# COMP3028
# Lecture 8 – Cryptography V

## Message Authentication, Hash Functions, Digital Signatures

# Message Authentication

- message authentication is concerned with:
  - protecting the integrity of a message
  - validating identity of originator
  - non-repudiation of origin (dispute resolution)
- will consider the security requirements
- then three alternative functions used:
  - message encryption
  - message authentication code (MAC)
  - hash function

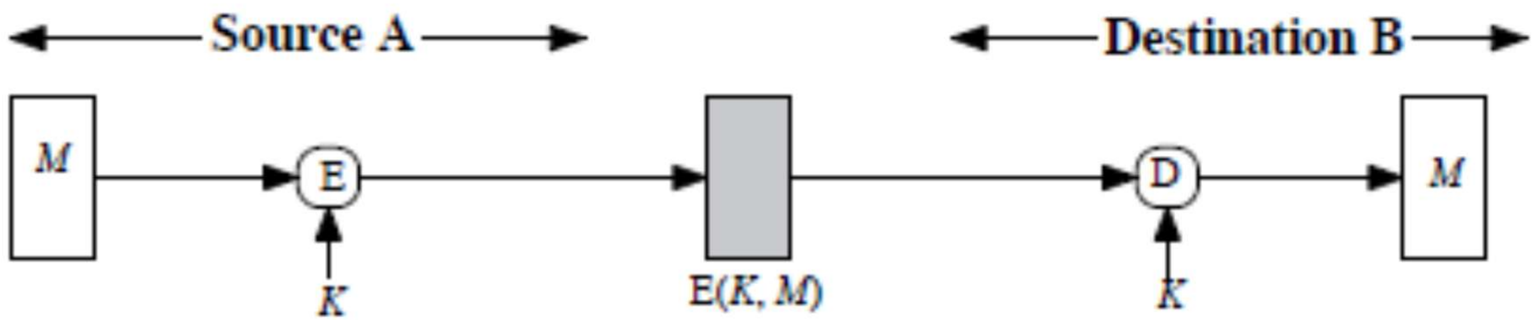
# Message Encryption

- message encryption by itself also provides a measure of authentication
- if symmetric encryption is used then:
  - receiver know sender must have created it
  - since only sender and receiver know key used
  - know content cannot of been altered
  - if message has suitable structure, redundancy or a checksum to detect any changes

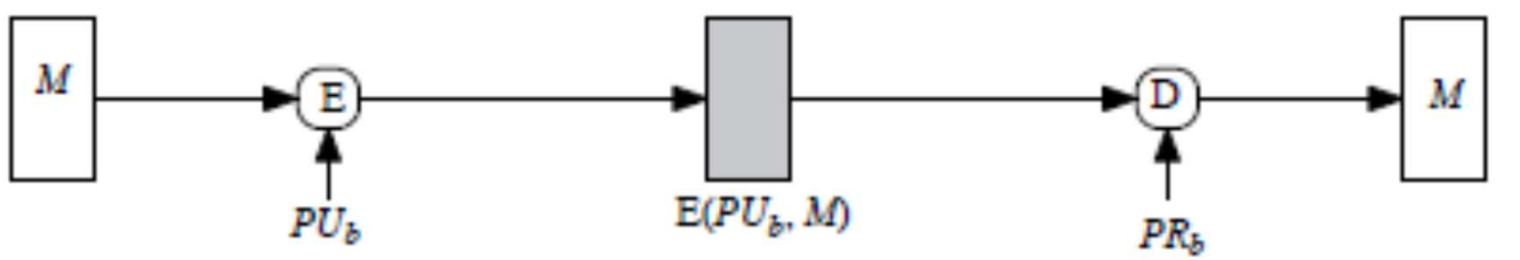
# Message Encryption

- if public-key encryption is used:
  - encryption provides no confidence of sender
  - since anyone potentially knows public-key
  - however if
    - sender **signs** message using their private-key
    - then encrypts with recipients public key
    - have both secrecy and authentication
  - again need to recognize corrupted messages
  - but at cost of two public-key uses on message

