Secure communication and computation

- Hardware reality: networks are insecure
  - Attacker can eavesdrop on data going over the wire
  - Attacker can modify data
  - Attacker can insert new data or messages
  - Attacker can delete data
  - Attacker can replay old messages (eavesdrop + insert)
  - Attacker can spoof identify by pretending to send a message with your identity
  - Attacker can launch “man-in-the-middle” attack: eavesdrop and delete the original message, then insert a new message that pretends to be from the original sender

Secure communication abstractions

- Abstractions
  - Confidentiality: attacker can not understand the data that is sent
  - Authentication: assure receiver that the message is from the right sender
  - Freshness: Attacker can not replay an old request
  - Availability: attacker can not deny service (we don’t know how to do this yet)
Encryption

- Encryption is the main tool used to provide secure communication
- Basic idea
  - encrypt(plain text, e-key) = cipher text
  - decrypt(cipher text, d-key) = plain text
- encrypt() and decrypt() algorithms are usually public

Symmetric key encryption

- e-key = d-key (i.e., symmetric)
  - Only sender and receiver know the key
  - Also called “secret key encryption”
- If you and I share a key, and you get a message encrypted with this key, you know:
  - I put the message there (authentication)
  - Nobody else has read the message (confidentiality)
- encrypt() / decrypt() tend to be very fast
- E.g., if I send message to registrar with a student’s grade:
  - encrypt(“B”, key)
  - Can someone modify the message?
  - How to protect against this?
How to get shared secret key?

- Hard to exchange secret key without already having a secure communication channel
- Trusted key server

Diffie-Hellman key exchange

- Public values
  - $p$ (large prime). All exponentiation carried out modulus $p$
  - $g$ (generator)
- A chooses secret: $a$
- B chooses secret: $b$
- A sends to B: $g^a$
- B sends to A: $g^b$
- A computes: $(g^b)^a = g^{ab}$
- B computes: $(g^a)^b = g^{ab}$
- Eavesdropper knows $g^a$ and $g^b$, but not $g^{ab}$

- Any problems with this?
Public-key (asymmetric) encryption

- e-key != d-key
- Typically, encrypt() == decrypt(). We’ll call this crypt().
  - crypt(clear text, e-key) = cipher-text-1
  - crypt(cipher-text-1, d-key) = clear text
  - crypt(clear text, d-key) = cipher-text-2
  - crypt(cipher-text-2, e-key) = clear text
- Note that
  - cipher-text-1 != cipher-text-2
  - crypt(cipher-text-2, d-key) != clear text
  - crypt(cipher-text-1, e-key) != clear text
- One key (say, e-key) is called the “public key”
  - Everyone should know the value of everyone’s public key
- The other key (say, d-key) is called the “private key”
  - Only the sender should know his/her own private key
- Difficult to guess private key, even if you know the public key, crypt(), and lots of encrypted pairs

Using public-key encryption

- Authenticate sender
  - “from pmchen” crypt(message, pmchen-private)
  - Anyone can read this message (no confidentiality)
  - Only pmchen can generate this message; others can verify that pmchen generated the message by decrypting with pmchen-public
  - Why include “from pmchen”?
  - This is called a “digital signature”
- Ensure confidentiality
  - crypt(message, receiver-public)
  - Anyone can send this message (no authentication)
  - Only receiver can read it
- Ensure authentication and confidentiality
  - crypt(“from pmchen”, crypt(message, pmchen-private), receiver-public)
  - Only receiver can read it; only pmchen can generate it
  - Does reversing the order of encryption work?
    - “from pmchen” crypt(crypt(message, receptor-public), pmchen-private)
Public-key encryption

• Used in
  – SSL (secure sockets layer)
  – ssh (secure shell)
  – pgp (secure mail)

• Problems
  – Computationally expensive
    • Solve by using public-key encryption to exchange a short-lived symmetric key (called a session key)
  – How to change my public key?
  – How to trust the authenticity of published public keys?

