UniteImaSapienza

Università degli Studi di Roma

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

# HASH

F. Parisi Presicce

# Network Attacks

1. Disclosure
2. Traffic analysis
3. Masquerade
4. Content modification
5. Sequence modification
6. Timing modification
7. *Source repudiation*
8. *Destination repudiation*

# Integrity, not secrecy

- First two concern the protection of message content (secrecy) dealt with by encrypting the message.
- Points 3-6 refer to modifications: how to prevent message modification without noticing (integrity), and to confirm the identity of the sender.
- Generically this is the problem of message authentication
  - in some applications (e.g.; e-commerce) it is arguably more important than secrecy, to which it is orthogonal.
- 7 is solved with digital signatures
- 8 with combination of signatures and protocol design

# Message Authentication

- message authentication is concerned with:
  - verifying the integrity of a message
  - validating the identity of the originator
  - non-repudiation of origin (dispute resolution)
- three alternative approaches used:
  1. message encryption (as for secrecy)
  2. hash function
  3. keyed hash functions (also called MAC)

# 1. Message Encryption

- message encryption, used  mainly for confidentiality, also provides a measure of authentication
- if symmetric encryption is used then:
  - receiver knows sender must have created it since only sender and receiver know the key used
  - content cannot have been altered by party not knowing key
  - if message has suitable structure, redundancy or a checksum used to detect any changes
  - NO non-repudiation

# 1. Message Encryption (cont.)

- if public-key encryption is used:
  - encryption provides **no** confidence in the sender since anyone potentially knows public key (public !)
  - however if
    - sender **signs** (encrypts) message using his/her private-key
    - then encrypts result with recipient's public key
    - have both secrecy (by public) and authentication (by private)
  - again need to recognize corrupted messages
  - but at cost of two public-key uses on message

# 2. Cryptographic Checksums

- Also known as **hash functions**
- Mathematical function to generate a set of *k* bits from a set of *n* (>=k) bits
  - (in general, from arbitrary length to fixed length, hence necessarily **non-injective**).
- Example: ASCII parity bit
  - ASCII has 7 bits; 8th bit is “parity”
  - Even parity: even number of 1 bits
  - Odd parity: odd number of 1 bits

# Checksums

- The result of applying a hash function is called *hash value, message digest,* or *checksum*.
- The last term creates frequent confusion because in communications, checksums often refer to error correcting codes, typically a cyclic redundancy check (CRC).
- Checksums used by anti-virus products, on the other hand, are not computed with a CRC but with a cryptographic hash function.

# Definition

Cryptographic checksum $h: A \rightarrow B$ :

1. For any $x \in A$, $h(x)$ is easy to compute
2. For any $y \in B$, it is computationally infeasible to find $x \in A$ such that $h(x) = y$ *[one-way]*
3. It is computationally infeasible to find two inputs $x, x' \in A$ such that $x \neq x'$ and $h(x) = h(x')$
   - Alternate form (stronger): Given any $x \in A$, it is computationally infeasible to find a different $x' \in A$ such that $h(x) = h(x')$.

# Example: Integrity Protection

- To protect a program ***p***, compute its hash ***h(p)*** in a clean environment and store it in a place where it cannot be modified, e.g. on CD-ROM.
- Protection of the hash value is important; computing the hash value requires no secret information, so anybody can create a valid hash for a given file.
- To check whether the program has been modified, re-compute the hash value and compare it with the value stored.

# Collisions

- ‘The Integrity protection example described needs more than the one-way property of $h$.
- not concerned about an attacker reconstructing the program from the hash, but concerned about attackers who can change a program $p$ to $p'$ so that $h(p') = h(p)$.
- Then, our integrity protection mechanism would fail to detect the change.
- there is a collision when two inputs $x$ and $x'$ map to the same hash value.

