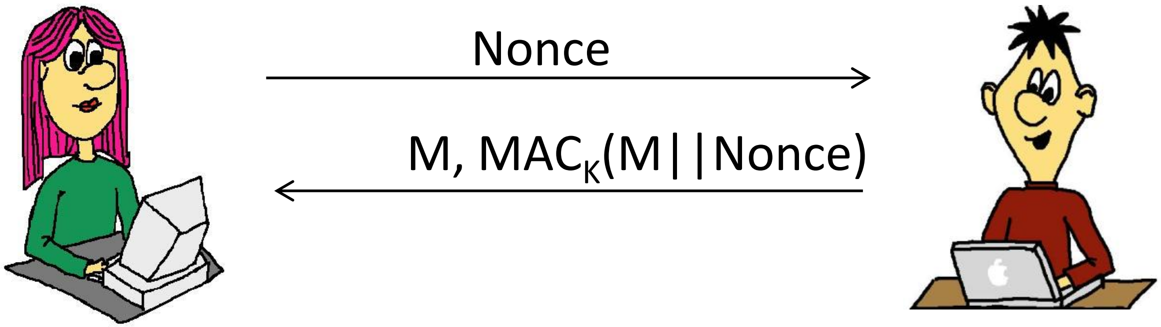
- Message authentication codes alone do not protect against a protected message being recorded and replayed at a later time
- Reply protection can be added to MACs with the help of additional input to the MAC:
  - Time stamps or other counters
  - Nonces
    - Nonce = Number used once

Replay Protection with Counters

- Examples for counters used in replay protection
  - time stamps
  - per session packet / frame counters
- Typical use
  - Sender increments counter on every packet he sends
  - Receiver accepts packet only if counter in packet received is larger then the counter in the last received packet
- Problem with counters
  - Require counter synchronization between sender and receiver
    - e.g. synchronized clocks, resetting the counters to some initial value for each new session
  - Using per-packet counters difficult if packets cannot be guaranteed to arrive in order

Replay Protection with Random Nonces

- Nonces are “numbers used once”
  - E.g. counters are actually nonces
- Random Nonces are randomly picked nonces
  - E.g. counters are *not* random nonces
- Typical use
  - Alice wants a guarantee that Bobs message is going to be fresh
  - She sends Bob a random nonce
  - Bob answers with the message concatenated with the nonce and a MAC on both



Integrity vs. Encryption

- Note: Although CMAC provides integrity protection, encryption alone does typically not provide integrity protection
  - Intuition: attacker may be able to modify message under encryption without learning what the message is
  - Example: if a stream cipher is used for encryption
    - Given a key stream K, encrypt M as $M \oplus K$
    - Attacker can replace M by $M \oplus M'$ for any M'
  - This is recognized by industry standards (e.g., PKCS)
    - "RSA encryption is intended primarily to provide confidentiality… It is not intended to provide integrity"
- If the plaintext is not recognizable non of the encryption modes can detect e.g. appending additional random ciphertext
- It is good practice ALWAYS to use different keys for integrity protection and encryption unless a special mode of operation is used, that provides both at once

Combining Integrity Protection and Encryption

- Encrypt, then MAC:
  - Encrypt the plaintext with K1, compute a MAC of the resulting ciphertext with K2, append MAC to ciphertext
- Encrypt and MAC:
  - Compute MAC of plaintext data with K2, encrypt only plaintext with K1, append MAC of plaintext to the resulting ciphertext
- MAC, then encrypt:
  - Compute MAC of plaintext with K2, encrypt plaintext || MAC with K1
- Use a mode of encryption that provides integrity protection as well, e.g.
  - Galoise Counter Mode (GCM) standardized by NIST
  - Counter mode with CBC MAC (CCM)

GCM

- Block-cipher-based MACs such as CBC-MAC or CMAC cannot be parallelized
- Counter mode makes any block cipher parallelizable and is therefore very efficient
- GCM is a mode of operation provides authentication and encryption while being parallelizable
- GCM can be used as MAC if no encryption is needed: GMAC
- GCM can use IVs of arbitrary length
- Easy to implement very efficiently in hardware
- Very good software performance

GCM Authenticated Encryption Operation (1)

