Cryptographic Hash Functions

Security Requirements

- Security Threats
- Attacks in the context of communications across a network may include:
  1. Disclosure
  2. Traffic analysis
  3. Masquerade
  4. Content modification
  5. Sequence modification
  6. Timing modification
  7. Repudiation

Security Requirements

- Threat Measures
- Measures to deal with first two attacks:
  - In the realm of message *confidentiality*, and are addressed with *encryption*
- Measures to deal with items 3 thro 6
  - Message *authentication*
- Measures to deal with items 7
  - *Digital signature*

Security Requirements

Message authentication

A procedure to verify that messages come from the alleged source and have not been altered

Message authentication may also verify sequencing and timeliness

Digital signature

An authentication technique that also includes measures to counter repudiation by either source or destination

Topics

- **Overview of Cryptography Hash Function** ✓
- Usages
- Properties
- Hashing Function Structure
- Attack on Hash Function
- The Road to new Secure Hash Standard

Cryptography Hash Function

- **Introduction to Hash Function(Done)**
- **Cryptography Hash Function**

A cryptographic hash function

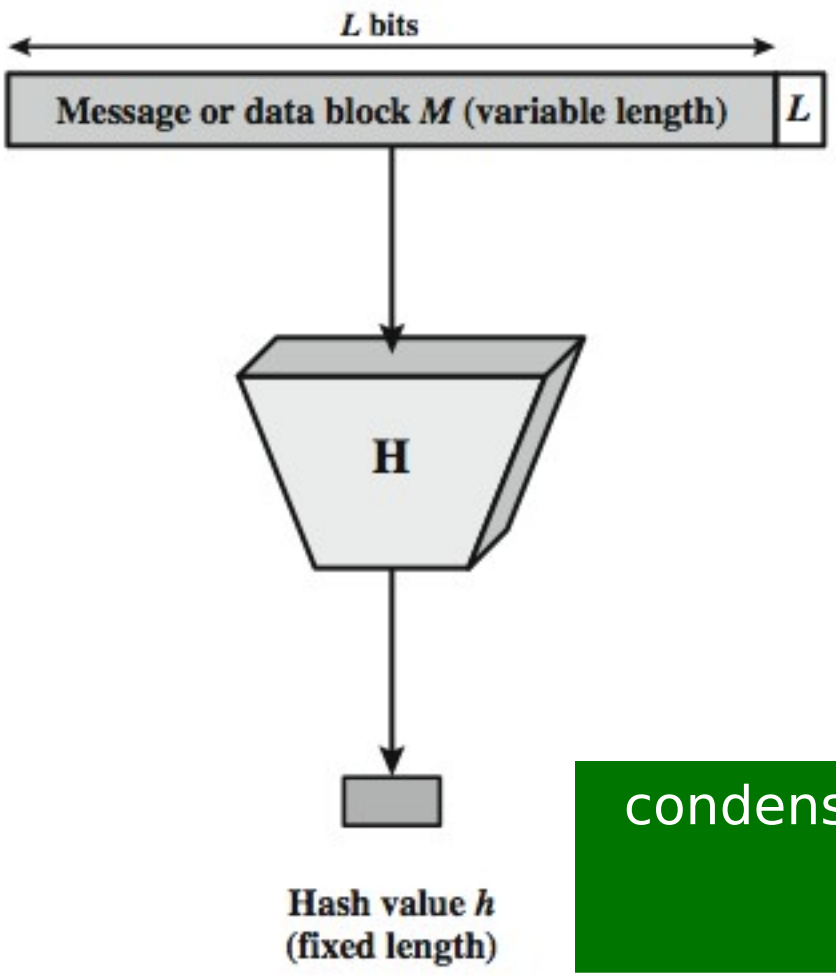
A cryptographic hash function

- A cryptographic hash function is a hash function that takes an arbitrary block of data and returns a fixed-size bit string, known as the ***cryptographic hash value***, such that any (accidental or intentional) change to the data will (with very high probability) change the hash value.
- The data to be encoded are often called the ***message***, and the hash value is sometimes called the ***message digest*** or simply ***digest***.

Definition

A hash function is a computationally efficient function mapping **binary strings** of arbitrary length to **binary strings** of some fixed length, called **hash-values**.

Cryptographic Hash Function



- The hash value represents concisely the longer message
  - may called the *message digest*
- A message digest is as a ``digital fingerprint'' of the original document

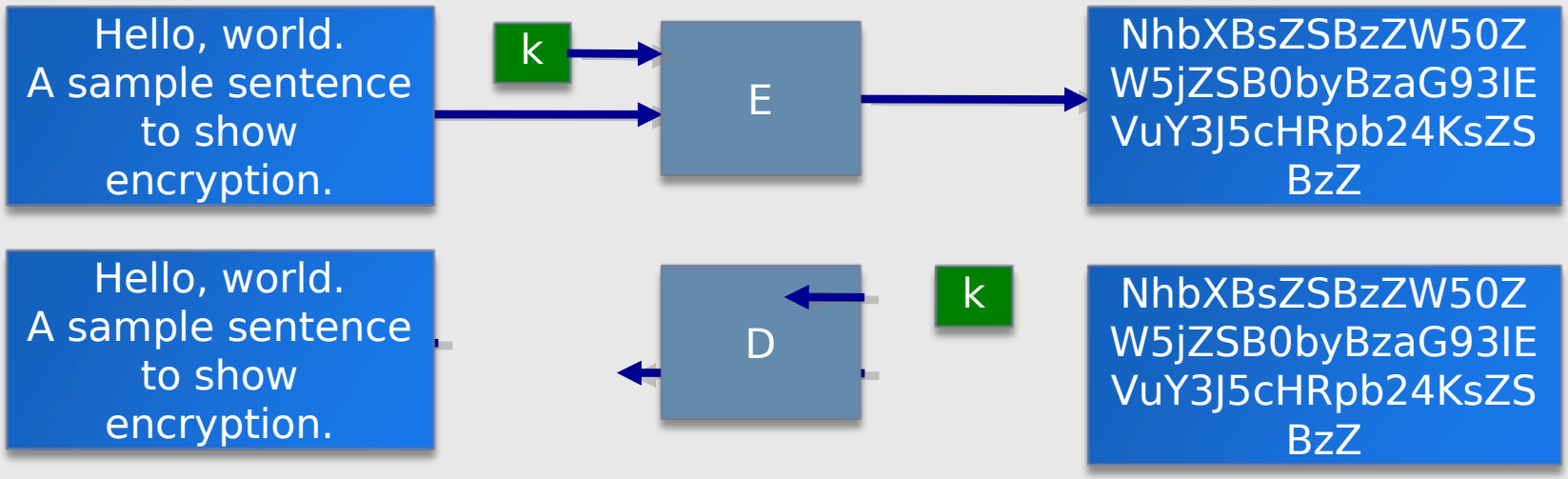
condenses arbitrary message to fixed size

$h = H(M)$

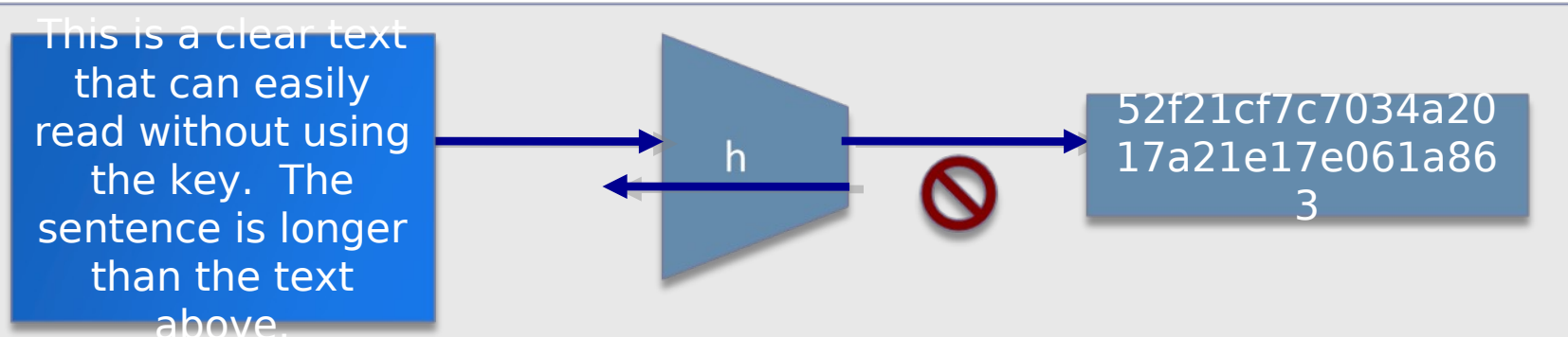
Cryptographic Hash Function

- Hashing function as “chewing” or “digest” function

Hashing V.S. Encryption



- Encryption is two way, and requires a key to encrypt/decrypt



- Hashing is one-way. There is no 'de-hashing'

Cryptographic Hash Function

- Motivation Behind Crypto Hashing
- Intuition
  - Re-examine the non-cryptographic checksum
  - Main Limitation
    - An attack is able to construct a message that matches the checksum
- Goal
  1. Design a transformational code where the original message can not be inferred based on its checksum and
  2. Such that an accidental or intentional change to the message will change the hash value

Cryptographic Hash Function

- Hashing functions are used to condense an arbitrary length message to a fixed size, usually for subsequent signature by a digital signature algorithm'
- To be of cryptographic use, a hash function h is typically chosen such that it is **computationally infeasible** to find two distinct inputs which hash to a common value (i.e., two colliding inputs x and y such that h(x) = h(y)), and that given a specific hash-value y, it is computationally infeasible to find an input (preimage) x such that h(x) = y.
- The basic idea of cryptographic hash functions is that a hash-value serves as a **compact representative image** (sometimes called an **imprint**, **digital fingerprint**, or **message digest**) of an input string, and can be used as if it were uniquely identifiable with that string.

Cryptographic Hash Function

General Classification

From a broader perspective, hash functions may be split into two classes:

1.Unkeyed hash functions, which uses a single input parameter (a message); and

2.Keyed hash functions, whose specification dictates two distinct inputs, a **message** and a **secret key**.

Cryptographic Hash Function

General Classification

Definition

A hash function (in the unrestricted sense) is a function **h** which has, as a minimum, the following two properties:

1.hash code map: h1 : keys → integers
2.compression map: h2 : integers → [0, . . . , N − 1]

The hash code map is applied before the compression map:
h(x) = h2 (h1 (x)) is the compressed hash function.

The compression map usually is of the form **h2 (x) = x mod N:**
The actual work is done by the hash code map.

Cryptographic Hash Function

Popular Compression Maps

Division: **h(k) = |k| mod N**

- the choice N = 2k is bad because not all the bits are location
- the table size N is usually chosen as a prime number
- certain patterns in the hash codes are propagated

Multiply, Add, and Divide (MAD): **h(k) = |ak + b| mod N**

- eliminates patterns provided a mod N ≠ 0
- same formula used in linear congruential (pseudo) random number generators

Topics

- Overview of Cryptography Hash Function
- **Usages**
- Properties
- Hashing Function Structure
- Attack on Hash Function
- The Road to new Secure Hash Standard

Crypto Hash Function Application

Application Areas

Cryptographic hash functions have many information security applications:

1.In digital signatures,
2.message authentication codes (MACs),
3.To index data in hash tables(Efficient Searching),
4.For fingerprinting,
5.To detect duplicate data or uniquely identify files,
6.As checksums to detect accidental data corruption especially during transmission.
7.Provide data integrity.

Crypto Hash Function Application

Ideal cryptographic hash function

The ideal cryptographic hash function has four main properties:

- It is easy to compute the hash value for any given message
- It is infeasible to generate a message that has a given hash
- It is infeasible to modify a message without changing the hash
- It is infeasible to find two different messages with the same hash.

