Cryptographic Hash Functions

FALGUNI ROY

AP, IIT, NSTU

What is hashing?

- a one-way process that converts input data of any size into fixed-length enciphered data.
- A hash function is a mathematical function that converts a numerical input value into another compressed numerical value
- Basically, you can take either a short sentence or an entire stream of data, run it through a hash function, and wind up with a string of data of a specific length.
- A way to hide your original data to make it as challenging as possible to reverse engineer.

What is hashing? (more)

- it’s a technique that uses a **mathematical operation** to **shrink** a random quantity of input data (called a hash key) into a fixed-length string of bits in a way that’s too impractical to reverse with modern computers.
- So, the definition of a hash function would be something that takes input data and uses it to create a fixed-length output value that’s unique and virtually irreversible (for all practical intents and purposes).

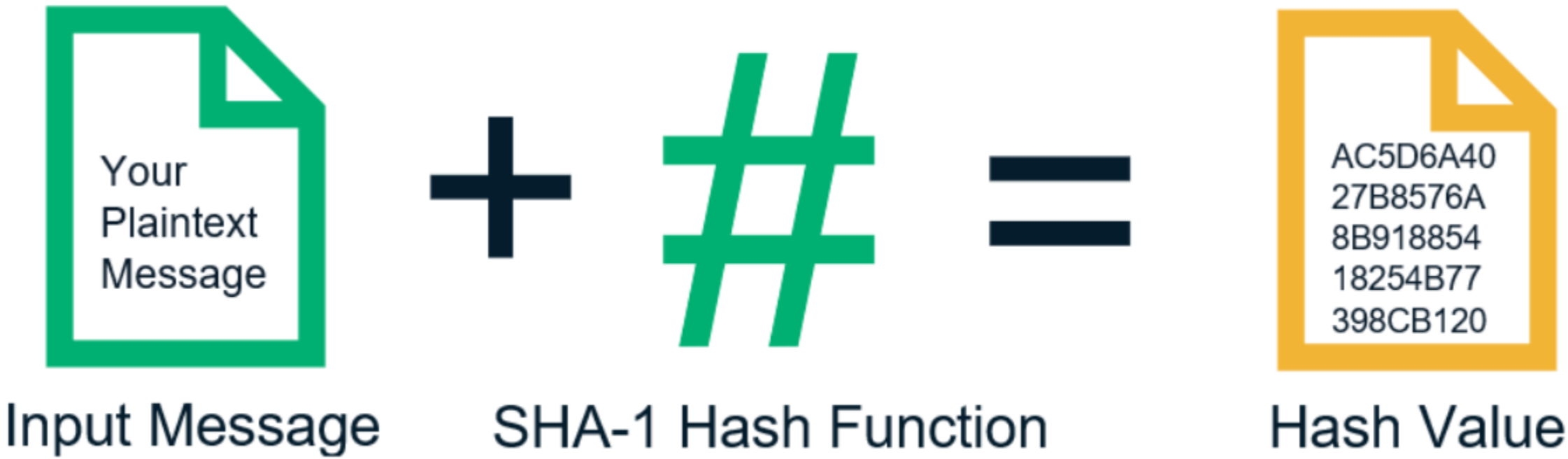
Output Values

- The output values returned by a hash function are called by a few different names:
    - Hash values,
    - Digests,
    - Hash codes, or
    - Hashes

Hashing vs Encryption

- **Encryption** is something you can use to convert plaintext (readable) data into something indecipherable using algorithms and a key.
- However, you can decrypt that data either by using the same (symmetric encryption) or a mathematically-different-but-related cryptographic key (asymmetric encryption).
- In **Hashing, o**nce you hash data, you can’t restore it to its original format because it’s a one-way process.

An Example of a Hash Function



Input Message SHA-1 Hash Function Hash Value

Length Of The Output

- depends on the hashing algorithm.
- Hash values can be 160 bits for SHA-1 hashes, or 256 bits, 384 bits, or 512 bits for the SHA-2 family of hashes.
- They’re typically displayed in hexadecimal characters.
- The input data’s quantity and size can be varied, but the output value always remains the same in terms of size.

For example, let's consider the following hash inputs and outputs:

Example Input Texts	Hash Values Using SHA-1
Hello	D364965C90C53DBF14064B9AF4BAABCA72196E2E
Hello! You are reading an article about the cryptographic hash function!	B26BACAB73C46D844CABEC26CE32B030FED1164F

- Hash functions are used in several different ways
  - Ensuring data integrity,
  - Creating and verify digital signatures (which are encrypted hashes), and
  - Facilitating secure password storage.

Types of Cryptographic Hash Algorithms

- The SHA family (SHA-1, SHA-2 [including SHA-256 and SHA-512], and SHA-3)
- The MD family (MD)
- Whirlpool,
- Tiger,
- NTLM, and
- LanMan (LM hash).

Cryptographic Hash Properties

- **Determinism** — Regardless of the size of the input or the key value, the operation should always result in the same consistent length output or hash value.
- **Computational Speed** — The speed of a hash function is important and should vary based on how it's being used. For example, in some cases, you need a fast hash function, whereas in others it's better to use a slow hash function.
- **Image Resistance** — Hashes should be extremely impractical to reverse (i.e., it should serve as a one-way function for all intents and purposes). The hash function should be so difficult and make the data so obscure that it would be improbable for someone to reverse engineer the hash to determine its original key value. Even one tiny change to the original input should result in an entirely different hash value.

