Introduction into Cyber Security

Chapter 3: Integrity Protection

WiSe 18/19

Chair of IT Security

Chapter Overview

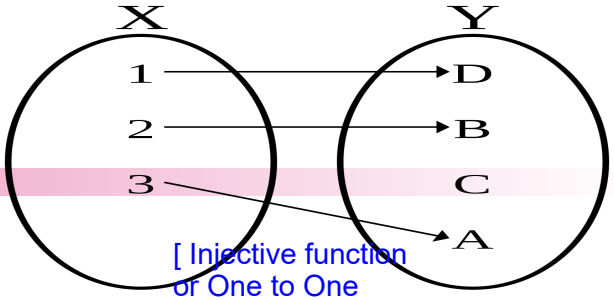
- Hash Functions in Cryptography
- Message Authentication Codes
  - From cryptographic hash functions
  - From block ciphers
- The replay problem
- Other applications of cryptographic hash functions

Hash Functions

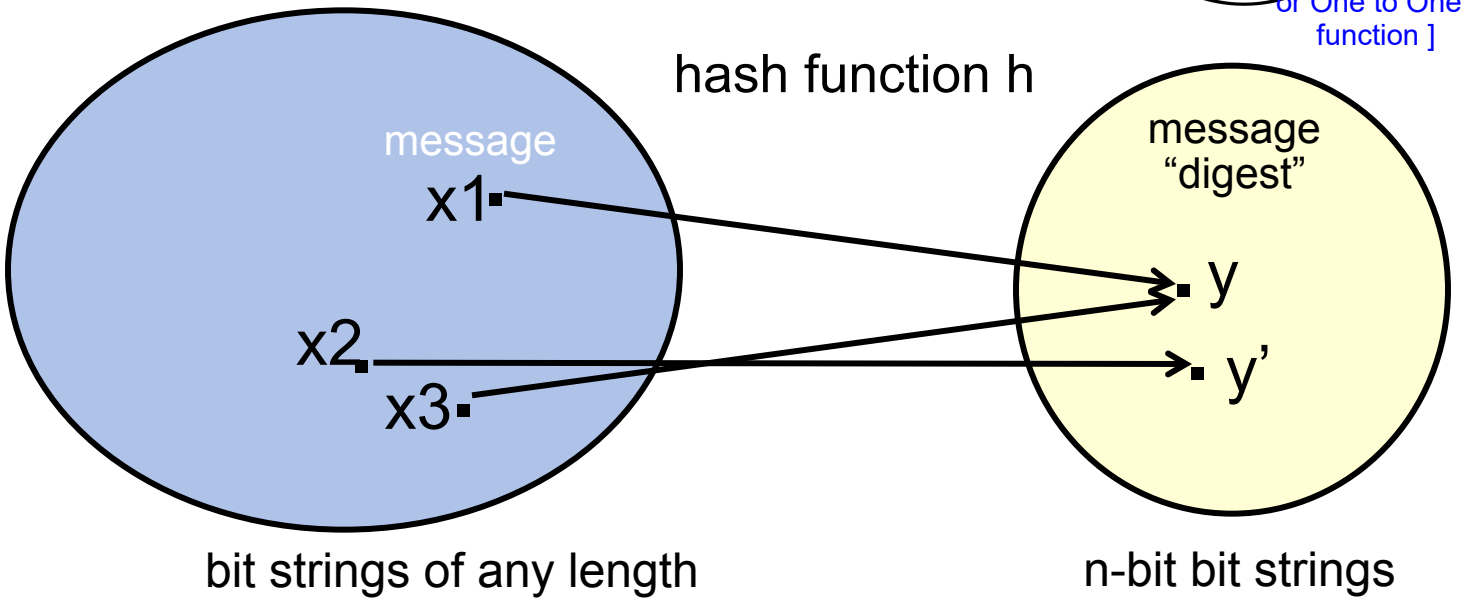
- A hash function is a function h which has the following two properties
  - compression — h maps an input x of arbitrary finite bit-length, to an output h(x) of fixed bit-length n.
  - ease of computation — given h and an input x, h(x) is easy to compute

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

Hash Functions



[Injective function or One to One function]



- 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

[In mathematics, an injective function or injection or one-to-one function is a function that preserves distinctness: it never maps distinct elements of its domain to the same element of its codomain. In other words, every element of the function's codomain is the image of at most one element of its domain.]

Hash Function Properties

- A "good" hash function produce outputs that are
  - Evenly distributed (spread equally)
  - Apparently random

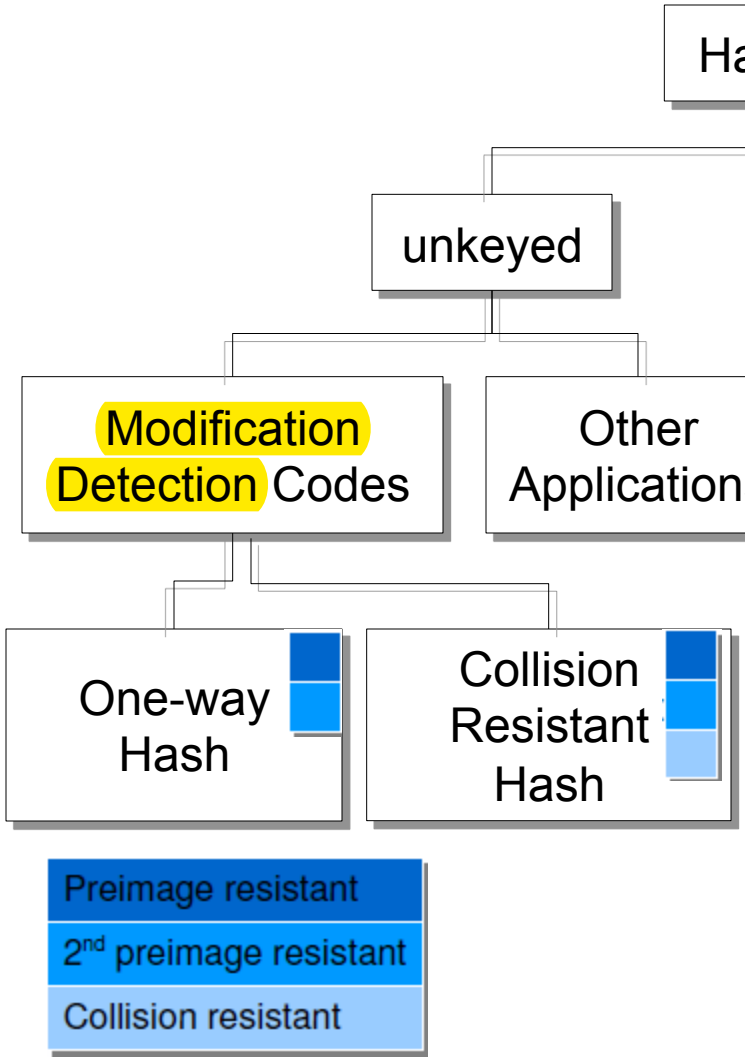
[It should be noted that distributions are used commonly in statistics, and using the term evenly distributed is going to mean something very specific to a statistician. If you're not sure your data is truly evenly distributed, it might be better to stick with a less formal term for it.]

