Digital Signatures

- Message authentication protects two parties who exchange messages from any third party.
- However, it does not protect the two parties against each other.
- In situations where there is not complete trust between sender and receiver, something more than authentication is needed.
- The more attractive solution is the digital signature

Overview

- 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 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 receivers public-key
- important that sign first then encrypt message & signature
- security depends on sender’s private-key

Arbitrated Digital Signatures

- involves use of arbiter A
  - validates any signed message
  - then dated and sent to recipient
- requires suitable level of trust in arbiter
- can be implemented with either private or public-key algorithms
- arbiter may or may not see message

Authentication Protocols

- used to convince parties of each other’s identity and to exchange session keys
- may be one-way or mutual
- key issues are
  - confidentiality – to protect session keys
  - timeliness – to prevent replay attacks

Replay Attacks

- where a valid signed message is copied and later resent
  - simple replay: The opponent simply copies a message and replays it later
  - repetition that can be logged: An opponent can replay a timestamped message within the valid time window.
  - repetition that cannot be detected: This situation could arise because the original message could have been suppressed and thus did not arrive at its destination; only the replay message arrives.

- backward replay without modification: This is a replay back to the message sender. This attack is possible if symmetric encryption is used and the sender cannot easily recognize the difference between messages sent and messages received on the basis of content

- countermeasures include
  - use of sequence numbers: it requires each party to keep track of the last sequence number for each claimant it has dealt with. It increases overhead, so impractical.

- challenge/response (using unique nonce): Party A, expecting a fresh message from B, first sends B a nonce (challenge) and requires that the subsequent message (response) received from B contains the correct nonce value

- timestamps (needs synchronized clocks): party A accepts a message as fresh only if the message contains a timestamp that, in A’s judgment, is close enough to A’s knowledge of current time. This approach requires that clocks among the various participants be synchronized.

Using Symmetric Encryption

- can use a two-level hierarchy of keys
- usually with a trusted Key Distribution Center (KDC)
  - each party shares own master key with KDC
  - KDC generates session keys used for connections between parties
  - master keys used to distribute these to them

Needham-Schroeder Protocol

- original third-party key distribution protocol
- for session between A, B mediated by KDC
- protocol overview is:

1. A$\rightarrow$KDC: $ID_A \,||\, ID_B \,||\, N_1$

2. KDC$\rightarrow$A: $\mathrm{E}_{\mathrm{Ka}}[\mathrm{Ks} \,||\, ID_B \,||\, N_1 \,||\, \mathrm{E}_{Kb}[Ks||ID_A]\,]$

3. A$\rightarrow$B: $E_{Kb}[Ks||ID_A]$

4. B$\rightarrow$A: $E_{Ks}[N_2]$

5. A$\rightarrow$B: $E_{Ks}[\mathrm{f}(N_2)]$

- Ka, Kb : Secret keys
- Ks : session key

Needham-Schroeder Protocol

- used to securely distribute a new session key for communications between A & B
- but is vulnerable to a replay attack if an old session key has been compromised
  - then message 3 can be resent convincing B that is communicating with A
- modifications to address this require:
  - timestamps
  - using an extra nonce

Using Public-Key Encryption

- have a range of approaches based on the use of public-key encryption
- need to ensure have correct public keys for other parties
- using a central Authentication Server (AS)
- various protocols exist using timestamps or nonce

Denning AS Protocol

- Denning 81 presented the following:
  1. A→AS: $ID_A \| ID_B$
  2. AS→A: $E_{KRas}[ID_A \| KU_a \| T] \| E_{KRas}[ID_B \| KU_b \| T]$
  3. A→B: $E_{KRas}[ID_A \| KU_a \| T] \| E_{KRas}[ID_B \| KU_b \| T] \| E_{KUb}[E_{KRas}[K_s \| T]]$
- note session key is chosen by A, hence AS need not be trusted to protect it
- timestamps prevent replay but require synchronized clocks

One-Way Authentication

- required when sender & receiver are not in communications at same time (eg. email)
- have header in clear so can be delivered by email system
- may want contents of body protected & sender authenticated

Using Symmetric Encryption

- can refine use of KDC but can't have final exchange of nonces, vis:
  1. A→KDC: $ID_A \| ID_B \| N_1$
  2. KDC→A: $E_{Ka}[Ks \| ID_B \| N_1 \| E_{Kb}[Ks \| ID_A]\ ]$
  3. A→B: $E_{Kb}[Ks \| ID_A] \| E_{Ks}[M]$
- does not protect against replays
  – could rely on timestamp in message, though email delays make this problematic

Public-Key Approaches

- have seen some public-key approaches
- if confidentiality is major concern, can use:

