Introduction

- Security in distributed systems has two parts
  - Secure channel, more specifically, authentication, message integrity, and confidentiality issues.
  - Authorization, which ensures that a process gets only those access rights to the resources it is entitled to. Techniques include access control.

Security Threats

- Interception:
  - unauthorized party has gained access to the data. Data are illegally got or copied.
- Interruption:
  - a file is corrupted or lost since someone maliciously makes data or service inaccessible.
- Modification:
  - unauthorized changing of the data, e.g., intercept and change the transmitted data.
- Fabrication:
  - additional data or activity are generated that would normally not exist, e.g., add a password entry.

Security Policy and Mechanisms

- Security policy describes precisely which actions the entities in a system are allowed to take and which ones are prohibited.
  - Entities include users, services, data, machines.
- Security mechanisms are used to enforce the policy
  - Encryption: ensure confidentiality.
  - Authentication: verify the claimed identity of the user.
  - Authorization: check whether the client is authorized to perform the requested action.
  - Auditing: to trace which client accessed what, in which way.
**Cryptography Terminology**

- Plaintext: text without encryption.
- Ciphertext: encrypted text.
- Intruder: the enemy who hears or even modify the ciphertext.
- Cryptology: the art of devising ciphers (cryptography) and breaking them (cryptanalysis) is collectively known as cryptology.
- Eavesdropping: obtaining copies of messages without authority.
- Message tampering: intercepting messages and altering their contents before passing them on to the intended recipient.
- Replaying: storing intercepted messages and sending them at a later date.

---

**The Encryption Model**

- Symmetric cryptosystem, the same key is used to encrypt and decrypt
  \[ P = D_K(E_K(P)) \]
- Asymmetric cryptosystem: the keys for encryption and decryption are different
  \[ P = D_K(D_{K^D}(E_{K^E}(P))) \]

---

**Cryptosystem**

- Each letter is replaced by another letter.
  - E: \( P \rightarrow (P+3) \) modular 26 where \( 0 \leq P \leq 25 \).
  - Plaintext "Julius" is transformed into "mxolxv".
- Improvement is to use a table map some letter to other letters.
  - Plain text: a (Q), b (W) c(E) k(A) t(Z)
  - Attack is transformed into QZZQEA
- For 26 letters, there are 26! Possible keys. It is almost impossible to break?

---

**Substitution Ciphers**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( K_{A,B} )</td>
<td>Secret key shared by ( A ) and ( B )</td>
</tr>
<tr>
<td>( K^P_A )</td>
<td>Public key of ( A )</td>
</tr>
<tr>
<td>( K^D_A )</td>
<td>Private key of ( A )</td>
</tr>
</tbody>
</table>
Breaking Techniques

- Use the statistical properties of natural language.
- In English, e, t, o, a, n, i are most commonly used letters. th, in, er, re, and an are most commonly used diagrams.
- Counting the relative frequencies of all letters in the ciphertext.
  - Tentatively assign e to the most common one.
  - If you have a tXe, it suggests that X is h.
- Guess a probable word or phase
  - Messages in an accounting firm may have “financial”. Then try to find a match.

Symmetric key crypto

DES: Data Encryption Standard
- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- How secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase was decrypted (brute force) in 4 months
  - making DES more secure:
    - use three keys sequentially (3-DES) on each datum
AES: Advanced Encryption Standard:
- new NIST standard, 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

DES

Public-Key Algorithm

- Although DES is secure, but how to distribute the key?
- In 1976, Diffie and Hellman at Stanford proposed the public key algorithm, which has three requirements.
  - D(E(p)) = P
  - It is almost impossible to deduce D from E.
  - E cannot be broken by a chosen plaintext attack.
- Each one has two keys: the public key and the private key.
  - Asymmetric cryptography.
RSA (Rivest, Shamir, Adelson): Choosing keys

1. Choose two large prime numbers $p, q$. (e.g., 1024 bits each)
2. Compute $n = pq, \ z = (p-1)(q-1)$
3. Choose $e$ (with $e < n$) that has no common factors with $z$. ($e, z$ are “relatively prime”).
4. Choose $d$ such that $ed-1$ is exactly divisible by $z$. (in other words: $ed \mod z = 1$).
5. Public key is $(n, e)$. Private key is $(n, d)$.