Crypto Hash Function Application

Note

1.Hash functions as discussed above are **typically publicly known** and involve no secret keys.

1. When used to detect whether the message input has been altered, they are called **Modification detection codes (MDCs)**.

1.Related to these are hash functions which involve **a secret key**, and provide **data origin authentication** as well as **data integrity**; these are called M**essage authentication codes (MACs).**

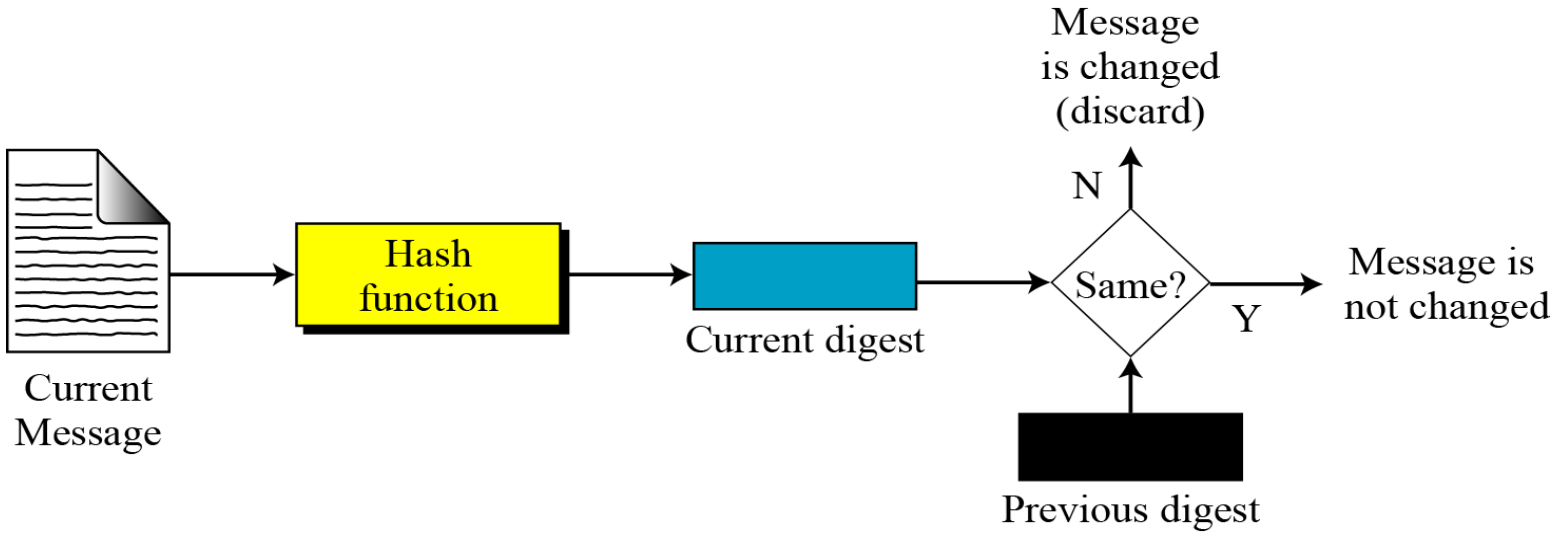
Crypto Hash Function Application

Hash Functions and Data Integrity(Verifying Files and Software),

Hash functions may be used for data integrity as follows.

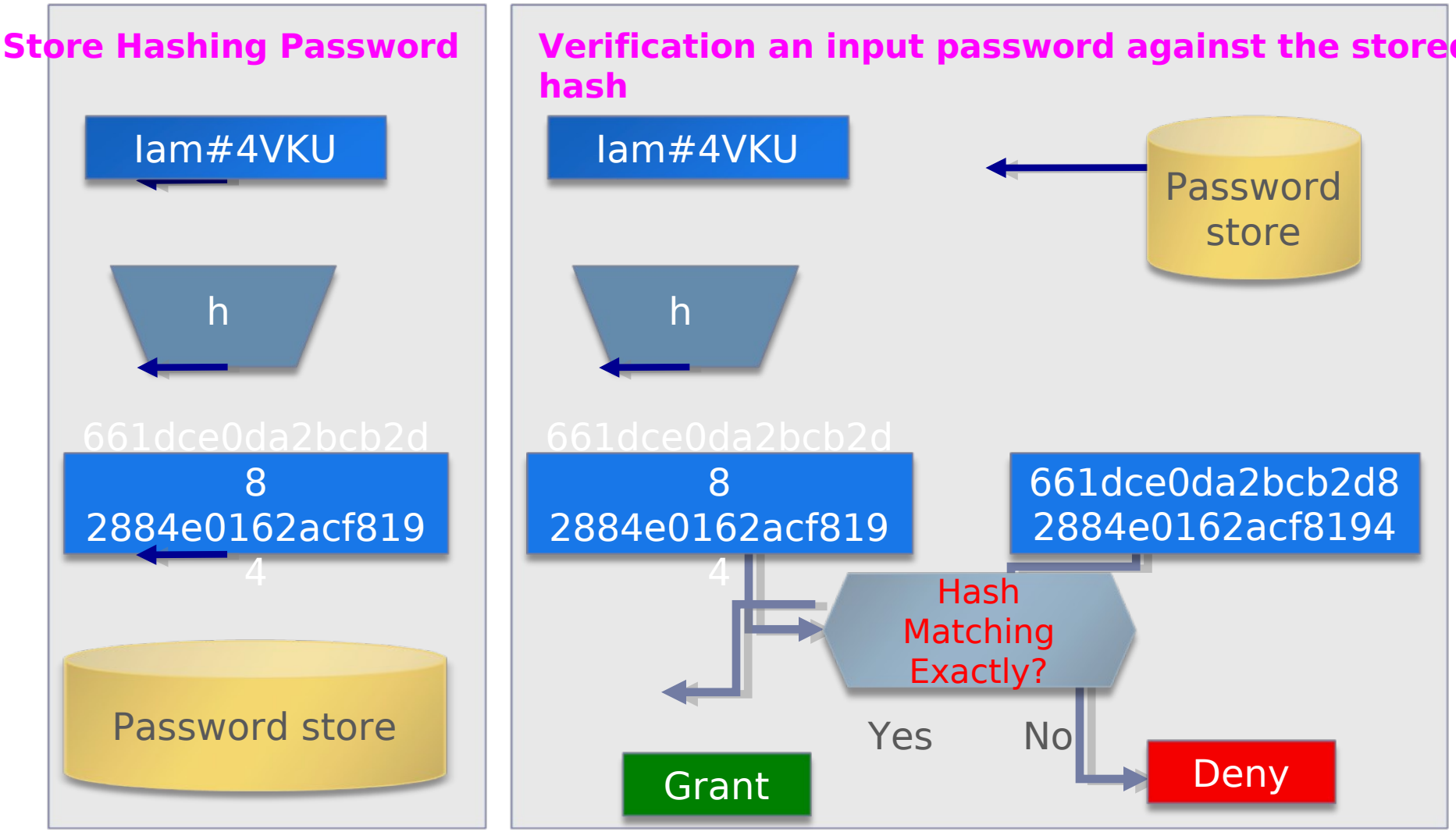
1.The **hash-value** corresponding to a particular input is computed at some point in time.
2.The integrity of this hash-value is protected in some manner.
3. At a subsequent point in time, to verify that the input data has not been altered, the **hash-value** is recomputed using the input at hand, and compared for equality with the original **hash-value**.
4.If they match data integrity is assured.
5.Specific applications include **virus protection** and **software distribution.(Th**

Message/File Integrity



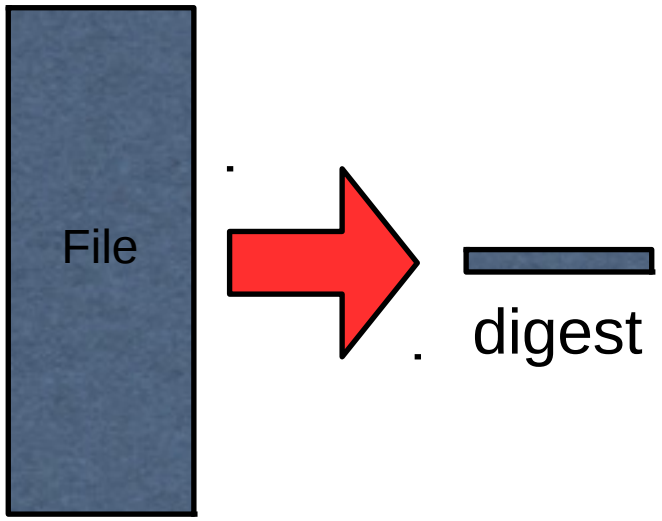
- Practical Application in:
  - To create a one-way password file
    - store hash of password not actual password
  - For intrusion detection and virus detection
    - keep & check hash of files on system

Password Verification

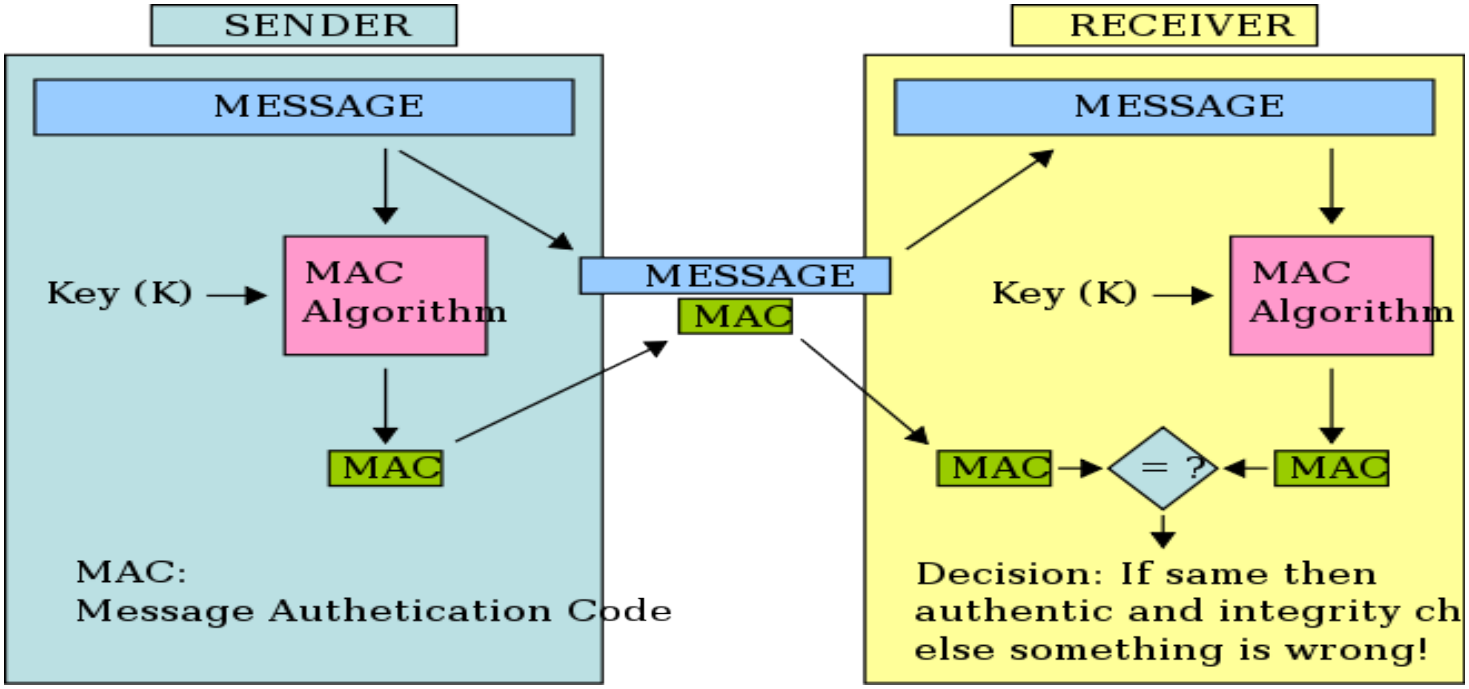


Message Digests

- Message Digests make a “fingerprint” of a file.
- Input: 1-2^{64} bytes
- Output: 128, 160, 256 or more bits

