Security

• Overview
  - Security Goals
  - The Attack Space

• Security Mechanisms
  - Introduction to Cryptography
  - Authentication
  - Authorization
  - Confidentiality

• Case Studies

Security Today...
Typical Attacks: Penetration Attempts

- Two basic forms:
  - completely bypass authentication mechanism
  - obtain information or alter the system so as to enter system as authorized user
- Attempts:
  - Wire tapping (active vs. passive)
  - Trial and error
  - Browsing
  - Trap doors
  - Trojan horse
  - Searching of waste

Typical Attacks: Man-In-The-Middle
Typical Attacks: Masquerading

- Passive tapping
  - Listen to communication without altering contents.
- Active wire tapping
  - Modify data being transmitted
  - Example:

```
user! \rightarrow intruder \rightarrow server!

Intruder! takes over identity of user (masquerading)
```

Man-In-The-Middle: Example

- Passive tapping
- Active wire tapping
  - Modify data being transmitted
  - Example:
Security Threats

- **Information Disclosure:**
  - unauthorized dissemination of information
  - result of theft or illegal action of who has access to information

- **Information Destruction:**
  - loss of internal data structures
  - loss of stored information
  - information may be destroyed without being disclosed

- **Unauthorized Use of Service:**
  - bypass system accounting policies
  - unauthorized use of some proprietary services

- **Denial of Service:**
  - prevent an authorized user from utilizing the system's services in a timely manner

Security Goals

- **Authentication** of Alice (the client)
- **Authorization** of request from Alice
- **Confidentiality** (e.g. protect the content of request)
- **Accountability** (non-repudiation)
- **Availability**
# Security: Systems Overview

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Authentication</th>
<th>Authorization</th>
<th>Confidentiality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitives</td>
<td>sign()</td>
<td>Access control lists, Capabilities, &quot;magic cookies&quot;</td>
<td>encrypt()</td>
</tr>
<tr>
<td></td>
<td>verify()</td>
<td></td>
<td>decrypt()</td>
</tr>
<tr>
<td>Cryptography</td>
<td></td>
<td></td>
<td>cyphers and hashes</td>
</tr>
</tbody>
</table>

# Cryptography

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Authentication</th>
<th>Authorization</th>
<th>Confidentiality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitives</td>
<td>sign()</td>
<td>Access control lists, Capabilities, &quot;magic cookies&quot;</td>
<td>encrypt()</td>
</tr>
<tr>
<td></td>
<td>verify()</td>
<td></td>
<td>decrypt()</td>
</tr>
<tr>
<td>Cryptography</td>
<td></td>
<td></td>
<td>cyphers and hashes</td>
</tr>
</tbody>
</table>

**Cryptography:**
- Closed-Design vs. Open-Design Cryptography
- Symmetric Encryption
- Asymmetric ("Public-Key") Encryption
**Closed-Design Cryptography**

```
"Alice"  "crypto box" (closed)  "de-crypto box" (closed)  "Bob"
```

**Open-Design Cryptography**

```
write  message m

key exchange

encryption key k

encryption algorithm E

encryption algorithm D

decryption key k

read  message m
```
Encryption

- Encryption algorithm consists of
  - Set of $K$ keys
  - Set of $M$ Messages
  - Set of $C$ ciphertexts (encrypted messages)
  - A function $E : K \rightarrow (M \rightarrow C)$. That is, for each $k \in K$, $E(k)$ is a function for generating ciphertexts from messages.
    - Both $E$ and $E(k)$ for any $k$ should be efficiently computable functions.
  - A function $D : K \rightarrow (C \rightarrow M)$. That is, for each $k \in K$, $D(k)$ is a function for generating messages from ciphertexts.
    - Both $D$ and $D(k)$ for any $k$ should be efficiently computable functions.

- An encryption algorithm must provide this essential property:
  - Given a ciphertext $c \in C$, a computer can compute $m$ such that $E(k)(m) = c$, only if it possesses $D(k)$.
  - Thus, a computer holding $D(k)$ can decrypt ciphertexts to the plaintexts used to produce them, but a computer not holding $D(k)$ cannot decrypt ciphertexts.
  - Since ciphertexts are generally exposed (for example, sent on the network), it is important that it be infeasible to derive $D(k)$ from the ciphertexts.

Symmetric Encryption

- Same key used to encrypt and decrypt
  - $E(k)$ can be derived from $D(k)$, and vice versa

- Examples:
  - Data Encryption Standard (DES)
  - Triple-DES
  - Advanced Encryption Standard (AES)
  - Twofish
Symmetric Encryption: Caesar Cipher

![Caesar Cipher Image]

MERRY CHRISTMAS

PHUUB FKULVWPDV

Symmetric Encryption: Jefferson’s Wheel Cipher

- **Sender:**
  - Assemble wheels in some (secret) order.
  - Align message on one line.
  - Choose any of the other lines as ciphertext.