# Message Encryption (cont.)




(a) Symmetric encryption: confidentiality and authentication




(b) Public-key encryption: confidentiality

# Message Encryption (cont.)

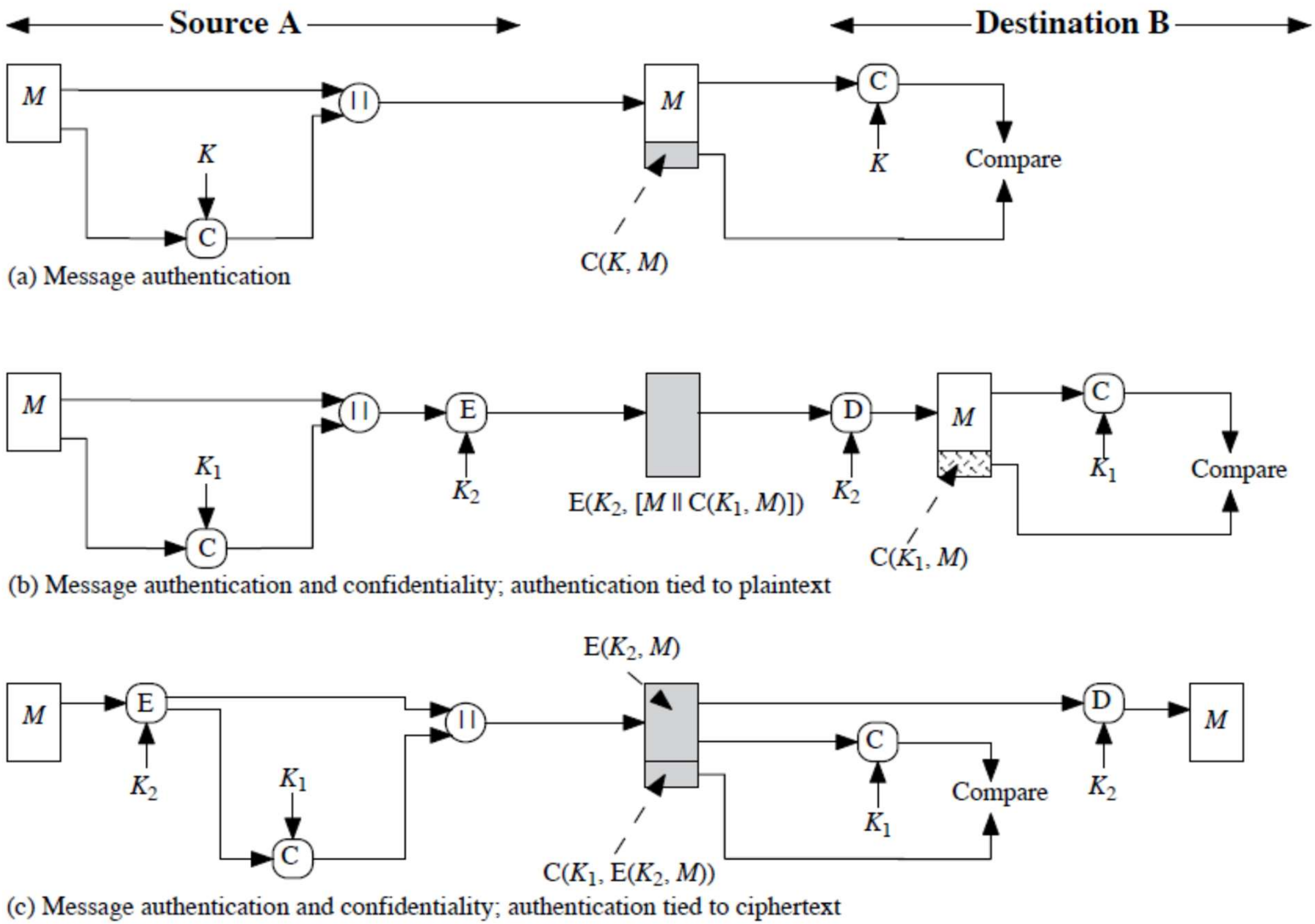

(c) Public-key encryption: authentication and signature


(d) Public-key encryption: confidentiality, authentication, and signature

# Message Authentication Code (MAC)

- generated by an algorithm that creates a small fixed-sized block
  - depending on both message and some key
  - like encryption though need not be reversible
- appended to message as a **signature**
- receiver performs same computation on message and checks it matches the MAC
- provides assurance that message is unaltered and comes from sender

# MAC (cont.)


Source A
Destination B
M
K
C
||
M
C
K
Compare
C(K, M)
(a) Message authentication
M
$K_1$
C
||
E
$K_2$
E($K_2$, [M || C($K_1$, M)])
D
$K_2$
M
C
$K_1$
Compare
C($K_1$, M)
(b) Message authentication and confidentiality; authentication tied to plaintext
M
E
$K_2$
$K_1$
C
||
E($K_2$, M)
C($K_1$, E($K_2$, M))
C
$K_1$
Compare
D
$K_2$
M
(c) Message authentication and confidentiality; authentication tied to ciphertext

