

CS458: Introduction to Information Security

Notes 6: Cryptographic Hash Functions

Yousef M. Elmehdwi

Department of Computer Science

Illinois Institute of Technology

yelmehdwi@iit.edu

Sep 28th, 2022

Slides: Modified from “Cryptography and Network Security”, 6/e, by William Stallings,
[Christof Paar and Jan Pelzl](#) & [Steven Gordon at Thammasat University](#)

- Hash Functions and Cryptographic Hash Functions
- Requirements and Security of Cryptographic Hash Functions
- Popular Hash Functions (MD5 and SHA)

Hash Functions

- Hash Functions H

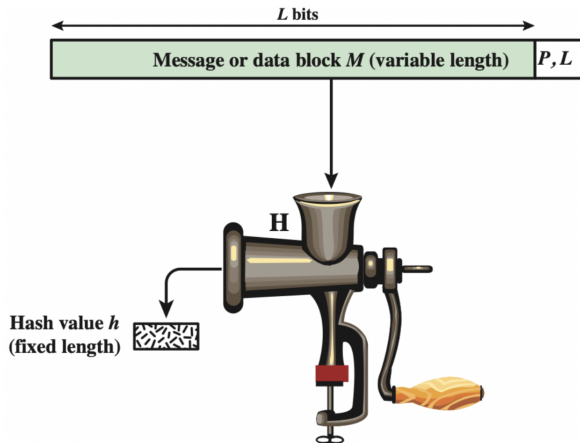
- **Input:** variable-length block of data M
- **Output:** fixed-size hash value $h = H(M)$
- Applying H to large set of inputs should produce evenly distributed and random looking outputs
- Values returned by a hash function are called message digest or simply hash values.

- Cryptographic hash function:

- An algorithm for which it is computationally infeasible¹ to find either:
 1. a data object that maps to a pre-specified hash result (the one-way property)
 - *i.e., it should be computationally hard to reverse a hash function*
 2. two data objects that map to the same hash result (the collision-free property)
 - *i.e., it should be hard to find two different inputs messages that produce the same hash value*
- Used to determine whether or not data has changed
 - Examples: message authentication, digital signatures, one-way password file, intrusion/virus detection, PRNG

¹ *infeasible means it would take too long to do that.*

Cryptographic Hash Function: $h = H(M)$



P, L = padding plus length field

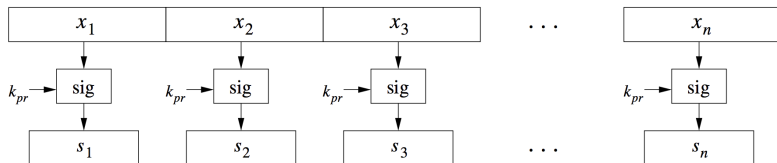
Hash function maps from input of some length L bits and produce a fixed small length output

Motivation: Signing Long Messages

- Suppose Bob signs x
- Bob sends x and $s = \text{sig}_{K_{pr,B}}(x)$ to Alice.
- Alice verifies that $\text{ver}_{K_{pub,B}}(x, s)$.
- Problem:
 - x is restricted in length, e.g., $|x| < 256$ Bytes (N is 2048 bits)
 - In practice the plaintext x will often be large.
- Q: How can we efficiently compute signatures of large messages?
 - Divide the message x into blocks x_i of size less than the allowed input size of the signature algorithm, and sign each block separately

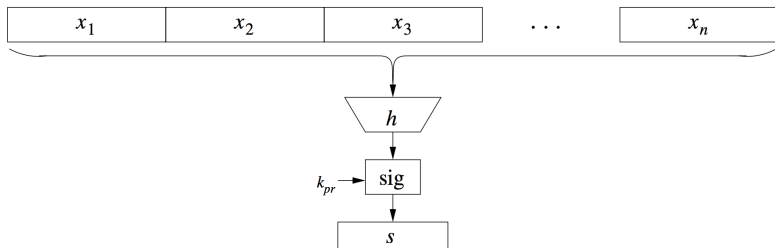
Motivation: Signing Long Messages: Problem

- Naïve signing of long messages generates a signature of same length.