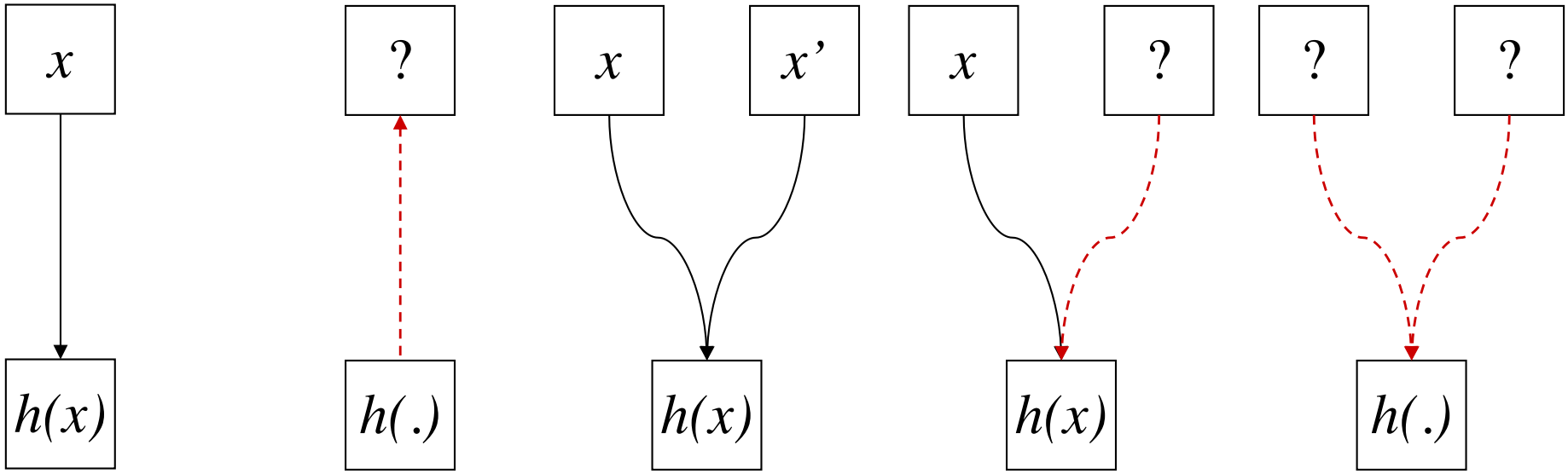
# Collisions

- If $x \neq x'$ and $h(x) = h(x')$, $x$ and $x'$ are a *collision*
  - ***Pigeonhole principle***: if there are $n$ containers for $n$+1 objects, then at least one container will have 2 objects in it.
  - Application: if there are 25 files and 8 possible cryptographic checksum values, at least one value corresponds to at least 4 files

# Collision Resistance

- Integrity protection requires collision-resistant hash functions
- distinguish between:
  - **2nd pre-image resistance** (weak collision resistance): given a value $h(x)$, it is computationally infeasible to find another input $x'$, $x \neq x''$, with $h(x) = h(x')$.
  - **Collision resistance** (strong collision resistance): it is computationally infeasible to find any two inputs $x$ and $x'$, $x \neq x'$, with $h(x) = h(x')$.

# Properties of One-way Functions




ease of computation

pre-image resistance

collision

2nd pre-image resistance

collision resistance

# Birthday Paradox

- How difficult is it to find collisions?
  - It depends on the bit-length of the hash
- Given an *n*-bit hash *y*, the expected number of tries before an *x* with $h(x) = y$ is found is $2^{n-1}$.
- Given *n*-bit hash values, a set of $2^{n/2}$ inputs is likely to contain a pair causing a collision.
- Birthday paradox:
  - put *m* balls numbered 1 to *m* into an urn;
  - draw a ball, list its number, put it back;
  - repeat;
  - for $m \rightarrow \infty$, the expected number of draws before a previously drawn number appears is $\mathit{sqrt}(\pi m/2)$.