RSA: Encryption, decryption

0. Given $(n, e)$ and $(n, d)$ as computed above
1. To encrypt bit pattern, $m$, compute $c = m^e \mod n$ (i.e., remainder when $m^e$ is divided by $n$)
2. To decrypt received bit pattern, $c$, compute $m = c^d \mod n$ (i.e., remainder when $c^d$ is divided by $n$)

RSA example:


- $e=5$ (so $e, z$ relatively prime).
- $d=29$ (so $ed-1$ exactly divisible by $z$).

<table>
<thead>
<tr>
<th>encrypt</th>
<th>m</th>
<th>$m^e$</th>
<th>c = $m^e \mod n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>12</td>
<td>1524832</td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>decrypt</th>
<th>c</th>
<th>$c^d$</th>
<th>$m = c^d \mod n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>481968572106750915091411825223071697</td>
<td>12</td>
<td>l</td>
</tr>
</tbody>
</table>

Useful number theory result: If $p, q$ prime and $n = pq$, then: $x^y \mod n = x^y \mod (p-1)(q-1) \mod n$

$(m^e \mod n)^d \mod n = m^{ed} \mod n$

= $m^{ed \mod (p-1)(q-1)} \mod n$  
(using number theory result above)

= $m^1 \mod n$  
(since we chose $ed$ to be divisible by $(p-1)(q-1)$ with remainder 1)

= $m$
RSA: another important property

The following property will be very useful later:

\( K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m)) \)

use public key first, followed by private key

use private key first, followed by public key

**Result is the same!**

Authentication Protocols

- Authentication deals with the questions of whether or not you are actually communicating with a specific process.
- Authorization is concerned with what that process is permitted to do.
- For example, a client contacts a file server and say “I am Scott’s process and I want to delete the file cookbook.old”.
  - Is this actually Scott’s process?
  - Is Scott allowed to delete cookbook.old?
- Challenge-response protocol: one party sends a random number to the other, who then transforms it in a special way and then returns the result.
  - A, B are the identities of Alice and Bob.
  - R_i’s are the challenges, where the subscript identifies the challenger.
  - K_i’s are keys, where i indicates the owner, K_s is the session key. A session key is used in the upcoming conversation.

Two-Way Authentication

Reflection Attack

- General rules
  - Have the initiator prove who she is before the responder has to.
  - Have the initiator and the responder use different keys for proof, even if this means having two shared keys.
  - Have the initiator and responder draw their challenges from different sets. For example, the initiator must use even numbers and the responder must use odd numbers.
Authentication Using KDC

- How to get the shared secret key?
  - By phone, arrange a meeting
- Key distribution center (KDC) approach.
  - Each user has a single key shared with the KDC. Authentication and session key management goes through the KDC.
- One problem of the first approach is that Alice may talk to Bob before Bob got the key.
- Let Alice notify Bob of the key, by the message $K_{B,KDC}(K_{A,B})$, which is known as a ticket.

Needham-Schroeder Authentication

Attacks

- A nonce is a random number that is used only once. By including $R_{A1}$ in $m2$, Alice knows that $m2$ is sent as a response to $m1$, not reply.
- Without nonce, suppose Chuck has stolen Bob’s old key, $K_{B,KDC}^{old}$, also intercepts $K_{A,KDC}(B,K_{A,B}, K_{B,KDC}^{old}(A,K_{A,B}))$, it fools Alice.
- In $m2$, without $B$ is also a problem. Chuck can replace $B$ by $C$ in $m1$. The KDC thinks that Alice wants to talk to Chuck and responds accordingly. When Alice contacts Bob, Chuck gets the message.
- In $m4$, $R_{A1}$ is not necessary.
- If Chuck got an old key $K_{A,B}$, he could reply $m3$, and get Bob to setup a channel. Solution is to relate $m3$ and $m1$. 
Kerberos