Cryptographic Hash Properties (more details)

- **Pre-Image Resistance**
  - This property means that it should be computationally hard to reverse a hash function.
  - if given y $\in$Y it is computationally infeasible to find a value x $\in$X s.t. h(x) = y
  - This property protects against an attacker who only has a hash value and is trying to find the input.

Cryptographic Hash Properties (more details)

- Second Pre-Image Resistance
  - This property means given an input and its hash, it should be hard to find a different input with the same hash.
  - In other words, if a hash function h for an input x produces hash value h(x), then it should be difficult to find any other input value y such that h(y) = h(x).
  - This property of hash function protects against an attacker who has an input value and its hash, and wants to substitute different value as legitimate value in place of original input value.

Cryptographic Hash Properties (more details)

- **Collision Resistance**
  - This property means it should be hard to find two different inputs of any length that result in the same hash. This property is also referred to as collision free hash function.
  - In other words, for a hash function h, it is hard to find any two different inputs x and y such that h(x) = h(y).
  - Since, hash function is compressing function with fixed hash length, it is impossible for a hash function not to have collisions. This property of collision free only confirms that these collisions should be hard to find.
  - This property makes it very difficult for an attacker to find two input values with the same hash.
  - Also, if a hash function is collision-resistant then it is second pre-image resistant.

Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	$H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value h, it is computationally infeasible to find y such that $H(y) = h$.
Second preimage resistant (weak collision resistant)	For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

Characteristics of a Hash Function in Cryptography

- **1) A Hash Function Is Practically Irreversible**
  - Hashing is often considered a type of one-way function. That's because it's highly infeasible (technically possible, though) to reverse it because of the amount of time and computational resources that would be involved in doing so. That means you can't figure out the original data based on the hash value without an impractical amount of resources at your disposal.
  - In other words, if the hash function is h, and the input value is x, the hash value will be h(x). If you have access to h(x) and know the value of hash function h, it's (almost) impossible to figure out the value of x.

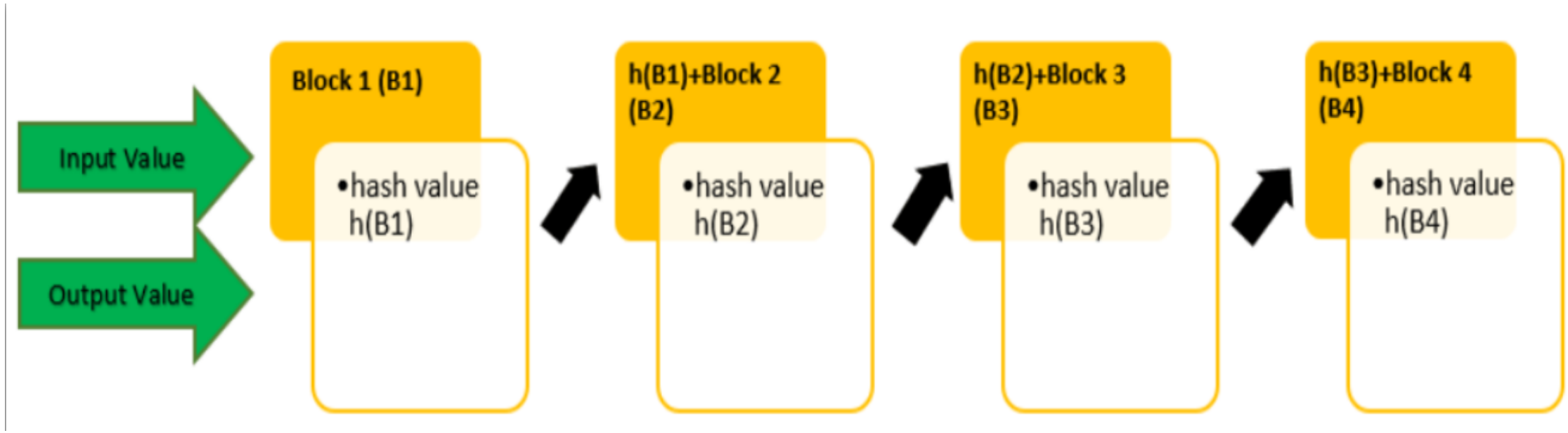
Characteristics of a Hash Function in Cryptography

- **2) Hash Values Are Unique**
  - No two different input data should (ideally) generate the same hash value. If they do match, it causes what's known as a collision, which means the algorithm isn't safe to use and is vulnerable to what are known as birthday attacks. Collision resistance is something that improves the strength of your hash and helps to keep the data more secure. That's because a cybercriminal would not only have to crack not only the hash value but the salt value, too.

  - So, if the hash function is h, and there are two different input data sets x and y, the hash value of h(x) should always be different than h(y). Hence, $h(x) \neq h(y)$. What this means is that if you make the slightest change in the original data, it's hash value changes. Hence, no data tampering goes unnoticed.

How Does Hashing Work?