- Takes 4 Input Values:
  - secret key **K**
  - Initialization vector **IV** of 1 to 2^{64} bit, recommended length: 96 bit
  - Plaintext **P** of length 0 to $2^{39} – 256$ bit
  - Additional authenticated data **A** of 0 to 2^{64} bit (will be authenticated but not encrypted)

- Ouputs
  - A ciphertext **C** of the same length as the **P**
  - An authentication tag **T** of **t** bit where **t** is 0 – 128

GCM Authenticated Encryption Operation (2)

$$
\begin{aligned}
H &= E(K, 0^{128}) \\
Y_0 &= \begin{cases} IV \| 0^{31}1 & \text{if } \mathrm{len}(IV) = 96 \\ \mathrm{GHASH}(H, \{\}, IV) & \text{otherwise.} \end{cases} \\
Y_i &= \mathrm{incr}(Y_{i-1}) \text{ for } i = 1, \ldots, n \\
C_i &= P_i \oplus E(K, Y_i) \text{ for } i = 1, \ldots, n-1 \\
C_n^* &= P_n^* \oplus \mathrm{MSB}_u(E(K, Y_n)) \\
T &= \mathrm{MSB}_t(\mathrm{GHASH}(H, A, C) \oplus E(K, Y_0))
\end{aligned}
$$

- IV is used as initial value for the counter value $\text{Counter}_i = Y_i$
- Y_i is used as input to the block cipher and resulting key block is xored with plaintextblock P_i to produce a ciphertext block C_i
- GHASH is defined on the next slide

$\mathrm{GHASH}(H, A, C) = X_{m+n+1}$

$$X_i = \begin{cases} 0 & \text{for } i = 0 \\ (X_{i-1} \oplus A_i) \cdot H & \text{for } i = 1, \ldots, m-1 \\ (X_{m-1} \oplus (A_m^* \| 0^{128-v})) \cdot H & \text{for } i = m \\ (X_{i-1} \oplus C_i) \cdot H & \text{for } i = m+1, \ldots, m+n-1 \\ (X_{m+n-1} \oplus (C_m^* \| 0^{128-u})) \cdot H & \text{for } i = m+n \\ (X_{m+n} \oplus (\mathrm{len}(A) \| \mathrm{len}(C))) \cdot H & \text{for } i = m+n+1. \end{cases}$$

