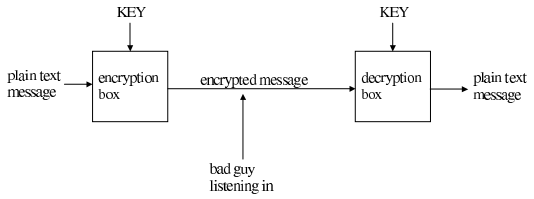
Encryption

CS 537 - Introduction to Operating Systems

Encryption

- Why use encryption?
- Can't deny access to everything
  - some files and/or information need to be transported across public lines
  - anyone can view this information
- Encryption makes the information look like gibberish to anyone but the destination
  - the destination can decrypt the message
- One important note
  - encryption algorithms should be made public

Encryption



Basic Idea

- Given a message *m* and a key *k*
  - use a function $E_k(m)$ to encrypt message
  - use a function $D_k(E_k)$ to decrypt the message
- Could use exclusive OR as the function
  - $E_k = m \oplus k$
  - $D_k = E_k \oplus k = m \oplus k \oplus k = m$
- Major problem with this
  - if *m* and E_k are known, can compute *k*

Cryptography

- Cryptography is the study of message encryption and decryption
- There exist functions E_k such that knowing E_k and *m* does not yield *k*
- $[m]^k$ means that message *m* is encrypted with key *k*
- Two major encryption algorithms
  - conventional private key encryption
  - public key encryption

Conventional Private Key

- Also called Neeham/Schroeder protocol
- Each of the two machines agree upon a private key that only they know
  - this will be different for each session
- All messages are then encrypted and decrypted with this key
- One major problem
  - how do they get the key to start with?

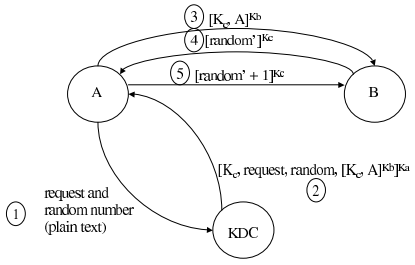
Key Distribution

- Only complete solution is "out-of-band" transmission
  - don't send it over a network
  - this is expensive
  - has other risks (watch James Bond sometime)
- Most systems actually use a network to get and transmit keys
- This requires trusting someone

Key Distribution

- Use a key distribution center (KDC)
  - everyone trusts this guy
- Every computer has a private key that only it and the KDC know
- When A wants to communicate with B, it contacts the KDC and it gives a random key, k_c, back to A
- A then transmits k_c to B
- A and B then use this key to communicate

Conventional Private Key



Conventional Private Key

1. Request a key for communication
   - random number will be used to prevent replay
2. KDC sends back the information for the following reasons
   - K_c: the key to be used by A and B
   - request: verifies whose key A is getting
   - random: prevents replay of previous session
   - $[K_c, A]^{Kb}$: encrypted message to send to B
   - encrypted with A's key so it is the only one that can decipher it
3. Send K_c to B using the encrypted response from KDC
   - B is the only one that can decode this message

Conventional Private Key

4. B replies with a random number encrypted using K_c
   - this is a challenge to A to prove it actually knows K_c
5. A replies by sending the random number + 1 back to B
   - this is A's response to prove it actually sent the original message

•At this point, A and B can now communicate with each other using the private key, K_c

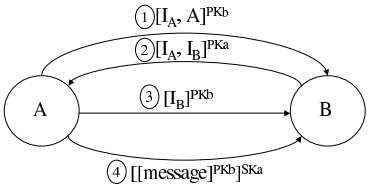
Public Key

- There exist some encryption algorithms that use a key pair
- If you encrypt with one key, you can only decrypt with the other key
- The way public key encryption works is that one of the pair is made public and the other is kept secret
  - hence, a secret key is only known by a single machine
  - everyone knows the public key

Key Distribution

- Now the public key can be sent unencrypted over the network
  - it does a bad guy no good unless he has the secret key
  - a machine will never share its secret key with anyone
- To actually communicate with someone
  - encrypt the message with their public key
  - they are the only one that can decrypt it

Public Key



Public Key

1. Send a message to B indicating a desire to communicate
   - only B can decipher this message
   - I_A is a random identifier that A created
2. B responds with a message that only A can decipher
   - by returning I_A B verifies it's really B
   - also includes I_B as a challenge to make sure A is really A
3. A responds to B's challenge with the identifier that B created
   - now both parties know the other is who they say they are

Public Key

④ All messages from A to B are encrypted with B's public key and then again with A's secret key
- anyone can decipher the message encoded with A's secret key
- all they would get is the encrypted message that only B can decode
- this is done to prevent others from injecting more into the message sent by A
- need to be careful of this because anyone could encrypt a message for B and attach it to A's message as it passes through

Public Key Cryptography

- So how does all of this math work?
- It's actually quite simple
  - to encode a message
    - $E(m) = m^e \bmod n = C$
  - to decode a message
    - $D(C) = C^d \bmod n = m$

Public Key Encryption

- How is *n* computed?
  - pick 2 big prime numbers (100 or more digits)
    - these numbers will be *p* and *q*
  - n = p x q
- Everyone knows the value of n
  - very difficult to calculate *p* and *q* given n

Public Key Encryption

- So what is the value of *d*?
  - *d* is a large random integer that is relatively prime to (p-1) x (q-1)
  - hence, the greatest common divisor of *d* and (p-1) x (q-1) is 1
- So what is the value of *e*?
  - *e* is the multiplicative inverse of:
    - d mod [(p-1) x (q-1)]
  - hence
    - e x d mod [(p-1) x (q-1)] = 1
  - like *n*, *e* is also made public

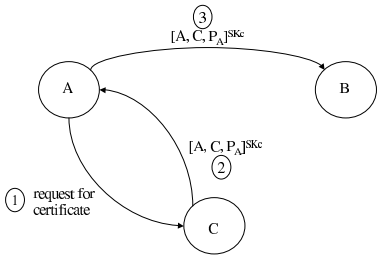
Public Key Encryption

- An example
  - let's say p = 5 and q = 7
  - n = 35
  - (p-1) x (q-1) = 24
  - several choices for *d*, we'll use d = 11
  - this means e x 11 mod 24 = 1 $\Longrightarrow$ e = 11
  - if m = 3
  - C = m^e mod n = 3^{11} mod 35 = 12
  - C^d mod n = 12^{11} mod 35 = 3 = m

Certificate

- One major problem
  - how does B guarantee that A's public key is really P_A?
  - have to trust someone again
- Assume there is a server C that both A and B trust
  - this server is someone like Verisine

Certificates



Certificates

1. Send a message to C requesting a certificate
2. C replies with the certificate
   - A is the identity of A
   - C is the identity of C
   - P_A is A's public key
   - All of this info is encrypted with C's secret K
     - anyone can decrypt this
     - used for authentication, not secrecy
3. A sends certificate to B
   - B then decodes it using C's public key
   - B now believes that it really has A's public key