CSCE 313 Introduction to Computer Systems

Instructor: Dr. Guofei Gu

http://courses.cse.tamu.edu/guofei/csse313

Security

- Overview
  - Security Goals
  - Security Threats
- Security Mechanisms
  - Introduction to Cryptography
  - Confidentiality
  - Authentication
- Case Studies
Security Goals

- **Confidentiality** (e.g. protect the content of request)
- **Integrity**
- **Authentication** of Alice (the client)
- **Others**
  - Authorization of request from Alice
  - 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><code>sign()</code></td>
<td><code>Access control lists</code></td>
<td><code>encrypt()</code></td>
</tr>
<tr>
<td></td>
<td><code>verify()</code></td>
<td><code>Capabilities</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>&quot;magic cookies&quot;</code></td>
<td></td>
</tr>
<tr>
<td>Cryptography</td>
<td></td>
<td></td>
<td><code>decrypt()</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>cyphers and hashes</code></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></td>
<td>encrypt()</td>
</tr>
<tr>
<td></td>
<td>verify()</td>
<td>Access control lists</td>
<td>decrypt()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Capabilities &quot;magic cookies&quot;</td>
<td></td>
</tr>
<tr>
<td>Cryptography</td>
<td></td>
<td>cyphers and hashes</td>
<td></td>
</tr>
</tbody>
</table>

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

## Closed-Design Cryptography

```
"Alice" → "crypto box" (closed) → "de-crypto box" (closed) → "Bob"
```
Open-Design Cryptography

Encryption

- Encryption algorithm consists of
  - Set of K keys
  - Set of M Messages
  - Set of C ciphertexts (encrypted messages)
  - A function E : K → (M → C). That is, for each k ∈ 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 → (C → M). That is, for each k ∈ 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:
  \[ \text{Given a ciphertext } c \in C, \text{ a computer can compute } m \text{ such that } E(k)(m) = c \text{ only if it possesses } D(k). \]
  - 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.
Computational Difficulty

- Algorithm needs to be efficient.
  - Otherwise only short keys can be used.
- Most schemes can be broken: depends on $$$.
  - E.G. Try all possible keys.
- Longer key is often more secure:
  - Brute-force cryptanalysis: twice as hard with each additional bit.
- Cryptanalysis tools:
  - Special-purpose hardware.
  - Parallel machines.
  - Internet coarse-grain parallelism.

Secret Key vs. Secret Algorithm

- Secret algorithm: additional hurdle
- Hard to keep secret if used widely:
  - Reverse engineering, social engineering
- Commercial: published
  - Wide review, trust
- Military: avoid giving enemy good ideas
Brute Force Attacks

- Number of encryption/sec: 1 million to 1 billion/sec
- 56-bit key broken in 1 week with 120,000 processors ($6.7m)
- 56-bit key broken in 1 month with 28,000 processors ($1.6m)
- 64-bit key broken in 1 week with $3.1 \times 10^7$ processors ($1.7b$)
- 128-bit key broken in 1 week with $5.6 \times 10^{26}$ processors

Types of Cryptography

- Hash functions: no key
- Secret key (Symmetric) cryptography: one key
- Public key (Asymmetric) cryptography: two keys - public, private
Secret Key Cryptography

- Same key is used for encryption and decryption
  - Symmetric cryptography
- Ciphertext approximately the same length as plaintext
- Substitution codes, DES, IDEA
- Message transmission:
  - Agree on key (but how?)
  - Communicate over insecure channel
- Secure storage: crypt

Secret Key Cryptography (Cont’d)

- Strong authentication: prove knowledge of key without revealing it:
  - Send challenge \( r \), verify the returned encrypted \( \{r\} \)
  - Fred can obtain chosen plaintext, ciphertext pairs
    - Challenge should chosen from a large pool
- Integrity check: fixed-length checksum for message
  - Send MIC along with the message
Public Key Cryptography

- Asymmetric cryptography
- Invented/published in 1975
- Two keys: private (d), public (e)
  - Encryption: public key; Decryption: private key
  - Signing: private key; Verification: public key
- Much slower than secret key cryptography

Public Key Cryptography (Cont’d)

- Data transmission:
  - Alice encrypts \( m_a \) using \( e_b \), Bob decrypts to \( m_a \) using \( d_b \).
- Storage:
  - Can create a safety copy: using public key of trusted person.
- Authentication:
  - No need to store secrets, only need public keys.
  - Secret key cryptography: need to share secret key for every person to communicate with.
Digital signatures
- Encrypt hash $h(m)$ with private key
  - Authorship
  - Integrity
  - Non-repudiation: can't do with secret key cryptography

Hash Algorithms
- Message digests, one-way transformations
- Length of $h(m)$ much shorter than length of $m$
- Usually fixed lengths: 48-128 bits
- Easy to compute $h(m)$
- Given $h(m)$, no easy way to find $m$
- Computationally infeasible to find $m_1, m_2$ s.t. $h(m_1) = h(m_2)$
- Example: $(m+c)^2$, take middle $n$ digits
Hash Algorithms (Cont’d)