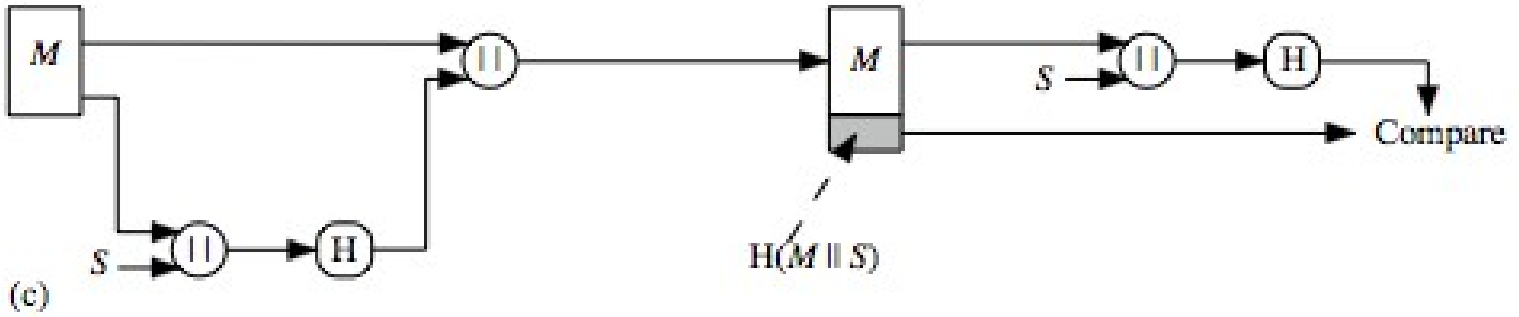


Message Authentication

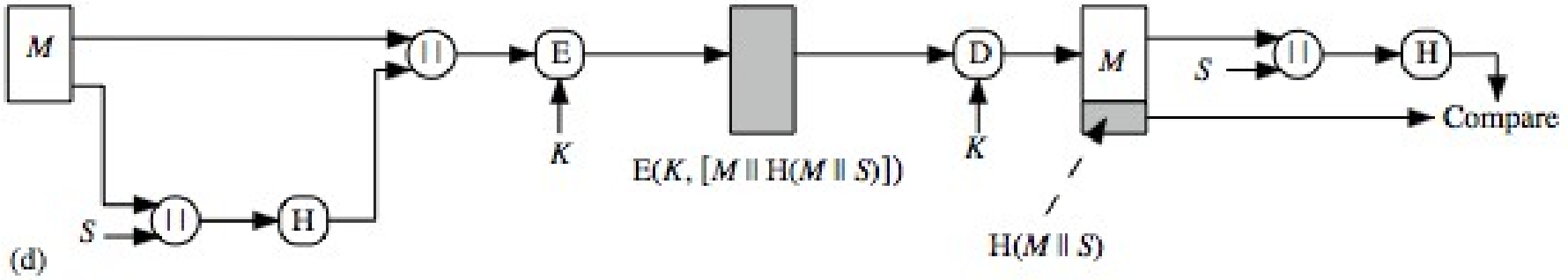


- Protects both a message's integrity as well as its authenticity , by allowing verifiers (who also possess the secret key) to detect any changes to the message content

Authentication

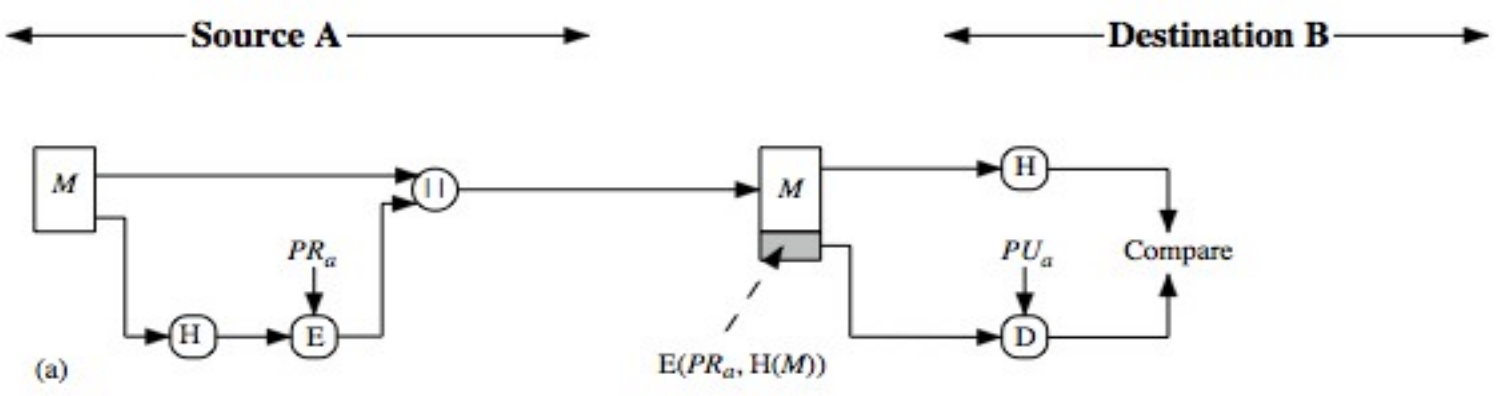


Message encrypted : Authentication (no encryption needed!)

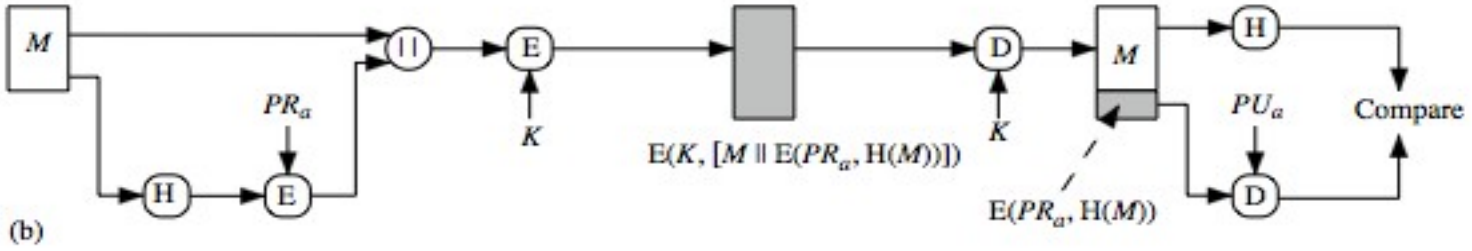


Message unencrypted: **Authentication, confidentiality**

.....Digital Signatures



Authentication, digital signature



Authentication, digital signature, confidentiality

Topics

- Overview of Cryptography Hash Function
- Usages
- **Properties** 
- Hashing Function Structure
- Attack on Hash Function
- The Road to new Secure Hash Standard

Hash Function Properties

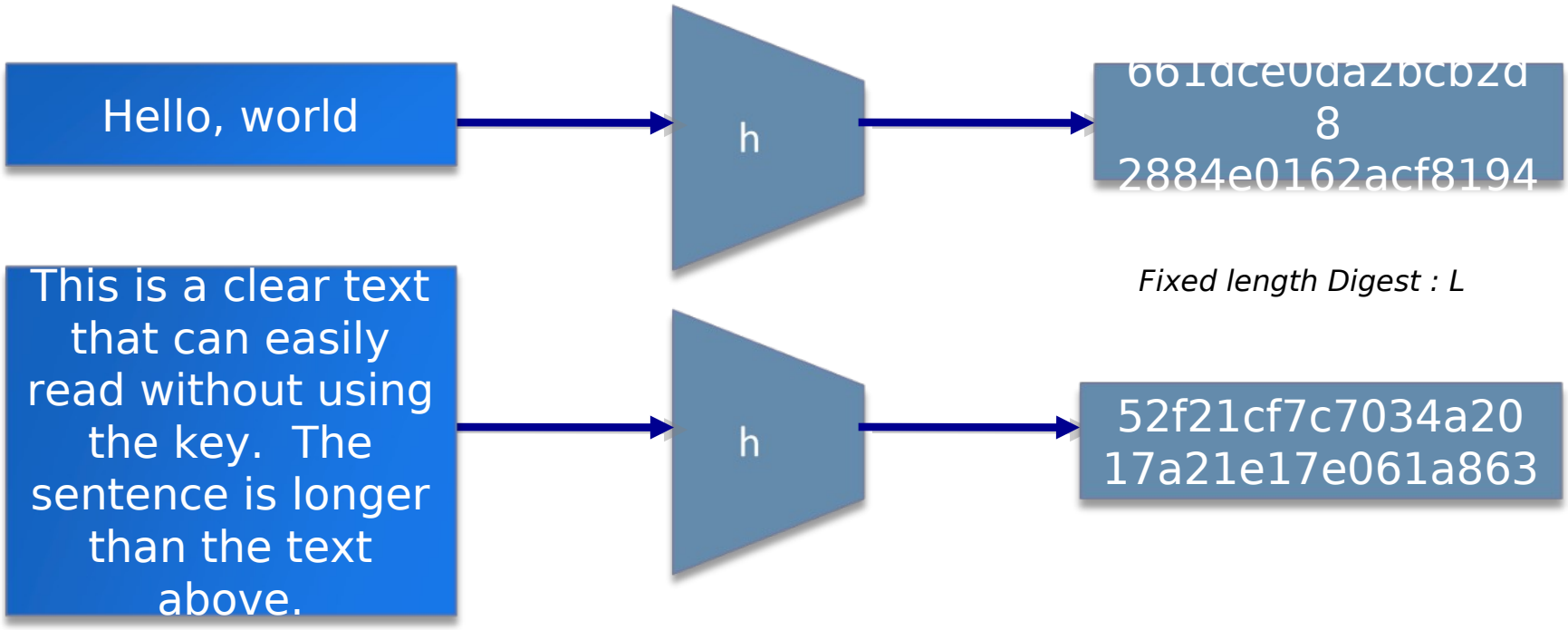
For an unkeyed hash function h with inputs x, x' and outputs y, y' .

1.preimage resistance — for essentially all prespecified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage x such that h(x) = y when given any y for which a corresponding input is not known.

2.2nd-preimage resistance — it is computationally infeasible to find any second input which has the same output as any specified input, i.e., given x, to find a 2nd-preimage x' = x such that h(x) = h(x').

3.Collision resistance — it is computationally infeasible to find any two distinct inputs x, x' which hash to the same output, i.e., such that h(x) = h(x').

