Key Management (2)

What’s To Address

- Key distribution
  - Kerberos (read lecture notes and textbook)
  - public key cryptography key exchange
- Key generation
- Key maintenance
- Key revocation (omitted)

Kerberos

- Uses revised Needham-Schroeder protocol
- A client: Alice
- A service provider: Gutterberg
- A Kerberos authentication server: Cerberus
- A ticket server: Barnum

Steps

1. Alice → Cerberus: AliceBarnum
2. Cerberus → Alice: \( \{ k_{A,B} \} k_{A,C} \oplus T_{A,B} \)
   
   \[ \text{[Ticket]} \quad T_{A,B} = \text{{Alice’s address valid time} } k_{A,B} \]

3. Alice → Barnum: Gutterberg II \( A_{A,B} \parallel T_{A,B} \)
   
   \[ T_{A,B} = \text{Bil(Alice’s address valid time} k_{A,B} ) k_B \]

   [Authentication] \( A_{A,B} = \text{All generation time} k_B \)

4. Barnum → Alice: Alice \( k_{A,G} k_{A,B} \parallel T_{A,G} \)

5. Alice → Gutterberg: \( A_{A,G} \parallel T_{A,G} \)

   \[ T_{A,G} = \text{Bil(Alice’s address valid time} k_{A,G} ) k_G \]

   \( A_{A,G} = \text{All generation time} k_G \)

6. Gutterberg → Alice: \( \{ t+1 \} k_{A,G} \)

(cont’d)
Public Key Cryptography Key Exchange

- Very easy
  
  \[ \text{Alice} \rightarrow \text{Bob} : \{k_s\} e_{\text{Bob}} \]
  
  – Alice and Bob then can securely communicate using a symmetric cryptosystem

- Flaw: Bob does not know who sent the message
  – Could be Eve forging such a message

Fix the Flaw: Sign \( k_s \)

\[ \text{Alice} \rightarrow \text{Bob} : \text{Alice}, \{\{k_s\} d_{\text{Alice}}\} e_{\text{Bob}} \]

- Question: What if Alice does not have Bob’s public key?

Key Generation

- Critical to generate keys that are hard to figure out by attackers

- Key generation issue is regarded as a randomness issue
  – Given a set of K potential keys, the minimum probability of successfully guessing the key is achieved when the key is randomly selected

Random Numbers

- A sequence of (cryptographically) random numbers is a sequence of numbers \( n_1, n_2, \ldots \) such that for any \( k \), an observer cannot predict \( n_k \) even if \( n_1, n_2, \ldots, n_{k-1} \) are known

- A random number generator requires a physical source of randomness
  – Disk latency at different time
  – Background radiation

Pseudorandom Numbers

- Physical sources of randomness are often not available

- A sequence of (cryptographically) pseudorandom numbers is a sequence of numbers intended to simulate a sequence of cryptographically random numbers
  – But generated by an algorithm
  – Such algorithm is difficult

Pseudorandom Number Generation

- A linear congruential generator
  
  \[ n_k = (a n_{k-1} + b) \mod n \] (\( a, b, n \) are relatively prime)

- A polynomial congruential generator
  
  \[ n_k = (a_1 n_{k-1} + a_2 n_{k-2} + \cdots + a_d n_{k-d}, a_0) \mod n \]

- A mixing function \( f \)
  
  \[ y = f(x_1, x_2, \ldots) \]
  
  where every bit of \( y \) is a nonlinear function of all the bits of \( x_1, x_2, \ldots \) (e.g. DES, MD4, MD5, SHA-1)

  – Initial input to \( f \) must be unpredictable, e.g. \( \text{date; ps auxa} \mid \text{md5} \)
Key Maintenance

- In classical cryptosystems, every key is shared between two (or more) entities
  - Then the key should be stored at every sharing entity during the key’s lifetime
  - Simple
- In public key cryptosystems, every entity has a public key and private key
  - Which are bound to the entity
- We focus on key maintenance for public key cryptography

Identification and Public Key

- Every node has an ID
- Every node has a public key
- The association between the ID and the key is critical
- A central question: is this the public key for node X?
  - X is the ID

A Preliminary Solution

- A node encrypts (signs) its public key with its private key
  \[ (e) \_d \rightarrow \text{recipient} \]
- The recipient can only decrypt using the public key in question
- Thus confirm that \( e \) is the public key of the signing guy
- But who is the signing guy?
- It won’t help by adding the ID, either
  \[ (e, \text{Alice}) \_d \rightarrow \text{recipient} \]
  since the \( e \) and \( d \) here can actually belong to Eve!