- A cryptographic hash function – for which it is computationally infeasible to find
  - x for a specific y
  - x1 and x2 that map to the same value

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 messages

Types of Hash Functions

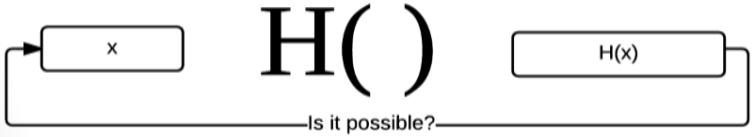


- Unkeyed hash functions only take a message as input
- Keyed hash functions take a message and a secret key as input

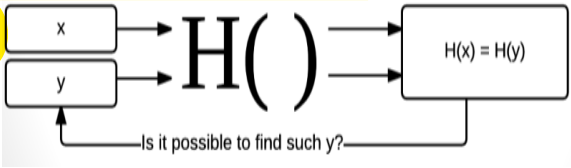
Cryptographic Hash Functions

Question: What is the properties of unkeyed Hash Function?

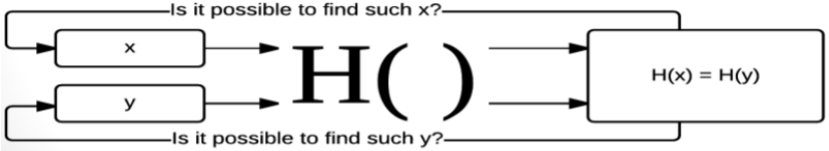
- … are unkeyed hash functions that can have the following properties



- Preimage resistant
  - Given y = h(x) but x unknown 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)

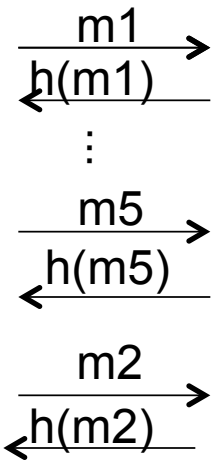


- Collision resistant
  - It is computationally infeasible to find any two different inputs x, x' that hash to the same value, i.e. such that h(x) = h(x')



Random Oracle Model

- Mathematical model for an ideal hash function
  - Upon receipt of a new message of any length the oracle produces 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 from any previously chosen digest



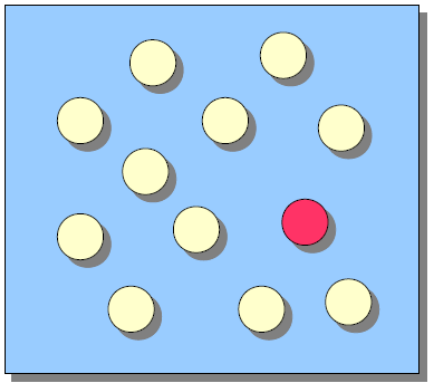
Oracle chooses hash values randomly and independently

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

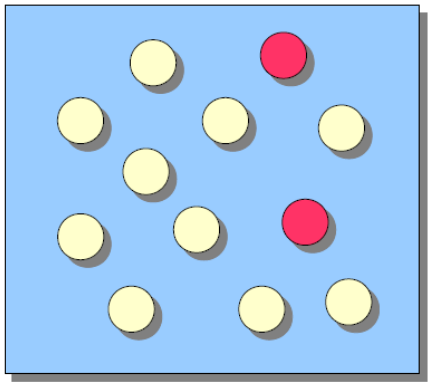
The Birthday Problems

1st problem



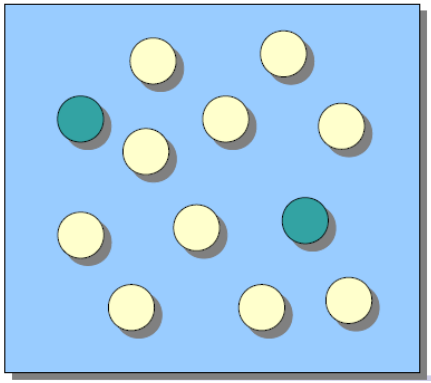
Find one with a particular fixed birthday

2nd problem



Select one and find one with the same birthday

3rd problem



Find any two with the same birthday

- Question in all
  - What is the minimal number k of students we need to have in a classroom to solve the problem with a probability >= ½

Birthday Problems (1)

Student number N=365, and given probability P=.5;

- Problem 1: 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)
- Problem 2: What is the minimum number k of students in a classroom such that with probability P at least one student has the same birthday as the student selected by the professor?

P(s)+P(d) =1 where P(s) = {some one share with some one else} and P(d) = {every one have different birthday}

#	*Probability*	*General Value of k*	*Value of k with P = 1/2*	*# of students (N=365, P=1/2)*
1	$P \sim 1 - e^{-k/N}$	$k \sim \ln[1/(1-P)]*N$	$k \sim 0.69 * N$	253
2	$P \sim 1 - e^{-(k-1)/N}$	$k \sim \ln[1/(1-P)]*N + 1$	$k \sim 0.69 * N + 1$	254

Sample space: for 3 people: 1 2 3 we got, P(365/365),P(364/365),P(363/365) = (365.364.363) / 365^3=365! / (362!*365^3)
so for 30 people in general, P(d)= 365!/{(365-30)!*365^30} =.2937
P(s) = 1-P(d) = 1- .2937 = .7063=70.63%

Example Proof for Problem 1

- 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 small values of x) gives us the probability
  - $P \sim 1- e^{-k/N}$

Birthday Problems (2)

- 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*	*General Value of k*	*Value of k with P = 1/2*	*# of students (N=365) P=1/2*
3	$P \sim 1 - e^{-k(k-1)/2N}$	$k \sim (2\ln[1/(1-P)]*N)^{1/2}$	$k \sim 1.18 * N^{1/2}$	23

How Preimage Resistant Can a Hash Function be?