- A real authentication system is Kerberos, which is based on a variant of Needham-Schroeder.
  - Designed in MIT to allow workstation users to access network resources in a secure way.
  - Assume the clock is fairly-well synchronized.
- It has three servers in addition to Alice (the client)
  - Authentication server (AS): verifies users during login.
  - Ticket-granting server (TGS): issues proof of identity tickets.
  - Bob (the server): do the requested work.
- The client can access the network in a secure way, and the password never has to go over the network.
**Digital Signatures**

- Need something to replace handwritten signatures.
- Digital signature has three requirements
  - The receiver can verify the claimed identity of the sender.
  - The sender cannot later deny the contents of the message.
  - The receiver cannot possibly modify the message.
- Use a one-way hash function that takes an arbitrarily long piece of plaintext and from it computes a fixed-length bit string. This hash function is called a message digest
  - Given \( m \), it is easy to compute \( H(m) \).
  - Given \( H(m) \), it is effectively impossible to find \( m \).
  - No one can generate two messages that have the same message digest.
- MD5, a hash function for computing a 128-bit fixed length message digest from an arbitrary length binary input string.

---

**Secure e-mail**

- Alice wants to send confidential e-mail, \( m \), to Bob.
  - \( K_S \equiv @ \)
  - \( K_S(m) \)
  - \( K_B(K_S) \)
  - \( K_B(K_S) - K_B(K_S) \)
  - \( K_B(K_S) \)
  - \( K_B(K_S) + K_B(K_S) \)
  - \( m \)
- Alice:
  - generates random symmetric private key, \( K_S \).
  - encrypts message with \( K_S \) (for efficiency)
  - also encrypts \( K_S \) with Bob's public key.
  - sends both \( K_S(m) \) and \( K_B(K_S) \) to Bob.
- Bob:
  - uses his private key to decrypt and recover \( K_S \)
  - uses \( K_S \) to decrypt \( K_S(m) \) to recover \( m \)

---

**Secure e-mail (continued)**

- Alice wants to provide sender authentication message integrity.
  - \( \odot K_A \)
  - \( K_A(H(m)) \)
  - \( K_A(H(m)) \)
  - \( K_A(H(m)) \)
  - compare
- Alice digitally signs message.
  - sends both message (in the clear) and digital signature.
Secure e-mail (continued)

- Alice wants to provide secrecy, sender authentication, message integrity.

\[
\begin{align*}
  m &\xrightarrow{H(\cdot)} K_A(\cdot) \\
  &\xrightarrow{K_A(H(m))} K_S(\cdot) \\
  &\xrightarrow{+} K_S(\cdot) \\
  &\xrightarrow{+} \text{Internet} \\
  m &\xrightarrow{K_S(\cdot)} K_B(\cdot) \\
  &\xrightarrow{K_B(K_S)} K_B(\cdot) \\
  &\xrightarrow{+} \text{Internet}
\end{align*}
\]

Alice uses three keys: her private key, Bob’s public key, newly created symmetric key.

Pretty good privacy (PGP)

- Internet e-mail encryption scheme, de-facto standard.

A PGP signed message:

```
---BEGIN PGP SIGNED MESSAGE--
Hash: SHA1
Bob: My husband is out of town tonight. Passionately yours, Alice
---BEGIN PGP SIGNATURE---
Version: PGP 5.0
Charset: noconv
yjhJ8RHhGJGhgg/12EpJ+1o8gE4vB3mqJhFEv2P9t6n7G6m5Gw2
---END PGP SIGNATURE---
```

Session Keys

- Session keys are only used once, and then discarded. The advantages are:
  - If a key is used frequently, it is easy to break.
  - Against replay attack.
  - By differentiate the keys for message integrity and confidentiality, the damage of key compromise can be reduced.
  - Before trusting the other party, don’t want to use the key to transmit a large amount of data at the beginning.

The Access Matrix Model

- Current objects (O)
  - A set of entities to which access is to be controlled; e.g., a file.
- Current subjects (S)
  - A set of entities that access current objects; e.g., a process.
- Generic rights
  - A finite set of generic rights; such as read, write, execute.
- Protection state
  - Represented by (S, O, P), where P is an access matrix.
Enforcing A Security Policy

- A security policy is enforced by validating every user access for appropriate access rights. Each access to an object is validated as follows:
  - A subject \( s \) requests an access \( \alpha \) to object \( o \).
  - The protection system presents triple \((s, \alpha, o)\) to the monitor of \( o \).
  - Check the access rights of \( s \) to \( o \). If \( \alpha \in P[s,o] \), then the access is permitted; else it is denied.

Implementations of the Access Matrix

- The access matrix model is very popular because of its simplicity and elegant structure.
- Since the matrix is very sparse, it may not be storage efficient.
  - Capability-based matrix: decomposing the matrix into rows and assigning the access rights contained in rows to their respective subjects. Null entries can be removed for efficiency.
  - Access control list: decompose the matrix by columns.

Capabilities

- Each subject \( s \) is assigned a list of tuples \((o, P[s,o])\) for all objects \( o \) that is allowed to access. The tuples are referred to as capabilities.
- A capability has two fields: object descriptor and access rights.
- Advantages: efficiency, simplicity, and flexibility.
**Capability-based Addressing**

- The system uses the capability ID to search the capability list to locate the capability, and then use the information of the object to get the data.
- With capability-based addressing, relocatability and sharing can be achieved.
  - An object can be relocated without any change to the capabilities that refer to it.
  - Several programs can share the same object.
- Implementation issues, a user should not be able to access a capability.
  - Partitioned approach: capabilities and ordinary data are stored separately.
  - Tagged approach: one or more bits are attached to the word, and users cannot manipulate words with their tag bits ON.

**Disadvantages**

- **Control of propagation**
  - After a subject passes a copy of capability for an object to another subject, the second subject can pass copies to others without the first subject’s knowledge.
  - Solution: a copy bit, or a depth counter.
- **Review**
  - Difficult to find out all subjects who have access to the object.
  - Simple in systems with the partitioned approach.
- **Revocation of access rights**
  - Since a subject may make copies of the capability, revoke of the access right is difficult.
- **Garbage collection**
  - An object may not be accessible to anyone.

**Access Control List**

- When a subject $s$ requests access $\alpha$ to object $o$, it is executed in the following way.
  - The system searches the access control list of $o$ to find out if an entry $(s, \phi)$ exists for subject $s$.
  - If an entry $(s, \phi)$ exists for subject $s$, the system checks to see if the requested access is permitted (i.e., $\alpha \in \phi$).
  - If the requested access is permitted, the request is executed, else it is denied.
- **Advantages**
  - Easy revocation
  - Easy review of an access
**Access Control List**

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Access Rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>read, write, execute</td>
</tr>
<tr>
<td>Jones</td>
<td>read</td>
</tr>
<tr>
<td>Lee</td>
<td>write</td>
</tr>
<tr>
<td>Grant</td>
<td>execute</td>
</tr>
<tr>
<td>White</td>
<td>read, write</td>
</tr>
</tbody>
</table>

**Implementation Issues**

- Execution efficiency is poor since an access control list (ACL) must be searched for each access to a protected object.
  - Solution: after the first access, the access rights of the subject are fetched and stored in a place, called shadow register, which acts like a capability.
  - Negative effect: revocation is complex.
- Efficiency of storage
  - Limiting the access rights to only a small number and assigning a bit in a vector.
  - Use protection groups.
- Authority to change an ACL
  - Self control, the owner can modify the ACL.
  - Hierarchical control: the owner specifies a set of other processes which have the right to modify the ACL.

**Protection Domains**

- Reducing the size of the ACL by using protection domains.
- A protection domain is a set of (object, access rights) pairs. Whenever a subject requests an operation at an object, the reference monitor checks the protection domain associated with the request. Different kinds of protection domains.
  - Groups, such as student, employee. Hierarchical groups.
  - Instead of letting the reference monitor do the work, each subject carries a certificate listing the groups it belongs to.
  - Role-based control: a user logs into the system with a specific role.

**Hierarchical Organization of Protection Domains**

- World
  - Employee
    - Employee_AMS
    - Employee_NYC
    - Employee_SF
      - Dick
      - Kees
    - Employee_ANON
      - …
**Firewall**

- Access control is good for stand alone systems. When outsiders are allowed to access resources, firewalls are used.

![Firewall Diagram]

**Secure Mobile Code**

- Modern distributed system has the ability to migrate code between hosts instead of just migrating data.
- Protect the agent against malicious hosts that try to steal or modify information carried by the agent.
- Protect the host from malicious agents.
  - Whether downloading the right program.
  - Should not get unauthorized access to the host’s resource.

**Protecting an Agent**

- Fully protecting an agent is impossible. An alternative is to organize agents in a way that modifications can be detected by follows.
- The **read-only state** of an agent consists of a collection of data that are signed by the owner.
- To allow collecting information, use **append-only logs**. Suppose server $S$ appends $X$ to the log. $N$ is a secret nonce.
  
  $C_{\text{init}} = K_{\text{owner}}^\dagger(N), \quad C_{\text{new}} = K_{\text{owner}}^\dagger(C_{\text{old}}, \text{sig}(S,X), S)$

- **Selective revealing** of state by providing an array of data items, where each entry is intended for a designated server. Each entry is encrypted with the server’s public key to ensure confidentiality.

**Protecting the Target**

- A sandbox is a technique by which a downloaded program is executed in such a way that each of its instructions can be fully controlled. If an attempt is forbidden, stop.
**Capability**

- Make sure that capability is unforgeable.
- One example system is Amoeba, which is an object-based distributed system.
- When an object is created, the server picks a random check field, and stores it both in the capability and its own table. All bits are on initially, called **owner capability**.
- To create a restricted capability, the client passes a capability to the server with a new right. The server creates a new capability with a new right and new check.
- The server always verifies if the received capability is valid.
- Sometimes, referred to as **attribute certificate**.

**Delegation**

- A user wants to have a big file to be printed at 2pm. Instead of sending the entire file, the user passes the file name to printer so that the printer can get the file and print it. How can the printer get the read permission?
- The user needs to delegate the access right to the printer server.
- One solution is to use **proxy**, which is a token that allows its owner to operate with the same or restricted right as the subject that granted the token. (different from that in client-server model)
- Issues: how to pass the certificate? Avoid illegal copying.
Using a proxy to delegate and prove ownership of access rights

Secure Electronic Transactions (SET)

- SET is a joint effort of Visa and Mastercard, with Netscape and Microsoft. It is a standard for purchasing goods over a network using credit cards.
- Alice wants to buy some goods from Bob. Signing the order and credit card makes the bank and Bob know too much information.
- Solution is dual signature.
  \[ [m1|m2]_A = [m1, H(m2), H(\text{concat}(m1,m2)), K_A(H(\text{concat}(m1,m2)))] \]

Different Steps in SET

1. \([\text{order | pay\_info}]_A, K_{A,\text{bank}}(\text{pay\_info}), K_{\text{bank}}^*(K_{A,\text{bank}})\]
2. \([K_{B1,\text{bank}}(\text{auth}), K_{B1,\text{bank}}^*(K_{B1,\text{bank}}), K_{A,\text{bank}}(\text{pay\_info}), K_{\text{bank}}^*(K_{A,\text{bank}})]\]
3. \([K_{B2,\text{bank}}(\text{auth\_OK}), K_{B2,\text{bank}}^*(K_{B2,\text{bank}}), K_{B3,\text{bank}}(\text{cap}), K_{B3,\text{bank}}^*(K_{B3,\text{bank}})]\]
4. \([K_{B4,\text{bank}}(\text{pay\_me}), K_{B4,\text{bank}}^*(K_{B4,\text{bank}}), K_{B3,\text{bank}}(\text{cap}), K_{B3,\text{bank}}^*(K_{B3,\text{bank}})]\]
5. \([K_{B5,\text{bank}}(\text{cap\_OK}), K_{B5,\text{bank}}^*(K_{B5,\text{bank}})]\]

[^1]: Source of diagrams: [Diagram Source](https://example.com)