# MAC (cont.)

- as shown the MAC provides confidentiality
- can also use encryption for secrecy
  - generally use separate keys for each
  - can compute MAC either before or after encryption
  - is generally regarded as better done before
- why use a MAC?
  - sometimes only authentication is needed
  - sometimes need authentication to persist longer than the encryption (eg. archival use)
- note that a MAC is not a digital signature

# MAC Properties

- a MAC is a cryptographic checksum

  $\texttt{MAC} = \texttt{C}_{\texttt{K}}(\texttt{M})$

  – condenses a variable-length message M
  – using a secret key K
  – to a fixed-sized authenticator
- is a many-to-one function
  – potentially many messages have same MAC
  – but finding these needs to be very difficult

# Requirements for MAC

- taking into account the types of attacks
- need the MAC to satisfy the following:
  1. knowing a message and MAC, is infeasible to find another message with same MAC
  2. MACs should be uniformly distributed
  3. MAC should depend equally on all bits of the message

# Using Symmetric Cipher for MAC

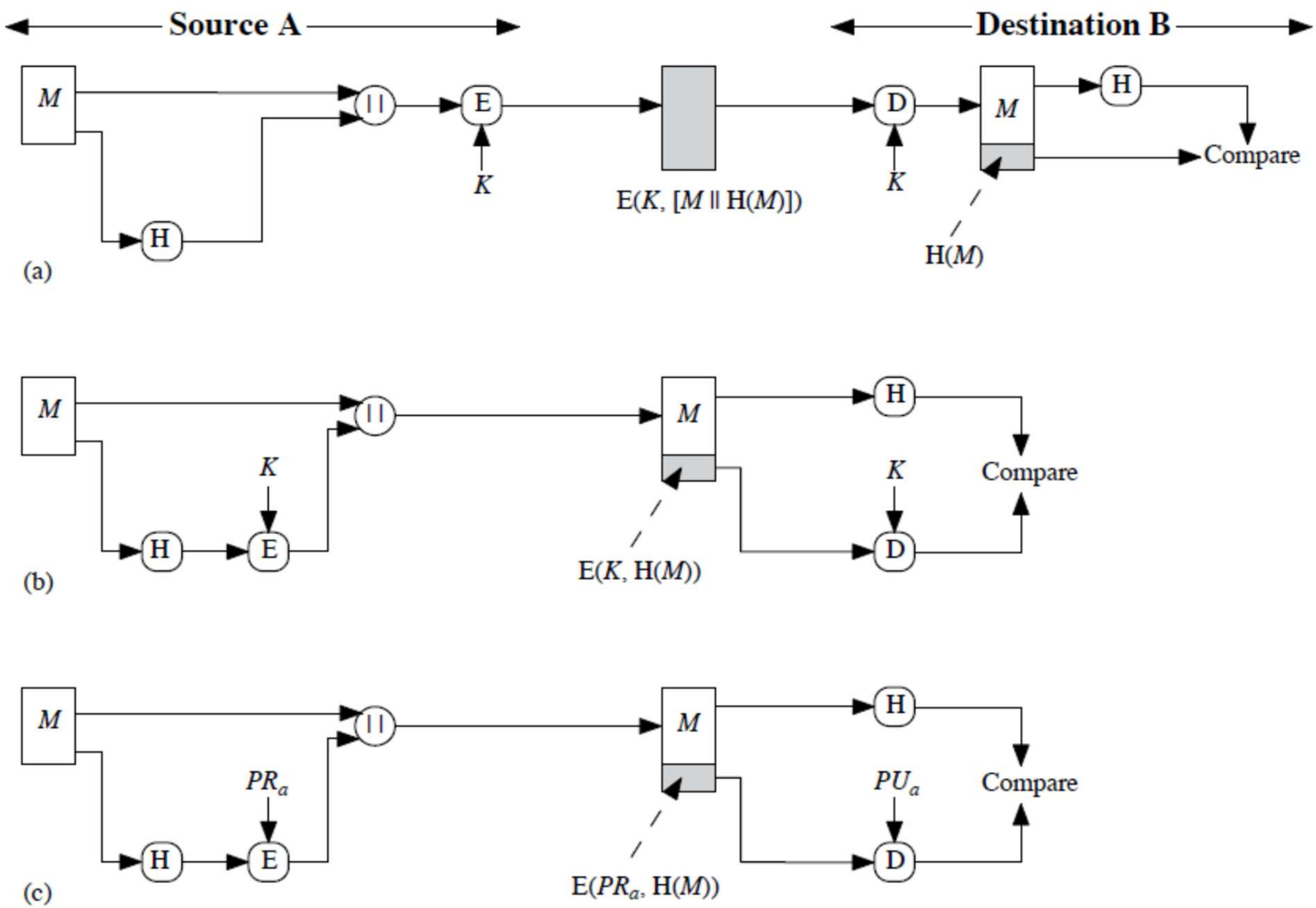
- can use any block cipher chaining mode and use final block as a MAC
- **Data Authentication Algorithm (DAA)** was a widely used MAC based on DES-CBC
  - using IV=**0** (64 bit of zero since DES is used) and zero-pad of final block
  - encrypt message using DES in CBC mode
  - and send just the final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