- Brute force attack to find preimage to a given digest y
  - Attacker randomly selects x', hashes it and compares the result to y until he finds a preimage
- 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

Here, N = 2^n, for n-bit hash function. For probability that attacker will success rate P(s) =0.5

- $O(2^n)$ hash values

How 2nd preimage Resistant Can a Hash Function Be

- Brute force attack to find a 2nd 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 2nd preimage with probability of ½
- Using the 2nd birthday problem
  - $0.69 * 2^n + 1$ hashes need to be calculated to brute force a 2nd preimage of an n-bit hash function
- $O(2^n)$ hash values

How Collision Resistant Can a Hash Function be at Max?

- 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 [occur at the same time]
- 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 values

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
  - Designed by the NSA in 1993
  - Updated to SHA-1 in 1995, SHA-2 in 2010, SHA-3 in 2015
  - Defined in FIPS PUP 180-2
- SHA-1 produces a hash of 160 bit
  - Computations of longer hashes possible with SHA-256, SHA-384, SHA-512 (collectively known as SHA-2)
- Whirlpool
- .....

Hash function:
MD5: produce a hash of 128 bit.

SHA: (SHA-1, SHA-2, SHA-3)
SHA-1: produces a hash of 160 bits.
SHA-2: Collectively known for SHA-256, SHA-384, SHA-512

MD-5 Overview

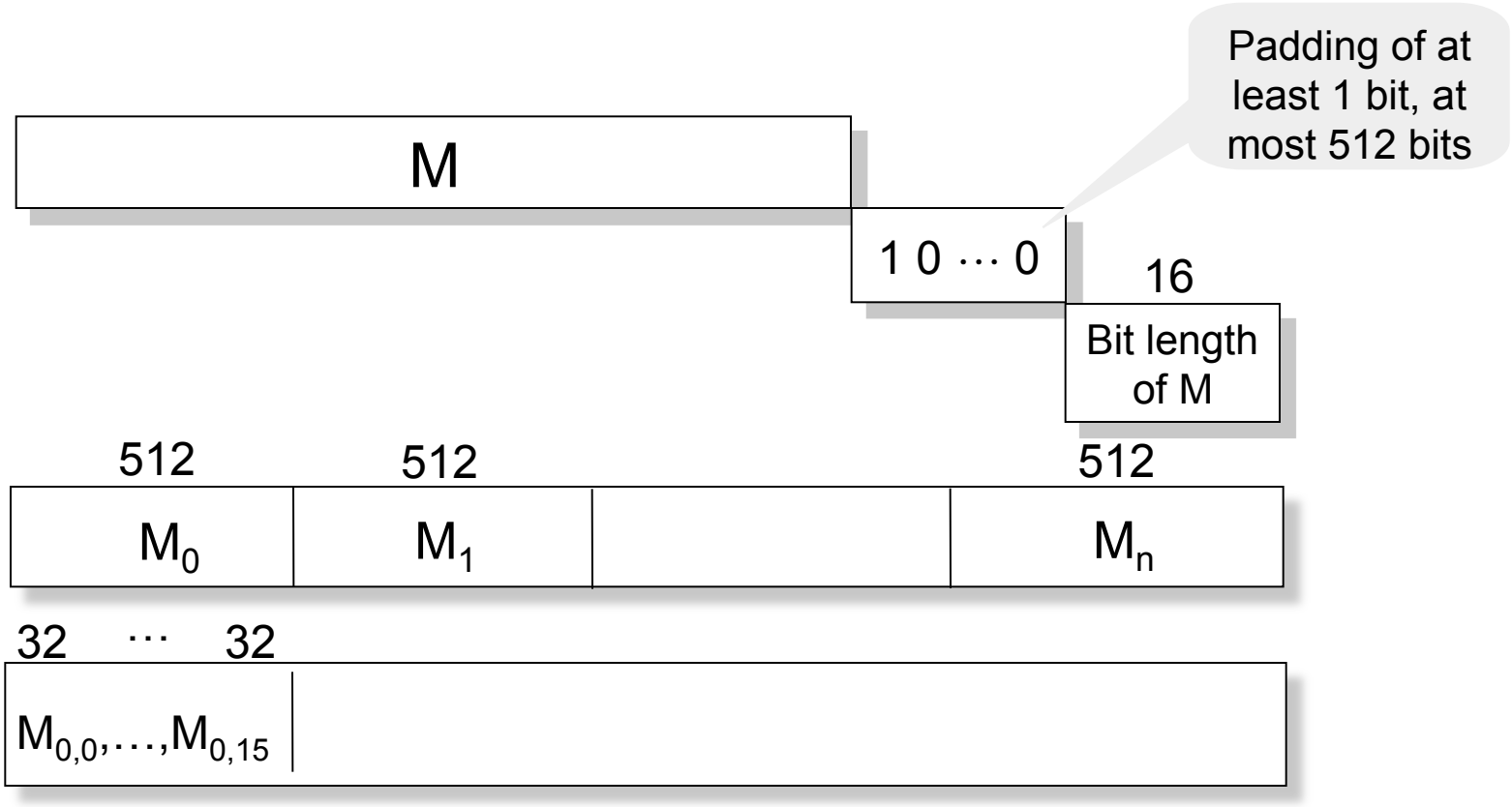
- Pad and add length of message to the message such that bit-length is a multiple of 512
- Initialize four word buffers
- Initialize a 64 elements table T[1],…. ,T[64]
- Shuffle each 512 bit block in 4 rounds and use the output as input to the next round

Plain text - 512 bit,
1. Append padding bits so, -> length =64< Multiple of 512.
2. Append 64 bit representation here, 1& 2 plaintext length = multiple of 512.
eg. plaintext 512-64=448,< 448 need padding + 64 =512 bits.
if it is set of 2 blocks i.e. 1024 -64 = 960< than 960 bit we need padding
3. Initialize the MD buffers (Buffer-32 bit, 4 buffer) , A, B, C, D.
4. Process the each block (512 bit)
5. Output (Message digest in Buffers)

1. pad message so its length is 448 mod 512
2. append a 64-bit length value to message
3. initialise 4-word (128-bit) MD buffer (A,B,C,D)
4. process message in 16-word (512-bit) blocks:
   - using 4 rounds of 16 bit operations on message block & buffer
   - add output to buffer input to form new buffer value
5. output hash value is the final buffer value