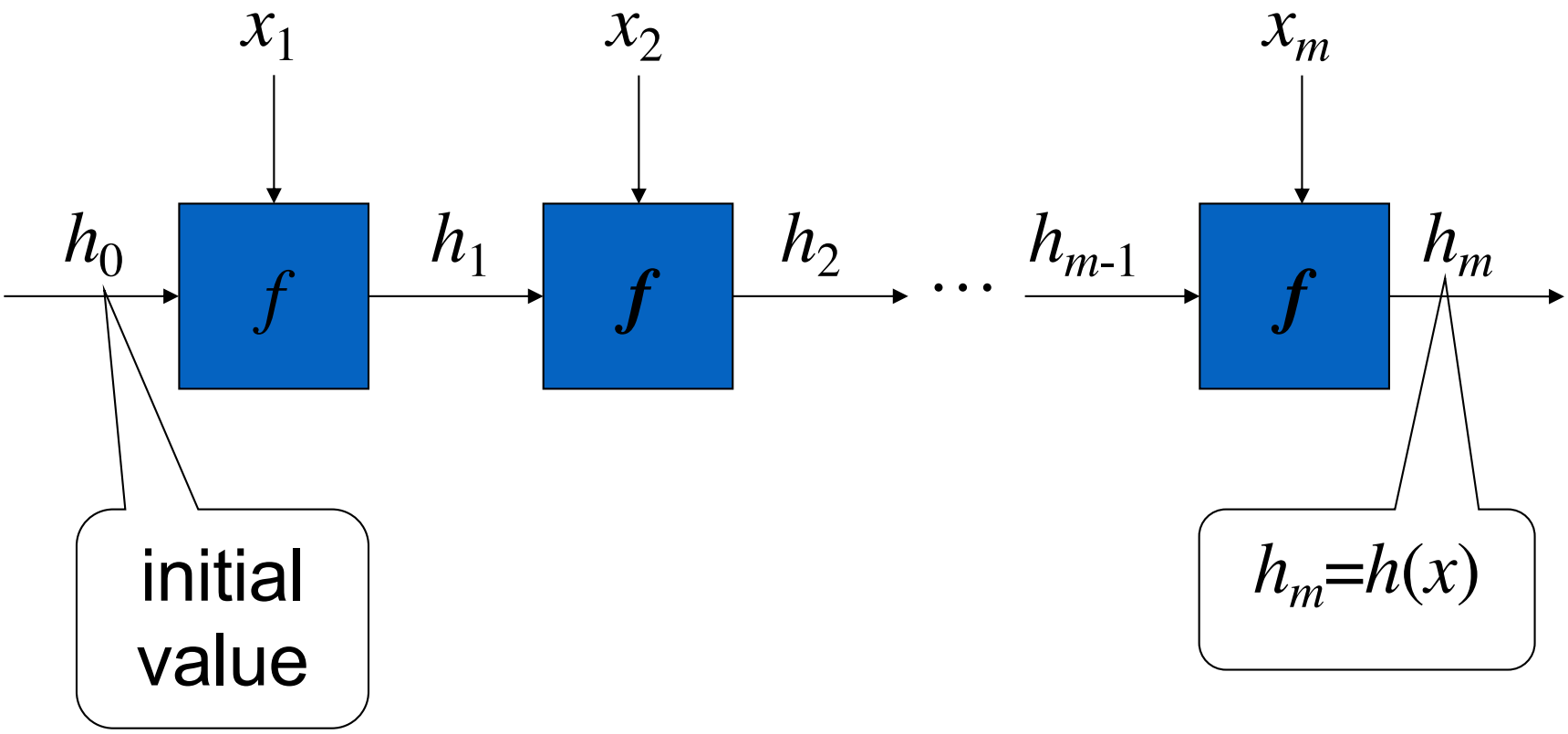
# Chances of Success

- Given a hash function which generates 64-bit digest ($n = 2^{64}$), randomly distributed and diffused
- Chance that a randomly chosen message maps to a given hash value is 1 in $n$ or $2^{-64}$ : seems secure
- but by **birthday attack** it is not: (digest of size m)
  - opponent generates $2^{m/2}$ variations of a valid message, all with essentially the same meaning
  - opponent also generates $2^{m/2}$ variations of a desired fraudulent message
  - 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

# Block Ciphers as Hash Functions

- can use block ciphers as hash functions
  - using $h_0$=0 and zero-pad of final block
  - compute: $h_i = E_{x_i} [h_{i-1}]$
  - and use final block as the hash value
  - similar to DES-CBC but without a key
- resulting hash is too small (64-bit)
  - due to direct birthday attack
- other variants also susceptible to attack