- This is Bad. Three Problems
 - **Computational overhead**
 - DSs are based on computationally intensive asymmetric operations
 - **Message overhead**
 - Double the message overhead
 - **Security limitations**
 - Attacker can remove individuals messages/signatures, reorder messages/signatures, or reassemble new messages/signatures, etc
- **Solution:** Instead of signing the whole message, sign only a digest (hash). Also secure, but much faster
 - i.e., somehow “compress” the message x prior to signing
- **Needed:** Hash Functions

Signing of long messages with a hash function



Message Authentication

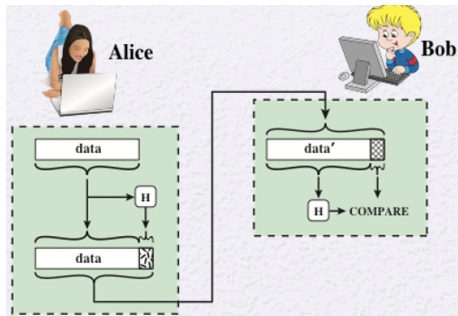
- Protects against active attacks
- Provide services to ensure the integrity of a message.
 - Ensure data received are exactly as sent (data Integrity)
 - i.e., contain no modification, insertion, deletion, or replay
 - Assure identity of the sender is valid (authentic)
- When a hash function is used to provide message authentication, the hash function value is often referred to as a **message digest**²

² *Message digest is a fixed size numeric representation of the contents of a message, computed by a hash function. A message digest can be encrypted forming a digital signature.*

Use Hash Function to Check Data Integrity

- The essence of the use of a hash function for message authentication is as follows.
 - ① **Alice** computes a hash value as a function of the bits in the message
 - ② **Alice** transmits both the hash value and the message.
 - ③ **Bob** performs the same hash calculation on the message bits
 - ④ **Bob** compares this value with the incoming hash value.
 - If there is a mismatch, **Bob** knows that the message (or possibly the hash value) has been altered

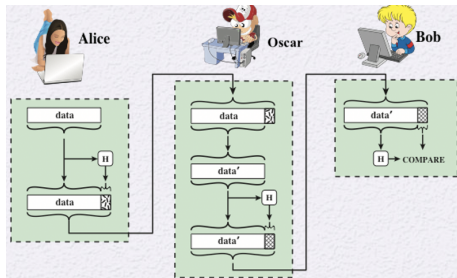
Use Hash Function to Check Data Integrity



Attack against Hash Function

- The hash function must be transmitted in a secure fashion.
- The hash function must be protected so that if an adversary alters or replaces the message, it is not feasible to also alter the hash value to fool the receiver
- **Man-in-Middle Attack**
 - ① Alice transmits a data block and attaches a hash value.
 - ② Oscar intercepts the message, alters or replaces the data block, and calculates and attaches a new hash value.
 - ③ Bob receives the altered data with the new hash value and does not detect the change.
- To prevent this attack, the hash value generated by Alice must be protected.

Attack against Hash Function

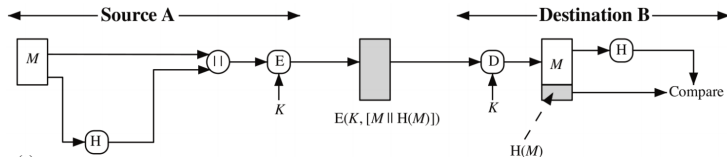


Message Authentication using Hash Function

- Hash function does not take a secret key as input.
- To authenticate a message, the message digest is sent with the message in such a way that the message digest is authentic.
- Next: illustrate different ways in which the message can be authenticated using a hash code

Message Authentication Example (a)

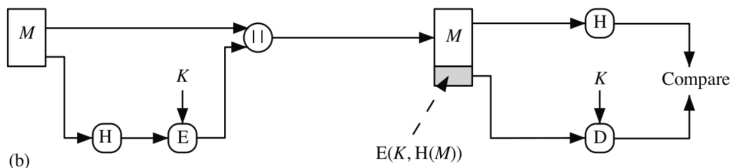
- Encrypt the message and hash code using symmetric encryption



- $||$ denotes concatenation.
- The hash code provides the structure or redundancy required to achieve authentication.
- Confidentiality is also provided
- Q) Do we have nonrepudiation here?