Padding

- Let M be the message to be hashed



Functions

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

Function:
F(B,C,D) = (B and C) or (not B and D)
G(B,C,D) = (B and D) or (C and not D)
H(B,C,D) = B XOR C XOR D
I(B,C,D) = C XOR (B or not D)

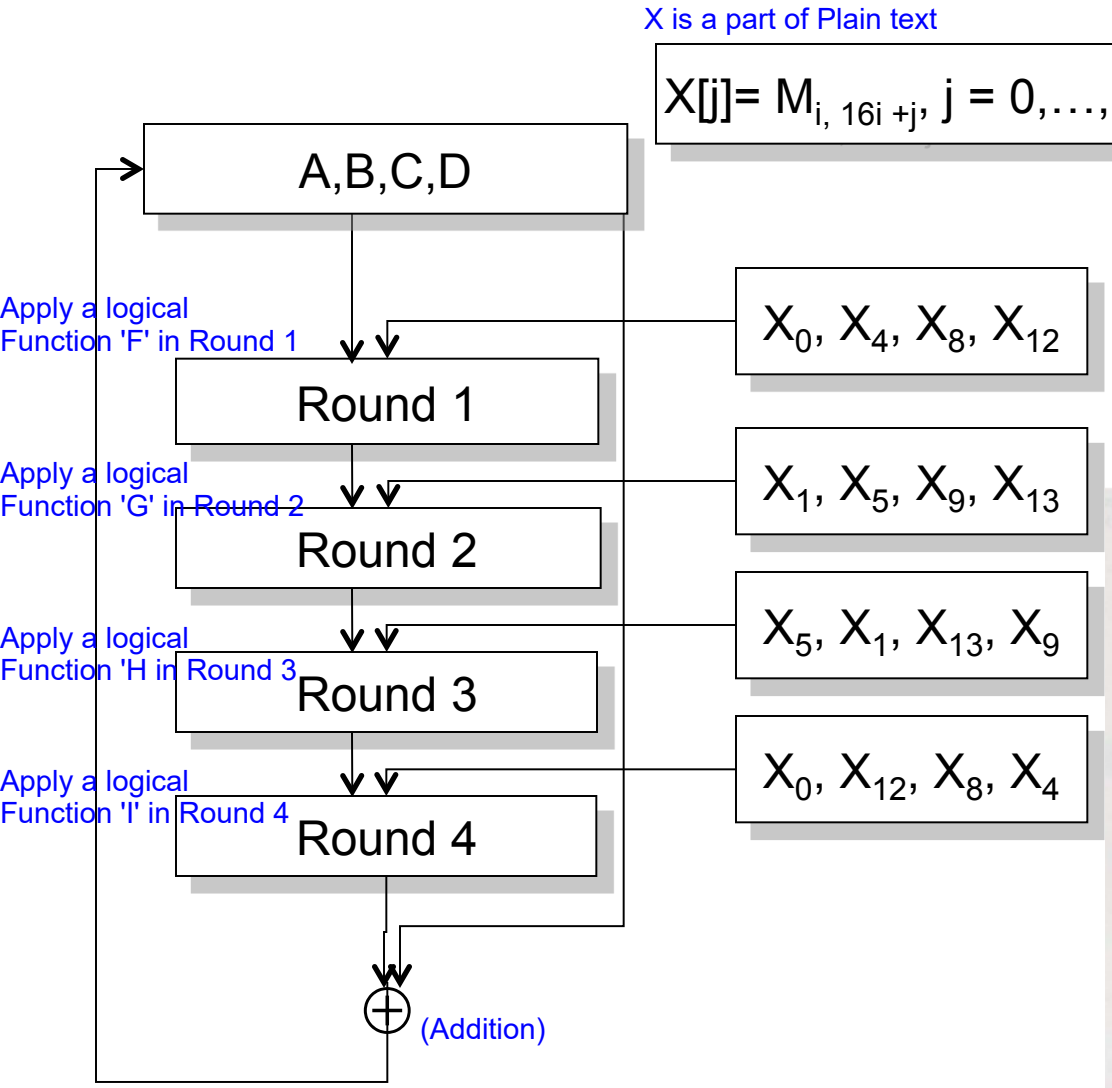
Addition Module 2^32
16 steps: first 16 step use F function.
A<- B Addition Module [(A Addition Module Function(B,C,D) Addition Module x[ ] Addition Module T[i]] <<< s, s circular left shift s-bits.

- F(X,Y,Z) = XY v not(X) Z ("if X then Y else Z")
- G(X,Y,Z) = XZ v Y not(Z)
- H(X,Y,Z) = X xor Y xor Z
- I(X,Y,Z) = Y xor (X v not(Z))

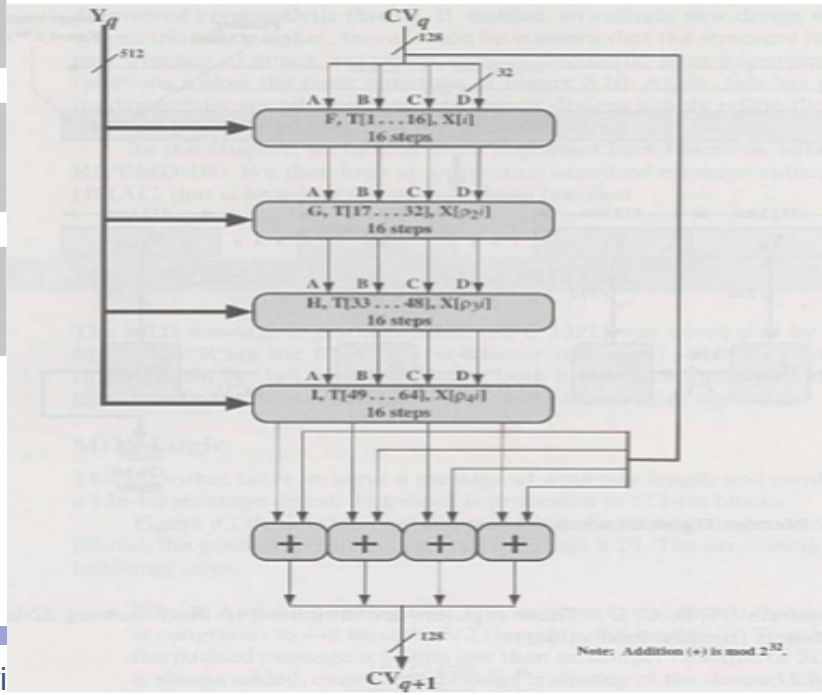
- T[i] = integer part of (2^{32} |sin(i)|), where |sin(i)| is the absolute value of sin(i)