Certificate-Based Solution

- A **certificate** is a token that binds an identity to a cryptographic key
  \[ C_{\text{Alice}} = (e_{\text{Alice}} \_\text{Alice} \_T) \_d_{\text{Cathy}} \]
- A **certificate authority** (CA) issues certificates

Certificate Verification

- Suppose Bob knows Cathy’s public key \( e_{\text{Cathy}} \)
- When Bob obtains \( C_{\text{Alice}} \)
  - Deciphers \( C_{\text{Alice}} \) using \( e_{\text{Cathy}} \)
  - Then knows that Cathy is vouching that \( e_{\text{Alice}} \) is Alice’s public key, issued at time \( T \)
  - If Bob trusts what Cathy believes
  - Then Bob knows \( e_{\text{Alice}} \) is Alice’s public key

But, Bob Has to Know \( e_{\text{Cathy}} \)

- Two solutions
  - Merkle’s Tree Authentication Scheme
    - Eliminates Cathy’s signature
  - Chain of certificates
    - There is another certificate for \( e_{\text{Cathy}} \)
Merkle’s Tree Authentication Scheme

- All <id, public key> pairs are stored in a file
- A cryptographic hash function creates a digest of the file
  - The digest is known to the public
- If any pair is changed, it will be detected
  - Since the digest will be different

Digest Algorithm

- A tree-based algorithm

Public Key Verification

- How can Bob verify whether or not Alice’s public key is 72384927894027.
- Bob will re-compute the digest, and compare that with the publicly known value of the digest
  - If Alice’s public key is not 72384927894027, a discrepancy will be detected

Authentication Path

- Bob knows $Y_3$
- Bob needs to know $h(4,4)$ and $h(1,2)$
- $Y_3$, $h(4,4)$ and $h(1,2)$ is the authentication path for Alice’s public key
  - They can put together and used for certifying Alice’s public key
- This is a certificate without signature!
Certificate Signature Chains

- X.509
- PGP

Tree-like CA hierarchy employed
- Every node has a local CA
- A local CA has its CA, the parent
- The parent CA has its parent
- And there is a root CA
- Together, a tree of CAs!

X.509

- X.509 defines certificate formats and validation in generic context
  - X.509v3 is the current version
- Format:
  - Version, serial number
  - issuer’s name, id, signature algorithm id
  - subject’s name, id, public key, validity interval
  - extensions
  - Signature

Can One CA Serve All?

- Probably not!
- Probably many CAs instead, each certifying some
  - Every node has a local CA

Certificate Chains

- Cathy certifies Dan’s public key
  - Cathy <<Dan>>
- If Dan <<Bob>>, Bob<<David>>, and Alice knows Cathy’s public key,
  - then a certificate chain is formed
  - Alice can validate Bob’s public key by going through the chain

PGP Certificate Chains

- PGP (Pretty Good Privacy) provides privacy for email
  - Can also be used to sign files
  - We look at OpenPGP below
- An OpenPGP certificate is a sequence of packets
  - A public key packet followed by 0+ signature packets
  - Each packet is a record with a tag describing its purpose

Public Key Packet

- Version
- Creation time
- Validity period
- Public key algorithms (and parameters)
- Public key (of course)
Signature Packet

- Version
- Signature type
- Creation time
- Key identifier of the signer
- Public key algorithm
- Hash algorithm
- Part of signed hash value
- Signature (of course!)

PGP Certificate Features

- PGP certificate allows multiple signatures
- Each signature has a different level of “trust”
- Different from X.509

PGP Certificate Chain Example

Alice is verifying Bob’s public key
- Ellen, Fred, Giselle, Bob <<Bob>>
- Henry, Irene, Giselle <<Giselle>>
- Ellen, Henry <<Henry>>
- Jack, Ellen <<Ellen>>

Then: Henry<<Henry>>, Henry<<Giselle>>, Giselle<<Bob>>
      Jack<<Ellen>>, Ellen<<Bob>>