$\oplus$ is the regular Xor operation

• is the multipilication in $GF(2^{128})$

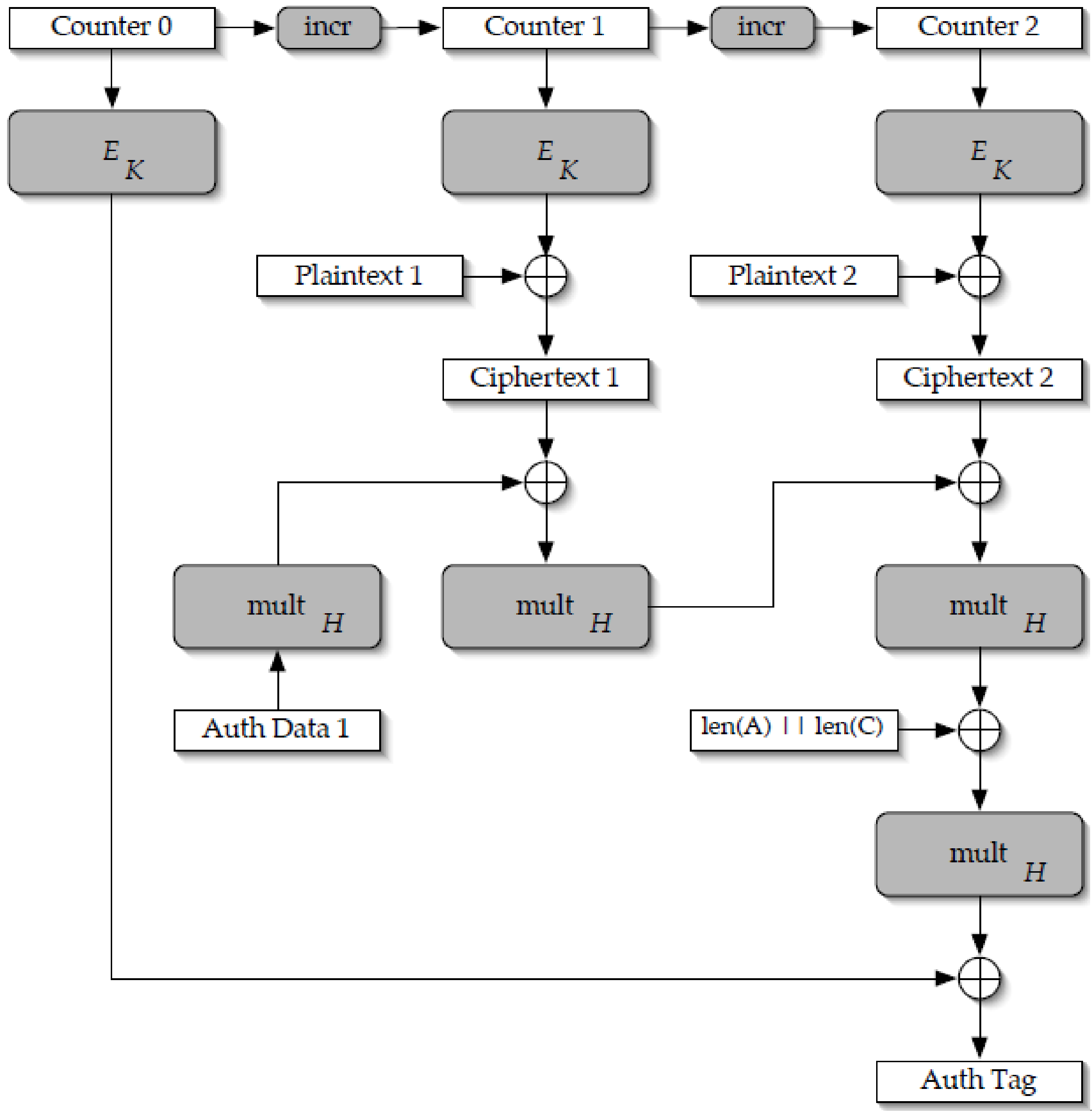
$H = E(K, 0^{128})$

Multiplication in $GF(2^{128})$

- $GF(2^{128})$ is the finite field with 2^{128} elements
- It is unique up to isomorphism
- GCM uses the irreducible polynomial $f(x) = 1 + x + x^2 + x^7 + x^{128}$
- Identify each 128 bit string $a = a_0 \ldots a_{127}$ with the polynomial $a(x) = \Sigma_{i=0,\ldots,127}\, a_i x^i$
- The addition of a and b in $GF(2^{128})$ is defined as coefficient-wise addition of the two polynomials which corresponds to bitwise xor of the bit representation
- Multiplication of a and b in $GF(2^{128})$ is then defined as bit string representation of a(x) b(x) mod f:

$$(\Sigma_{i=0,\ldots,127}\, a_i x^i)\ (\Sigma_{i=0,\ldots,127}\, b_i x^i) \mod f$$

Example Authenticated Encryption of GCM



GCM Authenticated Decryption Operation

$$H = E(K, 0^{128})$$

$$Y_0 = \begin{cases} IV \| 0^{31}1 & \text{if } \mathrm{len}(IV) = 96 \\ \mathrm{GHASH}(H, \{\}, IV) & \text{otherwise.} \end{cases}$$