  $A \rightarrow B: E_{KUb}[Ks] \| E_{Ks}[M]$

  – has encrypted session key, encrypted message
- if authentication needed use a digital signature with a digital certificate:

  $A \rightarrow B: M \| E_{KRa}[H(M)] \| E_{KRas}[T \| ID_A \| KU_a]$

  – with message, signature, certificate

Digital Signature Standard (DSS)

- US Govt approved signature scheme FIPS 186
- uses the SHA hash algorithm
- designed by NIST & NSA in early 90's
- DSS is the standard, DSA is the algorithm
- a variant on ElGamal and Schnorr schemes
- creates a 320 bit signature, but with 512-1024 bit security
- security depends on difficulty of computing discrete logarithms

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 $1<h<p-1$, $h^{(p-1)/q} \pmod{p} > 1$
- users choose private & compute public key:
  - choose $1<x<q$
  - 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:

$$r = (g^{k} \pmod{p}) \pmod{q}$$

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

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

DSA Signature Verification

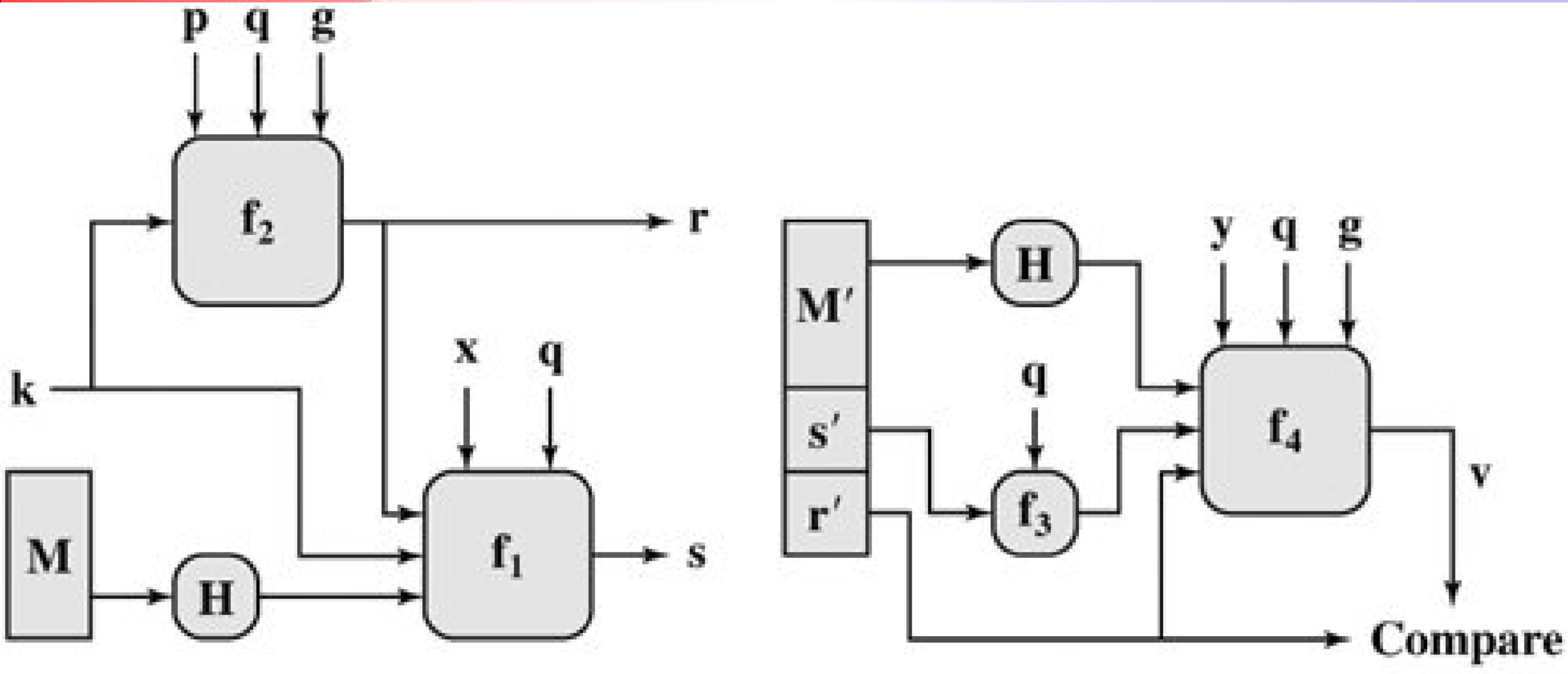
- having received M & signature `(r,s)`
- to **verify** a signature, recipient computes:

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

  $u1 = (H(M).w) \pmod q$

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

  $v = (g^{u1}.y^{u2} \pmod p) \pmod q$
- if `v=r` then signature is verified



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

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

(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