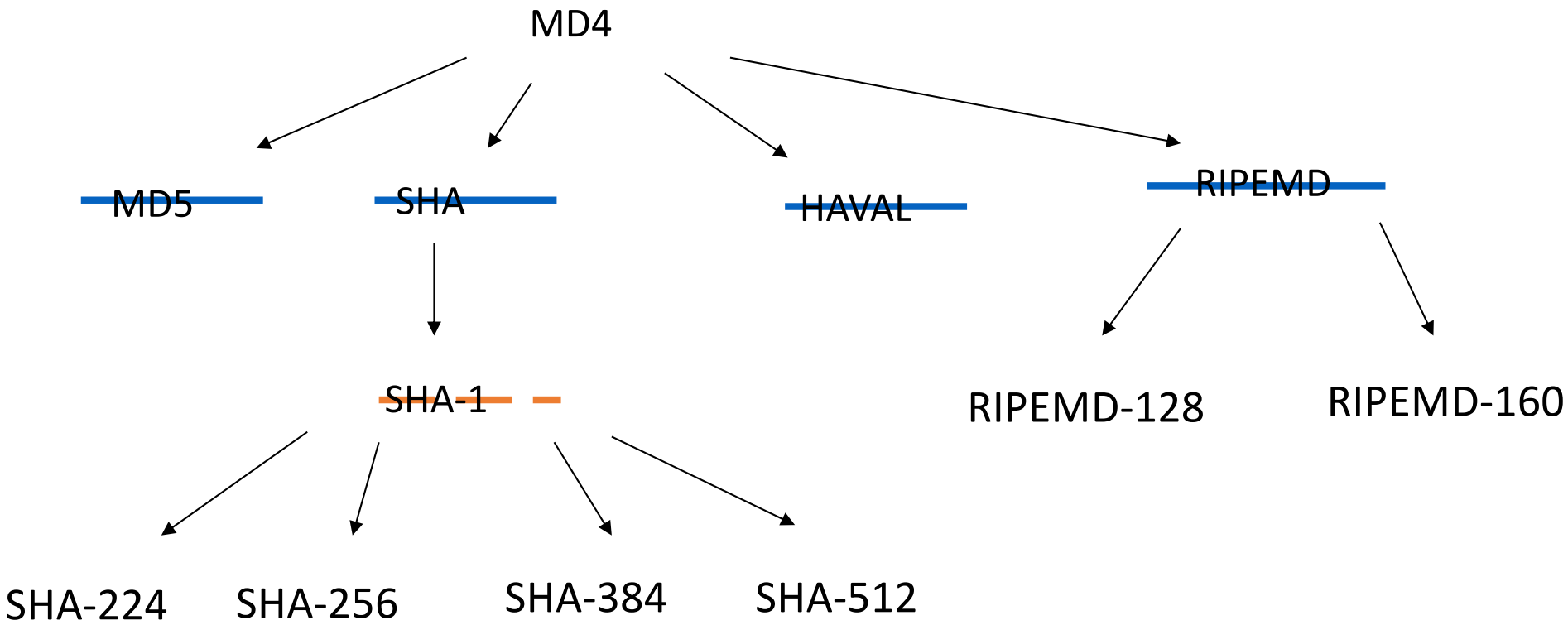
# Construction



# Frequently Used Hash Functions

- **MD4:** weak, it is computationally feasible to find meaningful collisions.
- **MD5:** (early 90s) standard choice in Internet protocols, broken and no longer recommended.
  - 128 bit message digest
  - 64 bits birthday attack
- **RIPEMD-160:** (late 90s) hash function frequently used by European cryptographic service providers.