- **Receive:**
  - Assemble wheels in same secret order.
  - Align ciphertext on one line.
  - Look for meaningful message on other lines.

Monticello Web Site: www.monticello.org/reports/interests/wheel_cipher.html
Symmetric Encryption: XOR

Symmetric Encryption: DES (Data Encryption Standard)
Asymmetric Encryption

Keys must be different

Asymmetric Encryption (cont.)

- Public-key encryption based on each user having two keys:
  - public key - published key used to encrypt data
  - private key - key known only to individual user used to decrypt data
- Must be an encryption scheme that can be made public without leaking the decryption scheme
  - Most common is **RSA block cipher**
  - Efficient algorithms exist for testing whether or not a number is prime
  - No efficient algorithm is known for finding the prime factors of a number
RSA (cont)

- If it is computationally infeasible to derive \( D(k_d, N) \) from \( E(k_e, N) \), \( E(k_e, N) \) need not be kept secret and can be widely disseminated
  - \( E(k_e, N) \) is the public key
  - \( D(k_d, N) \) is the private key
  - \( N \) is the product of two large, randomly chosen prime numbers \( p \) and \( q \) (for example, \( p \) and \( q \) are 512 bits each)
  - Encryption algorithm is \( E(k_e, N)(m) = m^{k_e} \mod N \), where \( k_e \) satisfies \( k_e k_d \mod (p-1)(q-1) = 1 \)
  - The decryption algorithm is then \( D(k_d, N)(c) = c^{k_d} \mod N \)

RSA: Example

- Make \( p = 7 \) and \( q = 13 \)
- We then calculate \( N = 7 \cdot 13 = 91 \) and \( (p-1)(q-1) = 72 \)
- We next select \( k_e \) relatively prime to 72 and < 72, yielding 5
- Finally, we calculate \( k_d \) such that \( k_e k_d \mod 72 = 1 \), yielding 29
- We now have our keys
  - Public key, \( (k_e, N) = (5, 91) \)
  - Private key, \( (k_d, N) = (29, 91) \)
- Encrypting the message 69 with the public key results in the ciphertext 62
  - 69\(^{5} \mod 91 = 62 \)
- Ciphertext can be decoded with the private key
  - 62\(^{29} \mod 91 = 69 \)
- Public key can be distributed in clear text to anyone who wants to communicate with holder of public key
RSA in Practice...

{m}^{k_{B_{pub}}} : A encrypts message with B's public key.

{m}^{k_{A_{priv}}} : A signs a message with A's private key.

Symmetric vs. Asymmetric Encryption

- **Symmetric** cryptography based on simple transformations
- **Asymmetric** based on time consuming mathematical functions
  - Asymmetric much more compute intensive
  - Typically not used for bulk data encryption
  - Used, instead, for short plaintexts, for example symmetric keys.
Key Exchange: Diffie Hellman

Step 1  Alice and Bob agree on a large prime \( m \) and "primitive root" \( g \mod m \).
Note: \( m \) and \( g \) need not be secret.

Step 2  Alice and Bob privately pick random integer \( x \) and \( y \), respectively.

Step 3  Alice and Bob exchange \( X = g^x \mod m \) and \( Y = g^y \mod m \), respectively.

Step 4  Alice and Bob privately compute \( k = Y^x \mod m \) and \( k' = X^y \mod m \), respectively.

\[
k = k' \mod m, \text{ since } k' = X^y = (g^x)^y = g^{xy} = Y^x = k \mod m
\]

Scheme can be broken if Eve succeeds to solve the equation
\[
g^x = X \mod m
\]
for \( x \), the "discrete logarithm base \( g \) of \( X \) modulo \( m \)."
Authentication

1. Who is making the request?
2. Is the received message the same as the sent message?
3. How do I build an audit trail?

1. Authentication
2. Message Integrity
3. Accountability / Non-Repudiation

Message Integrity

- Message Integrity can be guaranteed through Error-Detection Code. (e.g. cryptographic hash)

Message Integrity ≈ Authenticity ≈ Confidentiality
Authentication: Model

- Symmetric Encryption ($k_1 = k_2$):
  - $A(m)$ is "message authenticator"
- Asymmetric Encryption ($k_1 \neq k_2$):
  - $A(m)$ is "signature"
  - Example: $A(m) = \{\text{Hash}(m)\}^{k_{\text{priv}}}$
  - Cryptographically secure hash:
    - $\text{Prob}(\text{Hash}(m) = \text{Hash}(m'))$ is very low ("low collision prob.")
    - SHA1, SHA256, etc.