Properties : Fixed length



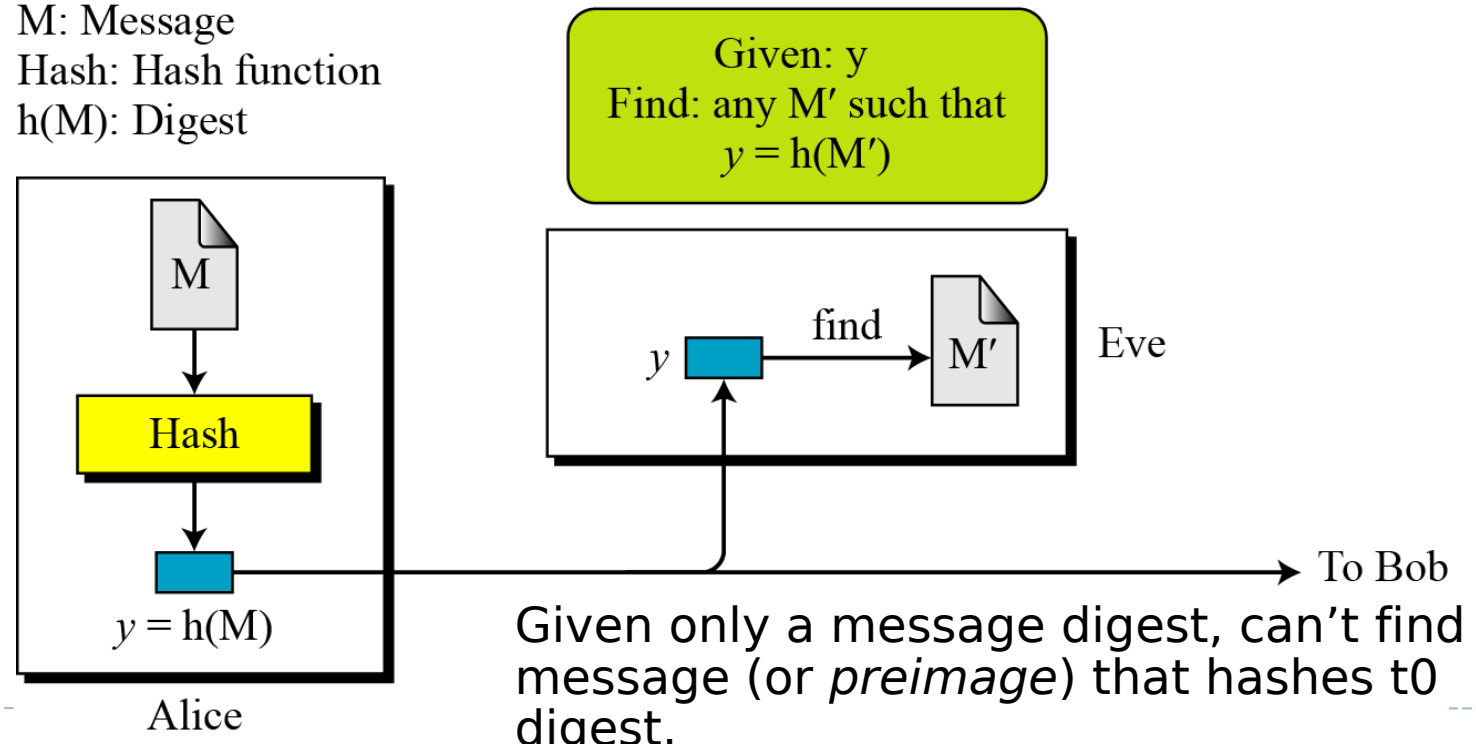
- Arbitrary-length message is hashed to fixed-length digest

Pre-image resistant

- This measures how difficult it is to devise a message which hashes to the known digest .
- Roughly speaking, the hash function must be one-way.

Preimage Attack

Given: y = h(M) **Find: M′ such that y = h(M′)**



Given only a message digest, can't find any message (or *preimage*) that hashes t0 digest.

Questions

- **Can we use a conventional lossless compression method such as *zip* as a cryptographic hash function?**

Answer : *No, a lossless compression method creates a compressed message that is reversible*.

- **Can we use a checksum function as a cryptographic hash function?**

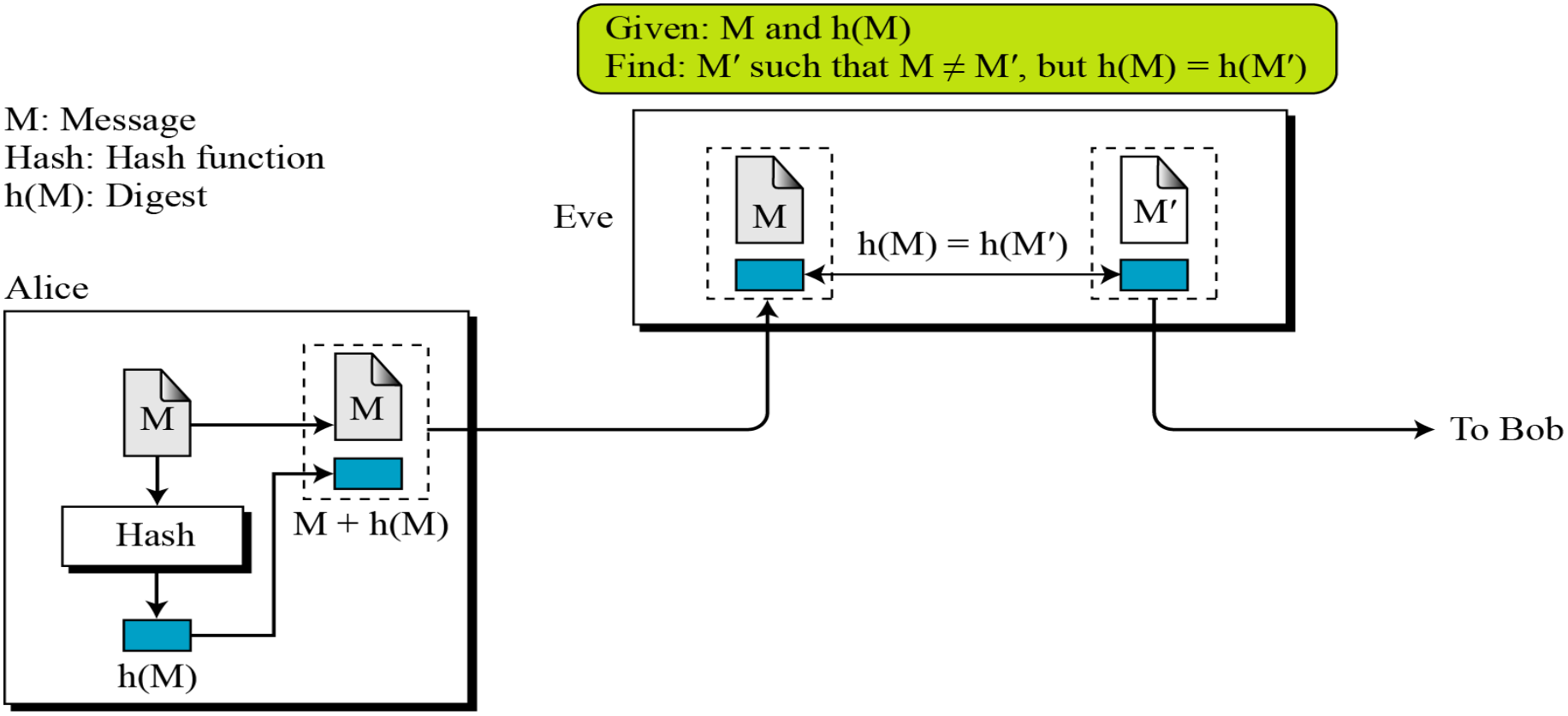
Answer : *No, a checksum function is not preimage resistant,* ***Eve*** *may find several messages whose checksum matches the given one.*

Second preimage resistant

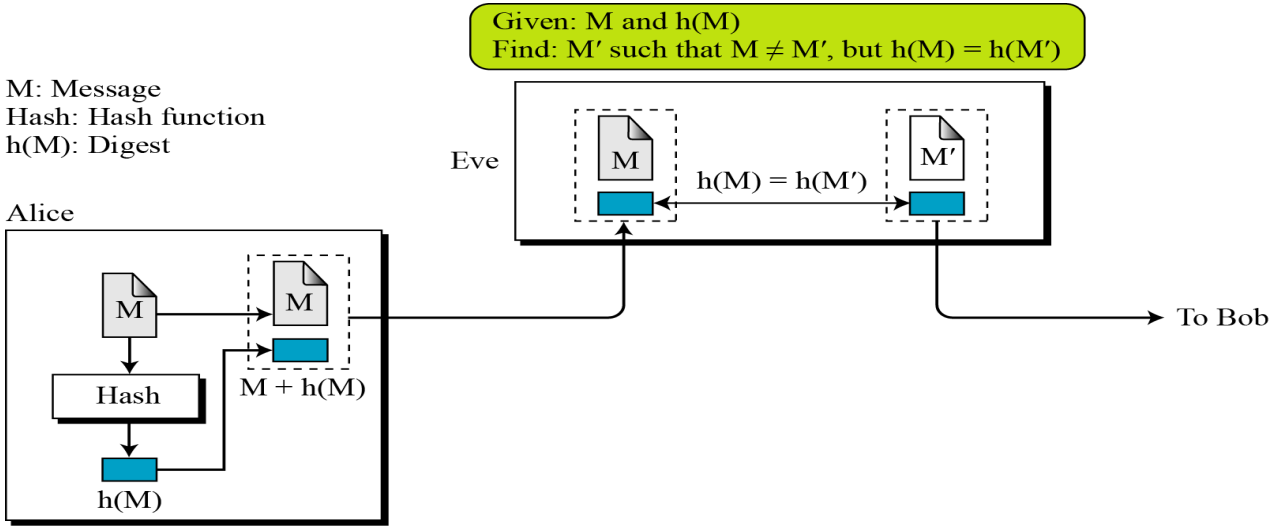
- This measures how difficult to devise a message which hashes to the known digest and its message

Second Preimage Attack

Given: M and h(M) **Find: M′ ≠ M such that h(M) = h(M′)**



Second preimage resistant



- Given one message, can't find another message that has the same message digest.
- An attack that finds a second message with the same message digest is a *second pre-image* attack.
  - It would be easy to forge new digital signatures from old signatures if the hash function used weren't second preimage resistant

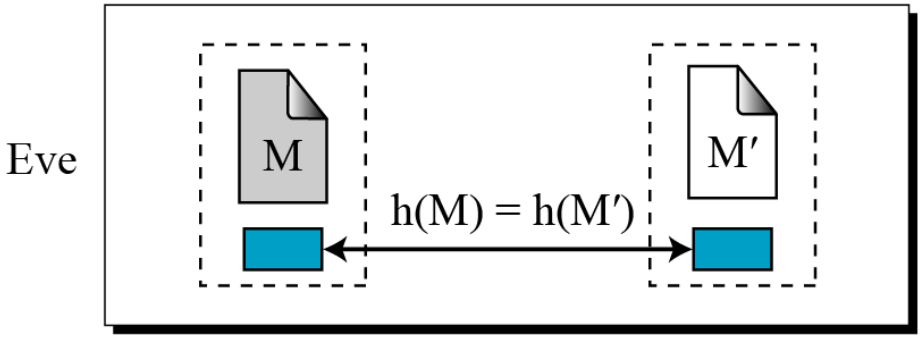
Collision Resistant

Collision Attack

Given: none **Find: $M' \neq M$ such that $h(M) = h(M')$**

M: Message
Hash: Hash function
h(M): Digest

Find: M and M′ such that $M \neq M'$, but $h(M) = h(M')$



- Can't find any two different messages with the same message digest
  - Collision resistance implies second preimage resistance
  - Collisions, if we could find them, would give signatories a way to repudiate their signatures

Hash Function Properties

Modification Detection Codes (MDCs)