- **Password hashing**
  - Doesn’t need to know password to verify it
  - Store \( h(p+s) \), \( s \) (salt), and compare it with the user-entered \( p \)
  - Salt makes dictionary attack less convenient

- **Message integrity**
  - Agree on a password \( p \)
  - Compute \( h(p|m) \) and send with \( m \)
  - Doesn’t require encryption algorithm, so the technology is exportable

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

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

1. Pick random number $k_e$, relative prime to $(p-1)(q-1)$
2. Compute $k_d$, such that $k_e k_d \mod (p-1)(q-1) = 1$

- Make $p = 7$ and $q = 13$
- We then calculate $N = 7 \times 13 = 91$ 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...

\[ (m)^{k_{\text{pub}}} : \text{A encrypts message with B's public key.} \]
\[ (m)^{k_{\text{priv}}} : \text{A signs a message with A's private key.} \]

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.

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

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**

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Authentication</th>
<th>Authorization</th>
<th>Confidentiality</th>
</tr>
</thead>
</table>
| Primitives    | \textit{sign()}  \
|               | \textit{verify()}  \
|               | Access control lists 
|               | Capabilities "magic cookies" |
|               | encrypt()  \
|               | decrypt()  |
| Cryptography  | cyphers and hashes |
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: $A(m) = \{\text{Hash}(m)\}^{kA}_{\text{priv}}$
  - 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 $V : K \rightarrow (M \times A \rightarrow \{\text{true, false}\})$. 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...

\[
\begin{align*}
\{m\}^{k_{A_{\text{pub}}}} & : \text{A encrypts message with B’s public key.} \\
\{\{m\}^{k_{A_{\text{pub}}}}\}^{k_{B_{\text{priv}}}} & : \text{B decrypts message with B’s private key.} \\
\{m\}^{k_{A_{\text{priv}}}} & : \text{A signs a message with A’s private key.} \\
\{\{m\}^{k_{A_{\text{priv}}}}\}^{k_{A_{\text{pub}}}} & : \text{B verifies a message with A’s public key.}
\end{align*}
\]

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. \( \{m, \text{Sign}(m, k_{Apriv})\} \)
   - \( k_{Apriv} \) is Alice’s private key.

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

3. \( \{m=’k_{Apub}=X’, \text{Sign}(m, k_{Cpriv})\} \)
   - \( k_{Cpriv} \) is the private key of the Certificate Authority.

Certificate Authority

- VeriSign
- Comodo
- GoDaddy
- Others

2007 Market Share (source: Secure Space)
Establishing a Secure Channel

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

Q: How to Exchange Shared Key?

Denning-Sacco Protocol (1982)

A Closer Look ... [Abadi 1994]

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.

User Authentication

- Crucial to identify user correctly, as protection systems depend on user ID
- User identity most often established through passwords, can be considered a special case of either keys or capabilities
  - Also can include something user has and/or a user attribute
- Passwords must be kept secret
  - Frequent change of passwords
  - Use of “non-guessable” passwords
  - Log all invalid access attempts
- Passwords may also either be encrypted or allowed to be used only once
Implementing Security Defenses

- **Defense in depth** is most common security theory – multiple layers of security
- Security policy describes what is being secured
- Vulnerability assessment compares real state of system / network compared to security policy
- Intrusion detection endeavors to detect attempted or successful intrusions
  - **Signature-based** detection spots known bad patterns
  - **Anomaly detection** spots differences from normal behavior
    - Can detect zero-day attacks
  - *False-positives* and *false-negatives* a problem
- **Virus protection**
- Auditing, accounting, and logging of all or specific system or network activities

Firewalling to Protect Systems and Networks

- A network firewall is placed between trusted and untrusted hosts
  - The firewall limits network access between these two security domains
- Can be tunneled or spoofed
  - Tunneling allows disallowed protocol to travel within allowed protocol (i.e. telnet inside of HTTP)
  - Firewall rules typically based on host name or IP address which can be spoofed
- **Personal firewall** is software layer on given host
  - Can monitor / limit traffic to and from the host
- **Application proxy firewall** understands application protocol and can control them (i.e. SMTP)
- **System-call firewall** monitors all important system calls and apply rules to them (i.e. this program can execute that system call)
Example: Windows XP

- Security is based on user accounts
  - Each user has unique security ID
  - Login to ID creates security access token
    - Includes security ID for user, for user’s groups, and special privileges
    - Every process gets copy of token
    - System checks token to determine if access allowed or denied
- Uses a subject model to ensure access security. A subject tracks and manages permissions for each program that a user runs
- Each object in Windows XP has a security attribute defined by a security descriptor
  - For example, a file has a security descriptor that indicates the access permissions for all users