# MD4 family of hash functions



# Non-keyed Message Digest Algorithms

- **SHA-1 (Secure Hash Algorithm)** by NIST
  - 160 bit message digest
  - used for US  Digital Signature Standard (DSA);
  - 80 bits birthday attack (65K longer time)(broken Feb 2005, with collision in 2**69 instead of 2**80)
  - After 2010 usable only for HMACs, KDFs and RNGs
- **SHA-2 Family** by NIST (2006) , block cipher based, not used much
- **SHA-3** winner of competition in 2012,  NIST standard in 2015, permutation based
- **BLAKE (-2)** another finalist in 2012, 256 and 512 output
- **SHA-256 -384** or **-512** when longer hash values are advisable.

# 3. MAC and its Properties

- a MAC is the value of a *keyed* cryptographic checksum
  - condenses a variable-length message M to a fixed-sized “authenticator” *using a secret key K*
- Since a cryptographic checksum is a many-to-one function, potentially many messages may have the same MAC but finding these needs to be very difficult
- needs satisfy the following:
  - knowing a message and MAC, it is unfeasible to find another message with same MAC
  - MAC values should be uniformly distributed
  - MAC should depend equally on all bits of the message

# Definition: authentication

- Authentication algorithm - A
- Verification algorithm - V  (“accept” / ”reject”)
- Authentication key - k
- Message space - usually binary strings
- Messages between Alice and Bob are pairs (m, $A_k(m)$) consisting of a message m (to be authenticated) and an authentication tag $A_k(m)$ for m

# Definition: authentication (cont.)

- Requirement - **$V_k(m, A_k(m)) =$ "accept"**
  - The verification consists of applying the authentication algorithm to m and comparing the result to $A_k(m)$
  - The authentication algorithm is sometimes called MAC (Message Authentication Code)
  - $A_k(m)$ is often denoted by $MAC_k(m)$

- In the context of public key, the function A uses a private key and the function V the corresponding public key $V_{kpub}(m, A_{kpriv}(m)) =$ "accept"

# Symmetric Ciphers for MACs

- can use any block cipher in chaining mode and use the final block as a MAC
- **Data Authentication Algorithm (DAA)** was a widely used MAC based on DES-CBC
  - using IV=0 and zero-pad of final block
  - encrypt message using DES in CBC mode
  - and send just the final block as the MAC
    - or the leftmost M bits (16≤M≤64) of final block
- but final MAC is (again) too small for security
- Idea can be used with any symmetric block cipher

# Symmetric Ciphers to combine Secrecy and Integrity

Given a message M consisting of n blocks M1, M2, …Mn,

- use CBC with secret key k1 to produce $MAC_{k1}(M)$ as the final block
- using CBC with a different key k2, produce the ciphertext blocks C1, C2, …, Cn
- Send the blocks C1, C2, …, Cn and the authentication tag $MAC_{k1}(M)$

Question: why is it necessary to use two different keys? What could happen if the same key were used for both? (Exercise)

# Current Generation MAC

- HMAC-MD5, HMAC-SHA
  - IETF standard
  - general technique for **constructing** a MAC from a message digest (unkeyed) algorithm
- Older MACs are based on secret key encryption algorithms (notably DES) and are still in use
  - DES based MACs are 64 bit and not considered strong enough anymore

# HMAC (proposed late 90s)

- Make keyed cryptographic checksums using keyless cryptographic checksums
- *h* keyless cryptographic checksum function that takes data in blocks of *b* bytes and outputs blocks of *l* bytes. *k′* is cryptographic key of length *b* bytes
  - If short, pad with 0 bytes; if long, hash to length *b*
  - *ipad* is 00110110 repeated *b* times
  - *opad* is 01011100 repeated *b* times
- HMAC-*h*(*k*, *m*) = *h*(*k′* ⊕ *opad* || *h*(*k′* ⊕ *ipad* || *m*))
  - ⊕ exclusive or, || concatenation

# Security of HMAC

- Depends heavily on size of secret key
- Most common attack brute force
- Simpler version MAC-$h(k, m) = h(k \,||\, m)$ suffers from collision attacks
  - Unless the function h is SHA-3
- Same for MAC-$h(k, m) = h(k \,||\, m \,||\, k)$
- Values of *ipad* and *opad* not critical but chosen for their large Hamming distance (few bits in common between outer and inner key)
- HMAC-MD5 does not suffer from shortcomings of MD5