# Hash Functions

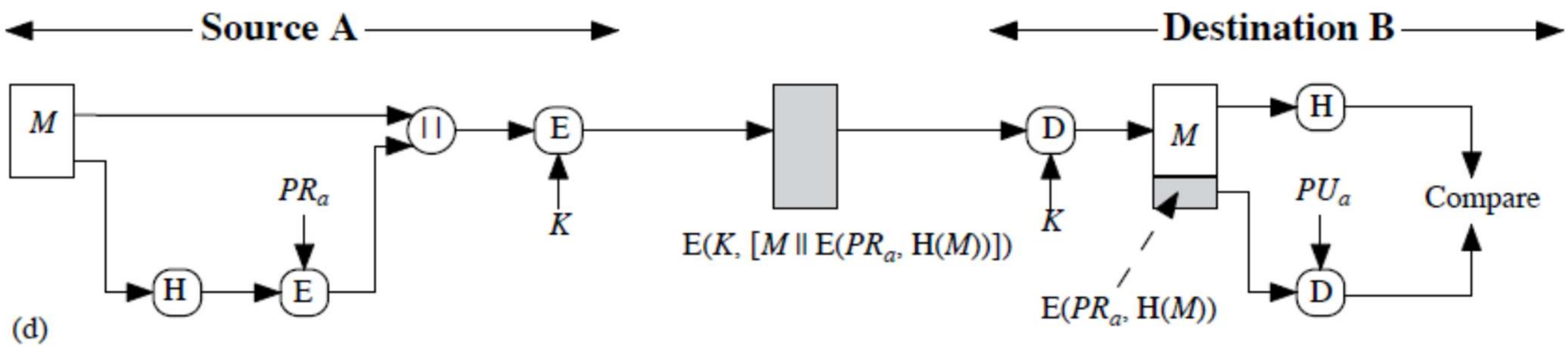
- condenses arbitrary message to fixed size

  `h = H(M)`
- usually assume that the hash function is public and not keyed
  - cf. MAC which is keyed
- hash used to detect changes to message
- can use in various ways with message
- most often to create a digital signature

