Authentication and Hash Functions

CSS 322 – Security and Cryptography

Contents

- Authentication using Encryption
- Principles of MAC and Hash Functions
- Example MAC and Hash Functions

Types of Attacks

Encryption

- Disclosure: release message contents
- Traffic Analysis: discover pattern of communication

- Masquerade: create messages pretending to be from someone else
- Content modification: change messages along a path
- Sequence modification: re-order messages along a path
- Timing modification: delay or replay messages
- Source repudiation: denial of transmission by source
- Destination repudiation: denial of transmission by destination

Authentication, Digital Signatures, …

Authentication Functions

- Authentication consists of two parts:
  - Function that produces low level authenticator
  - Protocol that enables receiver to verify identity
- Three types of functions:
  - Message Encryption
    - Ciphertext of entire message serves as authenticator
  - Message Authentication Code (MAC)
    - A function of message and secret key produces a fixed length authenticator
  - Hash Function
    - A function maps message to fixed-length hash value that serves as authenticator

Symmetric Key Encryption for Authentication

- Symmetric key encryption provides:
  - Confidentiality
  - Authentication: message must have come from A because A is only person with secret key K
- Assumes receiver can distinguish between legitimate and illegitimate ciphertext
  - When B decrypts ciphertext, must be able to determine that corresponding plaintext is legitimate
    - … and should be automatic detection
  - With English language plaintext, this is possible
  - What about binary messages (images, compressed files)?
    - The plaintext needs some known structure
  - Of all possible patterns of plaintext, only a small subset should be legitimate

Example

- Using Caesar cipher (K=1), legitimate ciphertext:

  `nbsftfbupbutboeepftfbupbutboemjuumfmbnctfbujwz`

- B decrypts to produce plaintext:

  `mareseatoatsanddoeseatoatsandlittlelambseativy`

- If attacker generated illegitimate ciphertext randomly:

  `zuvrsoevgqxkzwigamdvnmhpmccxiuureosfbcebtqxsxq`

- B would decrypt to:

  `ytuqrndufpwkyvhfzlcumlgolbbwhttqdnreabdaspwrwp`

- B would know this is illegitimate ciphertext!

Example

- What if this is legitimate ciphertext:

  `01110101011101001100010101101`

- B decrypts to produce plaintext:

  `11010010101110001010101001001`

- If attacker generated illegitimate ciphertext randomly:

  `01010111010010110100011010010`

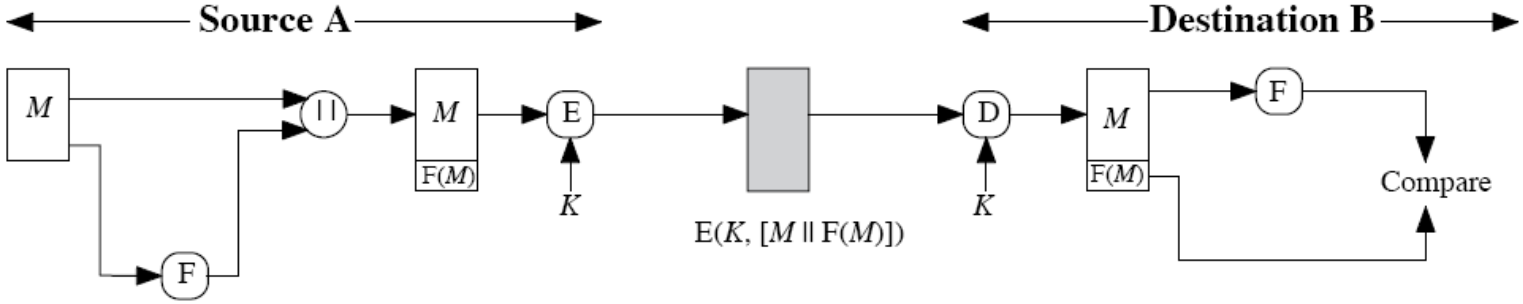
- B would decrypt to:

  `01011011011101000101010001101`

- How does B know whether it is legitimate or not???

Symmetric Key Encryption for Authentication

- Error checking codes (checksums) provide structure:
  - Append a checksum before encryption
  - At receiver, checksum must be correct



- Protocol headers provide structure:
  - For example, TCP contains a checksum and sequence numbers
  - If an attacker produces random ciphertext, the receiver B may detect that the TCP header is unstructured or contain incorrect fields

Public Key Encryption for Authentication

- Public key:
  - Encrypt with public key provides confidentiality
  - Encrypt with private key provides authentication
  - Same issues as using symmetric key for authentication

Message Authentication Code

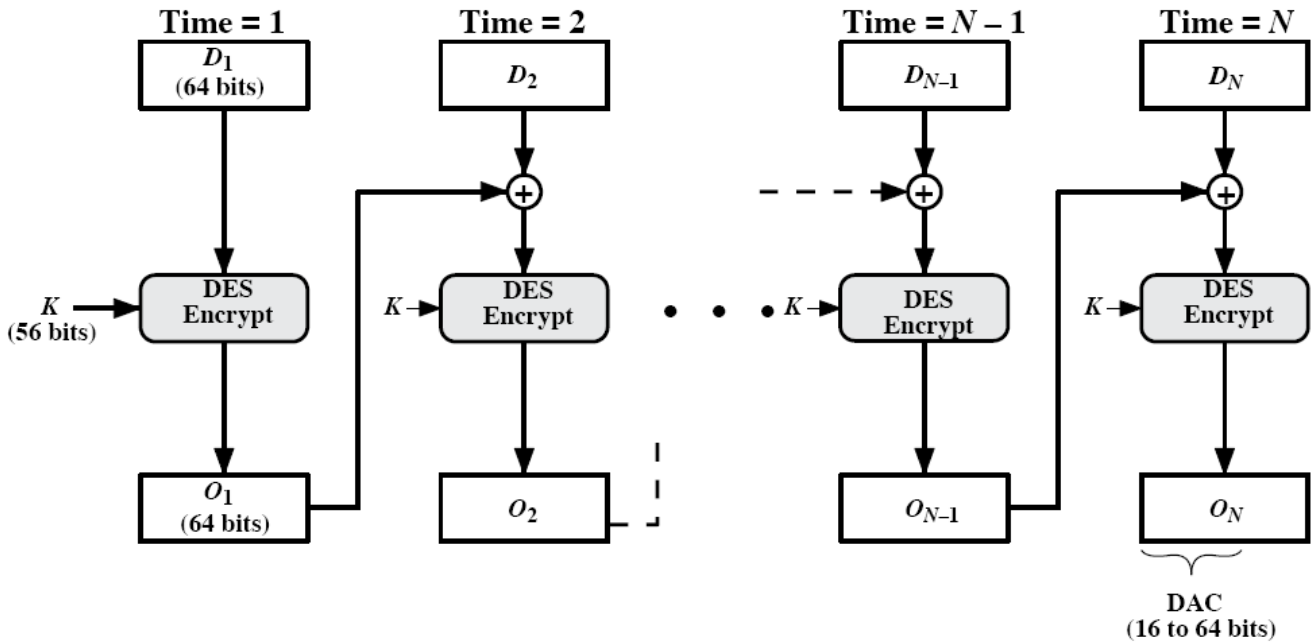
- Known as MAC or cryptographic checksum
- Encrypt input message M and shared secret key K with MAC function C:
  - MAC = C(K, M)
- Transmit the message M and MAC to recipient
- If received MAC matches transmitted MAC:
  - Receiver knows message has not been altered (otherwise MAC's wouldn't match)
  - Receiver knows message is from sender A (as no-one else has K)
  - If message has sequence numbers (e.g. TCP), then receiver knows proper sequence has been maintained

MAC and Encryption

- What is the difference?
  - MAC function is similar to encryption function, but can be made simpler
    - MAC does not need to be reversible (don't need decryption function)
    - The maths of MAC functions leads to MAC functions being stronger than encryption
- Why separate confidentiality and authentication?
  - Encryption is time consuming – some applications may not need encryption, but can do authentication to check sender
    - Example: SNMP for network management
  - Good to keep functionality separate – easier to design, implement and upgrade

Example: MAC based on DES

- Data Authentication Algorithm is based on DES
  - Was one of the most widely algorithms, but new faster and stronger algorithms in use
  - Use CBC and DES with initial value of 0



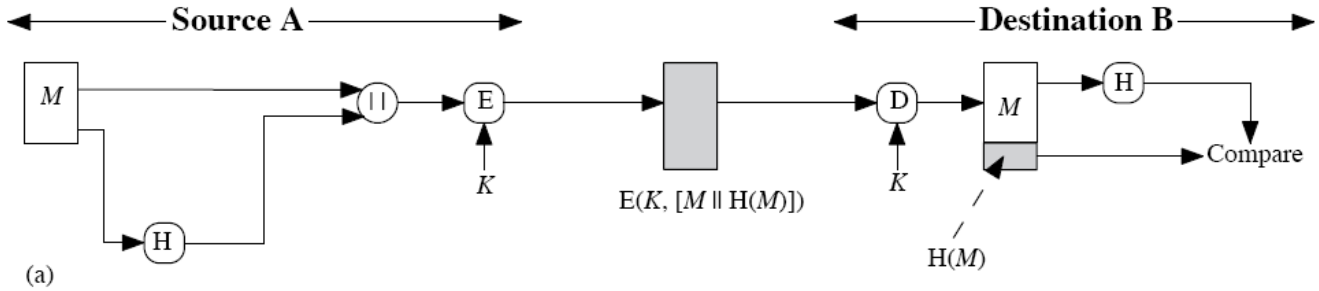
  - Data Authentication Code (DAC) is O_N or left most bits of O_N

Hash Function

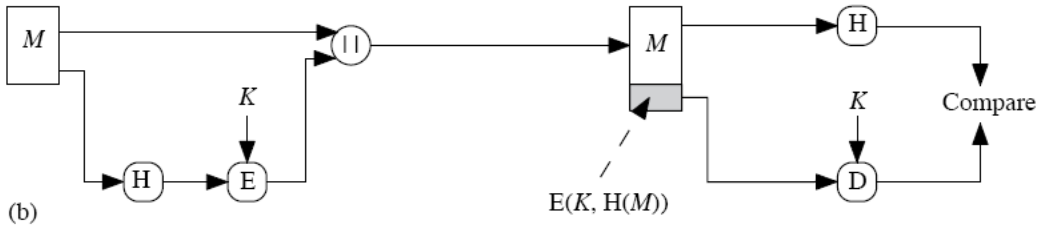
- Like a MAC, except do not need key:
  - Hash, h = H(M)
  - Output referred to as *hash value* or *message digest*
- Hash function has property that:
  - If you change any bits in the message, then the output has value will be different

Using Hash Functions

- Message and hash using symmetric key encryption
  - Confidentiality and authentication

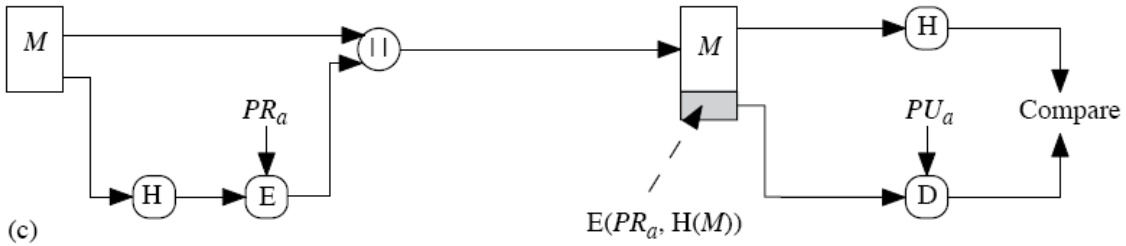


- Only hash encrypted with symmetric key
  - Authentication

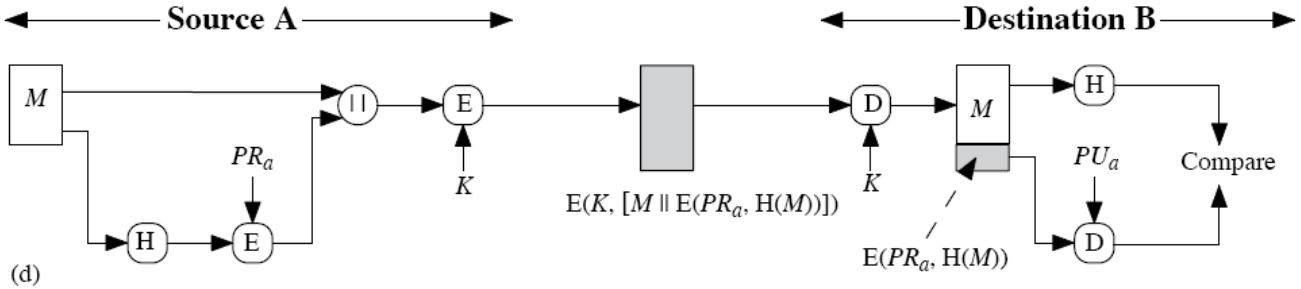


Using Hash Functions

- Only hash encrypted with sender's private key (using public key encryption)
  - Authentication and digital signature

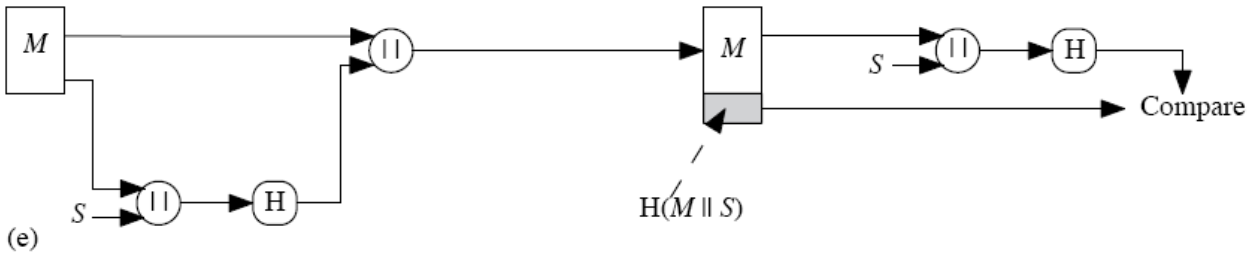


- Hash encrypted with sender's private key and message plus hash encrypted with symmetric shared key
  - Confidentiality, authentication and digital signature



Using Hash Functions

- Sender and receiver share secret value S; Hash the message and S, and send with message
  - Authentication, does not need encryption



- Encrypting only the hash code (instead of message) makes computations easier
- The final approach (above) does not rely on encryption
  - Useful because encryption software is often slow, encryption hardware is expensive and encryption algorithms may be patented

Requirements of Hash Function

- Hash function acts as a “fingerprint” of a message
- Hash function, H, should have the properties:
  - *H* can be applied to input message of any size
  - *H* produces fixed length output
  - *H* is easy to computer for an input message
  - For a given *h*, hard (infeasible) to find *x* such that *H(x) = h*
    - *One way property*
  - For a given message *x*, hard (infeasible) to find another message *y* such that *H(x) = H(y)*
    - *Weak collision resistance*
  - Hard (infeasible) to find pair (*x, y*) such that *H(x) = H(y)*
    - *Strong collision resistance*
- (First 3 properties are for practical implementations)

Simple Hash Function

- Break input message into *n*-bit blocks
  - Process each *n*-bit block at a time
- Hash bit *i* is XOR of all *i*-th bits in message (*m* blocks)

$$C_i = b_{i1} \oplus b_{i2} \oplus b_{i3} \oplus \ldots \oplus b_{im}$$

- Hash value is *C*

Security of MACs and Hash Functions

- Brute-force attack on Hash Functions with n-bit hash
  - One way property: 2^n
  - Weak collision resistance: 2^n
  - Strong collision resistance: $2^{n/2}$
  - Usually strong collision resistance is required
    - 128-bit hash can be broken in days; 160-bit is much more secure against brute force (but not other cryptanalysis!)
- Brute-force attack on MAC with n-bit MAC and k-bit key
  - Minimum of 2^n and 2^k
    - In practice, minimum of n, k is around 128 bits
- Cryptanalysis
  - Some efforts to use knowledge of algorithm, but attacks are quite hard and complex

MD5

- Developed by Ron Rivest in 1991
  - Improvements on previous Message Digest algorithms, MD1, …
- Standardised by IETF in RFC1321
- Generates 128-bit hash
- Commonly used in Unix passwords and checksums on files
- No longer secure
  - Collisions can be found using the birthday attack
  - Tools and databases of MD5 hashs and corresponding message are publicly available

Secure Hash Algorithm

- Developed by NIST and published in 1993
  - Revised version is called SHA-1
  - Also published by IETF in RFC 3174
  - SHA-1 uses 160 bit message digest size (hash value)
  - NIST recommend replacing SHA-1 with SHA-256, SHA-384, SHA-512

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

HMAC

- Aim to use a Hash function for a MAC
  - Hash Functions do not use key as input
- Motivation of HMAC:
  - Make use of existing Hash Functions, which are faster in software than DES
- HMAC provides a “wrapper” service for Hash functions
  - Turns Hash Functions into MACs
  - HMAC algorithm is generic – can use different Hash Functions
    - HMAC can use MD5, SHA-1, Whirlpool, … or any new hash functions