Message Authentication Example (b)

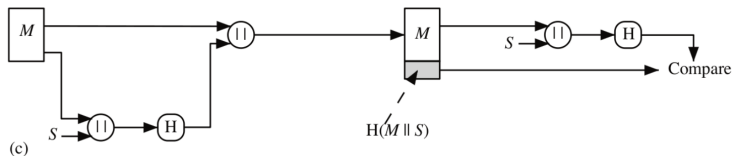
- Encrypt only hash code using symmetric encryption
- Reduces computation overhead when confidentiality not required



- *Desired property of the Hash Function: two different messages produce different hash values.*
- *Q) Do we need to have the hash function?*

Message Authentication Example (c)

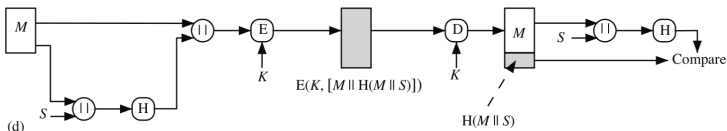
- It is possible to use a hash function but no encryption for message authentication
 - Shared secret S is hashed
 - No encryption needed



- Q) Can attacker find the shared secret S ?

Message Authentication Example (d)

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



Message Authentication Without Confidentiality

- It is possible to combine authentication and confidentiality in a single algorithm by encrypting a message plus its authentication tag
- Typically message authentication is provided as a separate function from message encryption
- Situations in which message authentication without confidentiality may be preferable include:
 - There are a number of applications in which the same message is broadcast to a number of destinations
 - An exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages
 - Authentication of a computer program in plaintext is an attractive service
- Thus, there is a place for both authentication and encryption in meeting security requirements

Authentication and Encryption

- Sometimes desirable to avoid encryption when performing authentication
 - Encryption in software can be slow
 - Encryption in hardware has financial costs
 - Encryption hardware can be inefficient for small amount of data
 - Encryption algorithms may be patented, need to pay license to use it, increasing costs to use
- More commonly, message authentication is achieved using a [Message Authentication Codes \(MAC\)](#)

Message Authentication Codes (MAC)

- Message Authentication Codes (or keyed hash function)
- Typically used between two parties that share a secret key to authenticate information exchanged between those parties
 - Take secret key K and a data block M as input; produce hash value (or MAC) which is associated with the protected message as output
 - If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the associated MAC value
 - An attacker who alters the message will be unable to alter the associated MAC value without knowledge of the secret key
 - Combining hash function and encryption produces same result as MAC; but MAC algorithms can be more efficient than encryption algorithms
 - MAC covered in next topic

Digital Signatures

- Another important application, which is similar to the message authentication application, is the **digital signature**
- Hash value of message encrypted with user's private key
- Anyone who knows the user's public key can verify integrity of message and author
- An attacker who wishes to alter the message would need to know the user's private key
- Implications of digital signatures go beyond just message authentication

Digital Signature Operations (Concept)

- **Signing**

- **Bob** signs a message by encrypting with own private key
 - $s = E_{k_{pr,B}}(x)$
- **Bob** attaches signature to message

- **Verification**

- **Alice** verifies a message by decrypting signature with signer's (Bob) public key
 - $x' = D_{k_{pub,B}}(s)$
- **Alice** then
 - compares received message x with decrypted x'
 - if identical, signature is verified

Digital Signature Operations (Practice)

No need to encrypt entire message; encrypt hash of message

- **Signing**

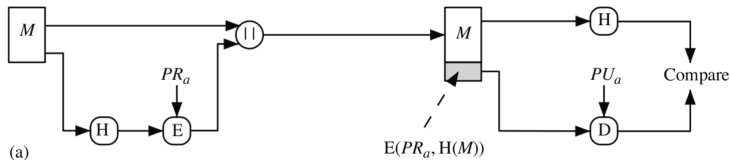
- Bob signs a message by encrypting **hash of message** with own private key
 - $s = E_{k_{pr,B}}(H(x))$
- Bob attaches signature to message

- **Verification**

- Alice verifies a message by decrypting signature with signer's (Bob) public key
 - $h' = D_{k_{pub,B}}(s)$
- Alice then
 - compares **hash** of received message, $H(x)$, with decrypted h' ;
 - if identical, signature is verified

Simplified Examples of Digital Signatures

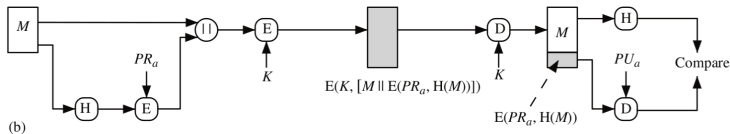
- The hash code is encrypted, using public-key encryption with the sender's private key.