# Hash Functions (cont.)



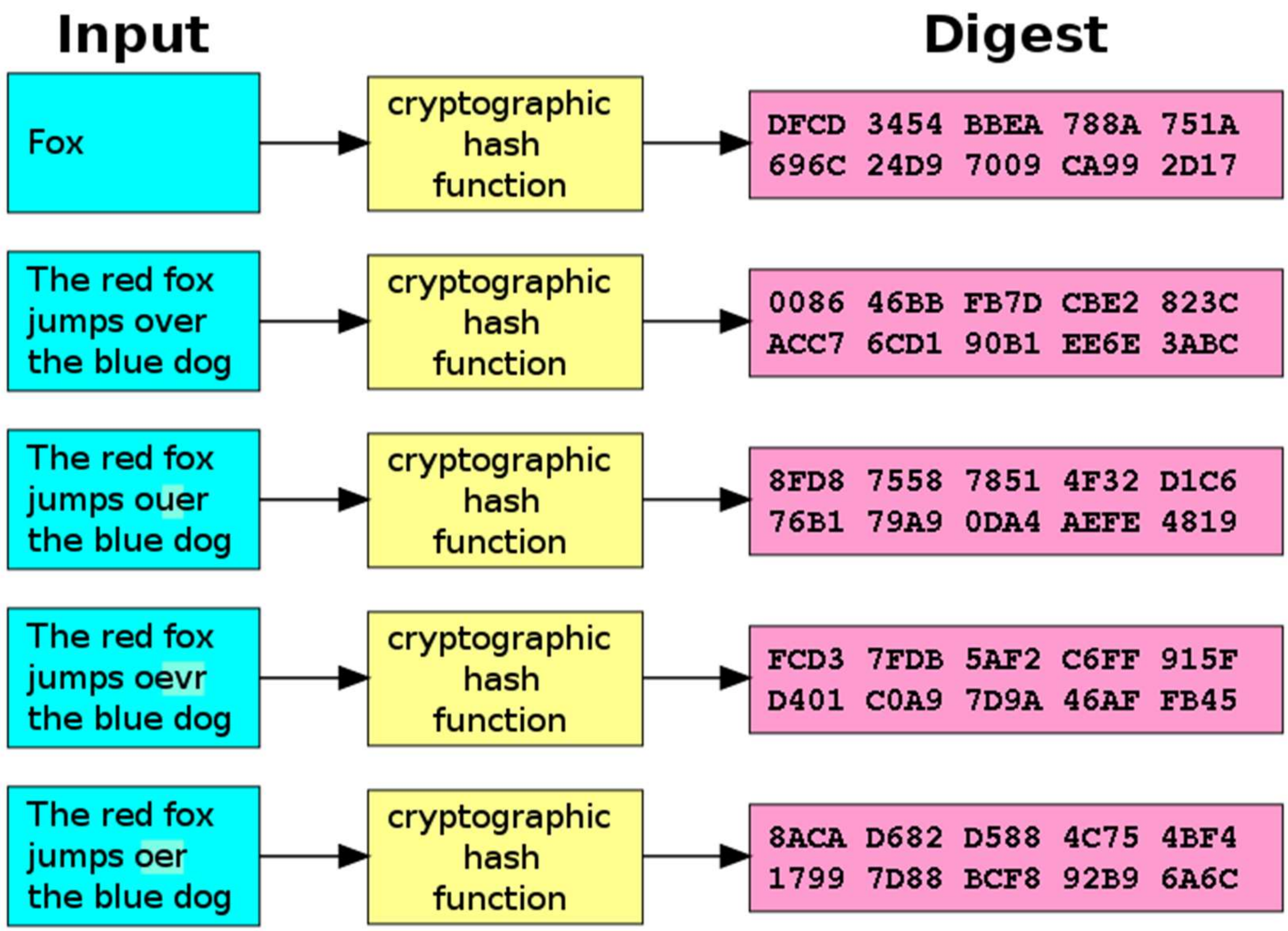
# Hash Functions (cont.)



# Requirements for Hash Functions

1. can be applied to any sized message $M$
2. produces fixed-length output $h$
3. is easy to compute $h=H(M)$ for any message $M$
4. given $h$ is infeasible to find $x$ s.t. $H(x)=h$
   - one-way property
5. given $x$ is infeasible to find $y$ s.t. $H(y)=H(x)$
   - weak collision resistance
6. is infeasible to find any $x,y$ s.t. $H(y)=H(x)$
   - strong collision resistance

# Avalanche Effect



# Simple Hash Functions

- There are several proposals for simple functions
- based on XOR of message blocks
- not secure since can manipulate any message and either not change hash or change hash also
- need a stronger cryptographic function

# Birthday Attacks

- might think a 64-bit hash is secure
- but by **Birthday Paradox** is not
- **birthday attack** works thus:
  - 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
- conclusion is that need to use larger MACs

# Block Ciphers as Hash Functions

- can use block ciphers as hash functions
  - using $H_0=0$ and zero-pad of final block
  - compute: $H_i = E_{M_i} [H_{i-1}]$
  - and use final block as the hash value
  - similar to CBC but without a key
- resulting hash is too small (64-bit)
  - both due to direct birthday attack
  - and to "meet-in-the-middle" attack
- other variants also susceptible to attack

# 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
  - nb. the algorithm is SHA, the standard is SHS
- 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

# Revised Secure Hash Standard

- NIST issued revision FIPS 180-2 in 2002
- adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1
- hence analysis should be similar
- but security levels are rather higher

# SHA-3

**SHA-1 has not yet been "broken"**

- No one has demonstrated a technique for producing collisions in a practical amount of time
- Considered to be insecure and has been phased out for SHA-2

**SHA-2 shares the same structure and mathematical operations as its predecessors so this is a cause for concern**

- Because it will take years to find a suitable replacement for SHA-2 should it become vulnerable, NIST decided to begin the process of developing a new hash standard