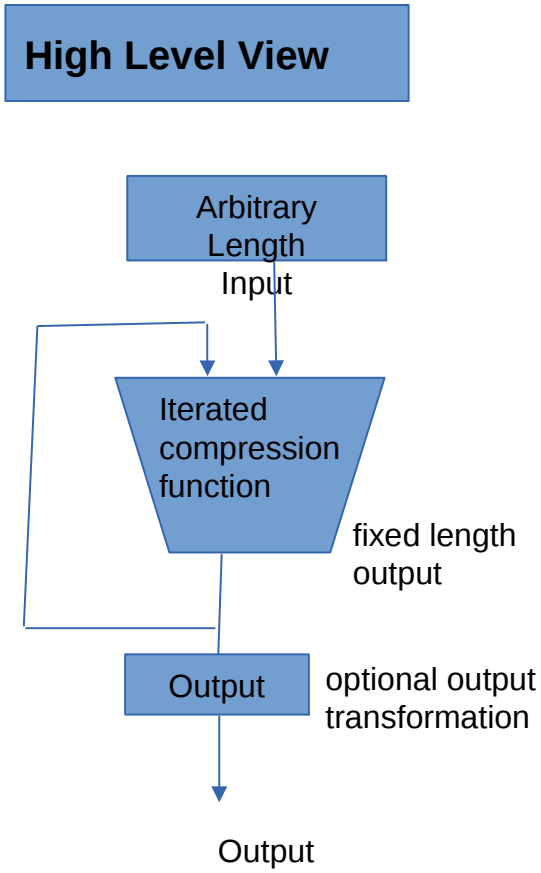
- Also known as manipulation detection codes**(MDCs)**, and less commonly as message integrity codes **(MICs),**
- The purpose of an **MDC** is to provide a representative image or hash of a message.
- The end goal is to facilitate, in conjunction with additional mechanisms data integrity assurances as required by specific applications.
- **MDCs** are a subclass of unkeyed hash functions, and can further be classified as:

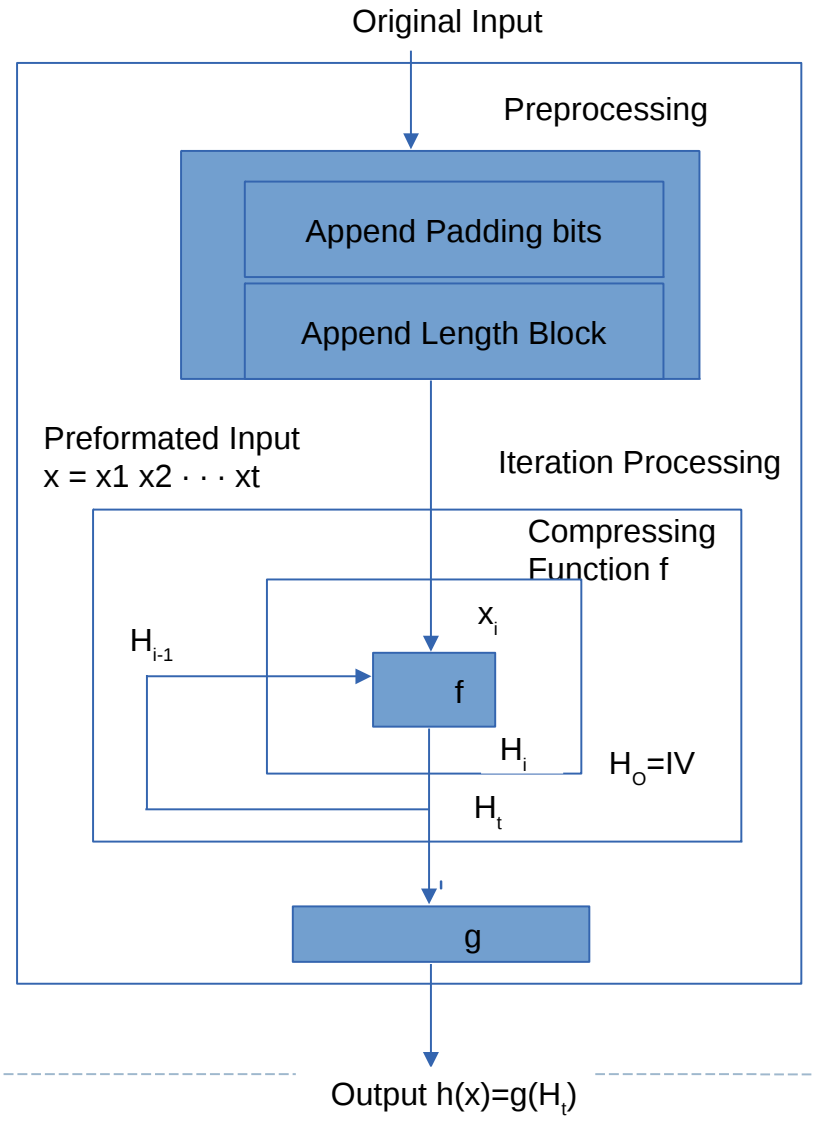
  - **One-way hash functions (OWHFs):** implying finding an input which hashes to a prespecified hash-value is difficult; (has a high preimage resistant)
  - **Collision resistant hash functions (CRHFs):** implying , finding any two inputs having the same hash-value is difficult. (has a high **Collision resistant**)

Topics

- Overview of Cryptography Hash Function
- Usages
- Properties
- **Hashing Function Structure** 
- Attack on Hash Function
- The Road to new Secure Hash Standard

The General Model for Iterated Hash Functions





The General Model for Iterated Hash Functions

Explanation

A hash input x of arbitrary finite length is divided into fixed-length r-bit blocks x i .

•This preprocessing typically involves appending extra bits (**padding**) as necessary to attain an overall bitlength which is a multiple of the blocklength r, and often includes a block or partial block indicating the bitlength of the unpadded input.

•Each block x_i then serves as input to an internal fixed-size hash function **f** , the compression function of **h**, which computes a new intermediate result of bitlength **n** for some fixed **n**, as a function of the previous **n-bit** intermediate result and the next input block xi .

•Let H_i denote the partial result after stage $_i$,

•The general process for an iterated **hash function with input x = x1 x2 . . . xt can be modeled as follows:**

$H_0 = IV$; $H_i = f(H_{i-1}, x_i)$, for $1 \le i \le t$; $h(x) = g(Ht)$.

The General Model for Iterated Hash Functions

Explanation

•The general process for an iterated **hash function with input x = x1 x2 . . . xt can be modeled as follows:**

$H_0 = IV$; $H_i = f(H_{i-1}, x_i)$, for $1 \le i \le t$; $h(x) = g(Ht)$.

Where:

1.H_{i-1} serves as the n-bit chaining variable between stage i− 1 and stage i, and

2.H_0 is a pre-defined starting value or initializing value (IV).

3.An optional output transformation g is used in a final step to map the n-bit chaining variable to an m-bit result $g(H_t)$;

4. g is often the identity mapping $g(H_t) = H_t$

Merkle–Damgard Scheme

The **Merkle–Damgård construction** or **Merkle–Damgård hash function** is a method of building **collision-resistant cryptographic** hash functions from **collision-resistant one-way compression** functions.

- The Merkle–Damgård hash function first applies an MD-compliant padding function to create an output whose size is a multiple of a fixed number (e.g. 512 or 1024) — this is because compression functions cannot handle inputs of **arbitrary size**.

- The hash function then breaks the result into blocks of fixed size, and processes them one at a time with the compression function, each time combining a block of the input with the output of the previous round.

- In order to make the construction secure, Merkle and Damgård proposed that messages be padded with a padding that encodes the length of the original message.
- This is called ***length padding*** or **Merkle–Damgård strengthening**.

Merkle-Damgard Scheme

- The one-way compression function is denoted by *f*, and transforms two fixed length inputs to an output of the same size as one of the inputs.

- The algorithm starts with an initial value, the **initialization vector (IV)**.
- The IV is a fixed value (algorithm or implementation specific)

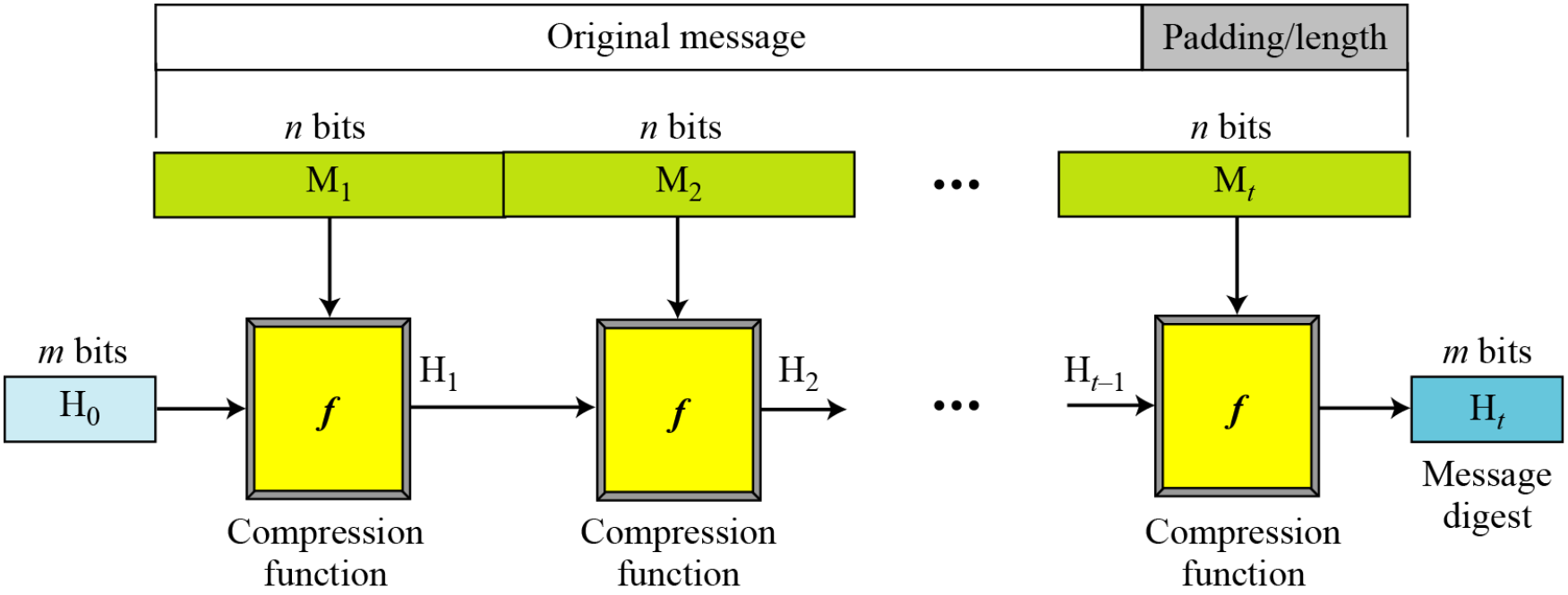
- For each message block, the compression (or compacting) function *f* takes the result so far, combines it with the message block, and produces an intermediate result.

- The last block is padded with zeros as needed and bits representing the length of the entire message are appended.

This construction was used in the design of many popular hash algorithms such as MD5, SHA1 and SHA2.