- Q) Can the attacker have the hash value?

Simplified Examples of Digital Signatures

- If confidentiality as well as a digital signature is desired, then the message plus the private-key-encrypted hash code can be encrypted using a symmetric secret key.
- This is a common technique.



Other Hash Function Uses

- Commonly used to create a **one-way password file**
 - When a user enters a password, the hash of that password is compared to the stored hash value for verification
 - This approach to password protection is used by most operating systems
- Can be used for **intrusion and virus detection**
 - Store $H(F)$ for each file on a system and secure the hash values
 - One can later determine if a file has been modified by recomputing $H(F)$
 - An intruder would need to change F without changing $H(F)$

Simple Hash Functions

- All hash functions operate using the following general principles.
 - The input (message, file, etc.) is viewed as a sequence of *n-bit* blocks.
 - The input is processed one block at a time in an iterative fashion to produce an *n-bit* hash function.
- One of the simplest hash functions is the **bit-by-bit exclusive-OR (XOR)** of every block.

Bit-by-Bit Exclusive OR

- $C_i = b_{i1} \oplus b_{i2} \oplus \dots \oplus b_{im}$
 - C_i is i^{th} bit of hash code, $1 \leq i \leq n$
 - m is number of n -bit blocks in input
 - b_{ij} is i^{th} bit in j^{th} block

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

Requirements and Security

- Preimage

- For a hash value $z = H(x)$, x is the **pre-image** of z
 - That is, x is a data block whose hash function, using the function H , is z
- Because H is a many-to-one mapping, for any given hash value z , there will in general be multiple pre-images

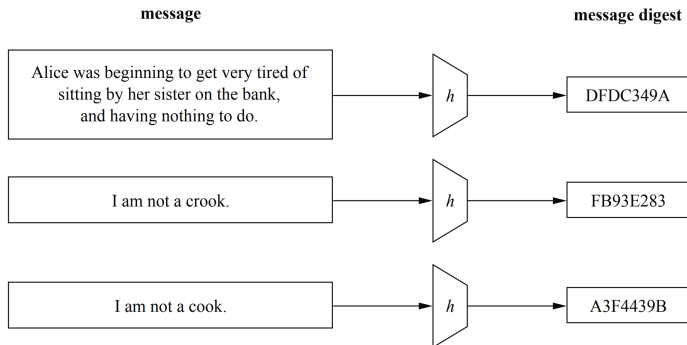
- Collision

- Occurs if we have $x_1 \neq x_2$ and $H(x_1) = H(x_2)$
- Because we are using hash functions for data integrity, collisions are clearly undesirable
- How many pre-images for given hash value?
 - If H takes b -bit input block, 2^b possible messages
 - For n -bit hash code, where $b > n$, 2^n possible hash codes
 - On average, if uniformly distributed hash values, then each hash value has 2^{b-n} pre-images

Requirements of Cryptographic Hash Function

- Crypto hash function H must provide:
 1. **Variable input size**: H can be applied to input block of any size
 2. **Fixed output size**: H produces fixed length output
 3. **Efficiency of operation**
 - $H(x)$ is relatively easy/fast to compute for any given x
 - Computationally hash functions are much faster than a symmetric encryption.
 4. **Pre-image resistance** (*One-way property*)
 - For a given hash value z , it is computationally infeasible to find any input x such that $H(x) = z$, i.e., $H(x)$ is one-way
 5. **Second pre-image resistance** (*Weak collision resistance*)
 - For any given block x_1 , and thus $H(x_1)$, it is computationally infeasible to find any $x_2 \neq x_1$ with $H(x_1) = H(x_2)$.
 6. **Collision resistance** (*Strong collision resistance*)
 - It is computationally infeasible to find any pairs $x_1 \neq x_2$, s.t., $H(x_1) = H(x_2)$.
- Actually, lots of collisions exist, but hard to find any

Principal input-output behavior of hash functions

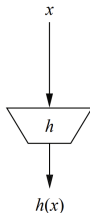


- Able to apply a hash function to messages x of any size
- Output of a hash function must be of fixed length, independent of the input size
- Computed hash value should be highly sensitive to all input bits.
⇒ minor modifications to the input x , hash value should look very different