T[i] = constant,
1st round = 16 constant use, T[1..16] so here 16 step,
2nd round =16 constant use T[17-32]
3rd round = 16 constant use T[33-48]
4th round = 16 constant use T[49 -64]

MD 5 – Word Processing Rounds

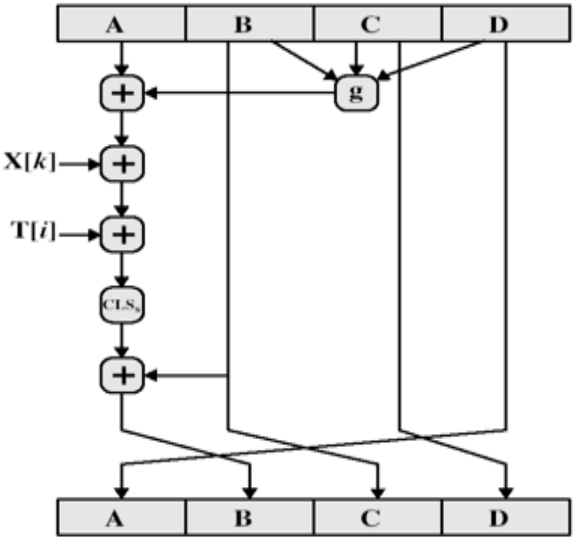


- The word processing takes place for each message block M_i, i=0,…,15



processing of a single 512-bit block (MD5 compression function)

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]

- each round has 16 steps of the form:
  `a = b+((a+g(b,c,d)+X[k]+T[i])<<<s)`
- a,b,c,d refer to the 4 words of the buffer, but used in varying permutations
  - note this updates 1 word only of the buffer
  - after 16 steps each word is updated 4 times
- where g(b,c,d) is a different nonlinear function in each round (F,G,H,I)
- T[i] is a constant value derived from sin

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 t] 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
- 2004: Wang et al. found collisions in MD5, MD4, RIPEMD, HAVAL-128
- 2005: Further enhanced to make collision finding feasible on a notebook (8 hours to find a collision)
- 2006: Black et al. implemented a toolkit that can be used to create collisions in MD5
- Greatly improved by the use of off-the-shelf GPUs
- As of 2010, md5 is considered as cryptographically broken and unsuitable for further use
- 2012 Flame malware exploited weaknesses in md5 to fake Microsoft digital signatures

Security of SHA-1

- 2004: Second preimage attack on SHA-1 in 2^{106}
- 2005: Attack found by Wang et al. that finds a collision with 2^{63} hash operations
- 2017: Attack reported that finds collision on PDF documents
  - Huge computational effort: 6500 years on a single CPU
- SHA-2 from 2002

- It is not recommended to use SHA-1 and SHA-2 (2016)
- Idea for the new hash function in 2012, known as SHA-3 from 2015

How Bad is That?

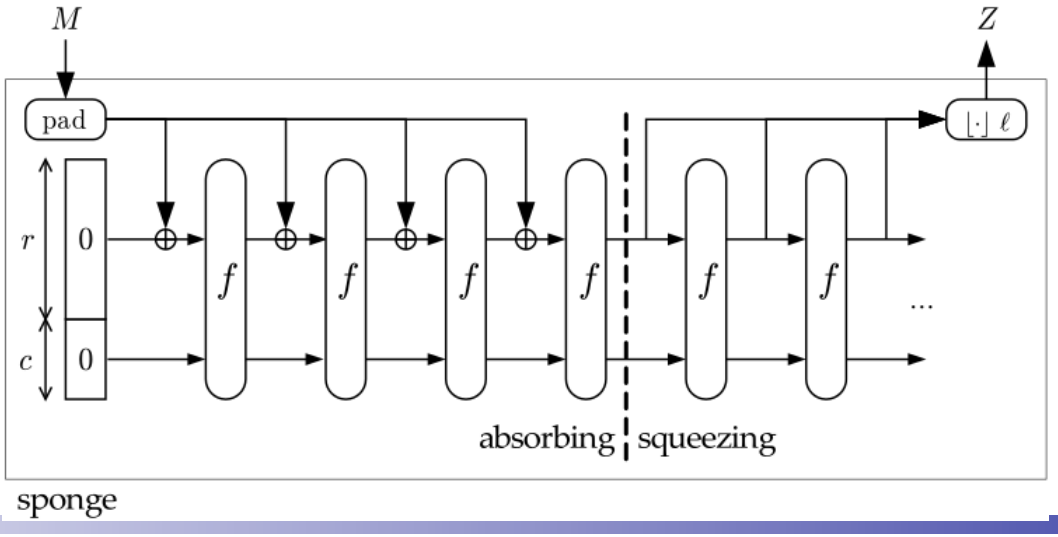
- As we will see later on hash functions are used for
  - Digital signatures on messages and certificates
    - Messages are hashed before they are signed
  - Calculation of responses in challenge response authentication protocols
  - Constructing message authentication codes
  - Constructing modification 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