- First of all, the hashing algorithm divides the large input data into blocks of equal size.
- The algorithm then applies the hashing process to each data block separately.
- Although one block is hashed individually, all of the blocks are interrelated.
- The hash value of the first data block is considered an input value and is added to the second data block.
- In the same way, the hashed output of the second block is lumped with the third block, and the combined input value is hashed again.
- the cycle continues until you get the final has output, which is the combined value of all the blocks that were involved.
- That means if any block's data is tampered with, its hash value changes.
- And because its hash value is fed as an input into the blocks that follow, all of the hash values alter.
- This is how even the smallest change in the input data is detectable as it changes the entire hash value.


Input Value
Output Value
Block 1 (B1)
•hash value h(B1)
h(B1)+Block 2 (B2)
•hash value h(B2)
h(B2)+Block 3 (B3)
•hash value h(B3)
h(B3)+Block 4 (B4)
•hash value h(B4)

Main Features of a Hash Function in Cryptography

- **It Enables Users to Identify Whether Data Has Been Tampered With**
  - When generated using a unique and random number, all hash values are different.
  - So, if an attacker tries to modify, alter, or remove any part of the original input data (text data, software, application, email content, and even the media file), its hash value changes.
  - As soon as the hash value changes, the users are notified about it.
  - Users will immediately know that a message's content or a software application is not in the same condition as it was sent or created by the original sender/developer.
  - Hence, if a hacker inserts malicious code into a software program, for example, the user gets a warning not to download or install it because it's been altered.
  - Likewise, if an attacker changes the content of an email to trick recipients into sharing their confidential information, transfer funds, or download a malicious attachment, users will know that the message was modified.
  - Therefore, they should not take any actions suggested in the message.

Main Features of a Hash Function in Cryptography

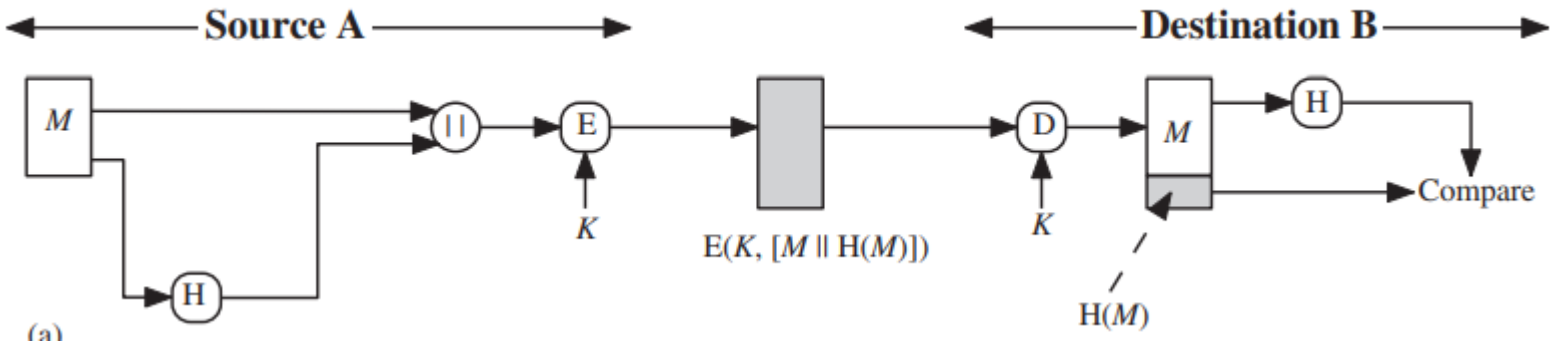
- **A Hash Function Prevents Your Data from Being Reverse Engineered**
  - Once you apply a hash function to data, you're left with an incomprehensible output.
  - So, even if an intruder manages to get their hands on the data's hashed values through a leaky database or by stealing it through a cyber attack, they can't easily interpret or guess the original (input) data.
  - Because the hash value can't be reversed easily using modern resources, it's improbably for hackers to decipher the hash value even if they know which hash function (algorithm) has been used to hash the data.
  - It's just infeasible due to the amount of resources and time such a process would require at scale.
  - cryptographic hash serves as a means of data protection while data is traveling or at-rest.

Applications of Cryptographic Hash Functions

- Digital signatures,
- Biometrics,
- Password storage,
- SSL/TLS certificates,
- Code signing certificates,
- Document signing certificates, and
- Email signing certificates.