1st Security properties of hash functions

1. Preimage resistance (*One-way property*)

- For a given output z , it is impossible to find any input x such that $h(x)=z$, i.e., $h(x)$ is one-way
 - *This property protects against an attacker who only has a hash value and is trying to find the input.*
- Bob sends $(E_K(x), sig_{K_{pr,B}}(z))$
 - Encrypts with AES and signs with RSA: $s = sig_{K_{pr,B}}(y) \equiv z^d \mod n$
- Oscar uses Bob's public key to calculate $s^e \equiv z \mod n$
- If $h(x)$ is not one-way then $x = h^{-1}(z)$
 - Thus, the symmetric encryption of x is circumvented by the signature, which leaks the plaintext. $\Rightarrow h(x)$ should be a one-way function

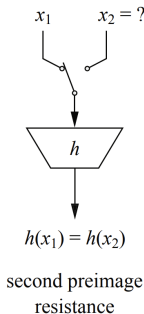


preimage resistance

2nd Security properties of hash functions

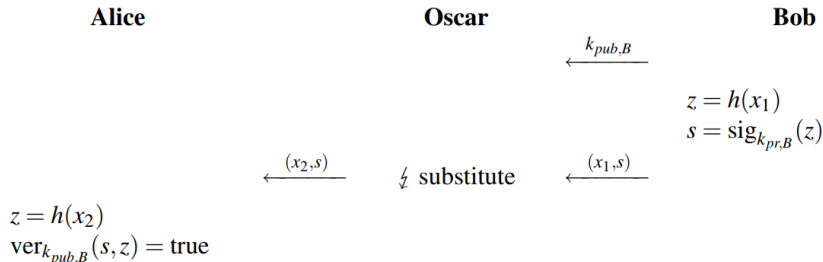
2. Second preimage resistance (*Weak collision resistance*)

- Given x_1 , and thus $h(x_1)$, it is computationally infeasible to find any other input value x_2 s.t., $h(x_1) = h(x_2)$.
- This property protects against an attacker who has an input value and its hash, and wants to substitute a different value as legitimate value in place of the original input value.*



2nd Preimage Attack

- Assume Bob hashes and signs a message x_1 .
- If Oscar is capable of finding a second message x_2 such that $h(x_1) = h(x_2)$, he can run the following **substitution attack**

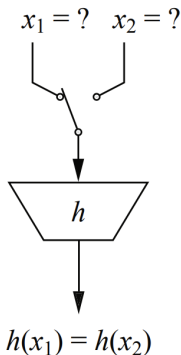


- There is always x_2 such that $h(x_1) = h(x_2)$ but it should be difficult to find
- “weak” collision, requires exhaustive search

3rd Security properties of hash functions

3. Collision resistance (*Strong collision resistance*)

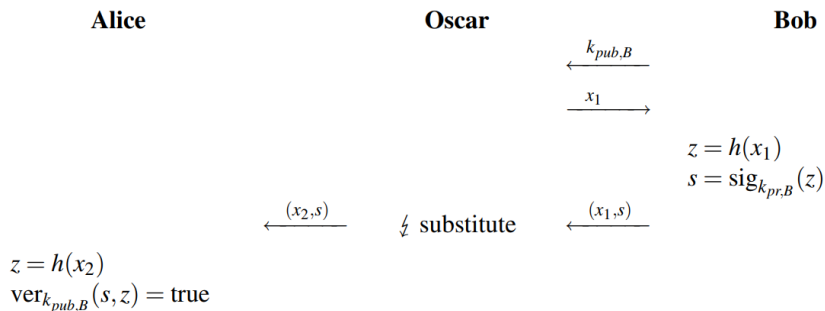
- It is computationally infeasible to find any pairs $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$.
- *This property makes it very difficult for an attacker to find two input values with the same hash.*



collision resistance

Collision Attack

- Oscar starts with two messages:
 $x_1 = \text{Transfer \$10 into Oscar's account}$
 $x_2 = \text{Transfer \$10,000 into Oscar's account}$
- He alters x_1 and x_2 at “non-visible” locations and continues until the condition $h(x_1) = h(x_2)$ is fulfilled.
- With the two messages, he can launch the following attack



Required Hash Properties for Different Applications

- **Weak hash function:** Satisfies 5 requirements listed in *slide 30* (but not collision resistant)
- **Strong hash function:** Also collision resistant
- Hash Function Resistance Properties Required for Various Data Integrity Applications

| | Preimage Resistant | Second Preimage Resistant | Collision Resistant |
|---|--------------------|---------------------------|---------------------|
| Hash + digital signature | yes | yes | yes* |
| Intrusion detection and virus detection | | yes | |
| Hash + symmetric encryption | | | |
| One-way password file | yes | | |
| MAC | yes | yes | yes* |