**NIST announced in 2007 a competition for the SHA-3 next generation NIST hash function**

- Winning design was announced by NIST in October 2012
- SHA-3 is a cryptographic hash function that is intended to complement SHA-2 as the approved standard for a wide range of applications

# The Sponge Construction

- Underlying structure of SHA-3 is a scheme referred to by its designers as a sponge construction
- Takes an input message and partitions it into fixed-size blocks
- Each block is processed in turn with the output of each iteration fed into the next iteration, finally producing an output block
- The sponge function is defined by three parameters:
  - f = the internal function used to process each input block
  - r = the size in bits of the input blocks, called the bitrate
  - pad = the padding algorithm
- https://csrc.nist.gov/projects/hash-functions/sha-3-project

# Hash Algorithms

| Name | Output size (bits) | Rounds | Security |
|---|---|---|---|
| MD5 | 128 | 64 | Broken (Collision attack) |
| SHA-1 | 160 | 80 | Theoretically vulnerable |
| RIPEMD-160 | 160 | 80 | Used in Bitcoin |
| Whirlpool | 512 | 10 | Based on AES |
| SHA-2 | 224,256,384,512 | 64,80 | Some theories, currently considered safe |
| BLAKE2 | 256,512 | 10,12 | Based on ChaCha Stream Cipher, SHA candidate |
| SHA-3 (Keccak) | 224,256,384,512 | 24 | Secure, but relatively untested |
| BLAKE3 | 256 but extensible | 7 | Very new, fast |

# Digital Signatures

- have looked at message authentication
  - but does not address issues of lack of trust
- digital signatures provide the ability to:
  - verify author, date & time of signature
  - authenticate message contents
  - be verified by third parties to resolve disputes
- hence include authentication function with additional capabilities

# Digital Signature Properties

- must depend on the message signed
- must use information unique to sender
  - to prevent both forgery and denial
- must be relatively easy to produce
- must be relatively easy to recognize & verify
- be computationally infeasible to forge
  - with new message for existing digital signature
  - with fraudulent digital signature for given message
- be practical to save digital signature in storage

# Direct Digital Signatures

- involve only sender & receiver
- assumed receiver has sender’s public-key
- digital signature made by sender signing entire message or hash with private-key
- can encrypt using recipient’s public-key
- important that sign first then encrypt message & signature
- security depends on sender’s private-key

# ElGamal Digital Signature

- signature variant of ElGamal, related to D-H
  - so uses exponentiation in a finite (Galois)
  - with security based difficulty of computing discrete logarithms, as in D-H
- use private key for encryption (signing)
- uses public key for decryption (verification)
- each user (eg. Alice) generates their key
  - chooses a secret key (number): $1 < x_A < q-1$
  - compute their **public key**: $y_A = a^{x_A} \bmod q$

# ElGamal Digital Signature

- Alice signs a message M to Bob by computing
  - the hash $m = H(M)$, $0 <= m <= (q-1)$
  - chose random integer $K$ with $1 <= K <= (q-1)$ and $GCD(K,q-1)=1$
  - compute temporary key: $S_1 = a^k \bmod q$
  - compute $K^{-1} \bmod (q-1)$
  - compute the value: $S_2 = K^{-1}(m-x_A S_1) \bmod (q-1)$
  - signature is: $(S_1, S_2)$
- Bob can verify the signature by computing
  - $V_1 = a^m \bmod q$
  - $V_2 = {y_A}^{S_1} {S_1}^{S_2} \bmod q$
  - signature is valid if $V_1 = V_2$

# ElGamal Signature Example

- use field GF(19) $q=19$ and $a=10$
- Alice computes her key:
  - A chooses $x_A=16$ & computes $y_A=10^{16} \bmod 19 = 4$
- Alice signs message with hash $m=14$:
  - choosing random $K=5$ which has $\gcd(18,5)=1$
  - computing $S_1 = 10^5 \bmod 19 = 3$
  - finding $K^{-1} \bmod (q-1) = 5^{-1} \bmod 18 = 11$
  - computing $S_2 = 11(14-16.3) \bmod 18 = 4$
- Bob can verify the signature by computing
  - $V_1 = 10^{14} \bmod 19 = 16$
  - $V_2 = 4^3.3^4 = 5184 = 16 \bmod 19$
  - since $16 = 16$ signature is valid

