Digital Signatures & Authentication Protocols

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 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

Conventional Encryption, Arbiter Sees Message

(1) X → A: $M \| E(K_{xa}, [ID_X \| H(M)])$

(2) A → Y: $E(K_{ay}, [ID_X \| M \| E(K_{xa}, [ID_X \| H(M)]) \| T])$

Conventional Encryption, Arbiter Does Not See Message

(1) X → A: $ID_X \| E(K_{xy}, M) \| E(K_{xa}, [ID_X \| H(E(K_{xy}, M))])$

(2) A → Y: $E(K_{ay}, [ID_X \| E(K_{xy}, M)]) \| E(K_{xa}, [ID_X \| H(E(K_{xy}, M)) \| T])$

Public-Key Encryption, Arbiter Does Not See Message

(1) X $\rightarrow$ A: $ID_X \| E(PR_x, [ID_X \| E(PU_y, E(PR_x, M))])$

(2) A $\rightarrow$ Y: $E(PR_a, [ID_X \| E(PU_y, E(PR_x, M)) \| T])$

- X = sender
- Y = recipient
- A = Arbiter
- M = message
- T = timestamp

Authentication Protocols

- used to convince parties of each others 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
  - repetition that can be logged
  - repetition that cannot be detected
  - backward replay without modification
- countermeasures include
  - use of sequence numbers (generally impractical)
  - timestamps (needs synchronized clocks)
  - challenge/response (using unique nonce)

Using Symmetric Encryption

- as discussed previously 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→KDC: $ID_A \mid\mid ID_B \mid\mid N_1$

2. KDC→A: $E_{Ka}[Ks \mid\mid ID_B \mid\mid N_1 \mid\mid E_{Kb}[Ks \mid\mid ID_A]\,]$

3. A→B: $E_{Kb}[Ks \mid\mid ID_A]$

4. B→A: $E_{Ks}[N_2]$

5. A→B: $E_{Ks}[f(N_2)]$

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 (Denning 81)
  - using an extra nonce (Neuman 93)

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 nonces

Denning AS Protocol

- Denning 81 presented the following:

  1. $A \rightarrow AS: ID_A \,||\, ID_B$

  2. $AS \rightarrow A: E_{KRas}[ID_A||KU_a||T] \,||\, E_{KRas}[ID_B||KU_b||T]$

  3. $A \rightarrow B: E_{KRas}[ID_A||KU_a||T] \,||\, E_{KRas}[ID_B||KU_b||T] \,||\, E_{KUb}[E_{KRas}[K_s||T]]$
- 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$\longrightarrow$KDC: $ID_A \,||\, ID_B \,||\, N_1$

  2. KDC$\longrightarrow$A: $E_{Ka}[Ks \,||\, ID_B \,||\, N_1 \,||\, E_{Kb}[Ks||ID_A]\,]$

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

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

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

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

DSA Signature Verification

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

  `w = s`$^{-1}$`(mod q)`

  `u1= (SHA(M).w)(mod q)`

  `u2= (r.w)(mod q)`

  `v = (g`^{u1}`.y`^{u2}`(mod p)) (mod q)`
- if `v=r` then signature is verified
- see book web site for details of proof why

Summary

- have considered:
  - digital signatures
  - authentication protocols (mutual & one-way)
  - digital signature standard