* Resistance required if attacker is able to mount a chosen message attack

Security of Hash Functions

- There are two approaches to attacking a secure hash function:
 1. **Cryptanalysis**
 - An attack based on weaknesses in a particular cryptographic algorithm
 - Seek to exploit some property of the algorithm to perform some attack other than an exhaustive search
 2. **Brute-force attack**
 - Does not depend on the specific algorithm, only depends on bit length
 - In the case of a hash function, attack depends only on the bit length of the hash value
 - Method is to pick values at random and try each one until a collision occurs
 - *Strength of hash function depends solely on the length of the hash code produced by the algorithm.*

Brute Attacks on Hash Functions

- Pre-image and Second Pre-image Attack

- Find a x that gives specific z ; try all possible values of x
- With $n\text{-bit}$ hash code, effort required proportional to 2^n

- Collision Resistant Brute Attack

- Find any two messages that have same hash values
- Effort required is proportional to $2^{n/2}$
- Due to [birthday paradox](#), easier than pre-image attacks

- Brute force attack can be applied against any hash algorithm.

- Practical Effort (other attacks take advantage of the algorithm design)

- Cryptanalysis attacks possible in theory; complex
- Collision resistance desirable for general hash algorithms
- MD5 uses 128-bits: collision attacks possible (2^{60}) (not much better than brute force attack)

Collision Attacks and the Birthday Paradox

- It turns out that collision resistance causes most problems and more difficult
- Collision attacks are much harder to prevent than 2^{nd} preimage attack.
- Q: Can we have hash function without collisions?
 - Since $|X| \gg |Z| \Rightarrow$ collision must exist. (“Pigeonhole Principle”)
 \Rightarrow We must make collision very hard to find!
- Q: How difficult it is to find collisions?

2nd preimage attack with brute-force

- Q: How difficult it is to find collisions?
- Probably this is as difficult as finding second preimages
- If $|Z| = 2^{80}$, where $|h(x)| = n = 80$
 \Rightarrow attack requires $\approx 2^{80}$ steps until to find a collision

Birthday Attacks

- For a collision resistant attack, an adversary wishes to find two messages or data blocks that yield the same hash function
 - The effort required is explained by a mathematical result referred to as the birthday paradox
- The **Birthday Attack** exploits the **birthday paradox**
 - The chance that in a group of people two will share the same birthday
 - Only 23 people are needed for a **Probability** > 0.5 of this.
 - Can generalize the problem to one wanting a matching pair from any two sets, and show need $2^{\frac{n}{2}}$ in each to get a matching *n-bit* hash.

Collision Attack

- It turns out that collision resistance causes most problems and more difficult to achieve
 - How hard is it to find a collision with a probability of 0.5?
 - Related Problem: How many people are needed such that two of them have the same birthday with a probability of 0.5?

$$P(\text{no collision among 2 people}) = 1 - \frac{1}{365}$$

$$P(\text{no collision among 3 people}) = (1 - \frac{1}{365})(1 - \frac{2}{365})$$

...

$$P(\text{no collision among } t \text{ people}) = \prod_{i=1}^{t-1} (1 - \frac{i}{365})$$

- for $t = 23$, $\prod_{i=1}^{22} (1 - \frac{i}{365}) = 0.507 \approx 50\%$
- No! Not $\frac{365}{2} = 183$. 23 are enough! This is called the **birthday paradox** (Search takes $\approx \sqrt{2^n}$ steps for 50% probability collision)
- To deal with this paradox, hash functions need a output size of at least 224 bits

How Long Should Hash Be

- Many input messages yield the same hash
 - e.g., 1024-bit message, 128-bit hash
 - On average, 2^{896} messages map into one hash
- With $n=64$, it takes 2^{32} trials to find a collision (doable in very little time)
- Today, need at least $n=160$, requiring about 2^{80} trials

- MD5: message-digest algorithm 5

- MD5 was most popular and widely used hash function for quite some years.
- The MD family comprises of hash functions MD2, MD4, MD5 and MD6.
- Developed by Ron Rivest in 1991
- Generates 128-bit hash
- Was commonly used by applications, passwords, file integrity; **no longer recommended**
 - e.g., 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.
- Collision and other attacks possible; tools publicly available to attack MD5, so it's broken