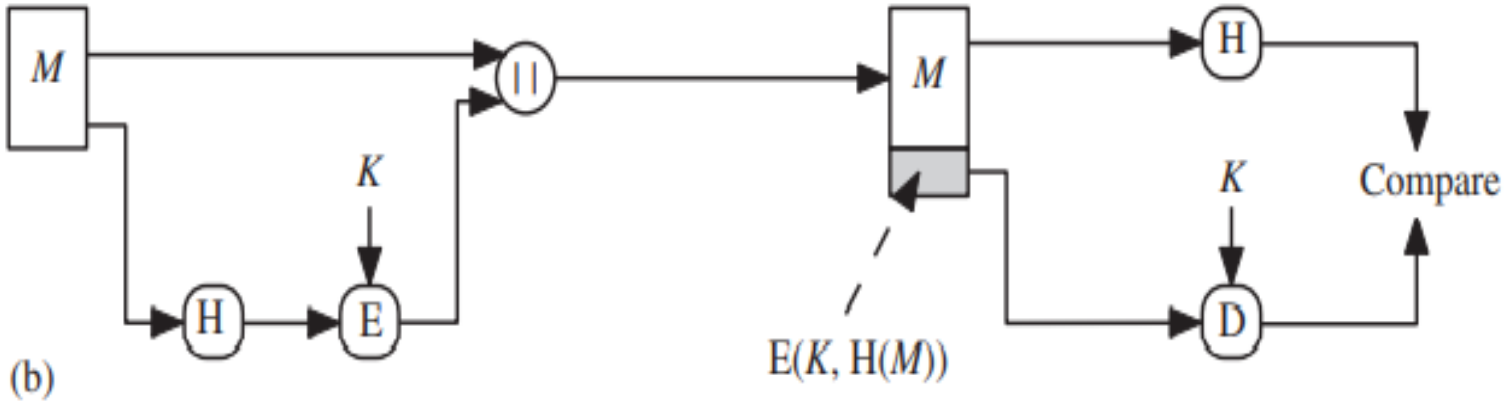
Hash Functions & Message Authentication

- The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.



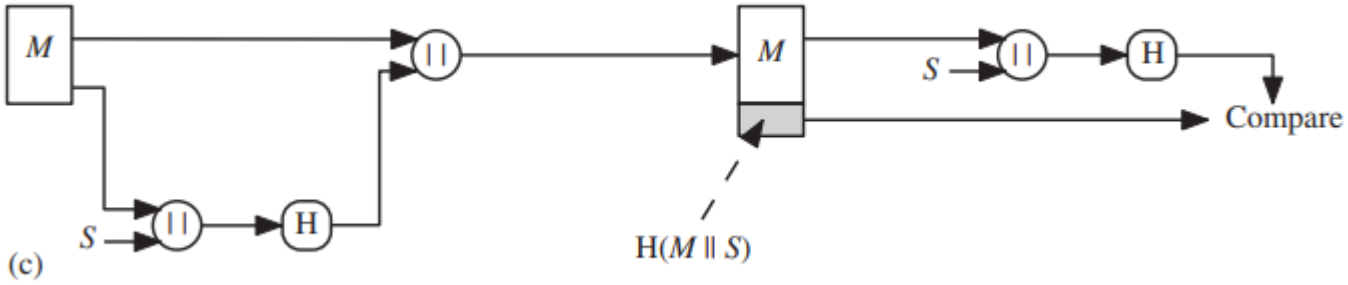
Hash Functions & Message Authentication

- Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.



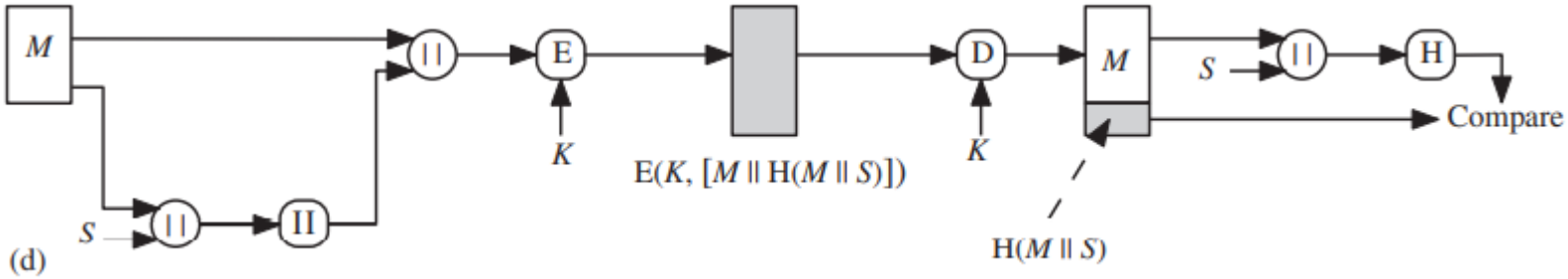
Hash Functions & Message Authentication

- It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value S. A computes the hash value over the concatenation of M and S and appends the resulting hash value to M. Because B possesses S, it can recompute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message



Hash Functions & Message Authentication

- Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code



Reason of growing interest in techniques that avoid encryption

- Encryption software is relatively slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.
- Encryption hardware costs are not negligible. Low-cost chip implementations of DES are available, but the cost adds up if all nodes in a network must have this capability.
- Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.
- Encryption algorithms may be covered by patents, and there is a cost associated with licensing their use.

How Hashing Works in Code Signing

- Say, you're a software publisher or developer who uses code signing certificates to digitally sign your downloadable software, scripts, applications, and executable.
- This certificate enables you to assure your users, clients, and their operating systems about your identity (i.e., that you're you) and that your product is legitimate.
- It also uses a hash function that warns them if it's been tampered with since you originally signed it.
- Once you have a final version of your code ready to go, you can put the code signing certificate to work. This means that the code signing certificate hashes the entire software, and the hash gets encrypted, which creates the publisher or developer's digital signature.

How Hashing Works in Code Signing

How Hashing Works in Code Signing

- So, when a user downloads your software, their OS generates a hash value to see whether it matches the original hash value of your software.
- If it does, that’s great and means they can proceed safely with that knowledge in mind.
- But if someone tries to pull a sleight of hand and change your software or your digital signature, the hash value they generate will no longer match your original hash, and the user will be notified about the compromise.
- This means that an unmodified hash value vouches for the integrity of your software. So, in this case, the cryptographic hash function ensures that no one can modify your software without someone noticing.