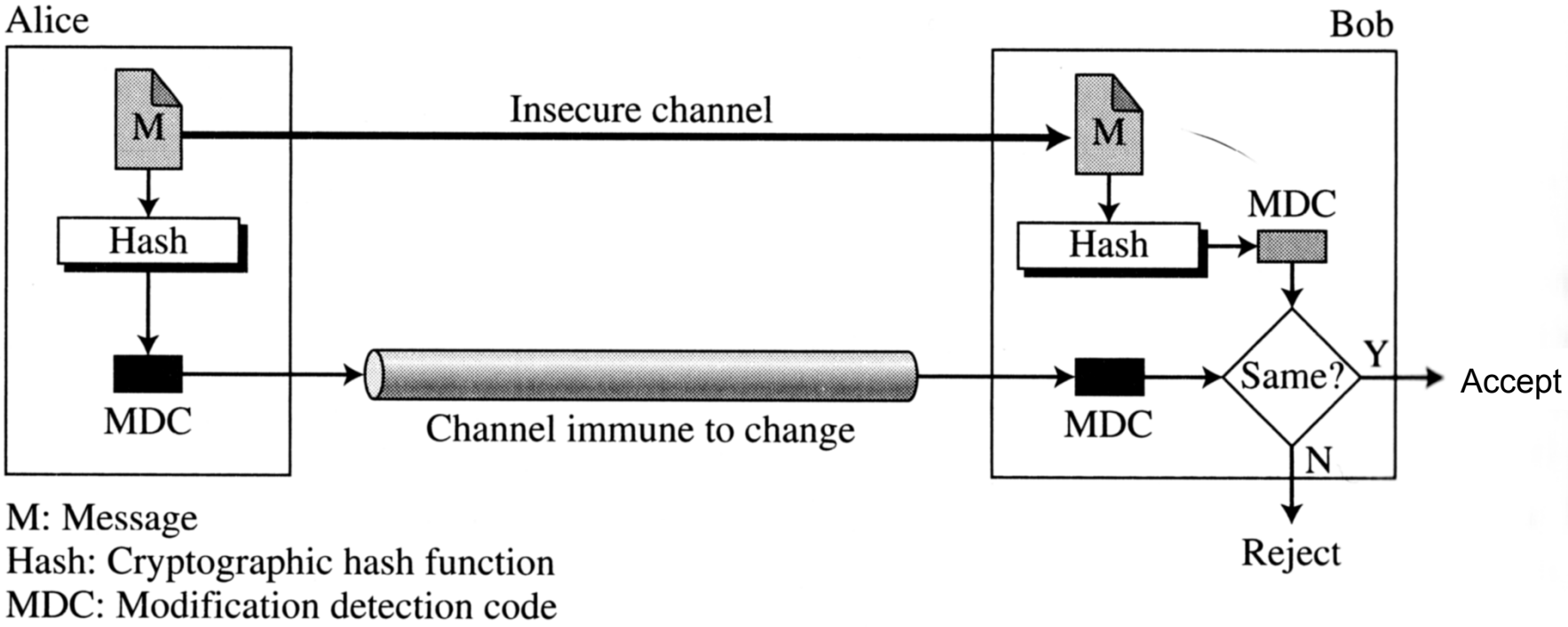
- November 2007: Call for a new hash function
- October 2008: Deadline for submissions
  - 64 submissions, 51 selected for the first round
- July 2009: Fourteen second round candidates announced
  - http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html
- Now: one year for public review of the second round candidates
    - August 2010 conference on the findings during that year

SHA3

- SHA-1/-2 work similarly to MD5 (Merkle-Damgard)
- 2007: public contest for SHA3
- 2012: winner Keccak (catch-ack)
  - 5 finalists, Keccak has entirely different construction
  - Can operate on much smaller states
  - Sponge construction (r-bitrate, c-capacity -- parameters)



Modification Detection Code



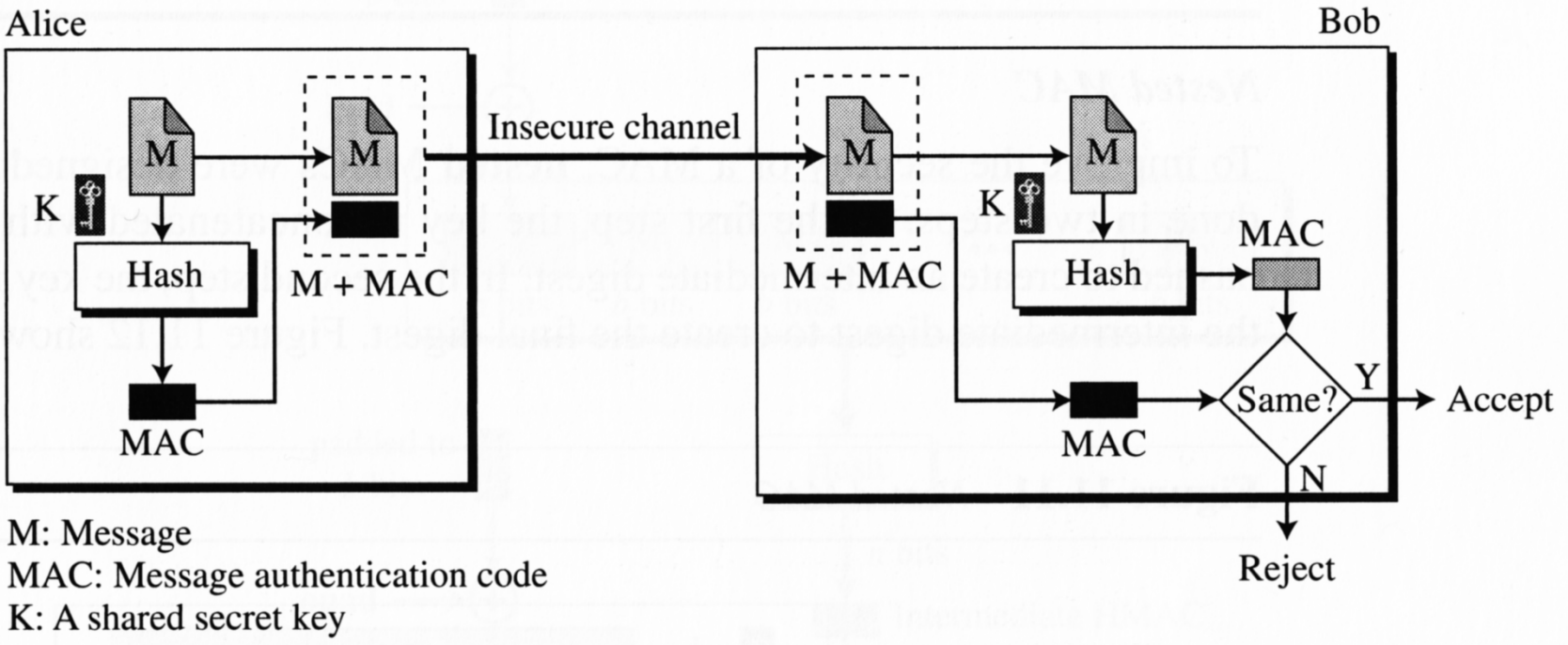
* Figure taken from: Fourouzan, Introduction to Cryptography and Network Security

Message Authentication Codes

Question : What is the properties for Keyed Hash function of MAC?