The SHA-2 hash function is implemented in some widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec

Merkle-Damgard Scheme



- Well-known method to build cryptographic hash function
- A message of arbitrary length is broken into blocks
  - length depends on the compression function *f*
  - padding the size of the message into a multiple of the block size.
  - sequentially process blocks , taking as input the result of the hash so far and the current message block, with the final fixed length output

Hash Functions Family

- **MD (Message Digest)**
  - Designed by Ron Rivest
  - Family: MD2, MD4, MD5
- **SHA (Secure Hash Algorithm)**
  - Designed by NIST
  - Family: SHA-0, SHA-1, and SHA-2
    - SHA-2: SHA-224, SHA-256, SHA-384, SHA-512
    - SHA-3: New standard in competition
- **RIPEMD (Race Integrity Primitive Evaluation Message Digest)**
  - Developed by Katholieke University Leuven Team
  - Family : RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320,

MD5, SHA-1, and RIPEMD-160

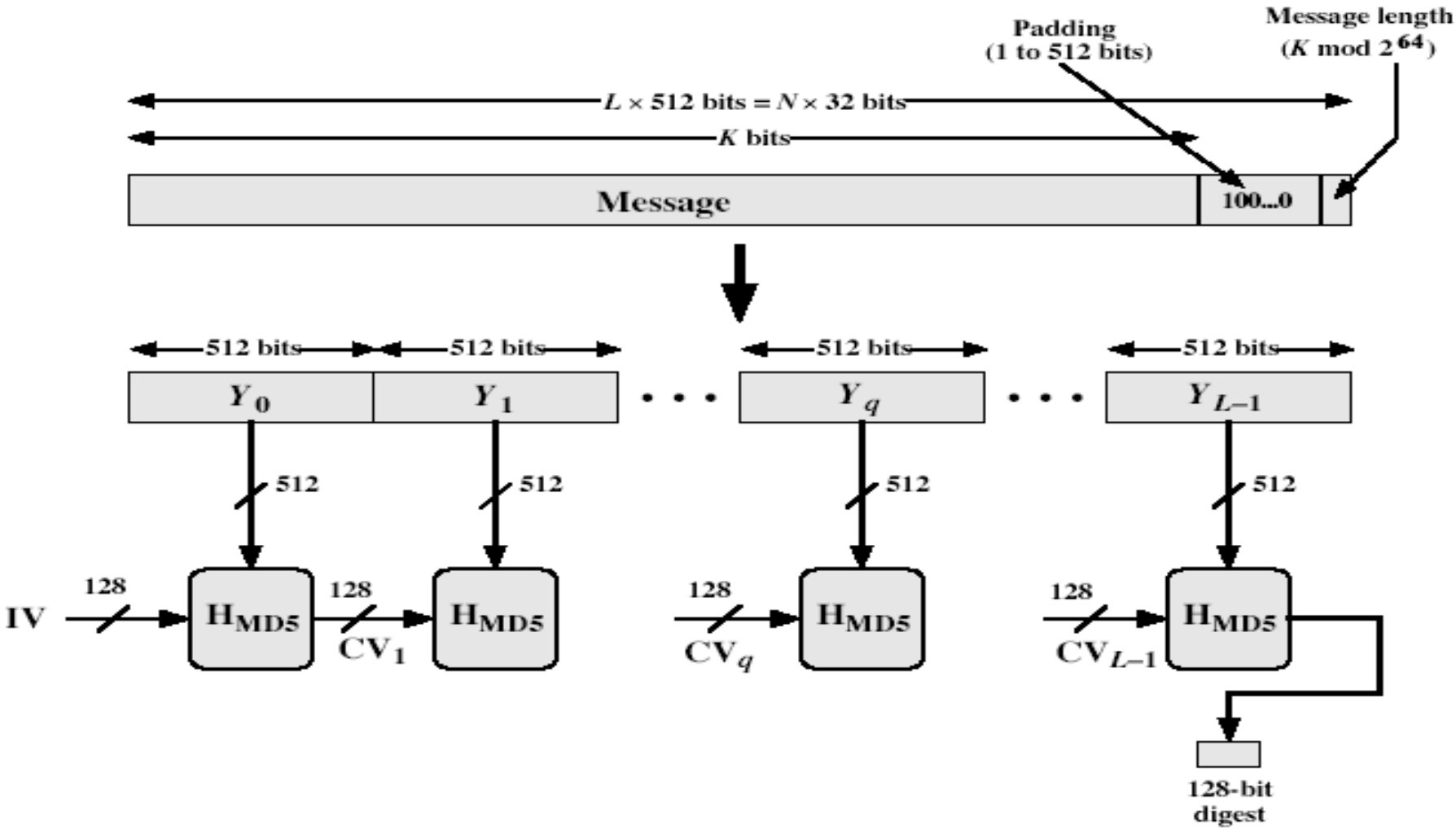
	MD5	SHA-1	RIPEMD-160
Digest length	128 bits	160 bits	160 bits
Basic unit of processing	512 bits	512 bits	512 bits
Number of steps	64 (4 rounds of 16)	80 (4 rounds of 20)	160 (5 paired rounds of 16)
Maximum message size	∞	$2^{64} - 1$ bits	$2^{64} - 1$ bits
Primitive logical functions	4	4	5
Additive constants used	64	4	9
Endianness	Little-endian	Big-endian	Little-endian

MD2, MD4 and MD5

- Family of one-way hash functions by Ronald Rivest
  - All produces 128 bits hash value

- **MD2: 1989**
  - Optimized for 8 bit computer
  - Collision found in 1995
- **MD4: 1990**
  - Full round collision attack found in 1995
- **MD5: 1992**
  - Specified as Internet standard in RFC 1321
  - since 1997 it was theoretically not so hard to create a collision
  - Practical Collision MD5 has been broken since 2004
  - CA attack published in 2007

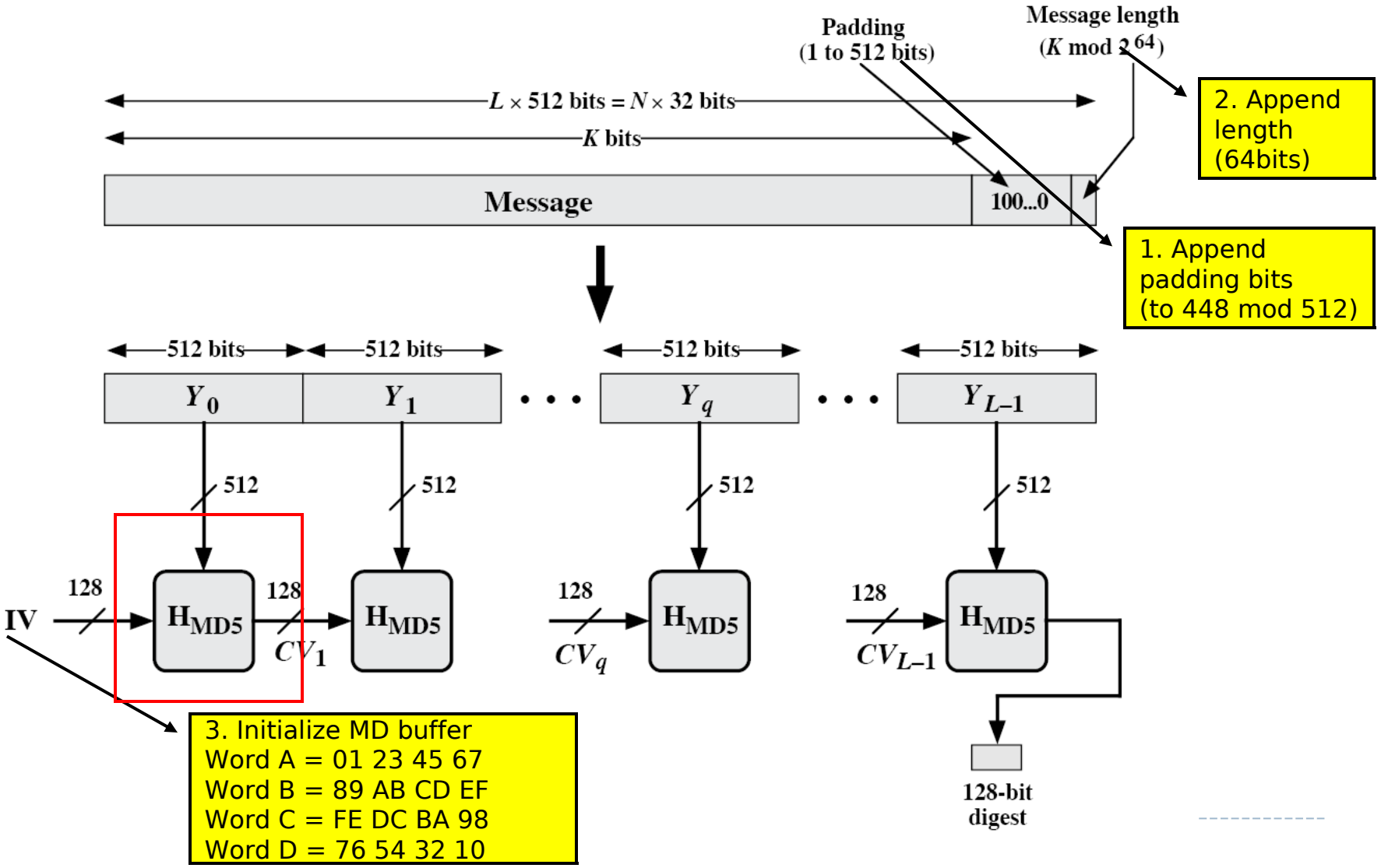
MD5 Overview



Topics

- Overview of Cryptography Hash Function
- Usages
- Properties
- **Hashing Function Structure**
  - **MD5**
  - SHA
- Attack on Hash Function
- The Road to new Secure Hash Standard

MD5 Overview



Secure Hash Algorithm

- SHA originally designed by NIST & NSA in 1993
- was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174
- Based on design of MD4 with key differences
- Produces 160-bit hash values
- Recent 2005 results on security of SHA-1 have raised concerns on its use in future applications

- **Linux Example**

```
# openssl sha1 myfile
SHA1(myfile)=da39a3ee5e6b4b0d3255bfef95601890afd80709
```

Secure Hash Algorithm

- **Linux Case Study : The Openssl Program**

- The digest functions output the message digest of a supplied file or files in hexadecimal form.
- They can also be used for digital signing and verification.

openssl dgst -c myfile

MD5(myfile)= d4:1d:8c:d9:8f:00:b2:04:e9:80:09:98:ec:f8:42:7e

Secure Hash Algorithm

- **Linux Case Study : The Openssl Program**

Standards COmmands

- **gendsa** : Generation of DSA Private Key from Parameters.
- **genpkey** : Generation of Private Key or Parameters.
- **genrsa** :Generation of RSA Private Key.
- **passwd** :Generation of hashed passwords.
- **pkey** :Public and private key management.
- **rand** :Generate pseudo-random bytes.
- **rsa** :RSA key management.
- **s_client** : This implements a generic SSL/TLS client which can establish a transparent connection to a remote server speaking SSL/TLS
- **s_server** : This implements a generic SSL/TLS server which accepts connections from remote clients speaking SSL/TLS.