Man-in-the-middle attack

• A wants to communicate with B
  – A and B must learn each other’s public keys (A-public, B-public)
  – Villain intercepts communication; replaces A-public with V-public1, and replaces B-public with V-public2

  – A signs message but sends with V-public2
    • crypt(“from A” crypt(message, A-private), V-public2)
  – Villain can read this message
  – Villain can modify message and generate
    • crypt(“from A”, crypt(message’, V-private1), B-public)
  – When B gets this, it will think message’ is authentically from A and confidential 😊
How to authenticate the published public key?

- pgp and ssh: verify the “fingerprint” of a public key via the telephone or trusted web server
- SSL/TLS example: your web browser wants to communicate with bank. You want to ensure that only bank can see your messages (confidentiality); bank wants to ensure that messages really are from you (authentication)
- Step 1: assure you that your messages can only be read by bank
  - Bank has public key, but how to learn this securely?
  - Certification authority (e.g., verisign) vouches for the authenticity of bank’s public key
    - Verisign generates certificate for bank: crypt(“bank’s public key is X”, verisign-private)
    - Bank sends you this certificate. You decrypt with verisign-public and see the verisign is vouching that bank’s public key is X
    - Use bank’s public key to set up shared session key
    - Problems?

How to authenticate the published public key?

- Step 2: assure bank that you are who you say you are
  - Most people wouldn’t have a certified public key from verisign
  - Send your password (encrypted with secret session key)
  - Bank decrypts with your password. This verifies that this session key is really from you.
Replay attacks

- Example using symmetric-key encryption (same is possible with public-key encryption)
  - I send message to bank: encrypt("transfer $100 to U-M", key)
  - Evil administrator eavesdrops and saves the encrypted message, then replays it later. Bank transfers another $100 to U-M.

- How to defend against this attack?

Picking a nonce

- Project 4: nonce is <session, sequence> tuple

- How to pick a nonce that doesn’t require anyone to keep any state (but may only provide probabilistic guarantee of freshness)?
General security

• Hardware reality
  – Collection of processors, memory, disks, network interfaces
  – Can be used by anyone to do anything (or by nobody to do nothing)
• OS abstraction: **controlled** access to hardware
• What primitives does the hardware provide for controlling access?

• OS will provide two abstractions on top of these primitives:
  – Identity (authentication): who are you?
  – Security policy (authorization): what are you allowed to do?

• Hardware already provides these two abstractions in a primitive way:

---

**Authentication: who are you?**

• Authentication is the process of you proving your identity to the operating system
  – It may also include the operating system proving its identity to you
• Many ways to do authentication
  – Password (something you know)
  – Physical token (something you have)
  – Physical token + password
  – Biometric
• How is authentication done today?

• How to improve this?
Authorization: what can you do?

- Access control matrix

<table>
<thead>
<tr>
<th></th>
<th>file1</th>
<th>file2</th>
<th>file3</th>
</tr>
</thead>
<tbody>
<tr>
<td>user1</td>
<td>rw</td>
<td>rw</td>
<td>rw</td>
</tr>
<tr>
<td>user2</td>
<td>-</td>
<td>r</td>
<td>rw</td>
</tr>
</tbody>
</table>

- Two approaches for storing this information: access control lists (ACLs) and capabilities

Access control lists

- At each object, store a list of who can access the object and in what ways they can access it
- E.g., at file 2, store <user1 rw; user2 r>
- On each access, check that the user (whose identity is authenticated at login time) has permission to access the file
- For convenience, create a group of several users
- To attack ACL system, control code that runs under someone else’s identity. E.g., sendmail runs as root, so subvert sendmail and make it run attacker’s code.
Capabilities

• At each user, store a list of objects the user is allowed to access, and how they are allowed to access it
• E.g., at user 2, store <file2, r> and <file3 rw>
• On each access, check that the user has a capability for this type of access
• Possession of the capability gives the power to access the file
  – Similar to car keys
• How to attack a capability system?

• Solution?

• How to handle revocation
  – In an ACL system
  – In a capability system