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

Message Authentication Codes



* Figure taken from: Fourouzan, Introduction to Cryptography and Network Security

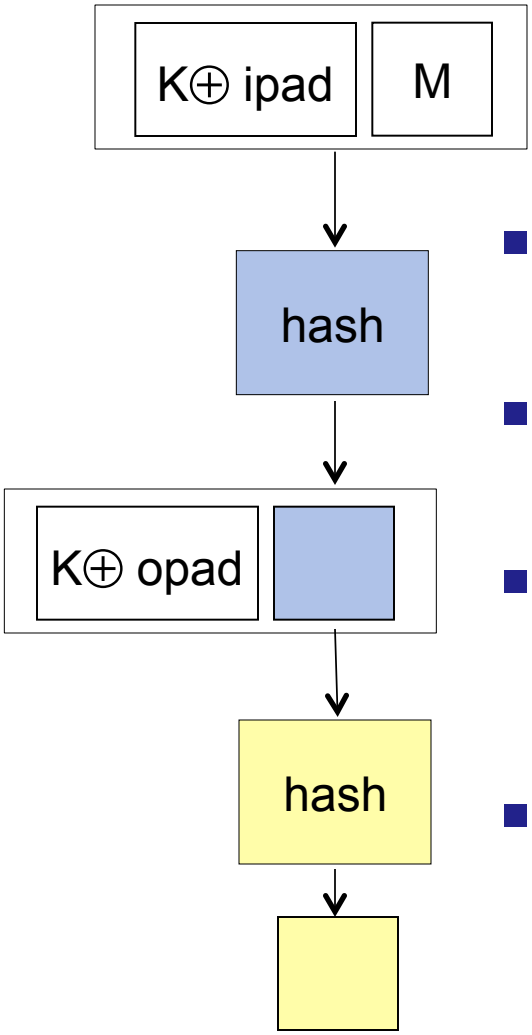
How Should a Hash Function be Keyed?

- For some hash functions simple constructions lead to insecure MACs, e.g.
  - h(k||m) leads to insecure MACs for certain hash functions
- 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)
- Mandatory for IP security, also used in SSL/TLS

HMAC: High-level view



HMAC is a MAC constructed from a cryptographic hash function h

- HMAC can be proven to be as secure as the underlying hash function
- h(K xor opad || h(K xor ipad || 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

CMAC Overview

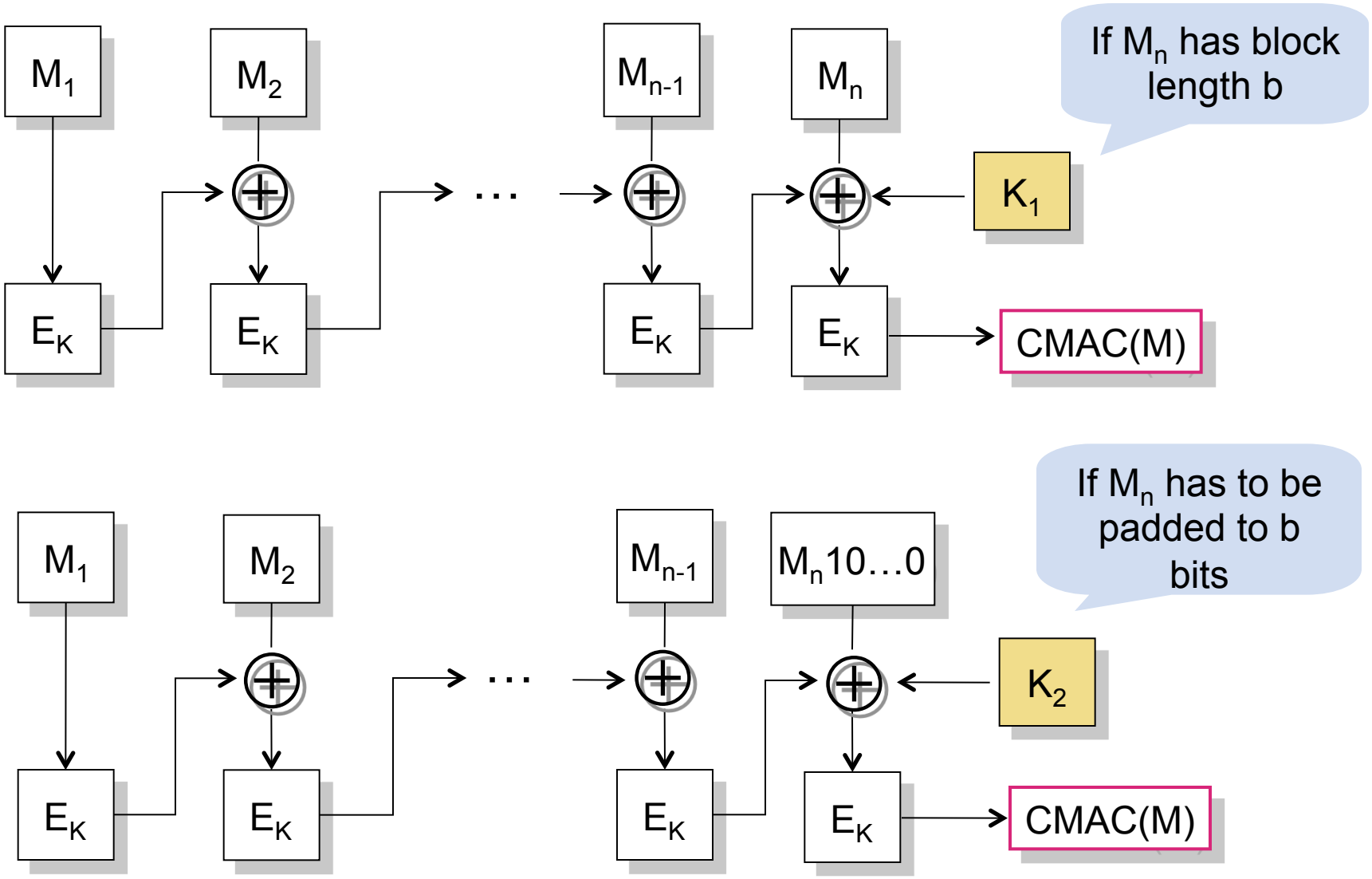
- Cipher-Based Message Authentication Code (CMAC)
- CMAC uses a block cipher E_K of block length b = 64 and b = 128
- A message M is split into n blocks of length b:

  $M = M_1 \,||\, M_2 \,||\, \ldots \,||\, M_n$
- 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