Secure Hash Algorithm

- **Linux Case Study : The Openssl Program**

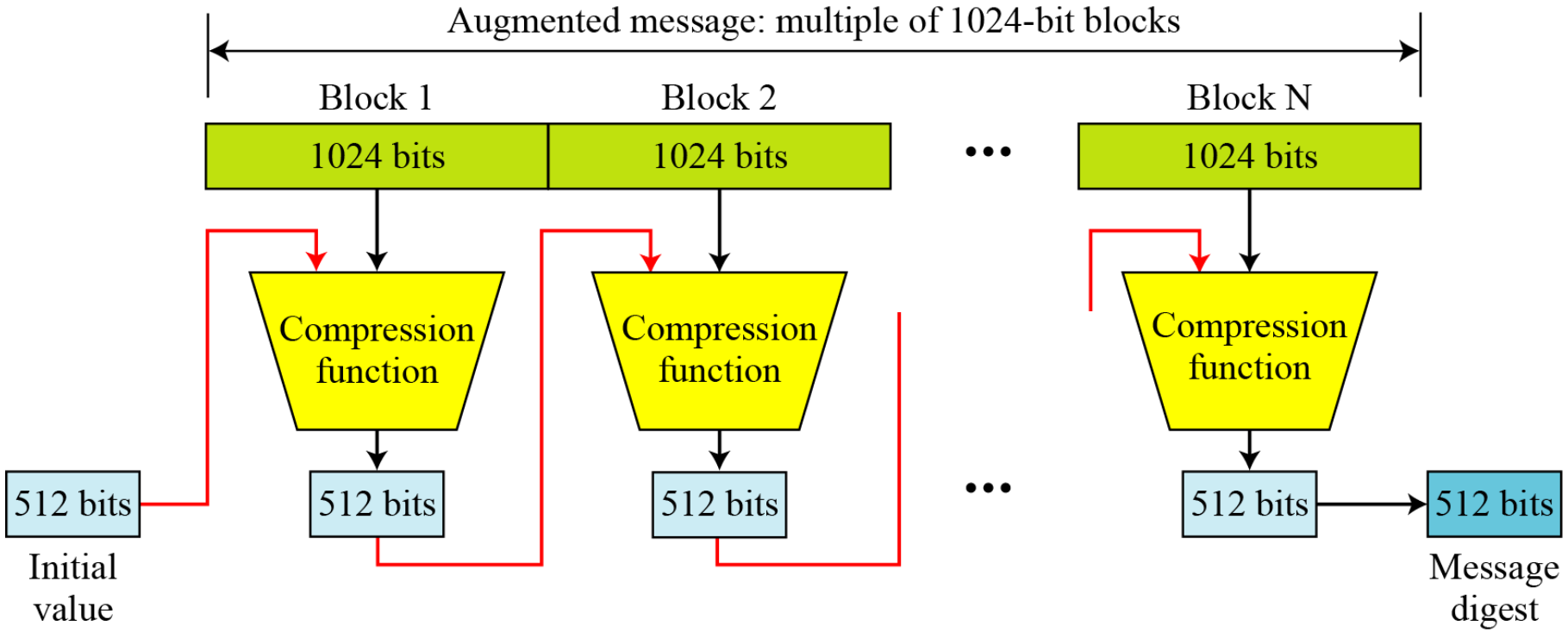
Standards COmmands

- **sess_id** SSL Session Data Management.
- **smime** S/MIME mail processing.
- **speed** Algorithm Speed Measurement.
- **ts** Time Stamping Authority tool (client/server)
- **verify** X.509 Certificate Verification.
- **version** OpenSSL Version Information.
- **x509** X.509 Certificate Data Management.

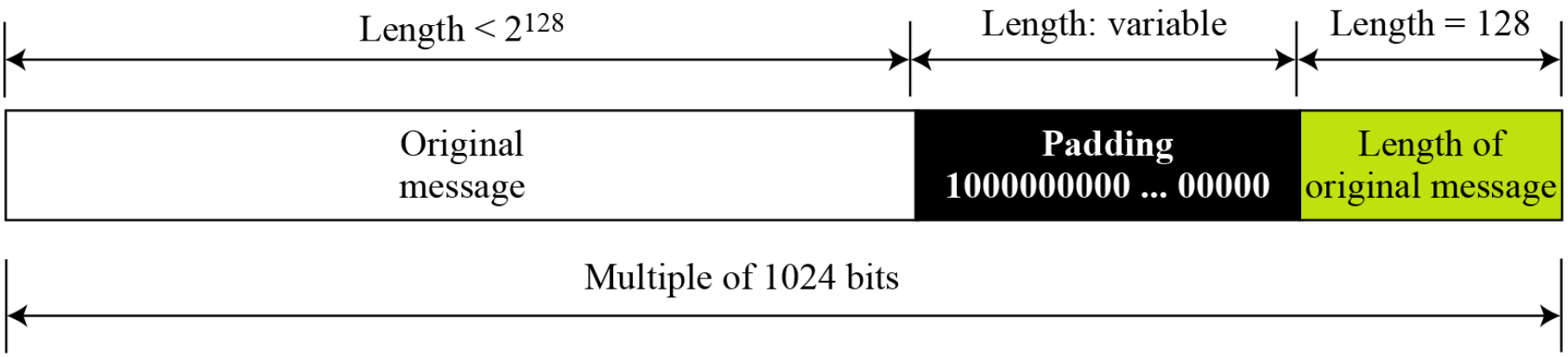
SHA Versions

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Digest size	160	224	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size	512	512	512	1024	1024
Word size	32	32	32	64	64
# of steps	80	64	64	80	80

SHA-512 Overview



Padding and length field in SHA-512

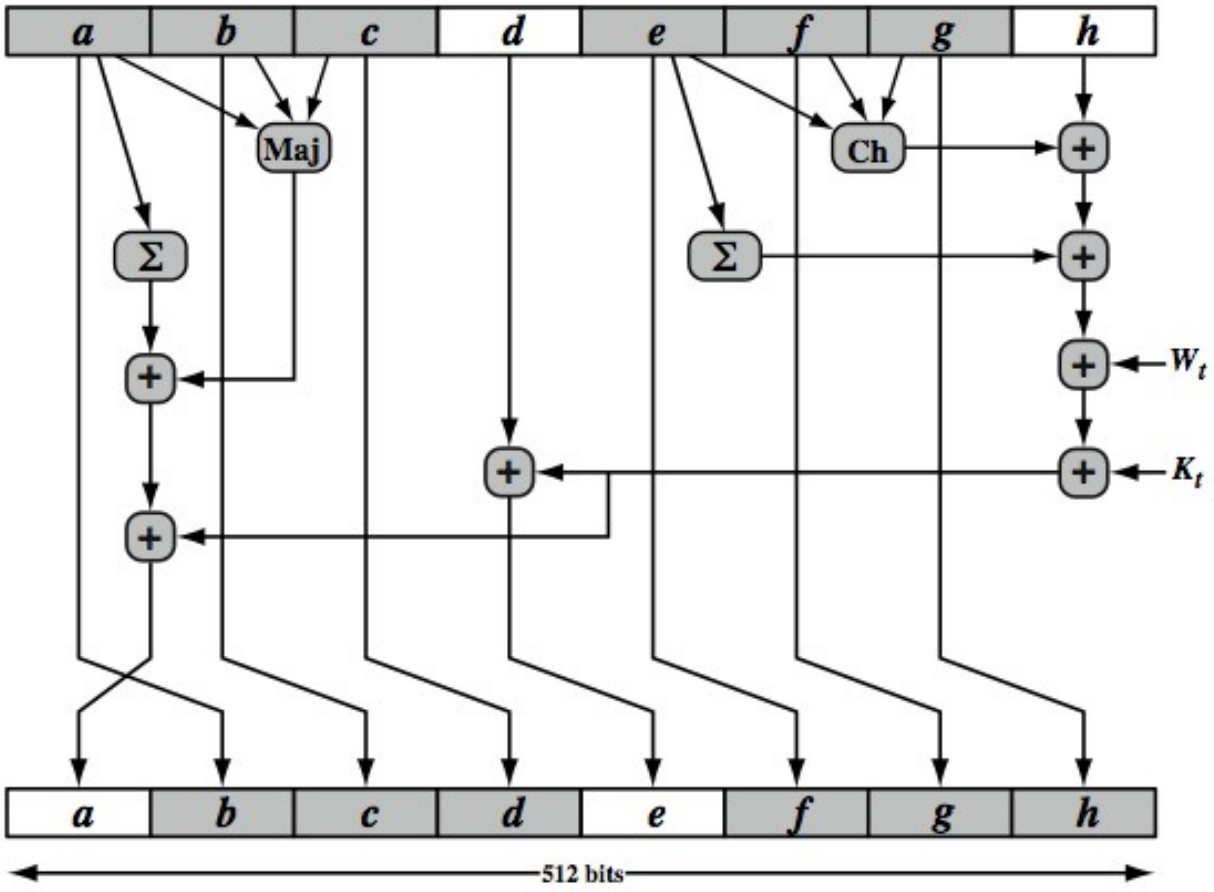


- **What is the number of padding bits if the length of the original message is 2590 bits?**
- We can calculate the number of padding bits as follows:

$$|P| = (-2590 - 128) \bmod 1024 = -2718 \bmod 1024 = 354$$

- The padding consists of one 1 followed by 353 0's.

SHA-512 Round Function

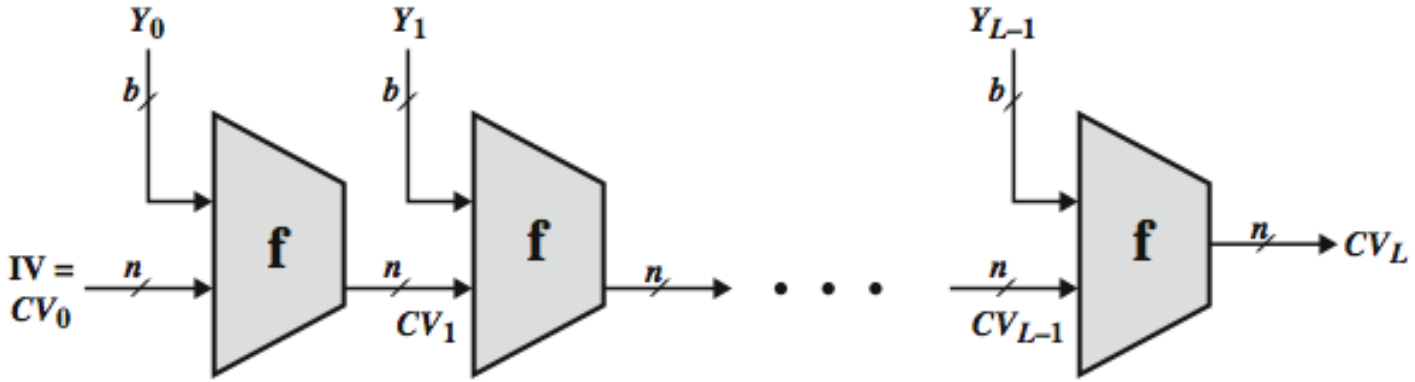


Topics

- Overview of Cryptography Hash Function
- Usages
- Properties
- Hashing Function Structure
  - MD5
  - SHA
- **Attack on Hash Function**
- The Road to new Secure Hash Standard

Hash Function Cryptanalysis

- cryptanalytic attacks exploit some property of alg so faster than exhaustive search
- hash functions use iterative structure
  - process message in blocks (incl length)
- attacks focus on collisions in function f



Attacks on Hash Functions

- Have brute-force attacks and cryptanalysis
- A preimage or second preimage attack
  - find *y* s.t. *H(y)* equals a given hash value
- collision resistance
  - find two messages x & *y* with same hash so `H(x) = H(y)`

Birthday Attack

- How many people do you need so that the probability of having two of them share the same birthday is > 50% ?
- N distinct values, k randomly chosen ones
  - P(N,i) = prob(i randomly selected values from 1..N have at least one match)
  - P(N,2) = 1/N
  - P(N,i+1) = P(N,i)+(1-P(N,i))(i/N)
- For P(N,k)>0.5, need $k \approx N^{1/2}$
- For m bits hash code, hence value $2^{m/2}$ determines strength of hash code against brute-force attacks
  - 128-bits inadequate, 160-bits suspect

Summary

- Hash functions are keyless
  - Applications for digital signatures and in message authentication codes
- The three security requirements for hash functions are
  - one-wayness, second preimage resistance and collision resistance
- MD5 is insecure
- Serious security weaknesses have been found in SHA-1
  - should be phased out
  - SHA-2 appears to be secure
  - But slow..
  - Use SHA-512 and use the first 256 bytes
- Next: Implementing OpenSSL in LInux

- HASHING CASE STUDY
- LINUX OpenSSL

Using OpenSSL In Hashing

- OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) network protocols and related cryptography standards required by them.
- It is program which has a command line tool for using the various cryptography functions of OpenSSL's crypto library from the shell.
- The program provides a rich variety of commands each of which often has a wealth of options and arguments
- The list of commands includes broadly:
  - List of Standard-commands
  - List of message-digest-commands
  - List of cipher-commands
  - List of public-key-algorithms supported.