Secure Hash Algorithm (SHA)

- Family of [SHA](#) comprise of four SHA algorithms; [SHA-0](#), [SHA-1](#), [SHA-2](#), and [SHA-3](#).
- [SHA-0](#):
 - Produces 160-bit hash values.
 - It had few weaknesses and did not become very popular.
 - Was revised in 1995 as [SHA-1](#).
- [SHA-1](#):
 - Most widely used of the existing [SHA](#) hash functions.
 - Employed in several widely used applications and protocols.
 - No longer considered secure against well-funded opponents³
- [SHA-2](#):
 - In 2002, NIST produced a revised version of the standard that defined three new versions of [SHA](#) with hash value lengths of 224, 256, 384, and 512.
 - Collectively known as [SHA-2](#)
 - No successful attacks have yet been reported.

³ [Announcing the first SHA1 collision](#)

- SHA-2 shares same structure and mathematical operations as its predecessors and causes concern
- Due to time required to replace SHA-2 should it become vulnerable, NIST announced in 2007 a competition to produce SHA-3
- **SHA-3 Requirements:**
 - Must support hash value lengths of 224, 256, 384, and 512 bits
 - Algorithm must process small blocks at a time instead of requiring the entire message to be buffered in memory before processing it
- SHA-3 standard was released by NIST as FIPS 202 (SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions)⁴ on August 5, 2015

⁴SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions

Comparison of SHA Parameters

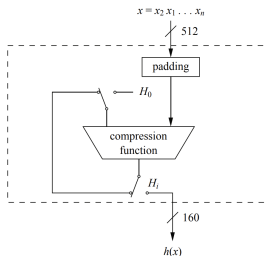
| | SHA-1 | SHA-224 | SHA-256 | SHA-384 | SHA-512 | SHA-512/224 | SHA-512/256 |
|---------------------|------------|------------|------------|-------------|-------------|-------------|-------------|
| Message size | $< 2^{64}$ | $< 2^{64}$ | $< 2^{64}$ | $< 2^{128}$ | $< 2^{128}$ | $< 2^{128}$ | $< 2^{128}$ |
| Word size | 32 | 32 | 32 | 64 | 64 | 64 | 64 |
| Block size | 512 | 512 | 512 | 1024 | 1024 | 1024 | 1024 |
| Message digest size | 160 | 224 | 256 | 384 | 512 | 224 | 256 |
| Number of steps | 80 | 64 | 64 | 80 | 80 | 80 | 80 |
| Security | 80 | 112 | 128 | 192 | 256 | 112 | 128 |

- Notes:

1. All sizes are measured in bits.
2. Security refers to the fact that a birthday attack on a message digest of size n produces a collision with a work factor of approximately $2^{n/2}$.

SHA-1 High Level Diagram

- Compression function consists of *80* rounds which are divided into four stages of *20* rounds each

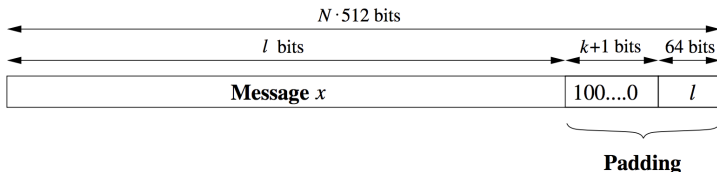


Merkle-Damgård Construction for Hash Functions

- Message is divided into fixed-size blocks and padded.
 - The input is viewed as a sequence of *512-bit* blocks
- The initial value H_0 is set to a predefined constant.
- The input is processed one block at a time in an iterative fashion to produce an *160-bit* hash function.

Example: SHA-1: Padding

- Message x has to be padded to fit a size of a multiple of 512 bit.
 - Let x with a length of l bit
 - To obtain an overall message size of a multiple of 512 bits
 - Append a single 1 followed by k zero bits and the binary 64-bit representation of l .
 - Consequently, the number of required zeros k is given by
$$k \equiv 512 - 64 - 1 - l = 448 - (l + 1) \bmod 512$$



SHA-1: Padding: Example

- Given is the message *abc* consisting of three 8-bit ASCII characters with a total length of $l = 24$ bits:

$$\underbrace{01100001}_a \quad \underbrace{01100010}_b \quad \underbrace{01100011}_c.$$