How Hashing Works in Password Storage

- If you're a business or organization that allows your users to store their passwords on your site, then this next part is especially important for you.
- When a user stores their password on your site (i.e., on your server), there's a process that takes place that applies a hash function to their plaintext password (hash input). This creates a hash digest that your server stores within its password list or database.
- There isn't a list of your users' original plaintext passwords anywhere on your server that your employees (or any cybercriminals) could get their hands on. The hashing process takes place within the server, and there's no "original file" of plaintext data for them to exploit.
- This is different from encryption, which involves the use of an encryption key to encrypt data and a decryption key that can decrypt it. Remember, with hashing, the goal is for the data to not be reverted to its original plaintext format (i.e., to only be a one-way function). With encryption, on the other hand, the goal is for the encrypted data to be decryptable with the right key (i.e., a two-way function).

What Is Salting & Why Do You Use It with Password Hash Functions?

- **Salting** means adding randomly generated characters to the input values before hashing them.
- It’s a technique that’s used in password hashing.
- It makes the hashing values unique and more difficult to crack.

What Is Salting & Why Do You Use It with Password Hash Functions?

- Suppose Bob and Alice have the same password (“Sunshine”) for a social media site. The site is using SHA-2 to store the passwords. Because the input value is same, their hash values are going to be the same “8BB0CF6EB9B17D0F7D22B456F121257DC1254E1F01665370476383EA776DF414.”

- Now, let’s suppose a hacker manages to discover Bob’s password (input value) using malware, brute force attacks, or by using other advanced hash cracking tools. They can bypass the authentication mechanism of all other accounts that have the same password “Sunshine.” They just need to see the table of hash values and find the user IDs having the same hash value in their password column.

What Is Salting & Why Do You Use It with Password Hash Functions?

- This is where salting comes in handy. Here, some random alphanumeric characters are added to the input values. So, suppose the salt “ABC123” is added to Bob’s password, and “ABC567” is added to Alice’s password. When the system stores the password, it stores the hash value for the inputs “SunshineABC123” and “SunshineABC567”. Now, even if both the original passwords are the same, their hash values are different because of the salts that were added. And the hacker can’t access Alice’s account even if they have managed to steal Bob’s password.

- This is the big difference between encryption and hashing. While encryption is also a process that converts plaintext data into incomprehensible format using a key, you can use the same or another key to decrypt it. With hashing, on the other hand, it uses a hash function to map your input data to a fixed-length output. This is something that you can’t restore because it essentially serves as a one-way process.

Hash Function Weaknesses

- In the past, there were incidences where popular algorithms like MD5 and SHA-1 were producing the same hash value for different data. Hence, the quality of collision-resistance was compromised.
- There is a technology named “rainbow tables” that hackers use to try to crack unsalted hash values. This is why salting before hashing is so crucial to secure password storage.
- There are some software services and hardware tools (called “hash cracking rigs”) that attackers, security researchers, or even government agencies use to crack the hashed passwords.
- Some types of brute force attacks can crack the hashed data.

Attacks On Hash Functions

- brute-force attacks and
- cryptanalysis

Brute-Force Attacks

- **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 $H(x) = H(y)$
- hence value $2^{m/2}$ determines strength of hash code against brute-force attacks
  - 128-bits inadequate, 160-bits suspect

Birthday Attacks

- might think a 64-bit hash is secure
- but by **Birthday Paradox** is not
- **birthday attack** works thus:
  - given user prepared to sign a valid message x
  - opponent generates $2^{m/2}$ variations x' of x, all with essentially the same meaning, and saves them
  - opponent generates $2^{m/2}$ variations y' of a desired fraudulent message y
  - two sets of messages are compared to find pair with same hash (probability > 0.5 by birthday paradox)
  - have user sign the valid message, then substitute the forgery which will have a valid signature
- conclusion is that need to use larger MAC/hash

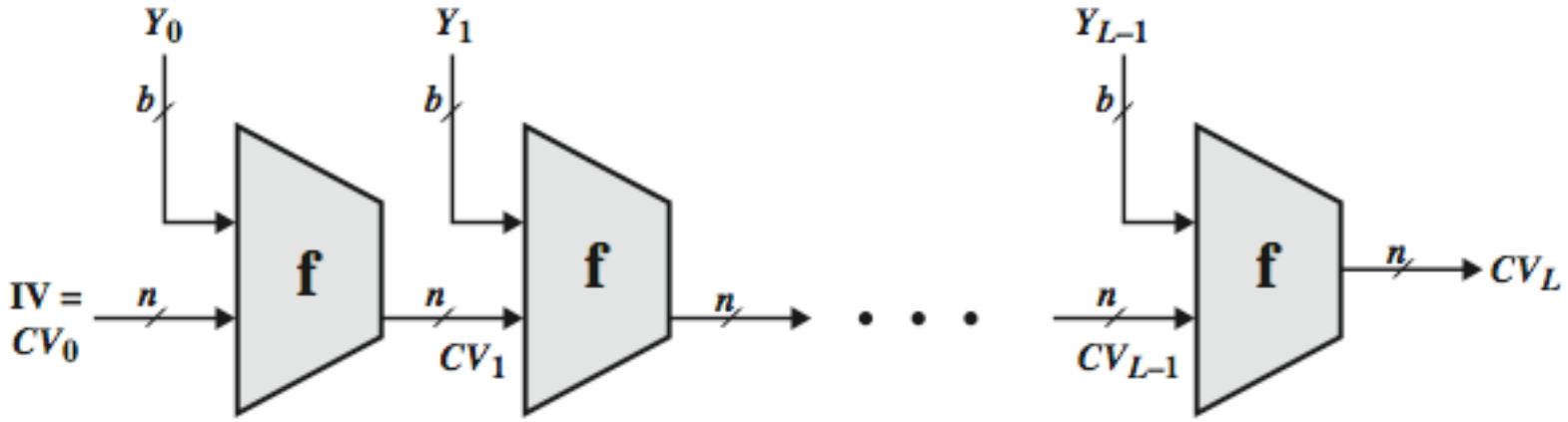
As { the / — } Dean of Blakewell College, I have { had the pleasure of knowing / known } Cherise Rosetti for the { last / past } four years. She { has been / was } { a tremendous / an outstanding } { asset to / role model in } { our / the } school. I { would like to take this opportunity to / wholeheartedly } recommend Cherise for your { school's / — } graduate program. I { am / feel } { confident / certain } { that / — } { she / Cherise } will { continue to / — } succeed in her studies. { She / Cherise } is a dedicated student and { thus far her grades / her grades thus far } { have been / are } { exemplary / excellent }. In class, { she / Cherise } { has proven to be / has been } a take-charge { person / individual } { who is / — } able to successfully develop plans and implement them.