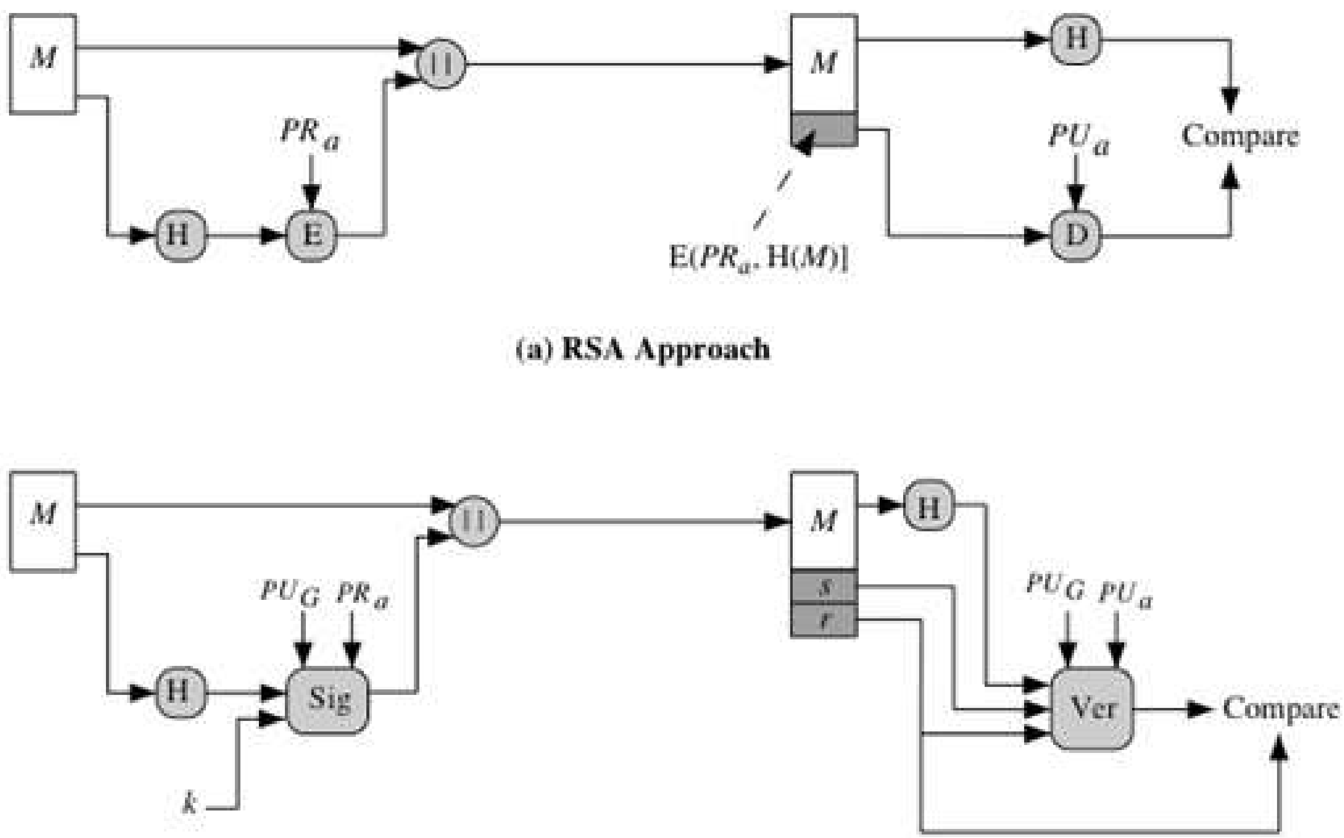
# Digital Signature Standard (DSS)

- US Govt approved signature scheme
- designed by NIST & NSA in early 90's
- published as FIPS-186 in 1991
- revised in 1993, 1996 & then 2000
- uses the SHA hash algorithm
- DSS is the standard, DSA was the algorithm
- The latest version, FIPS 186-5 (2020) also incorporates digital signature algorithms based on RSA and elliptic curve cryptography
  - DSA is only used for verification

# Digital Signature Algorithm (DSA)

- creates a 320 bit signature
- with 512-1024 bit security
- smaller and faster than RSA
- a digital signature scheme only
- security depends on difficulty of computing discrete logarithms

# Two Approaches to Digital Signatures




**(a) RSA Approach**

**(b) DSA Approach**

# DSA Key Generation

- have shared global public key values (p,q,g):
  - a large prime $p = 2^L$
    - where L= 512 to 1024 bits and is a multiple of 64
  - choose q, a 160 bit prime factor of p-1
  - choose $g = h^{(p-1)/q}$
    - where $h<p-1$, $h^{(p-1)/q} \pmod p > 1$
- users choose private & compute public key:
  - choose $x<q$
  - compute $y = g^x \pmod p$