Using OpenSSL In Hashing

- The openssl program can be used for :
  - Creation and management of private keys, public keys and parameters
  - Public key cryptographic operations
  - Creation of X.509 certificates, CSRs and CRLs
  - Calculation of Message Digests 
  - Encryption and Decryption with Ciphers
  - SSL/TLS Client and Server Tests
  - Handling of S/MIME signed or encrypted mail
  - Time Stamp requests, generation and verification

Using OpenSSL In Hashing

Syntax

1. **openssl** command [command_opts] [command_args]

   The commands Cloud be any commands from

   - list-standard-commands
   - List-message-digest-commands
   - list- cipher-commands
   - list-cipher-algorithms
   - list-message-digest-algorithms
   - list-public-key-algorithms

Using OpenSSL In Hashing

- **Examples of MESSAGE DIGEST COMMANDS**

- md2 MD2 Digest
- md5 MD5 Digest
- mdc2 MDC2 Digest
- rmd160 RMD-160 Digest
- sha SHA Digest
- sha1 SHA-1 Digest
- sha224 SHA-224 Digest
- sha256 SHA-256 Digest
- sha384 SHA-384 Digest
- sha512 SHA-512 Digest

Using OpenSSL In Hashing

- **To list down all the message digest algorithms Function**

  $ openssl list-message-digest-algorithms

- DSA
- DSA-SHA
- DSA-SHA1 => DSA
- DSA-SHA1-old => DSA-SHA1
- DSS1 => DSA-SHA1
- MD4
- MD5
- RIPEMD160
- RSA-MD4 => MD4
- RSA-MD5 => MD5
- RSA-RIPEMD160 => RIPEMD160
- RSA-SHA => SHA
- RSA-SHA1 => SHA1

Using OpenSSL In Hashing

- **To list down all the message digest algorithms Function**

  $ openssl list-message-digest-algorithms

- RSA-SHA1 => SHA1
- RSA-SHA1-2 => RSA-SHA1
- RSA-SHA224 => SHA224
- RSA-SHA256 => SHA256
- RSA-SHA384 => SHA384
- RSA-SHA512 => SHA512
- SHA
- SHA1
- SHA224
- SHA256
- SHA384
- SHA512
- DSA
- DSA-SHA
- dsaWithSHA1 => DSA

dss1 => DSA-SHA1

- ecdsa-with-SHA1
- MD4
- md4WithRSAEncryption => MD4
- MD5
- md5WithRSAEncryption => MD5
- ripemd => RIPEMD160
- RIPEMD160
- ripemd160WithRSA => RIPEMD160
- rmd160 => RIPEMD160
- SHA
- SHA1
- sha1WithRSAEncryption => SHA1
- SHA224
- sha224WithRSAEncryption => SHA224

SHA256

Using OpenSSL In Hashing

MESSAGE DIGEST COMMANDS Examples

- **Example 1:** Calculating an MD5 hash for a file:

- Create the a file plaintext with the content below

The programs included with the Centos system are free software;the exact distribution terms for each program are described in the individual files in /usr/share/doc//copyright.*

Centos comes with ABSOLUTELY NO WARRANTY, to the extent permitted byapplicable law

- Enter the command.

$ openssl dgst -md5 plaintext

- MD5(plaintext)= b96a46f1f55c293787c096d1e5f5ce31
- Modify the file, slightly by adding the line *"Centos is a free Open source Operating System"* to the end

$ openssl dgst -md5 plaintext

MD5(plaintext)= 5250fc32fdbe3025c7a8aff2853f74ae

Not that the hash is different

Using OpenSSL In Hashing

MESSAGE DIGEST COMMANDS Examples

- Other Digests that can be used include:

md4, md2, sha1, sha, mdc2, ripemd160

OPTIONS that can be used inlude:

-c print out the digest in two digit groups separated by colons, only relevant if hex format output is used.

-d print out BIO debugging information.

-hex digest is to be output as a hex dump. This is the default case for a "normal" digest as opposed to a digital signature.

-binary :output the digest or signature in binary form.

-out filename:filename to output to, or standard output by default.

-sign filename :digitally sign the digest using the private key in "filename".

Using OpenSSL In Hashing

MESSAGE DIGEST COMMANDS Examples

Generating different hashes and saving them to a file.

```
$ openssl dgst -md4 -binary -out dgst.bin plaintText

$ xxd dgst.bin
0000000: 5250 fc32 fdbe 3025 c7a8 aff2 853f 74ae  RP.2..0%.....?t.

#openssl dgst -sha1 -binary -out dgst.bin plaintText

# xxd dgst.bin
0000000: 207b 05d2 819c 8e42 d364 ef5c 5181 533d   {.....B.d.\Q.S=
0000010: 973b 87df
```

Note

The **xxd** command creates a hex dump of a given file or standard input. It can also convert a hex dump back to its original binary form.

Using OpenSSL In Hashing

MESSAGE DIGEST COMMANDS Examples

Generating different hashes and saving them to a file.

```
# openssl dgst -sha1 -c -out dgst.hex plaintText
# xxd dgst.hex

$ xxd dgst.bin
0000000: 5348 4131 2870 6c61 696e 7454 6578 7429  SHA1(plaintText)
0000010: 3d20 3230 3a37 623a 3035 3a64 323a 3831  = 20:7b:05:d2:81
0000020: 3a39 633a 3865 3a34 323a 6433 3a36 343a  :9c:8e:42:d3:64:
0000030: 6566 3a35 633a 3531 3a38 313a 3533 3a33  ef:5c:51:81:53:3
0000040: 643a 3937 3a33 623a 3837 3a64 660a       d:97:3b:87:df.
```

Using OpenSSL In Hashing

- **The Openssl Program**

Example 2: Using SHA-256 to generate digest,

```
$ openssl dgst -sha256 plaintext
SHA256(plaintText)=
   f530880da314ea13b5e90c30c00f370108196e0db5038f6d1d001ba701
   1ee3ec
$ openssl dgst -sha256 -binary -out dgst3.bin plaintext.in

$ xxd dgst3.bin
0000000: f530 880d a314 ea13 b5e9 0c30 c00f 3701  .0.........0..7.
0000010: 0819 6e0d b503 8f6d 1d00 1ba7 011e e3ec  ..n....m........
```

Using OpenSSL In Hashing

MESSAGE DIGEST COMMANDS Examples

Generating different hashes and saving them to a file.

```
#openssl dgst -ripemd160 -c -out dgst.hex plaintText
# xxd dgst.hex
0000000: 5249 5045 4d44 3136 3028 706c 6169 6e74  RIPEMD160(plaint
0000010: 5465 7874 293d 2037 393a 3932 3a34 633a  Text)= 79:92:4c:
0000020: 3033 3a66 333a 6232 3a31 613a 3437 3a30  03:f3:b2:1a:47:0
0000030: 353a 3366 3a65 393a 6437 3a31 373a 6433  5:3f:e9:d7:17:d3
0000040: 3a36 663a 3539 3a37 333a 6331 3a33 663a  :6f:59:73:c1:3f:
0000050: 6239 0a                                  b9.
```

Using OpenSSL In Hashing

MESSAGE DIGEST COMMANDS Examples

Generating different hashes and saving them to a file.

```
#openssl dgst -sha256 -c -out dgst.hex plaintText
# xxd dgst.hex
0000000: 5348 4132 3536 2870 6c61 696e 7454 6578  SHA256(plaintTex
0000010: 7429 3d20 6635 3a33 303a 3838 3a30 643a  t)= f5:30:88:0d:
0000020: 6133 3a31 343a 6561 3a31 333a 6235 3a65  a3:14:ea:13:b5:e
0000030: 393a 3063 3a33 303a 6330 3a30 663a 3337  9:0c:30:c0:0f:37
0000040: 3a30 313a 3038 3a31 393a 3665 3a30 643a  :01:08:19:6e:0d:
0000050: 6235 3a30 333a 3866 3a36 643a 3164 3a30  b5:03:8f:6d:1d:0
0000060: 303a 3162 3a61 373a 3031 3a31 653a 6533  0:1b:a7:01:1e:e3
0000070: 3a65 630a                                :ec.
```

Using OpenSSL In Hashing

Signing Message Digest

Step 1. Generate the private key using RSA algorithm and keep it in a file.(key.pem)

The genpkey command

The genpkey command generates a private key. It must use the option

-algorithm alg :

Where alg is any of the RSA, DSA or DH.

Using OpenSSL In Hashing

Signing Message Digest

Example

#openssl genpkey -algorithm RSA -out key.pem

cat key.pem

-----BEGIN PRIVATE KEY-----
MIICdQIBADANBgkqhkiG9w0BAQEFAASCAl8wggJbAgEAAoGBAJ1xtBvPLB5hD5WY
a51wd/ys8EtTcrQ9Hf7jwRpkvQniw8l10vPIE6Np1Vf22G16C/H+l1H+ZEzfjwfh
HbmJJ2vChxxCZMtUsQFd0SI3DkK3qg7r1oYn5wOTE2NkJ+7mjslcFqMAOEgJJTEi
TNir9VK5oTeYejbmo4anh6iBPdgTAgMBAAECgYBabXj90/LKDANQb3e3uGYh6Q4m
pWonHUdCI3vAdgWhTO4YoqSjwdGNtPaFDFDoKAX+WrR8snjXMjvLUb+p1Z2evz/r
+hB4UQbQ/C5HFQBfS+X0sxLHNJz9gVZQIXUNnN655LUEKCzYedRyV5tdJtC7WSBK
3lAT1ip+kF/uaSIi4QJBAMl6JAUOTwzVa8XDET/iaCETc+M185WvNHSmK3IWvk3c
snZMZVkhUqw8d+zBJGmsxyn6nX0dwS/HC7NplO2Ks/ECQQDIDRA4aSpdbFGAXW0s
QDGj8aXVaWgvH+364OileNMOAyIFcWKtMnGs8CYrTGhyJnh28eIiFnt635OdgI7K
vMBDAkAJG+eS/vGd/+wudcJK5B6XGD0OEbtgkhpKB9VBBDw4YvkCljOi0vnc5aL6
ZkUSLgiXIkiKhpTcZyBlTSm5j6LRAkB5wsG5pADeJfyhRbwaL+RG5eDKuUJpVGTT
yyqu9JB5OdzuNSobQtW/rdd9iR8VQ2cU9nptxwNXMVe2iLXgnevjAkBRYC03b6yO
QofFnpd7am/Za5UJ1RmQM87HdKzLT0UskX1oI5pNeFMFWxRuS+yXXQEMaGZw2p+1
XSfWnipmLaXS
-----END PRIVATE KEY-----

Using OpenSSL In Hashing

Signing Message Digest

Step 2. We use the -sign option to sign the digest before outputting the signed output to the file dgst.hex)

```
#openssl dgst -sha256 -c -sign key.pem  -out dgst.hex plaintText
```

```
# xxd dgst.hex
0000000: ab69 365c 01d4 d1b8 edf5 1ba5 2719 c204  .i6\........'...
0000010: 0f52 d599 1f44 7dd4 ec99 3250 d9a9 f844  .R...D}...2P...D
0000020: 6d1d f9ff 1eee 5c54 3da2 9656 52e9 1303  m.....\T=..VR...
0000030: 23c8 9c67 9fd4 048a 10f6 96f6 52a8 8bd1  #..g........R...
0000040: 1612 6bd0 6a2e b053 1c0d eda9 aae0 6b7c  ..k.j..S......k|
0000050: 7f61 b670 34e4 ff6e b5f2 0791 aa73 aa50  .a.p4..n.....s.P
0000060: e353 818f f844 7624 544e 0d47 9087 9972  .S...Dv$TN.G...r
0000070: dcfc ddce 99e8 838a a249 4d21 9d91 a82e  .........IM!....
```

Using OpenSSL In Hashing

- **Later WE will Use OpenSSL to:**
  - **Message Encryption/Decryption(Symmetric Key)**
  - **Message Encryption/Decryption (Public Key)**
  - **Message Authentication**
  - **Digital Signature**
  - **Signature Verification**
  - **Certificate management**