{ She / Cherise } has also assisted { us / — } in our admissions office. { She / Cherise } has { successfully / — } demonstrated leadership ability by counseling new and prospective students. { Her / Cherise's } advice has been { a great / of considerable } help to these students, many of whom have { taken time to share / shared } their comments with me regarding her pleasant and { encouraging / reassuring } attitude. { For these reasons / It is for these reasons that } I { highly recommend / offer high recommendations for } Cherise { without reservation / unreservedly }. Her { ambition / drive } and { abilities / potential } will { truly / surely } be an { asset to / plus for } your { establishment / school }.

Figure 11.7 A Letter in 2^{38} Variations

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



Popular algorithm

- MD5
- SHA

Message Digest (MD)

- The MD family comprises of hash functions MD2, MD4, MD5 and MD6. It was adopted as Internet Standard RFC 1321. It is a 128-bit hash function.

- MD5 digests have been widely used in the software world to provide assurance about integrity of transferred file. For example, file servers often provide a pre-computed MD5 checksum for the files, so that a user can compare the checksum of the downloaded file to it.

- In 2004, collisions were found in MD5. An analytical attack was reported to be successful only in an hour by using computer cluster. This collision attack resulted in compromised MD5 and hence it is no longer recommended for use.

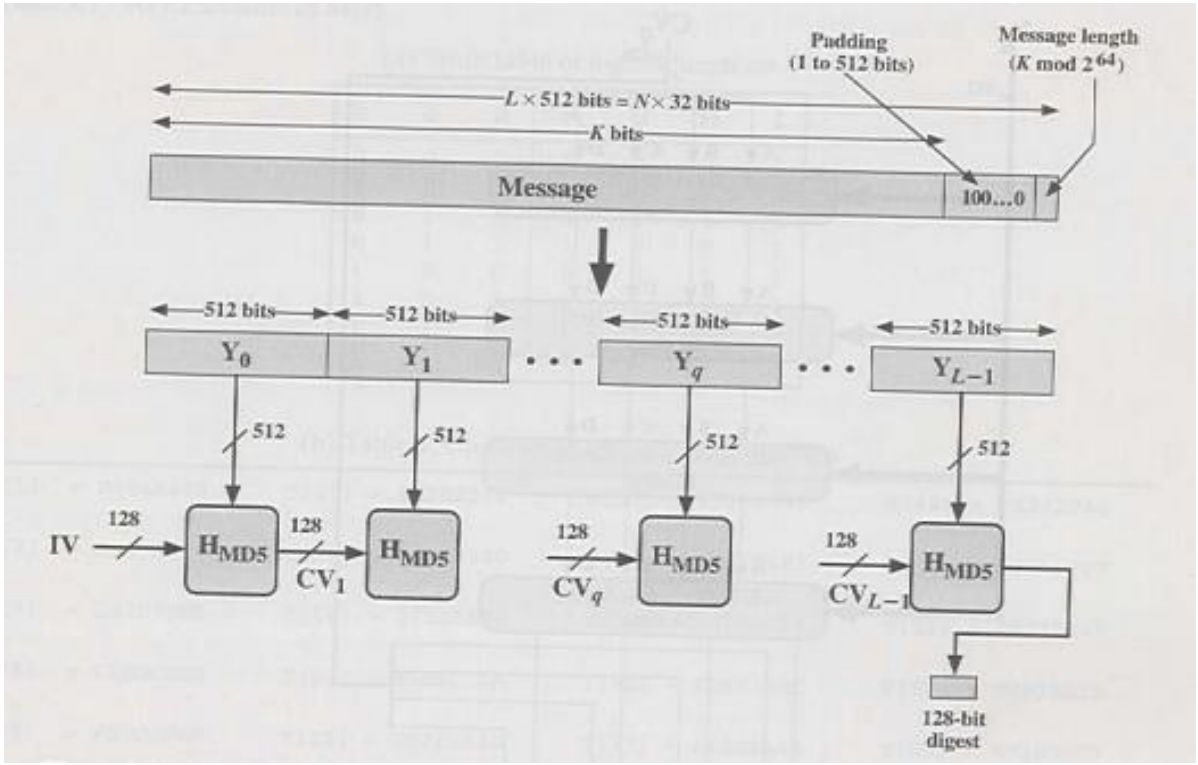
Message Digest (MD) Logic

- The algorithm takes as input a message of arbitrary length and produces as output a 128-bit message digest. The input is processed in 512-bit blocks. The processing consists of the following steps:
  - **Step 1: Appending padding bits**.
  - Step 2: Append length
  - Step 3: Initialize MD buffer
  - Step 4: Process message in 512-bit (16-word) blocks.
  - Step 5: Output