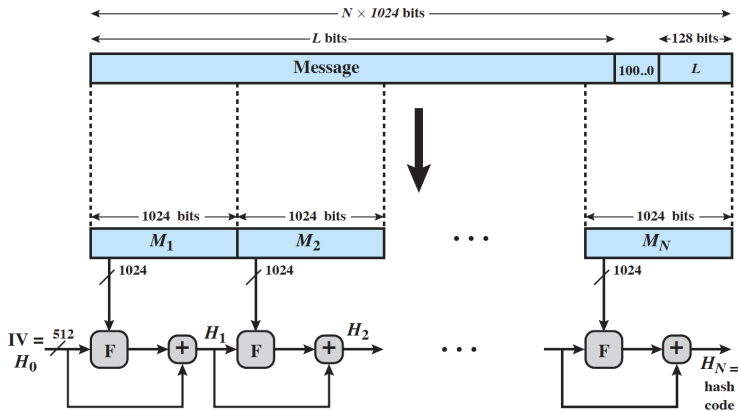
We append a “1” followed by $k = 423$ zero bits, where k is determined by

$$k \equiv 448 - (l + 1) = 448 - 25 = 423 \bmod 512.$$

Finally, we append the 64-bit value which contains the binary representation of the length $l = 24_{10} = 11000_2$. The padded message is then given by

$$\underbrace{01100001}_a \quad \underbrace{01100010}_b \quad \underbrace{01100011}_c \quad 1 \quad \underbrace{00\dots0}_{423 \text{ zeros}} \quad \underbrace{00\dots011000}_{l=24}.$$

Message Digest Generation Using SHA-512



$+$ = word-by-word addition mod 2^{64}

Message Digest Generation Using SHA-512

- **Step 1: Append padding bits:** message length is congruent to $896 \bmod 1024$
- **Step 2: Append length:** as a block of 128 bits being an unsigned 128 -bit integer length of the original message (before padding).
- **Step 3: Initialize hash buffer:** 512 -bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as eight 64 -bit registers
- **Step 4: Process the message in 1024 -bit (128 -word) blocks:** The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F in Figure in next slide.
- **Step 5: Output:** After all N 1024 -bit blocks have been processed, the output from the N^{th} stage is the 512 -bit message digest.

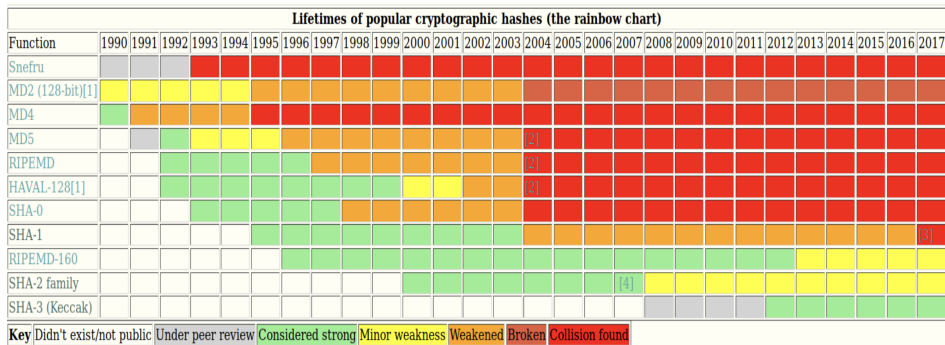
Choosing the length of Hash outputs

- The Weakest Link Principle:
 - A system is only as secure as its weakest link.
- Hence all links in a system should have similar levels of security.
- Because of the birthday attack, the length of hash outputs in general should double the key length of block ciphers
 - SHA-224 matches the 112-bit strength of triple-DES (encryption 3 times using DES)
 - SHA-256, SHA-384, SHA-512 match the new key lengths (128,192,256) in AES

Summary

- Hash functions are keyless.
- Applications of cryptographic hash functions
 - Message authentication
 - Digital signatures
 - Other applications
- Requirements and security
 - Security requirements for cryptographic hash functions
 - Brute-force attacks
 - Cryptanalysis
- Secure hash algorithm (SHA)

Hash function life cycles¹



- Historically, popular cryptographic hash functions have a useful lifetime of around 10 years

¹ Jan-aeke Larsson, Linköping University

- Computer Security: Principles and Practice
 - Chapters 2 and 21
- Understanding Cryptography: A Textbook for Students and Practitioners - *available online*
 - Chapter 11: Hash Functions