Authentication: Sign() and Verify()

- Algorithm components
  - A set $K$ of keys
  - A set $M$ of messages
  - A set $A$ of authenticators
  - A function $S : K \rightarrow (M \rightarrow A)$
    - That is, for each $k \in K$, $S(k)$ is a function for generating authenticators from messages
    - Both $S$ and $S(k)$ for any $k$ should be efficiently computable functions
  - A function $V : K \rightarrow (M \times A \rightarrow \{\text{true}, \text{false}\})$. That is, for each $k \in K$, $V(k)$ is a function for verifying authenticators on messages
    - Both $S$ and $V(k)$ for any $k$ should be efficiently computable functions
**RSA in Practice...**

1. **{m}kApub:** A encrypts a message with B's public key.
2. **{m}{kApub}kBpub:** B decrypts a message with B's private key.
3. **{m}kApriv:** A signs a message with A's private key.
4. **{m}{kApriv}kApub:** B verifies a message with A's public key.

---

**Authentication (Cont.)**

- For a message $m$, a computer can generate an authenticator $a \in A$ such that $V(k)(m, a) = \text{true}$ only if it possesses $S(k)$.
- Thus, computer holding $S(k)$ can generate authenticators on messages so that any other computer possessing $V(k)$ can verify them.
- Computer not holding $S(k)$ cannot generate authenticators on messages that can be verified using $V(k)$.
- Since authenticators are generally exposed (for example, they are sent on the network with the messages themselves), it must not be feasible to derive $S(k)$ from the authenticators.
Key Distribution Problem

- Q: How does Bob learn Alice's key?
  - Q.1: Alice's public key?
  - Q.2: Alice's shared key?

Key Distribution: Certificates

1. \( \{m, \text{Sign}(m, k_{Apriv})\} \)

2. \( \{\text{Alice}?!\} \)

3. \( \{m="k_{Apub}=X", \text{Sign}(m, k_{Cpriv})\} \)
Establishing a Secure Channel

1. Authenticate user using public key encryption.
2. Use shared-key encryption for communication.

Q: How to Exchange Shared Key?


1. \( \{A, B\} \)
2. \( \{A, k_{A}^{pub}, TS\}^{k_{C}^{priv}} \) (certificates)
3. \( \{A, k_{A}^{pub}, TS\}^{k_{C}^{priv}} \) (certificate)
   \( \{\{k_{AB}, TS\}^{k_{A}^{priv}}\}^{k_{B}^{pub}} \) (proposed key)
4. \( \{data, TS\}^{k_{AB}} \)

A Closer Look ...

[Abadi 1994]

1. \( \{A, B\} \)
2. \( \{A, k_{A}^{pub}, TS\}^{k_{C}^{priv}} \) (certificates)
3. \( \{A, k_{A}^{pub}, TS\}^{k_{C}^{priv}} \) (certificate)
   \( \{\{k_{AB}, TS\}^{k_{A}^{priv}}\}^{k_{B}^{pub}} \) (proposed key)
4. \( \{A, k_{A}^{pub}, TS\}^{k_{C}^{priv}} \) (certificate)
   \( \{\{k_{AB}, TS\}^{k_{A}^{priv}}\}^{k_{C}^{pub}} \) (proposed key)
5. \( \{data\}^{k_{AB}} \)

Problem:
Message 3 does not specify who it is intended to.
This opens door for impersonation attacks.
SSL

- Applications: HTTP, IMAP, FTP, etc...

- Client and server negotiate symmetric key that they will use for the length of the data session.

- Two phases in SSL:
  - Phase 1: Connection Establishment
  - Phase 2: Data Transfer

---

SSL: Connection Establishment

- **Step 1:** Client sends request to server, containing
  - SSL version; connection preferences; nonce (i.e. some random number)

- **Step 2:** Server chooses among preferences, and sends reply, containing
  - Chosen preferences; nonce; public-key certificate
  - Public-key certificate is a public key that has been digitally signed by a trusted authority.

- **Step 3:** Client can use certification authority's public key to check authenticity of server's public key.

- **Step 4:** Server can request public key of client and verify it similarly (optional)

- **Step 5:** Client chooses random number (premaster secret), encrypts it with server's public key, and sends it to server.

- **Step 6:** Both parties compute session key (used during data transfer) based on premaster secret and the two nonces.
  - Note: At no point is the session key transferred between client and server.
SSL: Data Transfer

- Messages are fragmented into 16kB portions.
- Each portion is optionally compressed.
- A **Message Authentication Code (MAC)** is appended
  - MAC is a hash derived from plaintext, two nonces, and pre-master secret
- Plaintext and MAC are encrypted using the symmetric key constructed during connection establishment.