Step 1: Appending padding bits

- The massage is padded so that its length in bits is congruent to 448 modulo 512 (length ≡ 448 mod 512).
- Padding is always is added, even if the message is already of the desired length.
- For example, if the message is 448 bits long, it is padded by 512 bits to a length of 960 bits.
- Thus, the number of padding bits is in the range of 1 to 512.
- The padding consists of a single 1-bit followed by the necessary number of 0-bits.

Step 1: Appending padding bits



Step 2: Append length

- A 64-bit representation of the length in bits of the original message (before the padding) is appended to the result of step 1 (least significant byte first).
- If the original length is greater than 264, then only the low-order 64 bits of the length are used.
- Thus, field contains the length of the original message, modulo 264.
- The outcome of the first two steps yields a message that is an integer multiple of 512 bits in length.
- Equivalently, the result is a multiple of 16 32-bit words.
- Let M[ ] 0 1 …N − denote the words of the resulting message, with N an integer multiple of 16. Thus, N = L ×16

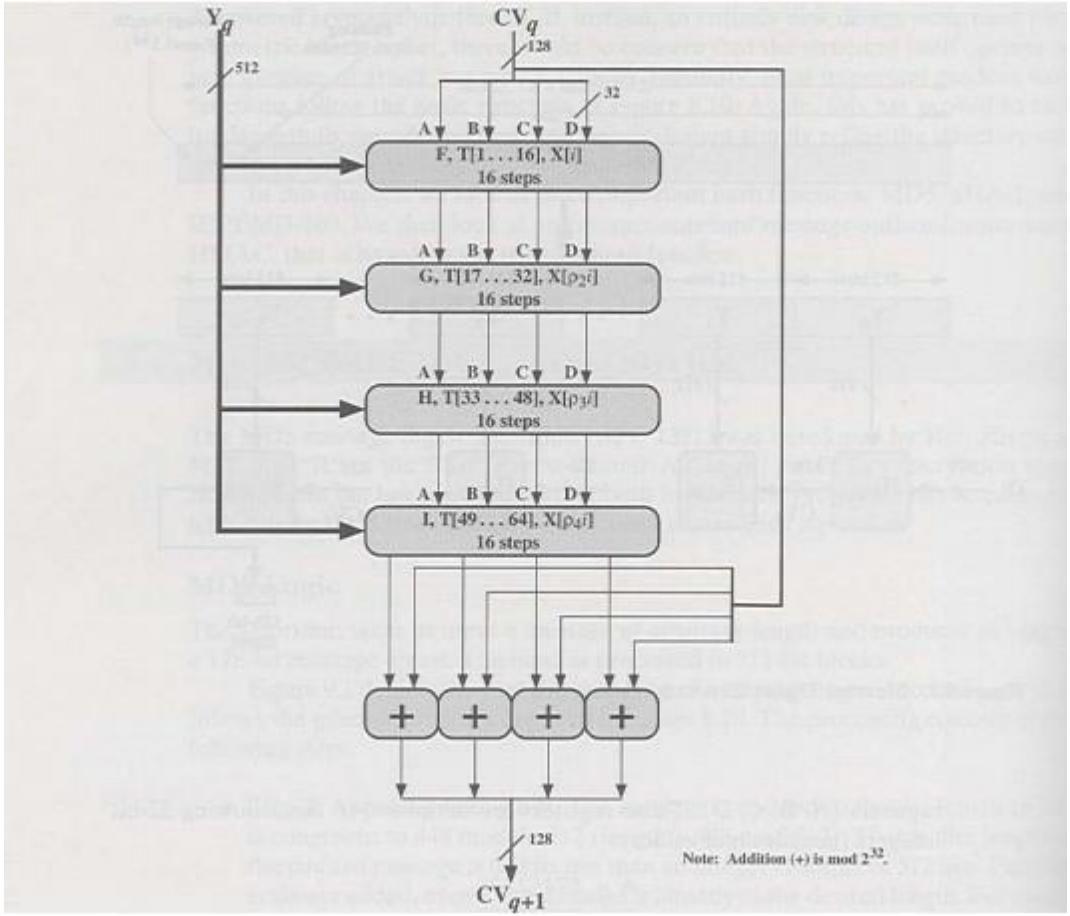
Step 3: Initialize MD buffer

- A 128-bit buffer is used to hold intermediate and final results of the hash function.
- The buffer can be represented as four 32-bit registers (A, B, C, D).
- These registers are initialized to the following 32-bit integers (hexadecimal values):
  - A = 67452301
  - B = EFCDAB89
  - C = 98BADCFE
  - D = 10325476

Step 3: Initialize MD buffer

- These values are stored in little-endian format, which is the least significant byte of a word in the low-address byte position.
- As 32-bit strings, the initialization values (in hexadecimal) appears as follows:
  - Word A: 01 23 45 67
  - Word B: 89 AB CD EF
  - Word C: FE DC BA 98
  - Word D: 76 54 32 10

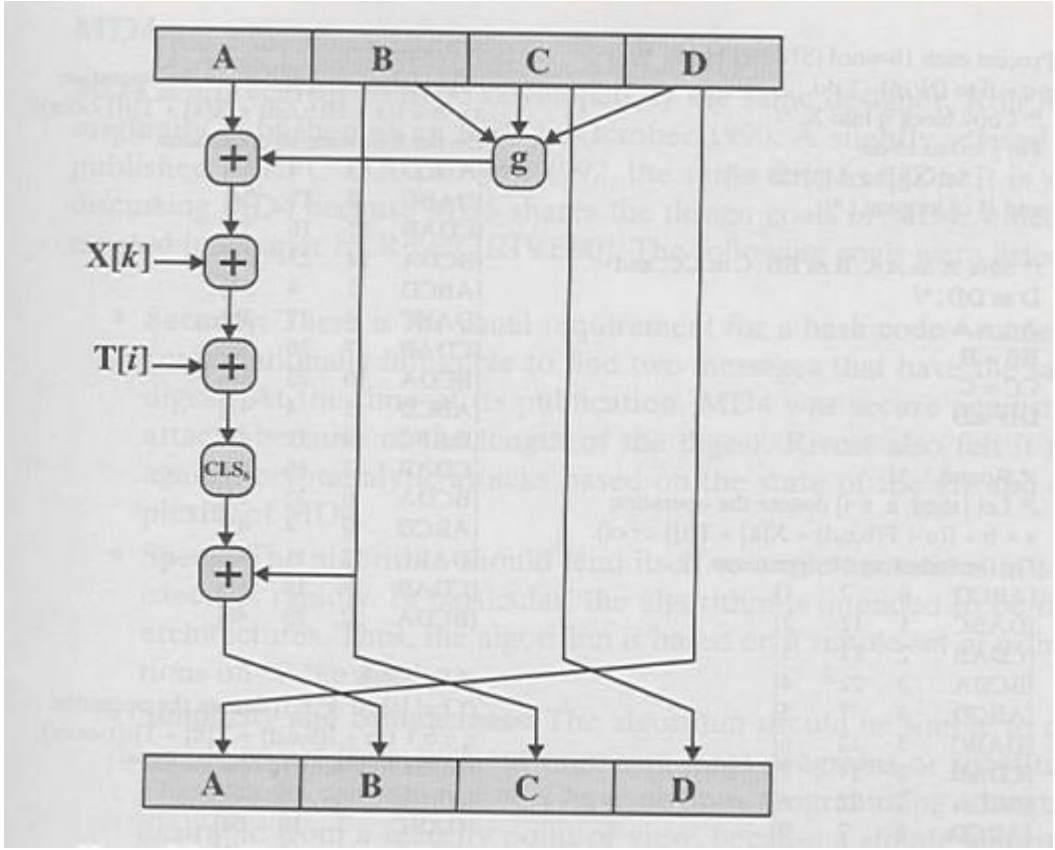
Step 3: Initialize MD buffer



Step 4: Process message in 512-bit (16-word) blocks

- The heart of the algorithm is a compression algorithm that consists of four “rounds” of processing; this module is labeled HMD5.
- The four rounds have the similar structure, but each uses a different primitive logical function, referred to as F, G, H, and I in the specification.
- Each round takes as input the current 512-bit block being processed (Yq) and the 28-bit buffer value ABCD and updates the contents of the buffer.
- Each round also makes use of one-fourth of a 64-element table T[1 64 …] , constructed from the sine function.
- The ith element of T, denoted T[i], has the value equal to the integer part of 232 × abs i (sin( )) , where I is in radians.

Step 4: Process message in 512-bit (16-word) blocks



Step 5: Output

- After all L 512-bit blocks have been processed, the output from the Lth stage is the 160-bit message digest.

$$CV_0 = IV$$
$$CV_{q+1} = SUM_{32}(CV_q, RF_I[Y_q, RF_H[Y_q, RF_G[Y_q, RF_F[Y_q, CV_q]]]])$$
$$MD = CV_L$$

where

IV - initial value of the ABCD buffer, defined in step 3

Y_q - the qth 512-bit block of the message

L - the number of blocks in the message (including padding and length fields)

CV_q - chaining variable processed with the qth block of the message

RF_x - round function using primitive logical function x

MD - final message digest value

SUM_{32} - addition modulo 2^{32} performed separately on each word of the pair of inputs

Presentation 22/04/2022

- Fardin Ahosan Shawon, Ratno Kumar: SHA algorithm
- Arnab Dey, Sourav Debnath:
  - SSL overview & protocols (SSL record protocol, Handshake protocol, Change-cipher spec protocol, Alert protocol),
  - SSL and the Man-in-the-Middle
  - SSL Connections

Presentation 24/04/2022

- Rayhan Billah, Rokonuzzaman
  - Details of IPSec (for subtopic check 10.3 of information security book)
- Redwan Hossain, Roichuddin Rana
  - Details of Kerberos & GSM (for subtopic check 10.4 & 10.5 of information security book)