# DSA Signature Creation

- to **sign** a message $M$ the sender:
  - generates a random signature key $k$, $k<q$
  - nb. $k$ must be random, be destroyed after use, and never be reused
- then computes signature pair:

$$r = (g^k \bmod p) \bmod q$$

$$s = (k^{-1}.SHA(M) + x.r) \bmod q$$

- sends signature $(r,s)$ with message $M$

# DSA Signature Verification

- having received M & signature `(r,s)`
- to **verify** a signature, recipient computes:

$w = s^{-1} \bmod q$

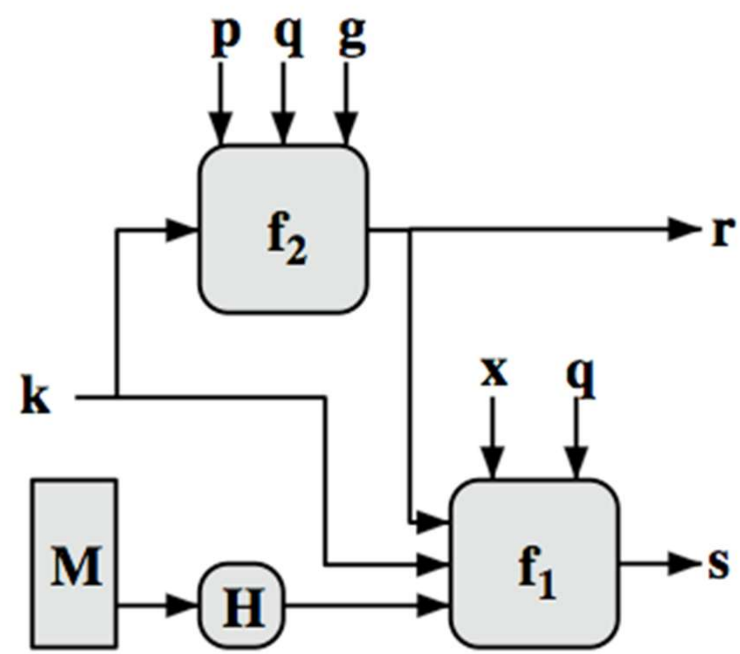
$u1 = (SHA(M).w) \bmod q$

$u2 = (r.w) \bmod q$

$v = (g^{u1}.y^{u2} \bmod p) \bmod q$

- if `v=r` then signature is verified

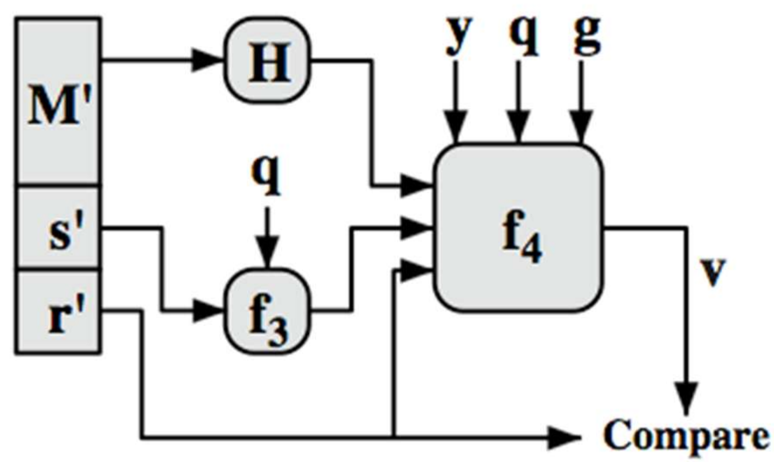
# DSS Overview




$s = f_1(H(M), k, x, r, q) = (k^{-1}\,(H(M) + xr)) \bmod q$

$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$

**(a) Signing**




$w = f_3(s', q) = (s')^{-1} \bmod q$

$v = f_4(y, q, g, H(M'), w, r')$

$= ((g^{(H(M')w) \bmod q}\, y^{r'w \bmod q}) \bmod p) \bmod q$

**(b) Verifying**

# Summary

- Message authentication code
- Hash functions
  - general approach & security
  - some hash algorithms
- Digital signature
  - Direct digital signature
  - Digital signature standard & algorithm