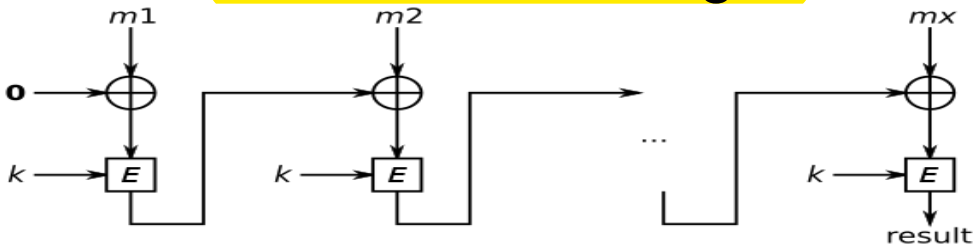
- Let $L = E_K(0^b)$, where 0^b is the bit string of b zeros
- Let $R_{128} = 0^{120}10000111$, $R_{64} = 0^{59}11011$
- Then K1 is computed by
  - If $MSB_1(L) = 0$, $K1 = L<<1$
  - Else $K1 = L \text{ xor } R_b$
- K2 is computed by
  - If $MSB_1(K1) = 0$, $K2 = K1<<1$
  - Else $K2 = (K1<<1) \text{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

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
  - $MAC(M) = E_K(M)$
  - $MAC(P) = E_K(P)$
  - $MAC(M \,||\, P \text{ xor } MAC(M)) = E_K(E_K(M \text{ xor } 0^b) \text{ xor } P \text{ xor } MAC(M)) = E_K(P \text{ xor } MAC(M) \text{ xor } MAC(M)) = E_K(P)$



- 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
- Replay protection can be added to MACs with the help of additional input to the MAC:
  - Time stamps [A timestamp is a sequence of characters or encoded information identifying when a certain event occurred, usually giving date and time of day, sometimes accurate to a small fraction of a second.]
  - Other counters
  - Random Nonces
    - Nonce = Number used once

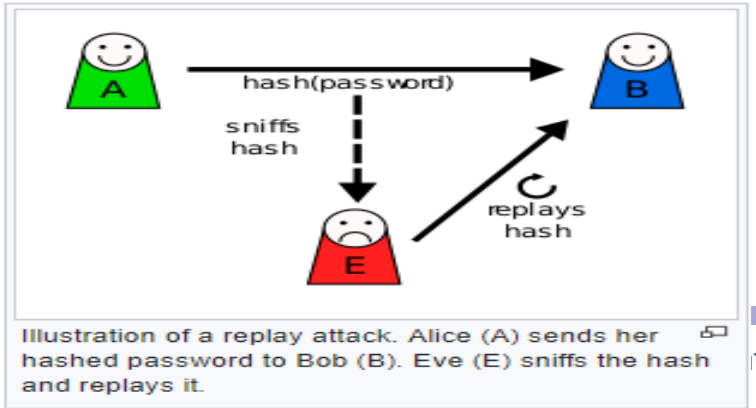
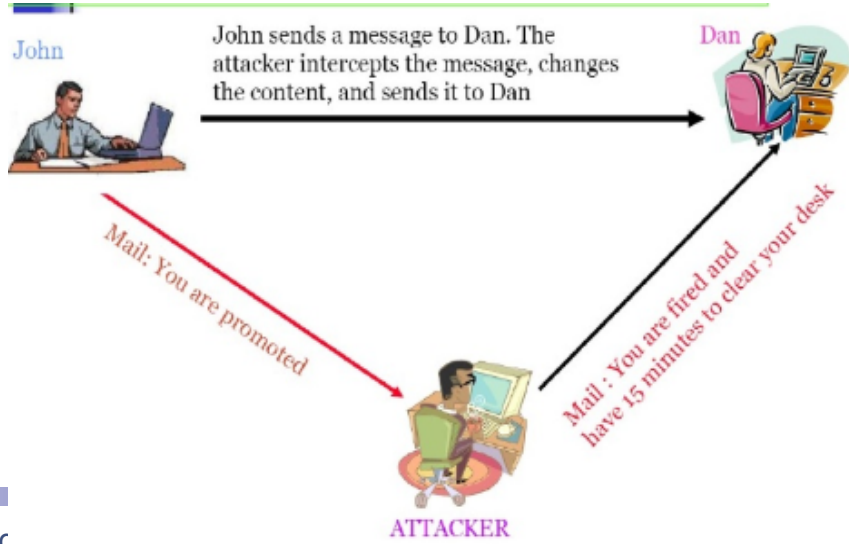


Illustration of a replay attack. Alice (A) sends her hashed password to Bob (B). Eve (E) sends the hash and replays it.



Replay Protection with Counters

ONE-TIME PASSWORDS

One-time passwords are similar to session tokens in that the password expires after it has been used or after a very short amount of time. They can be used to authenticate individual transactions in addition to sessions. These can also be used during the authentication process to help establish trust between the two parties that are communicating with each other.

- 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

Timestamps: Timestamping is another way of preventing a replay attack. Synchronization should be achieved using a secure protocol. For example, Bob periodically broadcasts the time on his clock together with a MAC. When Alice wants to send Bob a message, she includes her best estimate of the time on his clock in her message, which is also authenticated. Bob only accepts messages for which the timestamp is within a reasonable tolerance. The advantages of this scheme is that Bob does not need to generate (pseudo-) random numbers, and that Alice doesn't need to ask Bob for a random number. In networks that are unidirectional or near unidirectional, it can be an advantage.

The trade-off being that replay attacks, if they are performed quickly enough i.e. within that 'reasonable' limit, could succeed.Timestamps Timestamping is another way of preventing a replay attack. Synchronization should be achieved using a secure protocol. For example, Bob periodically broadcasts the time on his clock together with a MAC. When Alice wants to send Bob a message, she includes her best estimate of the time on his clock in her message, which is also authenticated. Bob only accepts messages for which the timestamp is within a reasonable tolerance. The advantages of this scheme is that Bob does not need to generate (pseudo-) random numbers, and that Alice doesn't need to ask Bob for a random number. In networks that are unidirectional or near unidirectional, it can be an advantage. The trade-off being that replay attacks, if they are performed quickly enough i.e. within that 'reasonable' limit, could succeed.

Replay Protection with Random Nonces

Nonces and MAC
Bob can also send nonces but should then include a message authentication code (MAC), which Alice should check.

- 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 sends a random nonce to Bob
  - Bob sends some messages appended with a MAC that uses the message, a shared secret key and the random nonce from Alice as input
  - Alice checks if Bob used the random nonce she sent. If this is the case, and Alice has not used the nonce before, Bob cannot have replayed the message

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 xor K
    - Attacker can replace M by M xor 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 none of the modes can detect e.g. appending additional random ciphertext
- It is good practice ALWAYS to use different keys for integrity protection and encryption

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

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
  - SHA-3
    - *https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.202.pdf*
  - CMAC: NIST Special Publication 800-38B
  - HMAC: specified in FIPS Pub 198-1