$$T' = \mathrm{MSB}_t(\mathrm{GHASH}(H, A, C) \oplus E(K, Y_0))$$

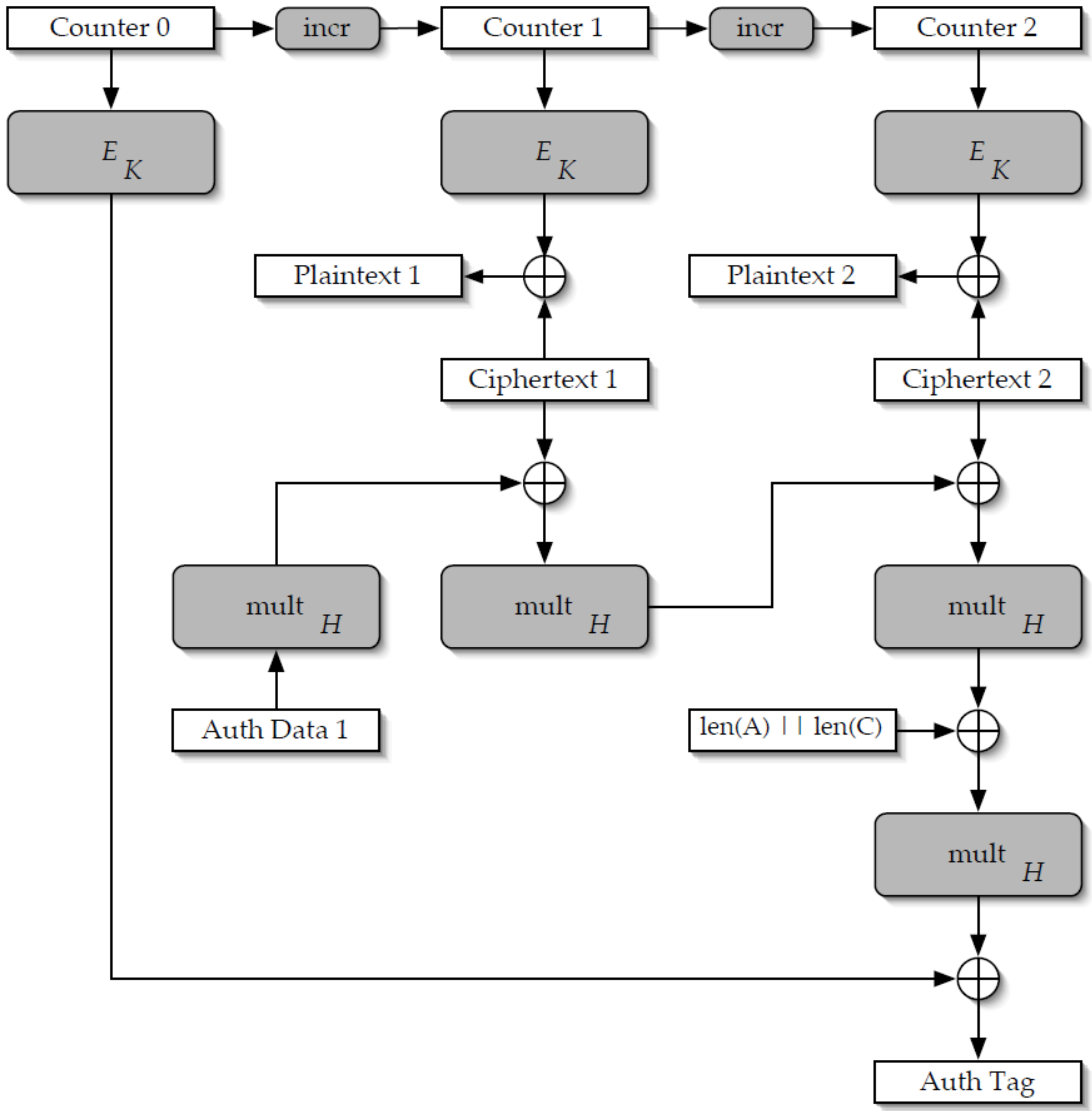
$$Y_i = \mathrm{incr}(Y_{i-1}) \text{ for } i = 1, \ldots, n$$

$$P_i = C_i \oplus E(K, Y_i) \text{ for } i = 1, \ldots, n$$

$$P_n^* = C_n^* \oplus \mathrm{MSB}_u(E(K, Y_n))$$

- Similar to encryption, but order of hash step and encryption step reversed
- T' is compared to T (sent along with the ciphertext) to check integrity, if T'≠ T

GCM Example Authenticated Decryption Operation



Reading

- Basic Reading
  - Forouzan, Introduction to cryptography and network security, Chapter 11
- Details on the mentioned algorithms
  - MD5: specified in RFC 1321 http://tools.ietf.org/html/rfc1321
  - SHA-1: specified in FIPS Pub 198-1
    - http://csrc.nist.gov/publications/PubsFIPS.html
  - CMAC: NIST Special Publication 800-38B
  - HMAC: specified in FIPS Pub 198-1
- Security Considerations for SHA-1 and MD5
  - SHA-1: http://tools.ietf.org/html/rfc6194
  - MD-5: http://tools.ietf.org/html/rfc6151
- NIST SHA-3 competition: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html
- Specification of GCM mode http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf
- Specification of CCM mode http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf