Computer Security: Principles and Practice

Fourth Edition

By: William Stallings and Lawrie Brown

Chapter 4

Access Control

Access Control Definitions 1/2

NISTIR 7298 defines access control as:

"the process of granting or denying specific requests to: (1) obtain and use information and related information processing services; and (2) enter specific physical facilities"

Access Control Definitions 2/2

RFC 4949 defines access control as:

"a process by which use of system resources is regulated according to a security policy and is permitted only by authorized entities (users, programs, processes, or other systems) according to that policy"

Access Control Principles

- In a broad sense, all of computer security is concerned with access control

- RFC 4949 defines computer security as:

"measures that implement and assure security services in a computer system, particularly those that assure access control service"

4.1 Access Control Principles

	Basic Security Requirements
1	Limit information system access to authorized users, processes acting on behalf of authorized users, or devices (including other information systems).
2	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.
	Derived Security Requirements
3	Control the flow of CUI in accordance with approved authorizations.
4	Separate the duties of individuals to reduce the risk of malevolent activity without collusion.
5	Employ the principle of least privilege, including for specific security functions and privileged accounts.
6	Use non-privileged accounts or roles when accessing nonsecurity functions.
7	Prevent non-privileged users from executing privileged functions and audit the execution of such functions.
8	Limit unsuccessful logon attempts.
9	Provide privacy and security notices consistent with applicable CUI rules.
10	Use session lock with pattern-hiding displays to prevent access and viewing of data after period of inactivity.
11	Terminate (automatically) a user session after a defined condition.
12	Monitor and control remote access sessions.
13	Employ cryptographic mechanisms to protect the confidentiality of remote access sessions.
14	Route remote access via managed access control points.
15	Authorize remote execution of privileged commands and remote access to security-relevant information.
16	Authorize wireless access prior to allowing such connections.
17	Protect wireless access using authentication and encryption.
18	Control connection of mobile devices.
19	Encrypt CUI on mobile devices.
20	Verify and control/limit connections to and use of external information systems.
21	Limit use of organizational portable storage devices on external information systems.
22	Control CUI posted or processed on publicly accessible information systems.

CUI = controlled unclassified information

Table 4.1

Access Control Security Requirements (SP 800-171)

(Table is on page 107 in the textbook)

4.1.1 Access Control Context

Core Components in the Model

- Authentication Function: Confirms user identity
- Access Control Function: Enforces permissions
- Authorization Database: Stores rules & privileges
- Security Administrator: Manages authorization data

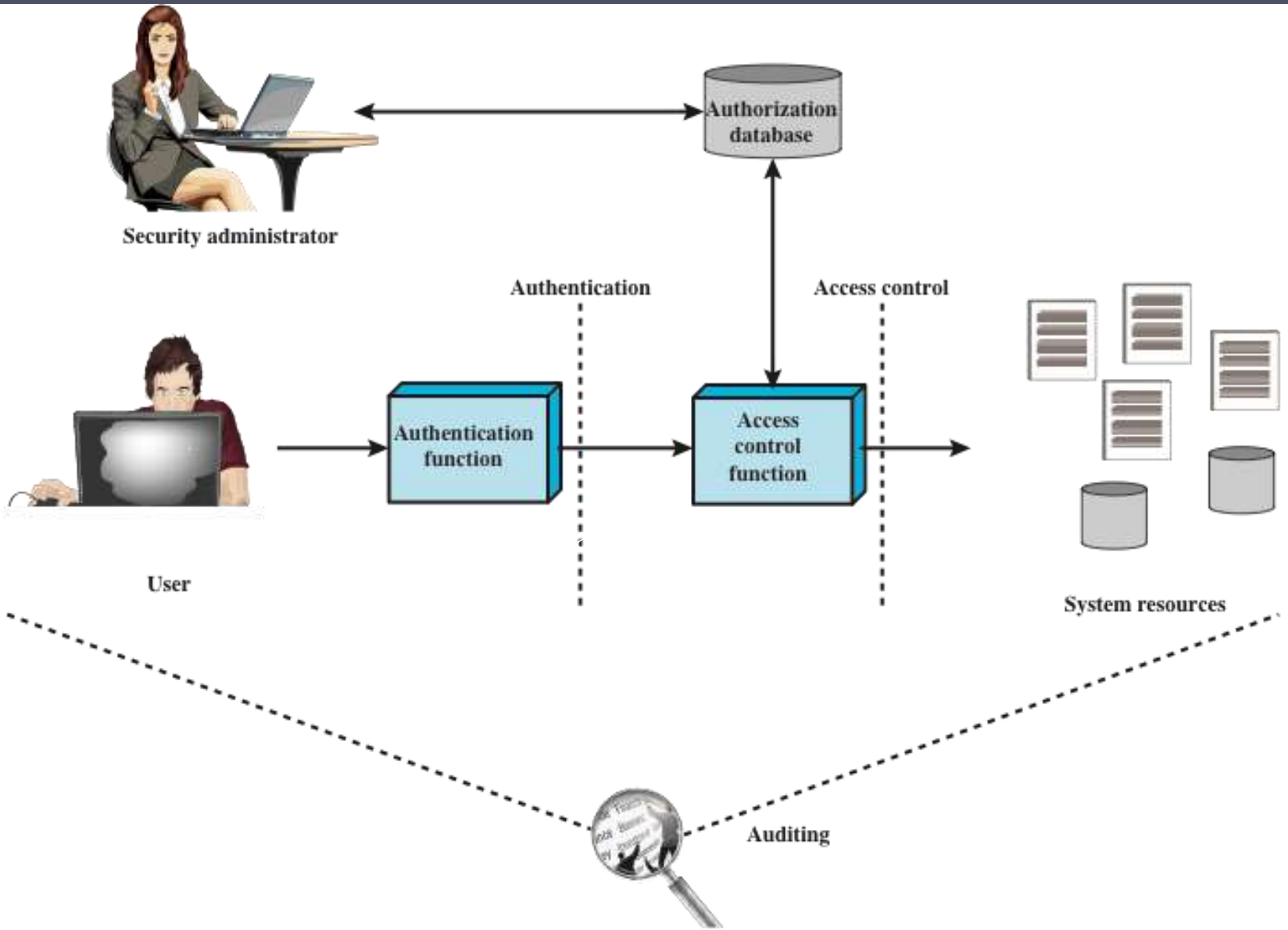


Figure 4.1 Relationship Among Access Control and Other Security Functions

Source: Based on [SAND94].

Authentication & Access Workflow

1. User requests access
2. Authentication function validates credentials
3. Access control function checks authorization rules
4. Authorization database returns permissions
5. Access to system resources allowed or denied

Role of Auditing

- Monitors all authentication and access events
- Creates logs for accountability and investigation
- Helps detect misuse or unauthorized activity

4.1.2 Access Control Policies

- Discretionary access control (DAC)
  - Controls access based on the identity of the requestor and on access rules (authorizations) stating what requestors are (or are not) allowed to do
- Mandatory access control (MAC)
  - Controls access based on comparing security labels with security clearances
- Role-based access control (RBAC)
  - Controls access based on the roles that users have within the system and on rules stating what accesses are allowed to users in given roles
- Attribute-based access control (ABAC)
  - Controls access based on attributes of the user, the resource to be accessed, and current environmental conditions

4.2 Subjects, Objects, and Access Rights

Subjects, Objects, and Access Rights

Subject

An entity capable of accessing objects

Three classes
- Owner
- Group
- World

Object

A resource to which access is controlled

Entity used to contain and/or receive information

Access right

Describes the way in which a subject may access an object

Could include:
- Read
- Write
- Execute
- Delete
- Create
- Search

4.3 Discretionary Access Control (DAC)

- Scheme in which an entity may be granted access rights that permit the entity, by its own violation, to enable another entity to access some resource
- Often provided using an access matrix
  - One dimension consists of identified subjects that may attempt data access to the resources
  - The other dimension lists the objects that may be accessed
- Each entry in the matrix indicates the access rights of a particular subject for a particular object

4.3.1 Access Matrix Model

Access Control Structures

- Access Matrix • ACLs • Capability Lists

What Is an Access Matrix?

- A core model in access control
- Rows = Subjects (users, processes)
- Columns = Objects (files, resources)
- Cells specify permissions: Read, Write, Own

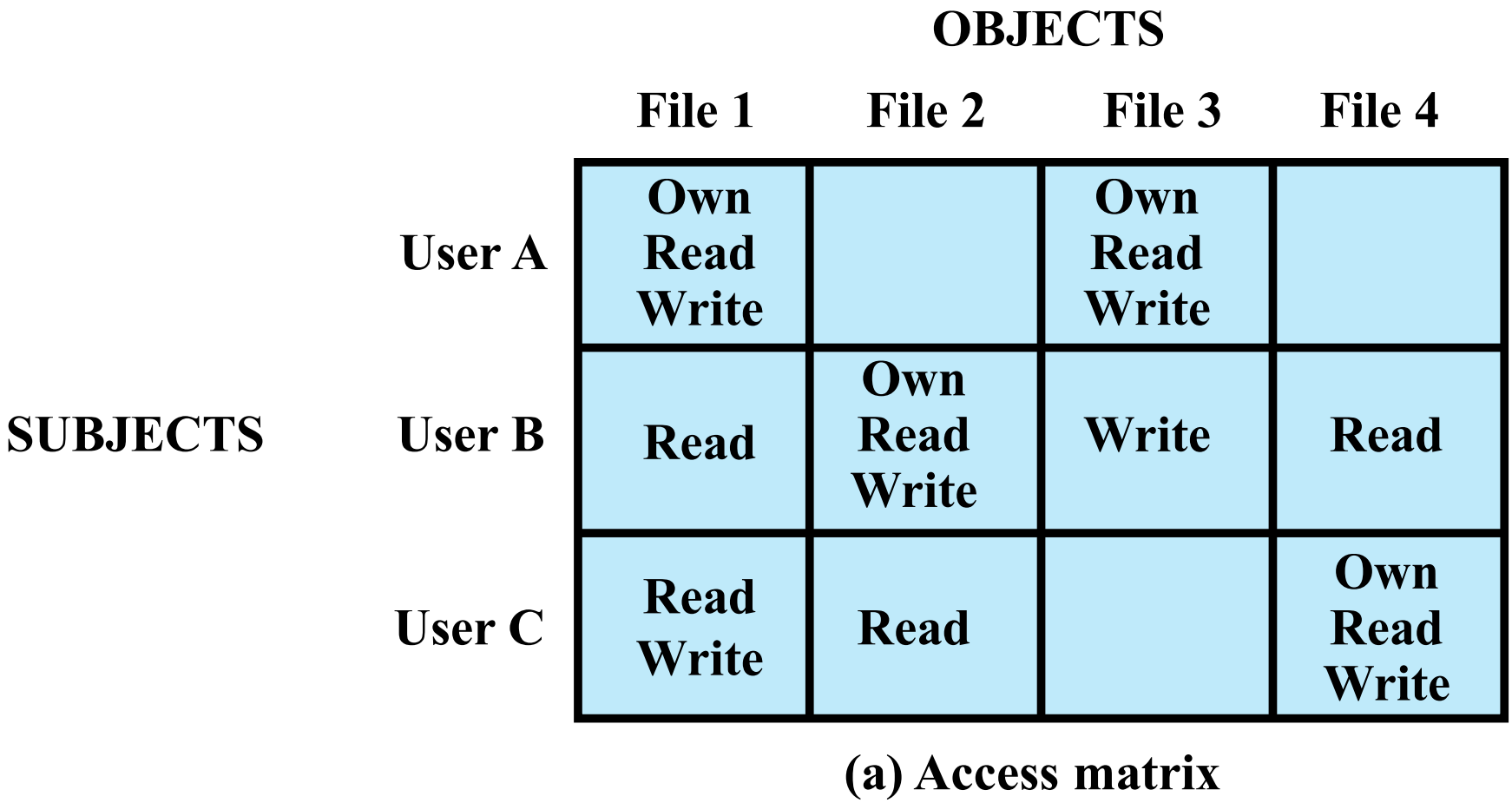


Figure 4.2 Example of Access Control Structures

Example Matrix Overview

- User A:
  - File 1: Own, Read, Write
  - File 3: Own, Read, Write
- User B:
  - File 2: Own, Read, Write
  - File 3: Write
  - File 4: Read
- User C:
  - File 1: Read, Write
  - File 2: Read
  - File 4: Own, Read, Write

How to Interpret the Matrix

- 'Own' typically includes the ability to modify permissions
- 'Read' allows viewing contents
- 'Write' allows modifying contents
- The matrix can be implemented via ACLs or capability lists

4.3.2 Access Control Lists (ACLs)

Access Control Lists (ACLs)

- Stored per object
- Each object lists users + permissions

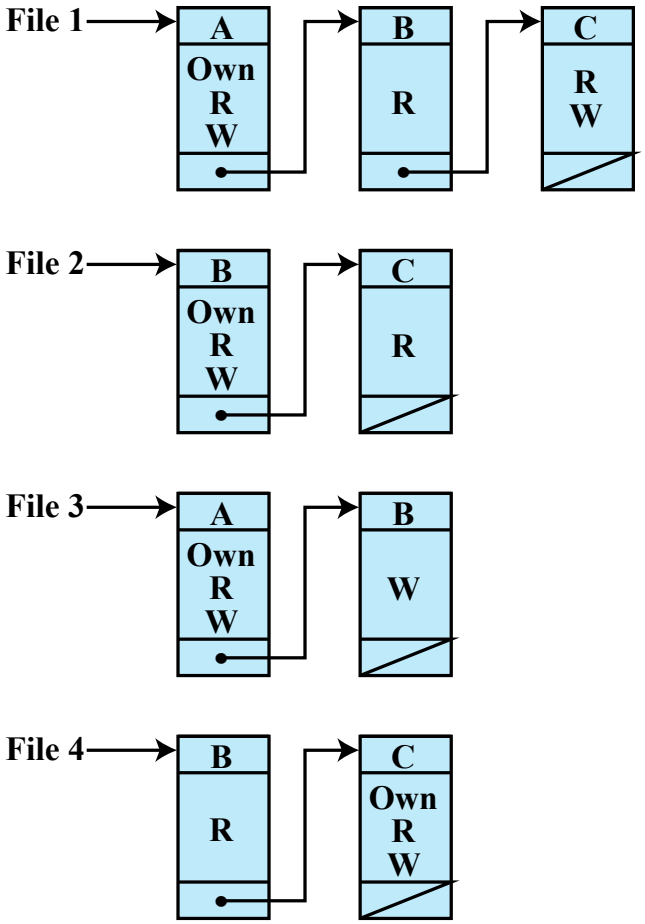
Example (File 1 ACL):

- User A: Own, Read, Write
- User B: Read
- User C: Read, Write

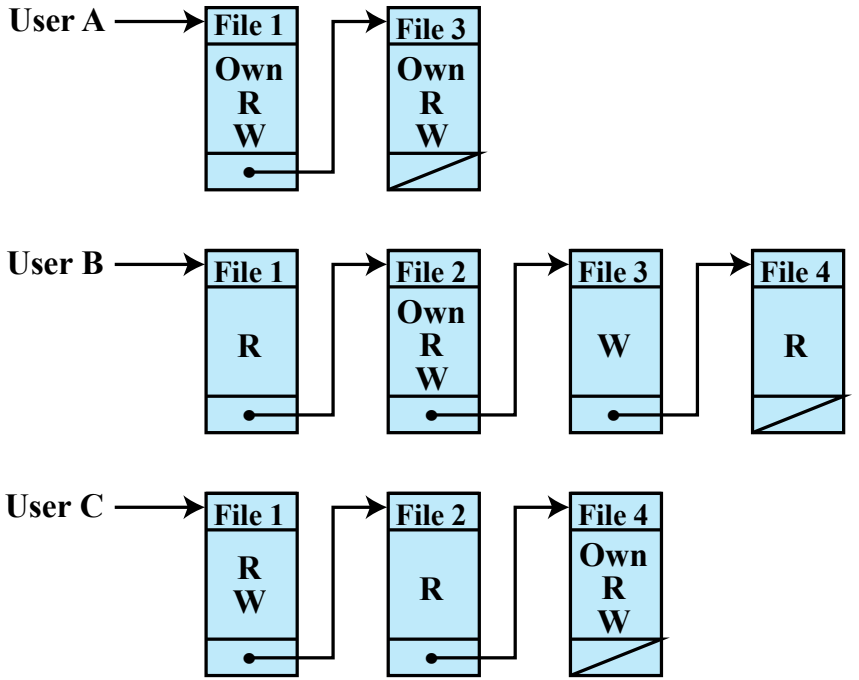
ACL Characteristics

- Advantages:
- • Easy to inspect per-resource permissions
- • Works well for centralized management

- Common Uses:
- • File systems
- • Routers / firewalls
- • Database permissions



(b) Access control lists for files of part (a)



(c) Capability lists for files of part (a)

Figure 4.2 Example of Access Control Structures

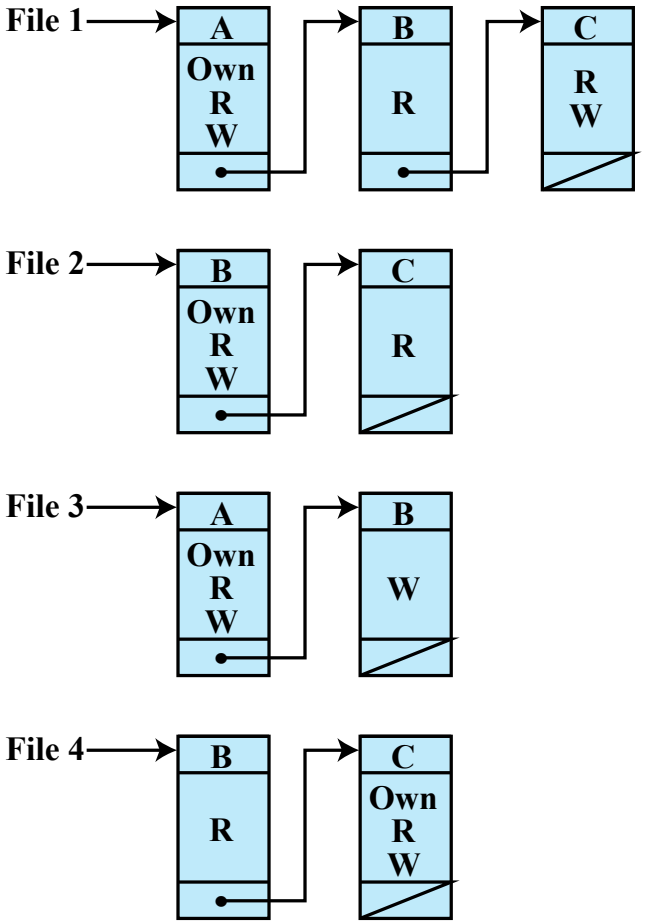
4.3.3 Capability Lists

Capability Lists

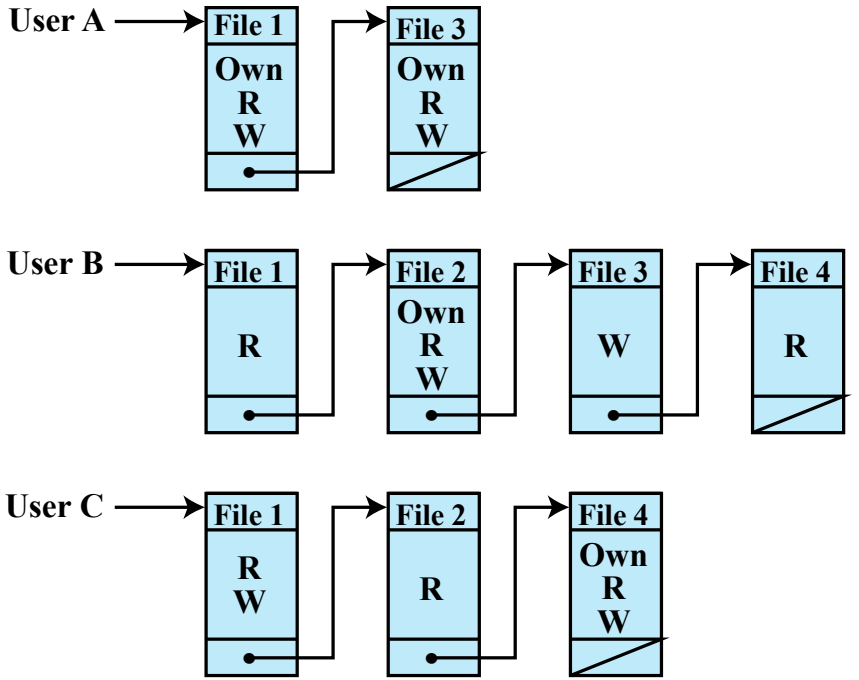
- Stored per subject (user/process)
- Each subject lists accessible objects + rights

Example (User A):

- File 1 – Own, Read, Write
- File 3 – Own, Read, Write



(b) Access control lists for files of part (a)



(c) Capability lists for files of part (a)

Figure 4.2 Example of Access Control Structures

Capability Characteristics

- Advantages:
  - Fast authorization checks
  - Great for distributed systems
  - Allows delegation via capability transfer

- Use cases:
  - Capability OS designs
  - Token-based access
  - Cloud identity systems

ACL vs Capability Comparison

- ACLs:
  - Object-focused
  - System Asks: "Who Can Access This Object ?"
- Capabilities:
  - Subject-focused
  - System asks: "What can this subject access?"

Security Considerations

- ACL Weaknesses:
  - Misconfigurations = privilege escalation
- Capability Weaknesses:
  - Capability theft
  - Must be unforgeable and securely stored

Summary

- Access matrix is the conceptual foundation
- ACLs represent object-side permissions
- Capabilities represent user-side permissions
- Each model excels in different environments

4.3.4 Authorization Table

Subject	Access Mode	Object
A	Own	File 1
A	Read	File 1
A	Write	File 1
A	Own	File 3
A	Read	File 3
A	Write	File 3
B	Read	File 1
B	Own	File 2
B	Read	File 2
B	Write	File 2
B	Write	File 3
B	Read	File 4
C	Read	File 1
C	Write	File 1
C	Read	File 2
C	Own	File 4
C	Read	File 4
C	Write	File 4

Table 4.2

Authorization Table for Files in Figure 4.2

(Table is on page 113 in the textbook)

Purpose of the Authorization Table

- Lists permissions in (Subject, Access Mode, Object) format
- Flattened form of access matrix from Figure 4.2
- Useful as a system-wide source of truth for permissions
- Can generate ACLs and capability lists from it

User A Permissions

- User A can access:
  - File 1 – Own, Read, Write
  - File 3 – Own, Read, Write

  These represent full control of Files 1 and 3.

User B Permissions

- User B can access:
  - File 1 – Read
  - File 2 – Own, Read, Write
  - File 3 – Write
  - File 4 – Read
- B has full control of File 2 and limited access to others.

User C Permissions

- User C can access:
  - File 1 – Read, Write
  - File 2 – Read
  - File 4 – Own, Read, Write
- C fully controls File 4.

Why This Table Matters

- Precise mapping of all permissions
- Helps define, audit, and analyze access control
- Enables generation of ACLs (object-centric)
- Enables generation of capability lists (subject-centric)
- Forms the foundation for security policy enforcement

4.3.5 Extended Access Control Matrix

What is an Extended Access Control Matrix?

- Framework defining permissions of subjects over objects
- Extends simple access matrix with richer rights
- Covers subjects, files, processes, disk drives

		OBJECTS								
		subjects			files		processes		disk drives	
		S_1	S_2	S_3	F_1	F_2	P_1	P_2	D_1	D_2
SUBJECTS	S_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
	S_2		control		write *	execute			owner	seek *
	S_3			control		write	stop			

* - copy flag set

Figure 4.3 Extended Access Control Matrix

Core Rights Explained

- control: manage permissions
- owner: full object authority
- read*, write*: rights with copy/propagation flag
- execute, stop, wakeup, seek

Matrix Structure

- Rows = subjects (S1, S2, S3)
- Columns = subjects, files, processes, devices
- Cells = allowed rights

Copy Flag (*)

- Indicates a right can be propagated
  - - e.g., read* allows granting read to others
- Enables controlled delegation

Security Implications

- Helps prevent unauthorized privilege escalation
- Reveals over-privilege and misconfigurations
- Supports principle of least privilege

Comparison to ACLs & Capabilities

- ACLs: object-centric view
- Capabilities: subject-centric view
- Access matrix generalizes both

4.3.6 Organization of the Access Control Function

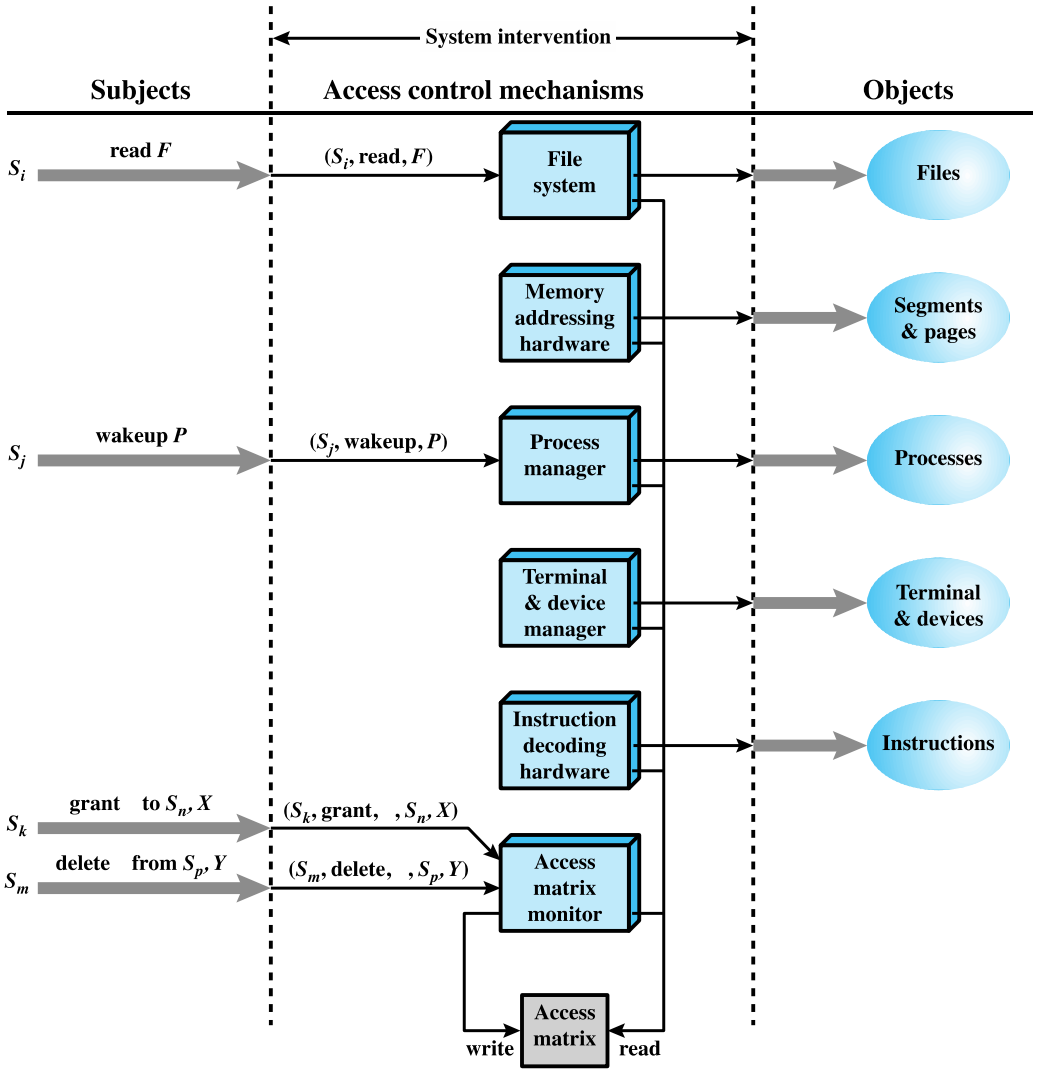


Figure 4.4 An Organization of the Access Control Function

What Access Control Does

- Mediates every subject → object interaction
- Ensures permissions and policies are enforced
- Central to system security

Subjects & Requests

- Subjects issue operations (read, write, execute…)
- System interprets them as (S, op, O)
- These requests trigger security checks

Access Control Mechanisms

- File system checks permissions
- Memory hardware enforces address bounds
- Process manager regulates process operations
- Device manager restricts device access
- Instruction decoder validates privileged instructions

Access Matrix Monitor

- Central entity holding the access matrix
- Evaluates each request using the matrix
- Supports grant/delete updates to permissions

Objects

- Files
- Memory segments
- Processes
- Devices
- Instructions

System Intervention Path

- Request → AC mechanism → Matrix check → Object access
- If permitted: access granted
- If denied: request blocked

Summary

- Access control is layered and distributed
- Matrix model underpins permission logic
- Every access is checked – no exceptions

4.3.7 Access Control System Commands

Overview of the Rule Set

- R1–R2: Delegation and granting of rights
- R3–R4: Removing rights and reading entries
- R5–R6: Creating and destroying objects
- R7–R8: Creating and destroying subjects

Rule	Command (by S_0)	Authorization	Operation
R1	**transfer** $\left\{ \begin{matrix} a* \\ a \end{matrix} \right\}$ **to** S, X	'a*' in $A[S_0, X]$	store $\left\{ \begin{matrix} a* \\ a \end{matrix} \right\}$ in $A[S, X]$
R2	**grant** $\left\{ \begin{matrix} a* \\ a \end{matrix} \right\}$ **to** S, X	'owner' in $A[S_0, X]$	store $\left\{ \begin{matrix} a* \\ a \end{matrix} \right\}$ in $A[S, X]$
R3	**delete** a **from** S, X	'control' in $A[S_0, S]$ or 'owner' in $A[S_0, X]$	delete a from $A[S, X]$
R4	$w \leftarrow$ **read** S, X	'control' in $A[S_0, S]$ or 'owner' in $A[S_0, X]$	copy $A[S, X]$ into w
R5	**create object** X	None	add column for X to A; store 'owner' in $A[S_0, X]$
R6	**destroy object** X	'owner' in $A[S_0, X]$	delete column for X from A
R7	**create subject** S	none	add row for S to A; execute **create object** S; store 'control' in $A[S, S]$
R8	**destroy subject** S	'owner' in $A[S_0, S]$	delete row for S from A; execute **destroy object** S

Table 4.3

Access Control System Commands

(Table is on page 116 in the textbook)

R1 Transfer

- Transfers a or a* to another subject
- Requires a* in $A[S_o,X]$
- Operation stores a or a* in $A[S,X]$

Rule	Command (by S_o)	Authorization	Operation
R1	**transfer** $\left\{ \begin{matrix} \alpha* \\ \alpha \end{matrix} \right\}$ **to** S, X	'α*' in $A[S_o, X]$	store $\left\{ \begin{matrix} \alpha* \\ \alpha \end{matrix} \right\}$ in $A[S, X]$

R2 Grant

- Grants a or a* without removing from grantor
- Requires 'owner' right
- Used for safe propagation

Rule	Command (by S_0)	Authorization	Operation
R2	**grant** $\{a^*, a\}$ **to** S, X	'owner' in $A[S_0, X]$	store $\{a^*, a\}$ in $A[S, X]$

R3 Delete

- Deletes right α from A[S,X]
- Requires 'control' or 'owner'

Rule	Command (by S_o)	Authorization	Operation
R3	**delete** α **from** S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	delete α from $A[S, X]$

R4 Read

- Reads the entry A[S,X] into w
- Requires 'control' or 'owner'

Rule	Command (by S_o)	Authorization	Operation
R4	$w \leftarrow$ **read** S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	copy $A[S, X]$ into w

R5 Create Object

- Adds new object column
- Assigns 'owner' to creator

Rule	Command (by S_o)	Authorization	Operation
R5	**create object** X	None	add column for X to A; store 'owner' in $A[S_o, X]$

R6 Destroy Object

- Removes object column
- Requires 'owner'

Rule	Command (by S_0)	Authorization	Operation
R6	**destroy object** X	'owner' in $A[S_0, X]$	delete column for X from A

R7 Create Subject

- Adds new subject row
- Assigns 'control' to creator

Rule	Command (by S_0)	Authorization	Operation
R7	**create subject** *S*	none	add row for *S* to *A*; execute **create object** *S*; store 'control' in *A*[*S*, *S*]

R8 Destroy Subject

- Deletes subject row
- Requires 'owner'

Rule	Command (by S_0)	Authorization	Operation
R8	**destroy subject** *S*	'owner' in $A[S_0, S]$	delete row for *S* from *A*; execute **destroy object** *S*

4.3.7 Protection Domains

Protection Domains: Core Idea

- A protection domain is a set of objects plus the access rights to those objects.
- Defines what a process, thread, or user may legally access.
- Basis of OS security and controlled resource use.

Access Matrix & Domain Structure

- In the access matrix model, each row represents a protection domain.
- Columns represent objects and allowed operations on them.
- Switching domains means switching the row a process operates under.

Process–Domain Association

- A process can be associated with a domain statically or dynamically.
- Users may spawn processes with only a subset of their rights.
- Capability-based systems allow finer-grained assignment of rights.

User Mode vs Kernel Mode

- User mode: Protected memory regions cannot be accessed.
- User mode: Privileged instructions cannot be executed.
- Kernel mode: OS may execute privileged instructions and access protected areas.
- CPU hardware enforces separation between domains.

4.4 UNIX File Access Control

UNIX Inode Administration

- UNIX files are administered using inodes (index nodes).
- Inodes store control structures with key information about each file.
- A single inode may be linked to several file names (hard links).
- Each active inode corresponds to exactly one file object in the system.
- File attributes, permissions, and control metadata reside inside the inode.
- The filesystem maintains an on-disk inode table that contains all inodes.
- When a file is opened, its inode is loaded into memory and kept in the in-memory inode table.

UNIX File Administration with Inodes

- UNIX manages files by inodes, not by names.
- Inode (index node) is the on-disk structure describing a file.
- Directory entries map names → inode numbers.
- Metadata (size, owner, permissions, etc.) lives in the inode, not in the filename.

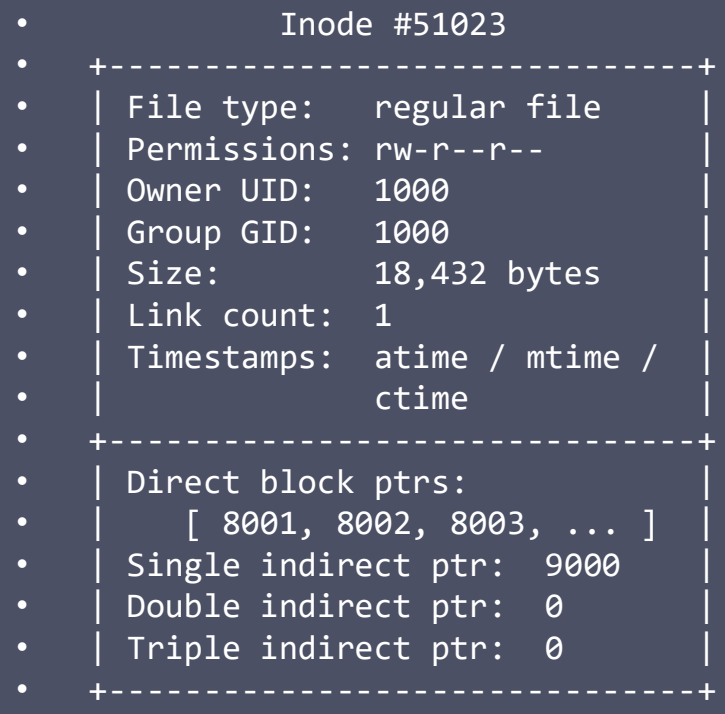
Directories: Name → Inode Mapping

- Directory: /home/alex

```
+-----------------+-----------+
| Name            | Inode #   |
+-----------------+-----------+
| .               |  42001    |  (this directory)
| ..              |  40000    |  (parent directory)
| report.txt      |  51023    |
| data.bin        |  51024    |
| notes           |  51025    |  (subdirectory)
+-----------------+-----------+
```

- A directory is just a list of (name, inode number) pairs.
- No metadata here: size, owner, permissions, etc. are stored in the inode.

Inode Structure (Conceptual)



- Inode holds metadata and pointers to data blocks.
- Data lives in blocks (e.g., 8001, 8002, …) on disk.
- Growing the file: allocate blocks and update inode’s pointers and size.

atime • mtime • ctime — What They Mean

- atime (Access Time)
  - Updated when the file is READ.
  - Examples: cat, less, any read action.

- mtime (Modification Time)
  - Updated when FILE CONTENT changes.
  - Examples: editing, appending, truncating.

- ctime (Change Time)
  - Updated when INODE METADATA changes.
  - Examples: chmod, chown, link count changes, renaming, moves within filesystem, writes (since mtime updates).

Note: ctime is NOT creation time; traditional Linux filesystems don't store crtime.

From Pathname to Inode to Data

- "/home/alex/report.txt"

- "/" (inode 2)
- |
- | "home" -> inode 40000
- v
- Directory inode 40000
- |
- | "alex" -> inode 42001
- v
- Directory inode 42001
- |
- | "report.txt" -> inode 51023
- v
- Inode 51023 --> data blocks [8001, 8002, ...]

- Kernel walks directories, resolving each name to an inode.
- Final component's inode describes the file completely.
- After open(), operations use the inode, not the pathname.

Displaying Inode Numbers in Linux (with Examples)

- Quick view: ls -i myfile.txt
  Output: 1234567 myfile.txt

- Detailed info: stat myfile.txt
  Output: Inode: 1234567 Links: 1

- List all files with inodes: ls –li myfile.txt

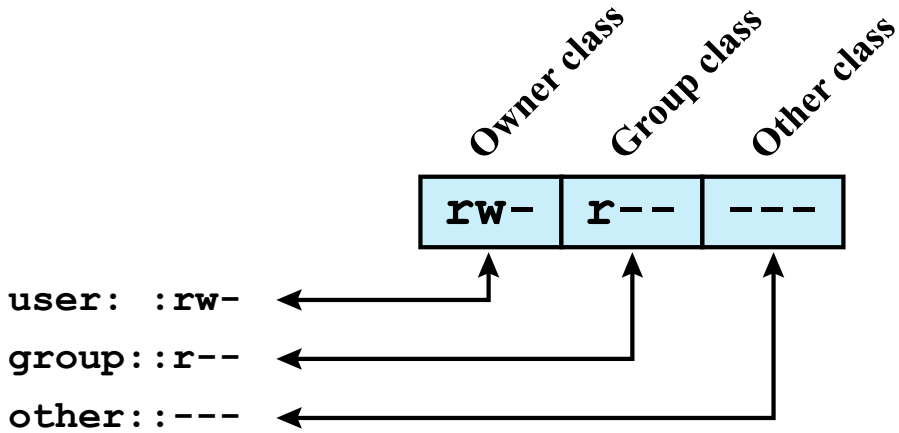
  Output: 1234567 -rw-r--r-- 1 user user 42 Jan 1 myfile.txt

- Search by inode: find . -inum 1234567
  Output: ./myfile.txt

- Print inode + path: find . -printf "%i %p\n"
  Output: 1234567 ./myfile.txt

UNIX
File Access Control

- Unique user identification number (user ID)
- Member of a primary group identified by a group ID
- Belongs to a specific group
- 12 protection bits
  - Specify read, write, and execute permission for the owner of the file, members of the group and all other users
- The owner ID, group ID, and protection bits are part of the file's inode



(a) Traditional UNIX approach (minimal access control list)

Figure 4.5 UNIX File Access Control

The 12 Protection Bits in Linux

- 9 Standard Permission Bits:
  - User: r w x
  - Group: r w x
  - Others: r w x

- 3 Special Permission Bits:
  - setuid (run with owner's privileges)
  - setgid (run with group's privileges/inherit group)
  - sticky (restrict delete/rename in shared dirs)

Total = 9 + 3 = 12 protection bits

Why 10 Characters Represent 12 Protection Bits

Permission String Example:
    -rwsr-Sr-t

10 characters, but 12 protection bits:

- 9 regular bits:
  - User: r w x
  - Group: r w x
  - Others: r w x

- 3 special bits (embedded in x positions):
  - setuid → shows as s/S in user execute field
  - setgid → shows as s/S in group execute field
  - sticky → shows as t/T in others execute field

The first character '-' is file type (not a protection bit).

All 12 bits are visible in octal mode:
    stat -c "%a" myfile.txt   → e.g., 4755

Difference Between rwx and rws in Linux

- rwx:
  - r = read
  - w = write
  - x = execute (normal execution as the invoking user)

- rws:
  - r = read
  - w = write
  - s = setuid + execute

Meaning of 's':
  - Program executes with the FILE OWNER'S privileges.
  - 's' replaces 'x' when setuid is ON in user permissions.

Example:
  -rwsr-xr-x /usr/bin/passwd
  → passwd runs with root's privileges even for normal users.

SGID: Meaning of 's' in Group Permissions

- SGID (Set Group ID):
  - 's' appears in the GROUP execute position.
  - Replaces 'x' when SGID bit is ON.

Effects:

- Executable runs with the FILE'S GROUP privileges.
- On directories: new files inherit the directory's group.

Example:

```
-rwxr-sr-x  myscript
```

→ Program runs with the file's group permissions.

Sticky Bit: Meaning of 't'/'T' in Others Permissions

- Sticky Bit:
  - Appears in the OTHERS execute position.
  - Shows as 't' (if x is set) or 'T' (if x is not set).

Effect (on directories):
  - Users may delete/rename ONLY their own files.
  - Prevents others from deleting shared directory files.

Common Example:
  drwxrwxrwt /tmp
  → World-writable directory but protected by sticky bit.

On files (rare):
  - Historically used for caching executables; mostly obsolete today.

Traditional UNIX File Access Control

- "Set user ID"(SetUID)
- "Set group ID"(SetGID)
  - System temporarily uses rights of the file owner/group in addition to the real user's rights when making access control decisions
  - Enables privileged programs to access files/resources not generally accessible
- Sticky bit
  - When applied to a directory it specifies that only the owner of any file in the directory can rename, move, or delete that file
- Superuser
  - Is exempt from usual access control restrictions
  - Has system-wide access

Access Control Lists (ACLs) in UNIX

Modern UNIX systems support ACLs

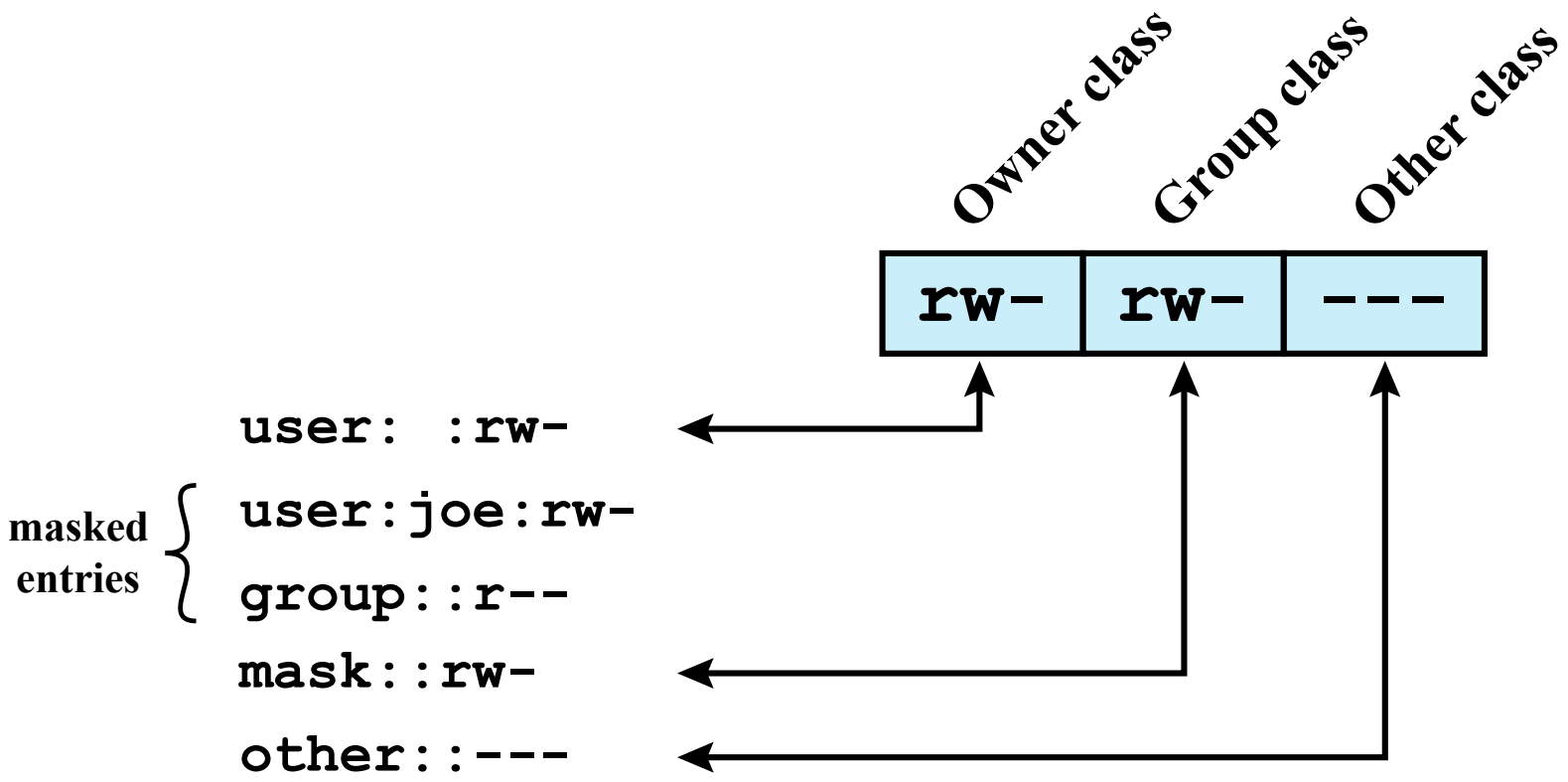
- FreeBSD, OpenBSD, Linux, Solaris

FreeBSD

- Setfacl command assigns a list of UNIX user IDs and groups
- Any number of users and groups can be associated with a file
- Read, write, execute protection bits
- A file does not need to have an ACL
- Includes an additional protection bit that indicates whether the file has an extended ACL

When a process requests access to a file system object two steps are performed:

- Step 1 selects the most appropriate ACL
- Step 2 checks if the matching entry contains sufficient permissions



(b) Extended access control list

Figure 4.5 UNIX File Access Control

Step 1 — Initial ACL (Before ACL Entries)

- `touch testfile`
- `getfacl testfile`

- Result:

```
user::rw-
group::---
mask::---
other::---
```

- Only owner has permissions; no extended ACL entries.

Step 2 — Add ACL Entry for User demos

- `setfacl -m user:demos:rw- testfile`
- `getfacl testfile`

- Result:

```
user::rw-
user:demos:rw-      #effective:---
group::---          #effective:---
mask::---
other::---
```

- Mask remains '---', so demos has no effective permissions.

Step 3 — Add Named Group and User robot

- `setfacl -m group:iugroup:r--,user:robot:--- testfile`
- `getfacl testfile`

- Result:

```
user::rw-
user:robot:---       #effective:---
user:demos:rw-       #effective:---
group::---           #effective:---
group:iugroup:r--    #effective:---
mask::---
other::---
```

- Mask still blocks all non-owner entries.

Step 4 — Set the Mask to Enable ACL Entries

- `setfacl -m mask:rw- testfile`
- `getfacl testfile`

- Result:
- `user::rw-`

```
user:robot:---       #effective:---
user:demos:rw-       #effective:rw-
group::---           #effective:---
group:iugroup:r--    #effective:r--
mask::rw-
other::---
```

- Mask now allows demos and iugroup to have their intended permissions.
- The mask is the effective permissions cap applied to all ACL entries except the owner and “other”.

4.5 Role-Based Access Control (RBAC)

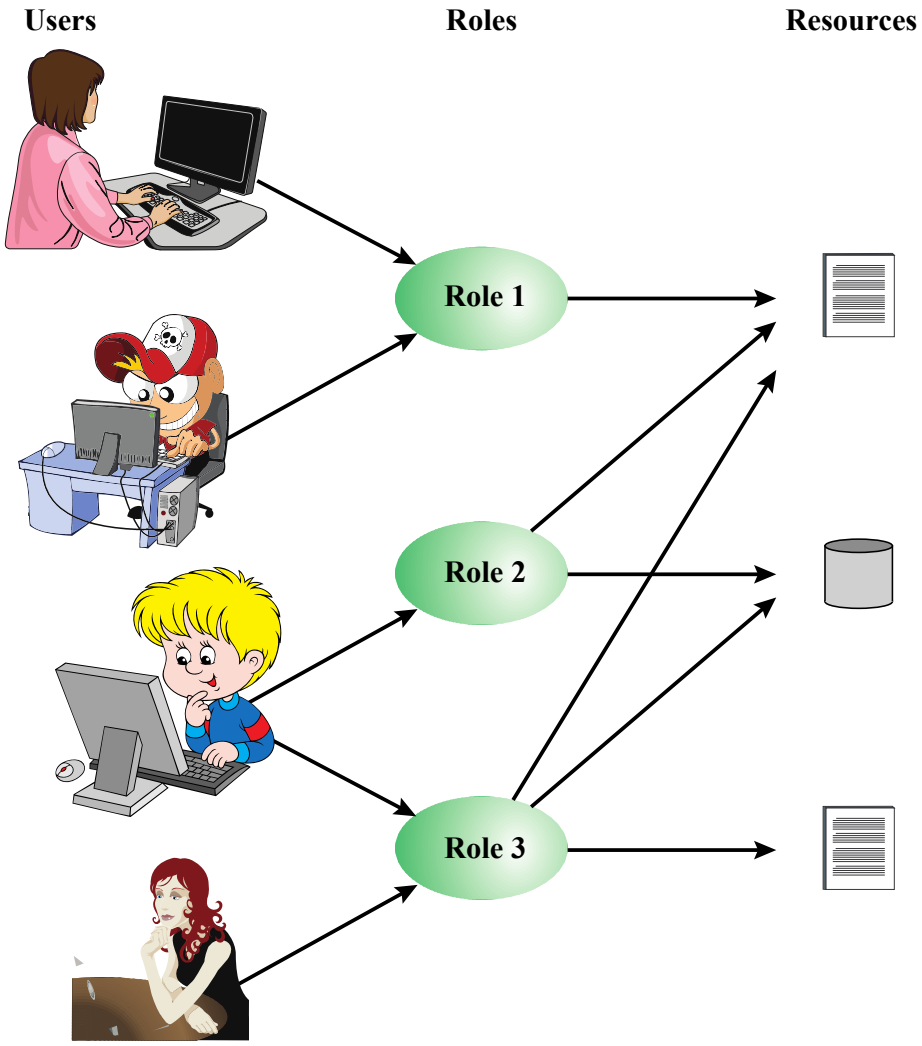


Figure 4.6 Users, Roles, and Resources

Users

- Individuals who need access to system resources
  - Examples: employees, students, admins, contractors
- Users do NOT receive direct permissions
- Access comes only through assigned roles

Roles

- Represent job functions or responsibilities
  - Examples: Reader, Analyst, Manager
- Permissions are assigned to roles—not users
- Simplifies admin and enforces consistent access

Resources

- Objects users need access to
  - Examples: files, databases, applications, printers
- Roles determine what actions can be performed
- Users access resources indirectly via roles

How the Mapping Works

1. Users → are assigned to roles
2. Roles → contain permissions
3. Permissions → allow access to resources

- Users never receive permissions directly
- Roles act as intermediaries between users and resources

Benefits of RBAC

- Highly scalable for large organizations
- Easy onboarding and offboarding
- Supports least-privilege access
- Simplifies auditing and compliance
- Minimizes permission errors

Key Idea of RBAC

- RBAC separates WHO a user is from WHAT they can do
- Permissions attach to roles, not individuals
- Users gain authority solely through role membership
- More predictable and secure than direct permission assignment

Access Control Matrix Representation of RBAC

- RBAC can be represented using two related matrices:

1) User–Role Assignment Matrix:

- Rows represent users (U1…Um)
- Columns represent roles (R1…Rn)
- An 'X' marks which roles each user belongs to

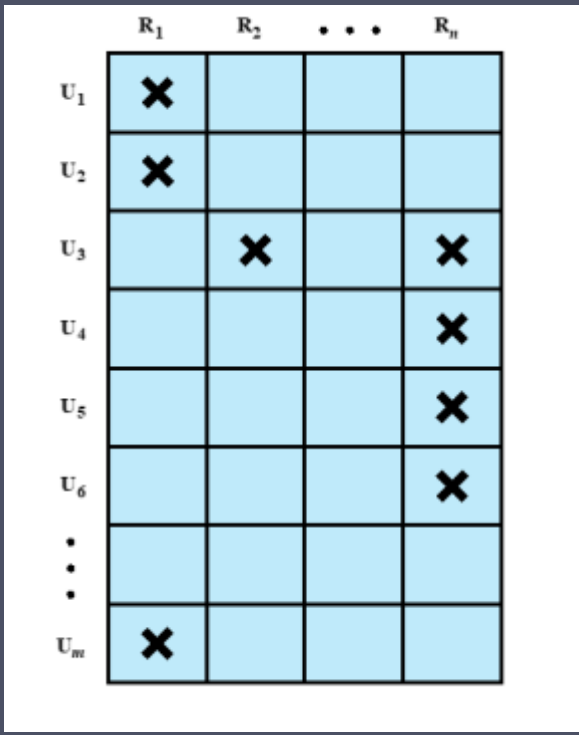


Figure 4.7 Access Control Matrix Representation of RBAC

Access Control Matrix Representation of RBAC

- RBAC can be represented using two related matrices:

2) Role–Permission/Object Matrix:

- Rows represent roles
- Columns represent objects/resources (files, devices, processes)
- Each cell lists the permissions that role has on the object

- Access decision:

  User → assigned to Role → Role has permissions on Object

- This structure cleanly separates users from permissions.

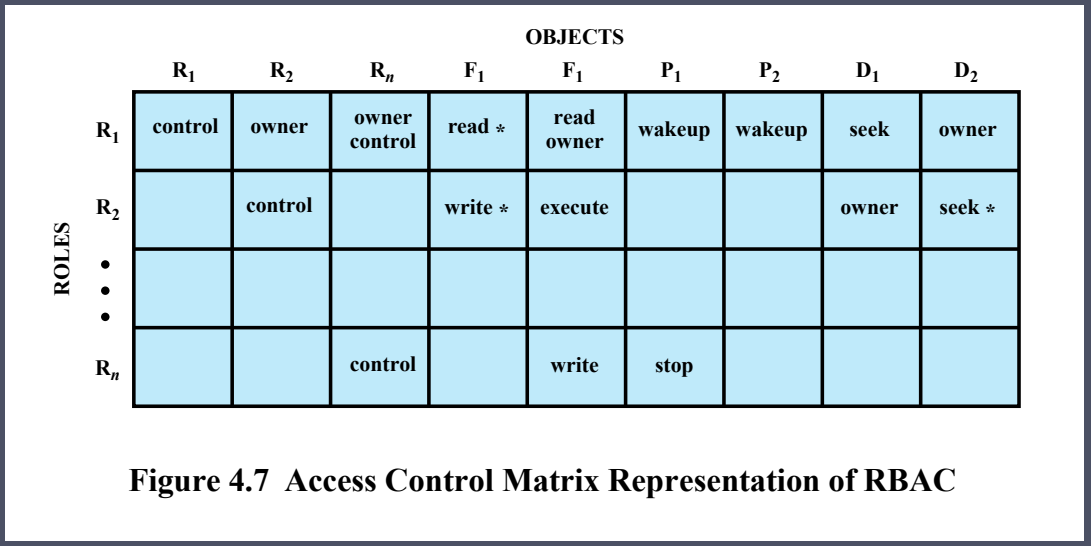


Figure 4.7 Access Control Matrix Representation of RBAC

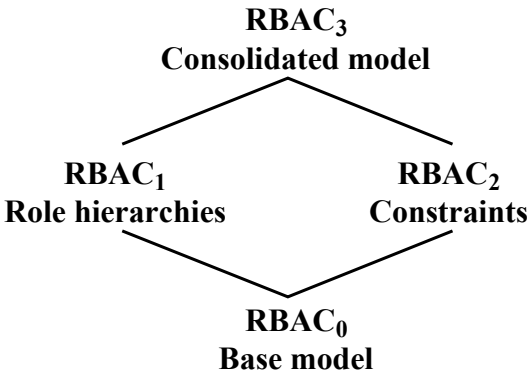
4.5.1 RBAC Reference Models

RBAC Model Hierarchy ($RBAC_0 \rightarrow RBAC_3$)

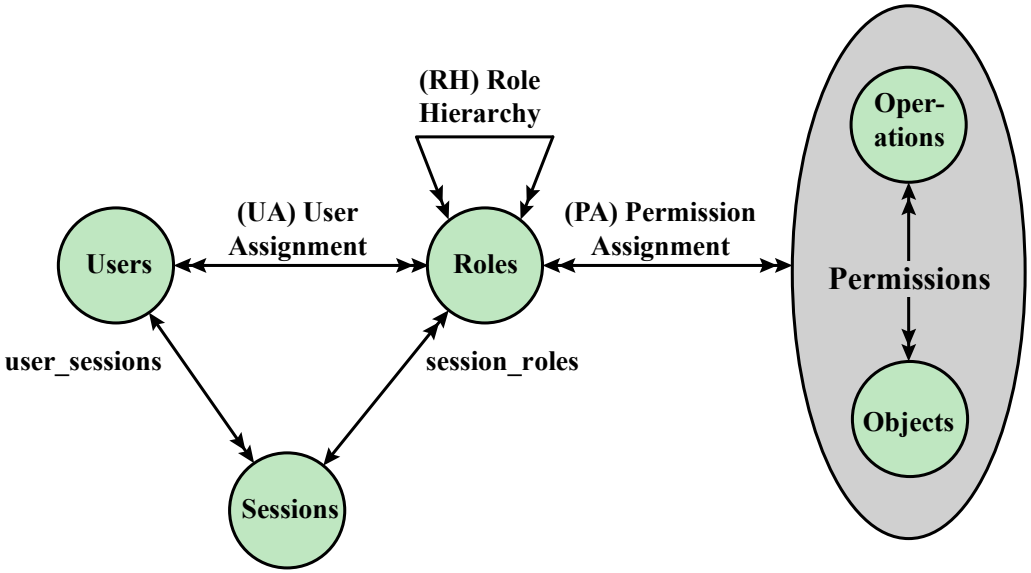
- $RBAC_0$ — Base Model: Users, Roles, Permissions, Sessions
- $RBAC_1$ — Adds Role Hierarchies (senior ↔ junior roles)
- $RBAC_2$ — Adds Constraints (separation of duty, prerequisites)
- $RBAC_3$ — Consolidated Model ($RBAC_1$ + $RBAC_2$)
- Represents increasing expressive power and security control

Table 4.4
Scope RBAC Models

Models	Hierarchies	Constraints
$RBAC_0$	No	No
$RBAC_1$	Yes	No
$RBAC_2$	No	Yes
$RBAC_3$	Yes	Yes



(a) Relationship among RBAC models



(b) RBAC models

Figure 4.8 A Family of Role-Based Access Control Models.

4.5.1.1 Base Model—$RBAC_0$

- Users: Human or system entities needing access
- Roles: Named job functions grouping permissions
- Permissions: Operations allowed on objects
- Objects: Files, databases, applications, resources
- Sessions: Mapping of a user to an activated subset of roles

4.5.1.2 $RBAC_1$ – Role Hierarchies

- Role hierarchies reflect organizational job structures.
- Higher responsibility = greater authority/access.
- Subordinate roles inherit permissions from superior roles.
- Supports structured, scalable access control.

Concept of Inheritance

- A superior role includes all permissions of its subordinate roles.
- Example:
  - Manager inherits all permissions of Employee.
  - Senior Doctor inherits permissions of Doctor.
- Simplifies permission assignment and reduces redundancy.

Example Role Hierarchy

Example organizational hierarchy:

- Admin

  ↓ inherits
- Manager

  ↓ inherits
- Employee

Result: Admin has all rights of Manager and Employee.

Benefits of Role Hierarchies

- Reduces duplication of permissions.
- Reflects real-world job structures.
- Easier to manage large organizations.
- Enables scalable access control.

Role Hierarchy Explained

- Director sits at the top and inherits all permissions below.
- Project Leads inherit permissions from engineers in their project.
- Production/Quality Engineers hold specialized permissions.
- Engineer roles aggregate permissions from their subroles.
- Engineering Department is the base role for all engineers.
- Permissions flow upward—lower roles → higher roles.

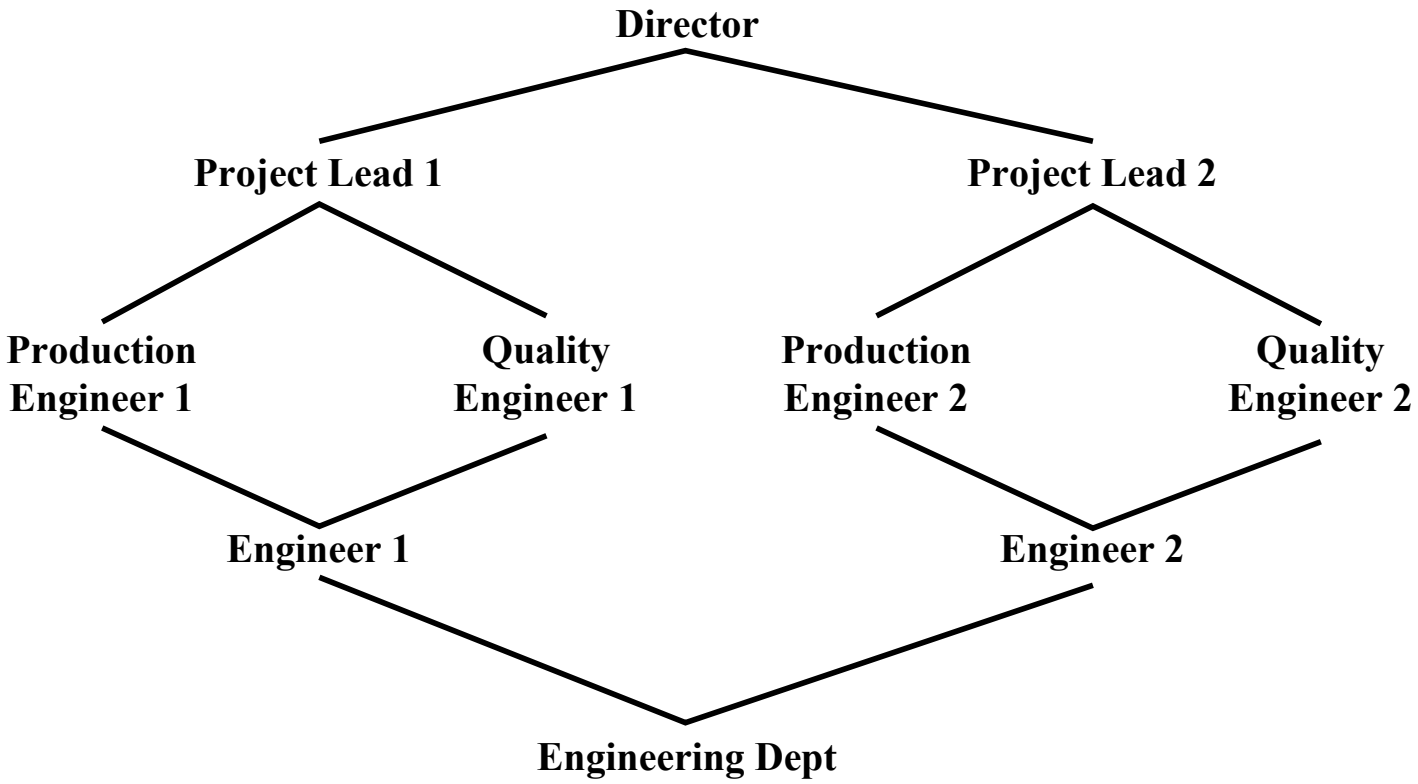


Figure 4.9 Example of Role Hierarchy

RBAC Relationships (UA, PA, RH)

- UA — User Assignment: User ↔ Role mapping
- PA — Permission Assignment: Role ↔ Permission mapping
- RH — Role Hierarchy: Senior roles inherit junior role permissions
- Sessions enable temporary activation of roles at login

Static vs Dynamic Separation of Duty (SoD)

• Separation of Duty (SoD): prevents excessive privilege by dividing critical tasks.

Static Separation of Duty (SSoD):

– Restricts role assignment.

– A user CANNOT be assigned to conflicting roles at the same time.

– Example: A user cannot be both 'Accountant' and 'Auditor'.

Dynamic Separation of Duty (DSoD):

– Restricts role activation within a session.

– A user may hold conflicting roles but CANNOT activate them simultaneously.

– Example: User has both roles but can only activate ONE per login session.

RBAC Model Comparison Table

- $RBAC_0$: Base model with users, roles, permissions — no hierarchies or constraints.
- $RBAC_1$: Adds role hierarchies (senior roles inherit junior permissions).
- $RBAC_2$: Adds constraints such as Separation of Duty (SoD).
- $RBAC_3$: Consolidated model combining $RBAC_1$ + $RBAC_2$.

4.5.1.3 Constraints in RBAC ($RBAC_2$)

- Constraints adapt RBAC to specific administrative and security policies.
- A constraint is a defined relationship among roles or a condition related to roles.
- Common types of constraints:
  - Mutually exclusive roles
  - Cardinality constraints
  - Prerequisite roles

4.5.1.3.1 Mutually Exclusive Roles (MER) (1/2)

- User can only be assigned to one role in a mutually exclusive set.
- Supports separation of duties and reduces risk of fraud or abuse.
- Example:
  - Payment Requester vs Payment Approver
  - One person cannot request AND approve payments.

MER – Permissions (2/2)

- Roles in a mutually exclusive set should not have overlapping permissions.
- If two users are assigned different roles in the set, they must have non-overlapping permissions.
- This reduces the chance of collusion or a single user gaining excessive power.

4.5.1.3.2 Cardinality Constraints

- Limit the maximum number of users for a given role.
- Or limit the maximum number of roles a user can activate in a session.
- Example:
  - Only one Project Lead per project.
  - A user can activate at most 3 roles in one session.

4.5.1.3.3 Prerequisite Roles

- A user can be assigned to a role only if they already hold some other role(s).
- Used to implement least privilege and role hierarchy.
- Example:
  - Must be Engineer before becoming Senior Engineer.
  - Project Lead requires holding Production Engineer or Quality Engineer role first.

Combined Example Scenario

- MER: Accountant and Auditor cannot be assigned to the same user.
- Cardinality: Only one Project Lead per project.
- Prerequisite: Engineer → Senior Engineer; Production/Quality Engineer → Project Lead.
- Together, these constraints enforce separation of duties and limit powerful permissions.

4.6 Attribute-based Access Control

Attribute-Based Access Control (ABAC)

Can define authorizations that express conditions on properties of both the resource and the subject

Strength is its flexibility and expressive power

Main obstacle to its adoption in real systems has been concern about the performance impact of evaluating predicates on both resource and user properties for each access

Web services have been pioneering technologies through the introduction of the eXtensible Access Control Markup Language (XAMCL)

There is considerable interest in applying the model to cloud services

Attribute-Based Access Control (ABAC)

- ABAC uses attributes of subjects, objects, and environment to make access decisions.
- Policies evaluate these attributes to determine permission.
- Highly flexible and expressive.
- Commonly used in cloud, APIs, and distributed systems.

Why ABAC?

- Can express complex and fine-grained rules.
- Evaluates conditions on both user and resource.
- Performance is acceptable due to modern computing power.
- Supports dynamic scenarios like cooperative web services.

4.6.1 Attributes

Key Elements of ABAC

- Attributes – properties describing subject, object, or environment.
- Policy Model – defines ABAC rules.
- Architecture Model – enforces access control using these rules.

Subject Attributes

- • A subject is an active entity that causes information to flow among objects or changes the system state
- Attributes define the identity and characteristics of the subject
- Describe the entity initiating the request (user or process).
- Examples:
  - Identifier, name, organization
  - Job title, role, clearance level
- Used to characterize the subject making the request.

Object Attributes

- An object (or resource) is a passive information system-related entity containing or receiving information
- Objects have attributes that can be leverages to make access control decisions
- Describe the resource being accessed.
- Examples:
  - File type, owner, classification
  - Metadata like subject, date, author
- Important for fine-grained content control.

Environment Attributes

- Describe the operational, technical, and even situational environment or context in which the information access occurs
- These attributes have so far been largely ignored in most access control policies
- Describe the context of the access attempt.
- Examples:
  - Current time and date
  - Network security level (internal/external)
  - Device trust level or threat conditions
- Often ignored historically but vital for cloud and distributed systems.

ABAC Rule Example

- Example:

  "Allow access if user.department = 'Finance'
  AND document.type = 'Report'
  AND current_time < 20:00
  AND network = 'Internal'."

ABAC

Distinguishable because it controls access to objects by evaluating rules against the attributes of entities, operations, and the environment relevant to a request

Relies upon the evaluation of attributes of the subject, attributes of the object, and a formal relationship or access control rule defining the allowable operations for subject-object attribute combinations in a given environment

Systems are capable of enforcing DAC, RBAC, and MAC concepts

Allows an unlimited number of attributes to be combined to satisfy any access control rule

4.6.2 ABAC logical Architecture

ABAC (Attribute-Based Access Control) Scenario

- ABAC uses attributes to decide access instead of roles or identities.
- Subject Attributes: user details (name, clearance, affiliation, etc.).
- Object Attributes: resource characteristics (type, owner, classification).
- Environmental Attributes: context factors (time, location, security state).
- Policies define allowed combinations of attributes.
- Access Mechanism evaluates attributes and issues Permit or Deny.

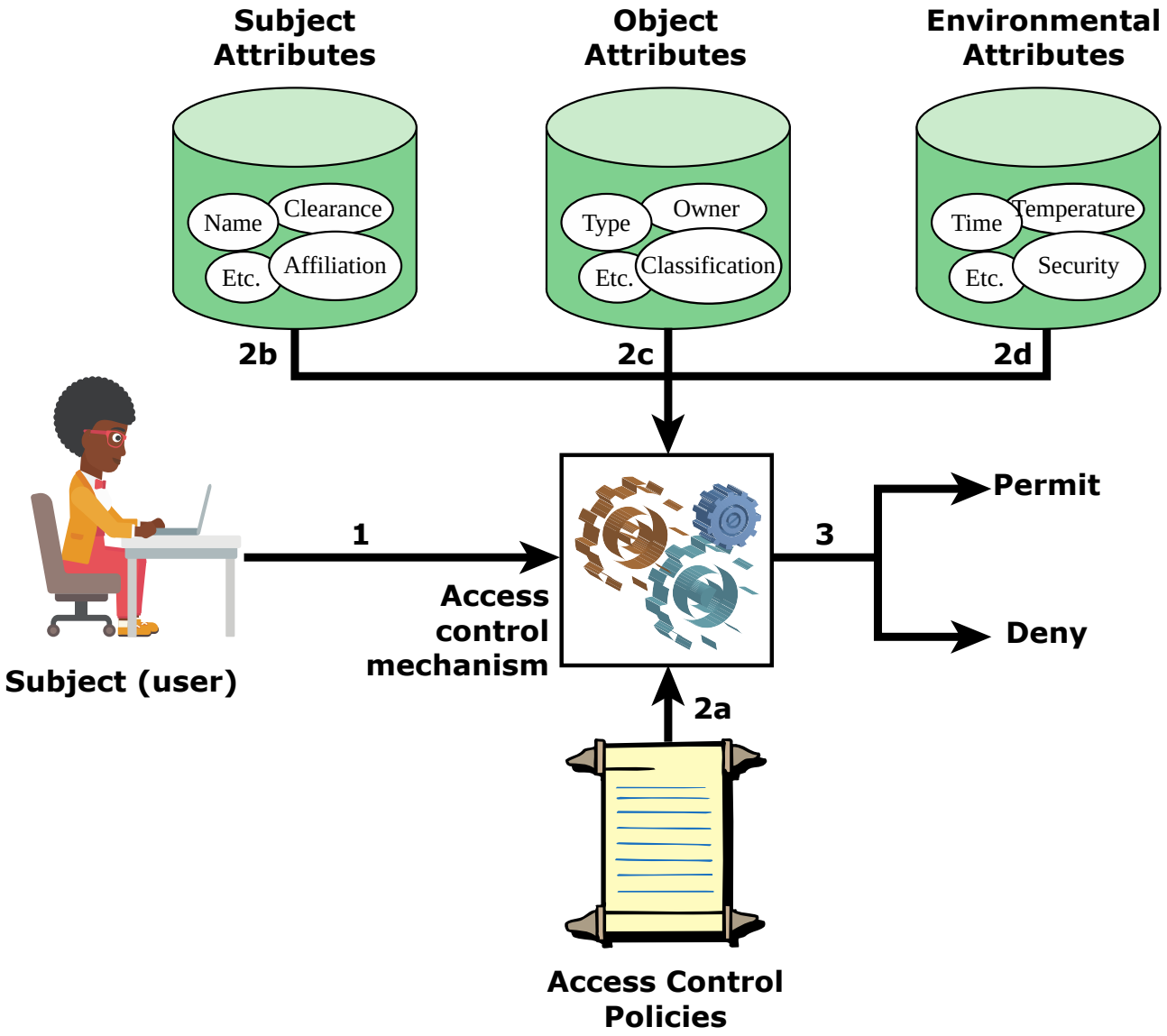
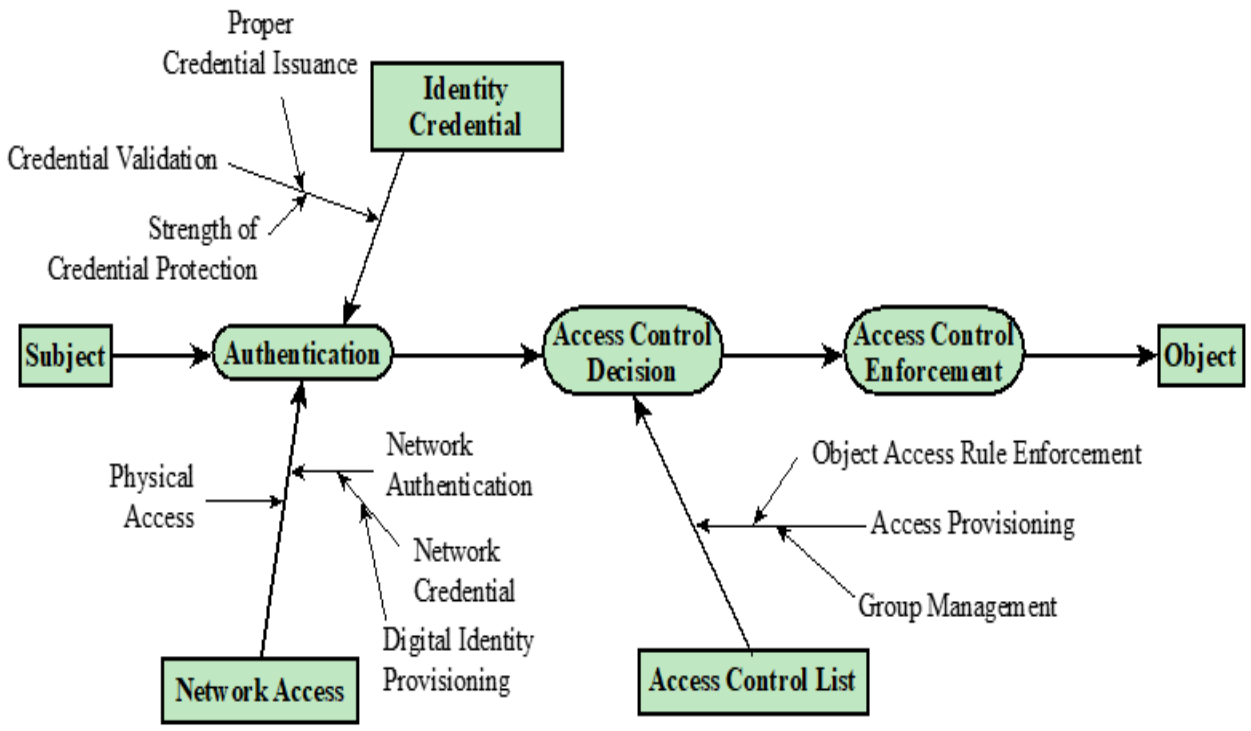


Figure 4.10 ABAC Scenario

ACL Trust Chain Explained

- Subject presents identity credentials.
- Authentication verifies credentials (passwords, tokens, etc.).
- Network Access grants connectivity after identity validation.
- Access Control Decision uses Access Control Lists (ACLs) to determine rights.
- Access Control Enforcement applies the decision to objects.
- Trust depends on valid credentials, accurate ACLs, and correct enforcement.

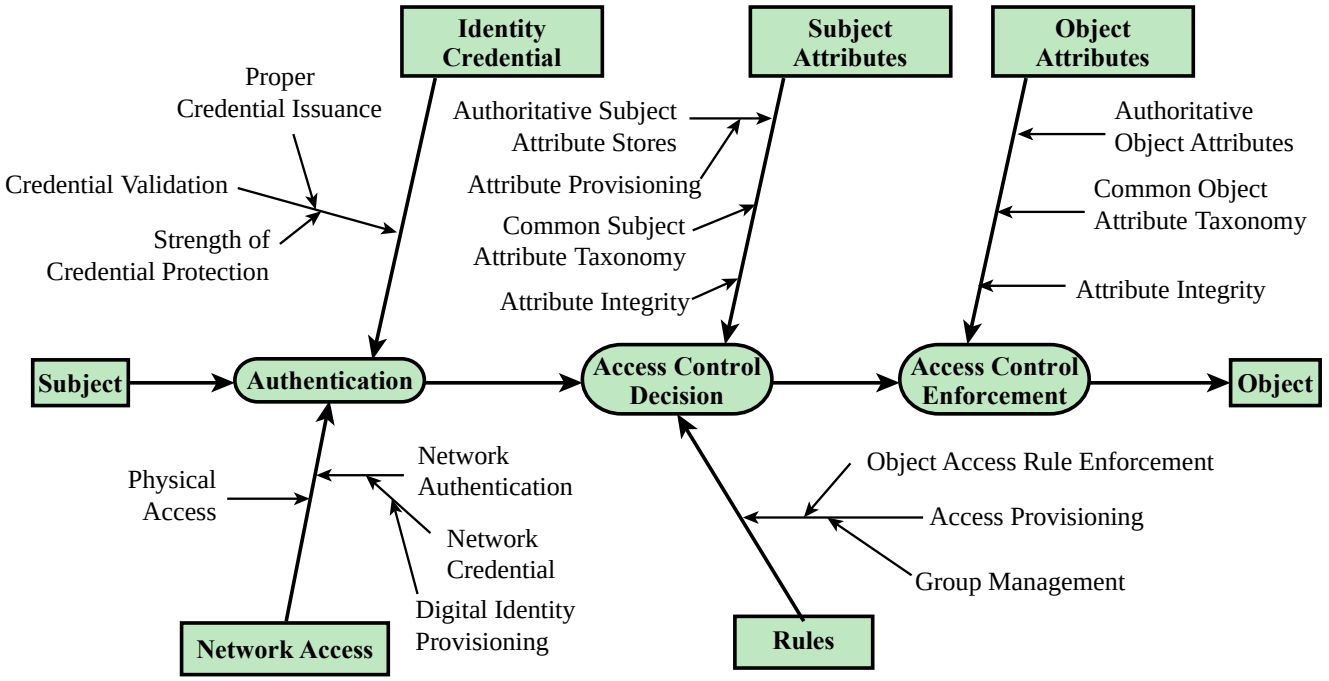


(a) ACL Trust Chain

Figure 4.11 ACL and ABAC Trust Relationships

ABAC Trust Chain Explained

- Subject authenticates using identity credentials.
- Network Access is granted after successful authentication.
- Subject, Object, and Environmental Attributes are retrieved.
- Attributes come from authoritative, trusted attribute stores.
- Access Control Decision evaluates ABAC rules using these attributes.
- Rules define how attribute combinations allow or deny access.
- Access Control Enforcement applies the decision to the object.
- Trust relies on: identity integrity, attribute integrity, rule correctness, enforcement.



(b) ABAC Trust Chain

Figure 4.11 ACL and ABAC Trust Relationships

4.6.3 ABAC Policies

A policy is a set of rules and relationships that govern allowable behavior within an organization, based on the privileges of subjects and how resources or objects are to be protected under which environment conditions

Typically written from the perspective of the object that needs protecting and the privileges available to subjects

Privileges represent the authorized behavior of a subject and are defined by an authority and embodied in a policy

Other terms commonly used instead of privileges are: rights, authorizations, and entitlements

ABAC vs RBAC vs DAC vs MAC

- DAC: Owner-controlled; access based on object owner's discretion.
- MAC: System-controlled; uses strict labels and classifications.
- RBAC: Role-based; users inherit permissions through assigned roles.
- ABAC: Attribute-based; highly flexible and context-aware.
- • Summary: DAC = simplest; MAC = strictest; RBAC = most scalable; ABAC = most flexible.

4.7 Identity, Credential, and Access Management (ICAM)

Identity, Credential, and Access Management (ICAM)

- A comprehensive approach to managing and implementing digital identities, credentials, and access control
- Developed by the U.S. government
- Designed to:
  - Create trusted digital identity representations of individuals and nonperson entities (NPEs)
  - Bind those identities to credentials that may serve as a proxy for the individual of NPE in access transactions
    - A credential is an object or data structure that authoritatively binds an identity to a token possessed and controlled by a subscriber
  - Use the credentials to provide authorized access to an agency's resources

ICAM Diagram Explained

- Identity Management verifies user identity and manages attributes.
- Credential Management issues and maintains trusted credentials.
- Access Management controls physical and logical access based on policies.
- Provisioning/Deprovisioning synchronizes access with identity changes.
- Identity Federation enables access for trusted external entities.

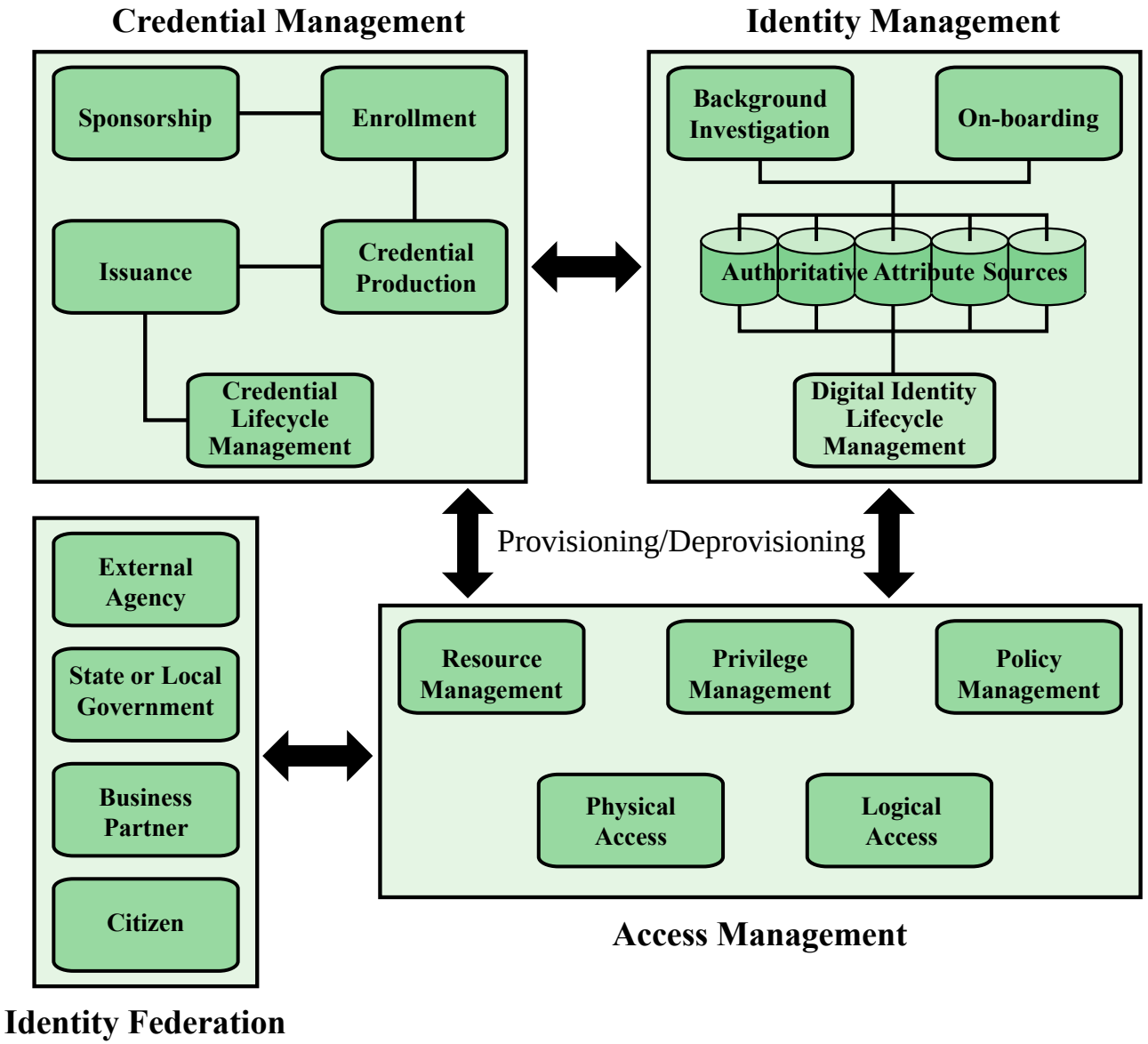
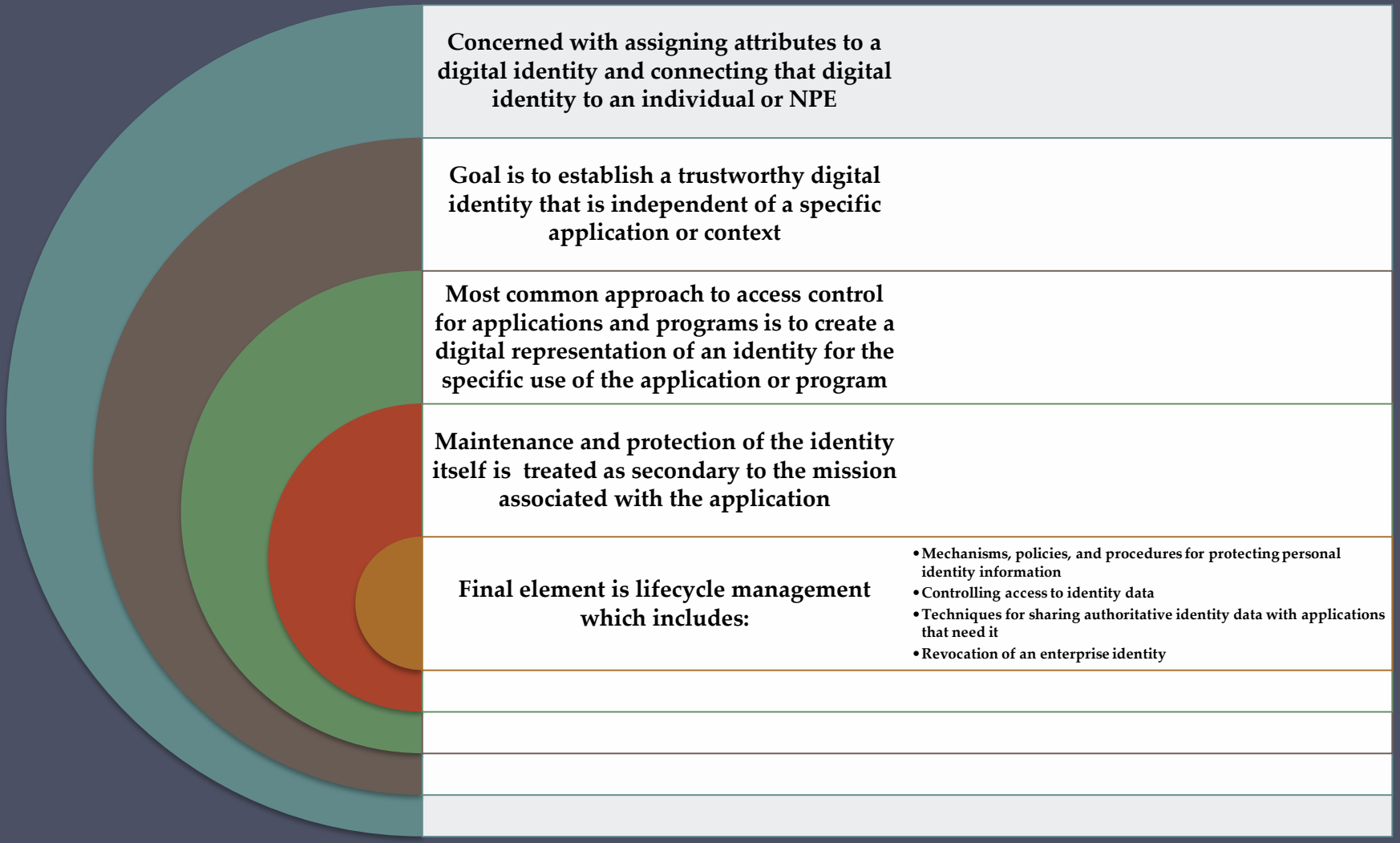


Figure 4.12 Identity, Credential, and Access Management (ICAM)

4.7.1 Identity Management



Credential Management

The management of the life cycle of the credential

Examples of credentials are smart cards, private/public cryptographic keys, and digital certificates

Encompasses five logical components:

An authorized individual sponsors an individual or entity for a credential to establish the need for the credential

The sponsored individual enrolls for the credential

- Process typically consists of identity proofing and the capture of biographic and biometric data
- This step may also involve incorporating authoritative attribute data, maintained by the identity management component

A credential is produced

- Depending on the credential type, production may involve encryption, the use of a digital signature, the production of a smart card or other functions

The credential is issued to the individual or NPE

A credential must be maintained over its life cycle

- Might include revocation, reissuance/replacement, reenrollment, expiration, personal identification number (PIN) reset, suspension, or reinstatement

Access Management

Deals with the management and control of the ways entities are granted access to resources

Covers both logical and physical access

May be internal to a system or an external element

Purpose is to ensure that the proper identity verification is made when an individual attempts to access a security sensitive building, computer systems, or data

Three support elements are needed for an enterprise-wide access control facility:

- Resource management
- Privilege management
- Policy management

Three support elements are needed for an enterprise-wide access control facility:

Resource management

- Concerned with defining rules for a resource that requires access control
- Rules would include credential requirements and what user attributes, resource attributes, and environmental conditions are required for access of a given resource for a given function

Privilege management

- Concerned with establishing and maintaining the entitlement or privilege attributes that comprise an individual's access profile
- These attributes represent features of an individual that can be used as the basis for determining access decisions to both physical and logical resources
- Privileges are considered attributes that can be linked to a digital identity

Policy management

- Governs what is allowable and unallowable in an access transaction

Identity Federation (1/2)

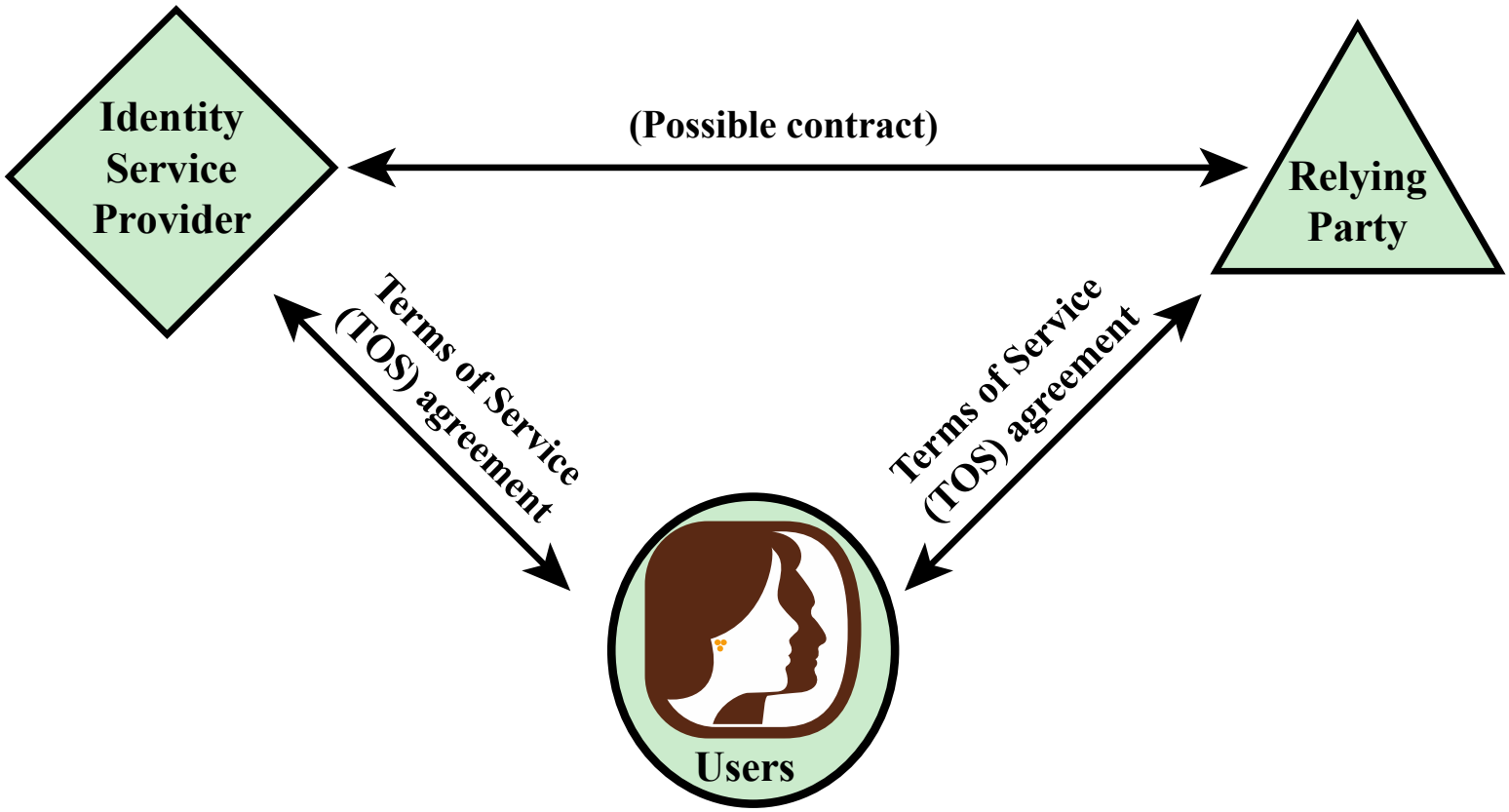
- Term used to describe the technology, standards, policies, and processes that allow an organization to trust digital identities, identity attributes, and credentials created and issued by another organization
- Addresses two questions:
  - How do you trust identities of individuals from external organizations who need access to your systems
  - How do you vouch for identities of individuals in your organization when they need to collaborate with external organizations

Identity Federation (2/2)

- Allows external entities to use their own identity systems.
- Supports collaboration with agencies, partners, and citizens.
- Reduces need to create duplicate identities in each system.
- Relies on trust frameworks and standardized authentication protocols.
- Enables secure cross-organization access management.

4.8 Trust Frameworks

4.8.1 Traditional Triangle of Identity Exchange Approach



(a) Traditional triangle of parties involved in an exchange of identity information

Figure 4.13 Identity Information Exchange Approaches

Identity Service Provider (IdP)

- Manages and stores user identities.
- Authenticates users and issues credentials.
- Provides identity information to relying parties.
- May or may not have a formal contract with the relying party.
- Examples:
  - Google
  - Facebook
  - University login system
  - Corporate authentication provider

Users

- Individuals whose identity is being used.
- Maintain Terms of Service (TOS) with both IdP and RP.
- Trust IdP to manage identity securely.
- Trust RP to use their identity data responsibly.

Relying Party (RP)

- System that requests identity information from IdP.
- Relies on IdP to authenticate the user correctly.
- Uses identity info to grant access or personalize services.

Examples: websites, mobile apps, service portals.

Key Relationships in the Triangle

- User ↔ Identity Provider: governed by Terms of Service.
- User ↔ Relying Party: also governed by TOS agreements.
- Identity Provider ↔ Relying Party: may involve a contract or trust protocol.
- All three rely on accurate identity handling and secure data exchange.

4.8.2 Open Identity Trust Framework

OpenID — Overview

- Open standard for decentralized user authentication.
- Allows users to log in to multiple sites using a single identity.
- Websites that accept OpenID are called Relying Parties.
- Identity Provider (IdP) handles authentication on behalf of the user.
- Reduces password fatigue and eliminates site-specific login systems.
- Users pick their preferred OpenID provider to manage their digital identity.

Examples of OpenID Identity Providers

  - Historically: Google, Yahoo, AOL, MyOpenID.
    Modern equivalents evolved into **OpenID Connect** (e.g., Google Sign-In).

OIDF — OpenID Foundation

- International nonprofit enabling and promoting OpenID technologies.
- Supports interoperability, adoption, and security of OpenID.
- Provides infrastructure and community support.
- Helps guide global identity standards efforts.

ICF — Information Card Foundation

- Nonprofit community supporting the Information Card ecosystem.
- Information Cards are user-controlled digital identities.
- Cards include visual icons and names for easy selection.
- Helps evolve identity metasystems and user-centric identity.

OITF — Open Identity Trust Framework

- Open, standardized trust framework for identity and attribute exchange.
- Jointly developed by OIDF and ICF.
- Defines roles, governance rules, and trust requirements.
- Provides a foundation for interoperable identity systems.

OIX — Open Identity Exchange

- Independent, neutral international organization.
- Certifies trust frameworks using OITF guidelines.
- Ensures frameworks meet privacy and security standards.
- Promotes trusted digital identity ecosystems worldwide.

AXN — Attribute Exchange Network

- Internet-scale gateway for attribute exchange.
- Connects identity service providers and relying parties.
- Enables secure sharing of user-asserted, permissioned attributes.
- Supports scalable, federated identity environments.

Open Identity Trust Framework

OpenID

- An open standard that allows users to be authenticated by certain cooperating sites using a third party service

OIDF

- OpenID Foundation is an international nonprofit organization of individuals and companies committed to enabling, promoting, and protecting OpenID technologies

ICF

- Information Card Foundation is a nonprofit community of companies and individuals working together to evolve the Information Card ecosystem

OITF

- Open Identity Trust Framework is a standardized, open specification of a trust framework for identity and attribute exchange, developed jointly by OIDF and ICF

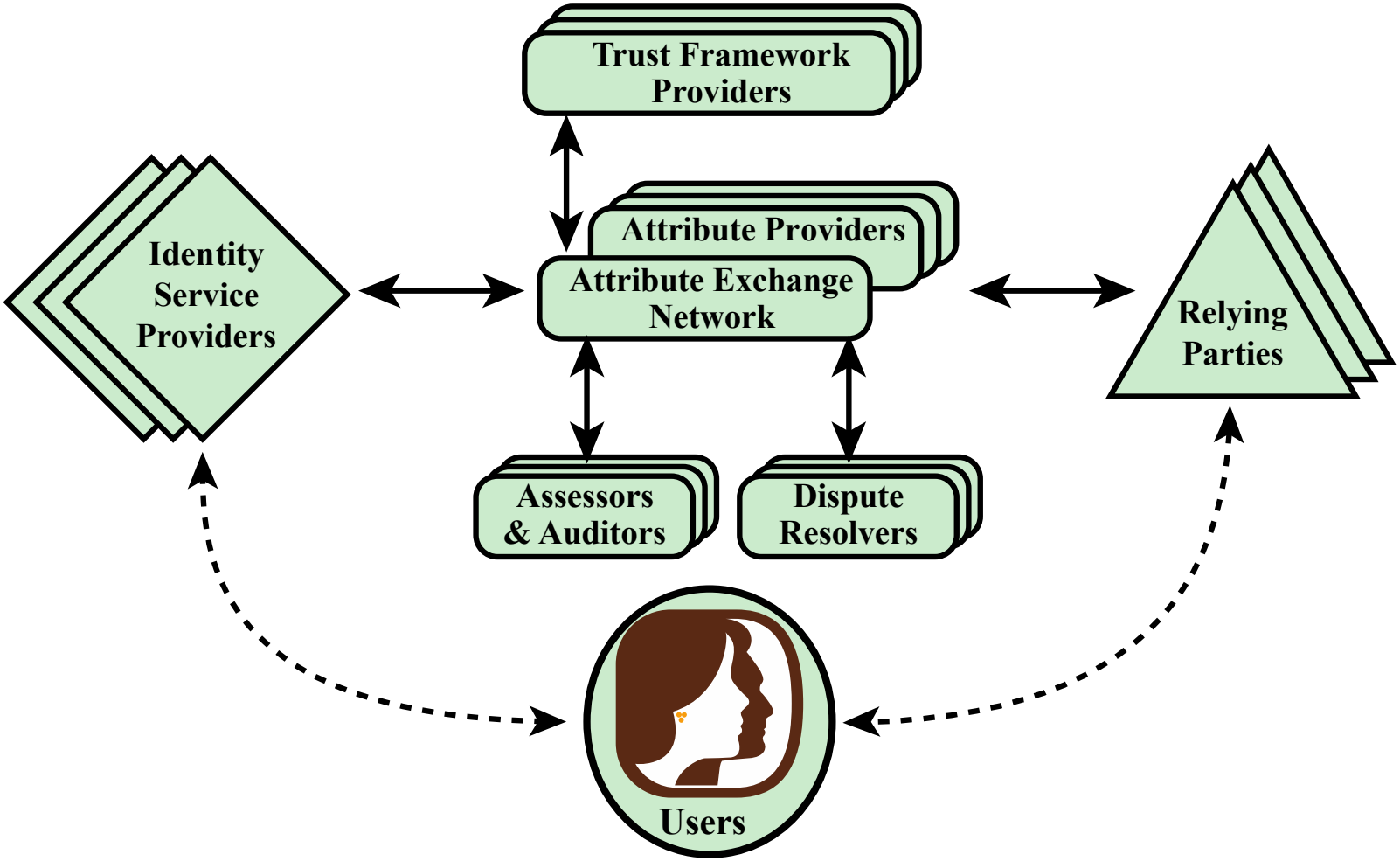
OIX

- Open Identity Exchange Corporation is an independent, neutral, international provider of certification trust frameworks conforming to the OITF model

AXN

- Attribute Exchange Network is an online Internet-scale gateway for identity service providers and relying parties to efficiently access user asserted, permissioned, and verified online identity attributes in high volumes at affordable costs

4.8.2 Identity Attribute Exchange Ecosystem



(B) Identity attribute exchange elements

Figure 4.13 Identity Information Exchange Approaches

Users

- Individuals whose identity attributes are being exchanged.
- Provide consent for sharing personal information.
- Depend on secure, privacy-preserving identity ecosystems.
- Examples: employees, students, citizens, customers.

Identity Service Providers (IdPs)

- Authenticate users and issue verified identity attributes.
- Provide identity assurance and login capabilities.
- First step in trusted identity exchange.
- Examples:

  Google, Microsoft, universities, government identity systems.

Attribute Providers

- Maintain validated attributes about users.
- Provide supplemental verified data (e.g., status, role, address).
- Feed attributes into the Attribute Exchange Network.

Examples:

HR databases, registrars, DMV, healthcare providers.

DMV = Department of Motor Vehicles

Attribute Exchange Network (AXN)

- Internet-scale intermediary enabling secure attribute requests.
- Connects IdPs, Attribute Providers, and Relying Parties.
- Ensures consent, privacy, and standardized attribute formatting.
- Core routing and policy enforcement for attribute exchange.

Trust Framework Providers

- Define rules, policies, and governance for identity ecosystems.
- Specify assurance levels and identity proofing standards.
- Ensure interoperability and trustworthiness among participants.
- Examples:

  OIX, Kantara Initiative.

Relying Parties (RPs)

- Services or systems that request attributes for access decisions.
- Depend on trusted identity and attribute sources.
- Use attributes to authorize or personalize user interactions.
- Examples:
- websites, banks, government portals, enterprise apps.

Assessors & Auditors

- Evaluate compliance with trust framework requirements.
- Verify identity proofing and attribute integrity controls.
- Ensure ongoing security and policy adherence.
- Provide accountability across the ecosystem.

Dispute Resolvers

- Handle identity-related conflicts, errors, and complaints.
- Resolve misidentification, incorrect attributes, or misuse.
- Provide legal and operational fairness for users and organizations.
- Maintain trust in the identity exchange ecosystem.

Examples of Dispute Resolvers

- Identity Governance Offices (university, corporate, government).
- Privacy or Compliance Ombudsman Offices.
- Third-party arbitration services (OIX/Kantara certified).
- Regulatory agencies (FTC, Data Protection Authorities).
- Identity Provider dispute channels (Google, Microsoft, Govt eID).
- Relying Party dispute desks (banks, healthcare, insurance).

Federal Trade Commission (FTC)

Identity Attribute Exchange — Summary

- Users authenticate through Identity Service Providers.
- Relying Parties request attributes via the AXN.
- Attribute Providers supply verified data.
- Trust Frameworks regulate the flow and security of information.
- Assessors and Dispute Resolvers maintain integrity and fairness.
- Together form a trusted, scalable identity exchange ecosystem.

4.9 Case Study: RBAC System For A Bank

How RBAC Works in This Banking Case Study

- Roles (A, B, C, ...) represent job categories in the bank.
- Each role is linked to a function (financial analyst, technician, etc.).
- Official position determines seniority and responsibility.
- Permissions depend on the financial applications each role can access.
- Inheritance allows senior roles to automatically gain lower-level permissions.

Table 4.5
Functions and Roles for Banking Example

(a) Functions and Official Positions

Role	Function	Official Position
A	financial analyst	Clerk
B	financial analyst	Group Manager
C	financial analyst	Head of Division
D	financial analyst	Junior
E	financial analyst	Senior
F	financial analyst	Specialist
G	financial analyst	Assistant
•••	•••	•••
X	share technician	Clerk
Y	support e-commerce	Junior
Z	office banking	Head of Division

Table 4.5
Functions and Roles for Banking Example

(b) Permission Assignments

Role	Application	Access Right
A	money market instruments	1, 2, 3, 4
	derivatives trading	1, 2, 3, 7, 10, 12
	interest instruments	1, 4, 8, 12, 14, 16
B	money market instruments	1, 2, 3, 4, 7
	derivatives trading	1, 2, 3, 7, 10, 12, 14
	interest instruments	1, 4, 8, 12, 14, 16
	private consumer instruments	1, 2, 4, 7
• • •	• • •	• • •

(c) PA with Inheritance

Role	Application	Access Right
A	money market instruments	1, 2, 3, 4
	derivatives trading	1, 2, 3, 7, 10, 12
	interest instruments	1, 4, 8, 12, 14, 16
B	money market instruments	7
	derivatives trading	14
	private consumer instruments	1, 2, 4, 7
• • •	• • •	• • •

Principle of Least Privilege (POLP)

- Users receive only the minimum permissions required to perform their job.
- Reduces blast radius of compromised accounts.
- Limits access misuse—intentional or accidental.
- Essential for financial institutions (high-risk operations).

Role Hierarchy Example (Bank)

- Head of Division → Group Manager → Senior Analyst → Junior Analyst → Clerk
- A simple hierarchy:
  - A (Clerk)
  - B (Group Manager inherits A)
  - C (Head of Division inherits A and B)

Static vs Dynamic Separation of Duty (SoD)

Static SoD:

- Prevents conflicting roles from being assigned to the same user.
- Example:

  A user cannot be both ‘Financial Analyst’ and ‘Auditor’.

Dynamic SoD:

- User may hold multiple roles but cannot activate conflicting ones in the same session.
- Example:

  User with Analyst + Approver role cannot approve their own transaction.

Access Control Decision Walkthrough (1/3)

- Step 1: User attempts to access derivatives trading system.
  - System identifies user's role (e.g., Role B: Group Manager).
  - System loads all permissions assigned to Role B.

Access Control Decision Walkthrough (2/3)

- Step 2: Apply role inheritance:
  - Role B inherits Role A permissions.
  - Combined permission set: {1,2,3,7,10,12,14} + inherited {1,2,3,4}.
  - System merges rights, removes duplicates.

Access Control Decision Walkthrough (3/3)

- Step 3: Enforce Separation of Duty and Least Privilege:
  - Check for any SoD conflicts before granting access.
  - Verify only necessary permissions are activated.
  - Final decision: Permit or Deny based on conflict + permissions.

Human Resources → Roles, Functions, Positions

- HR assigns User IDs, job positions, and functions.
- Bank Example: Clerk, Junior Analyst, Senior Analyst, Group Manager, Head of Division.
- HR maps: User → Position → Function → Role.
- Example:
  - A = Clerk (Financial Analyst), B = Group Manager, C = Head of Division.

Application Administration → Applications & Access Rights

- Each banking application defines its own access rights.
- Rights (e.g., 1,2,3,7,10,12) represent actions the role can perform.
- This matches ‘Application ↔ Access Right’ in the diagram.
- Examples:
  - Money Market Instruments, Derivatives Trading, Interest Instruments.

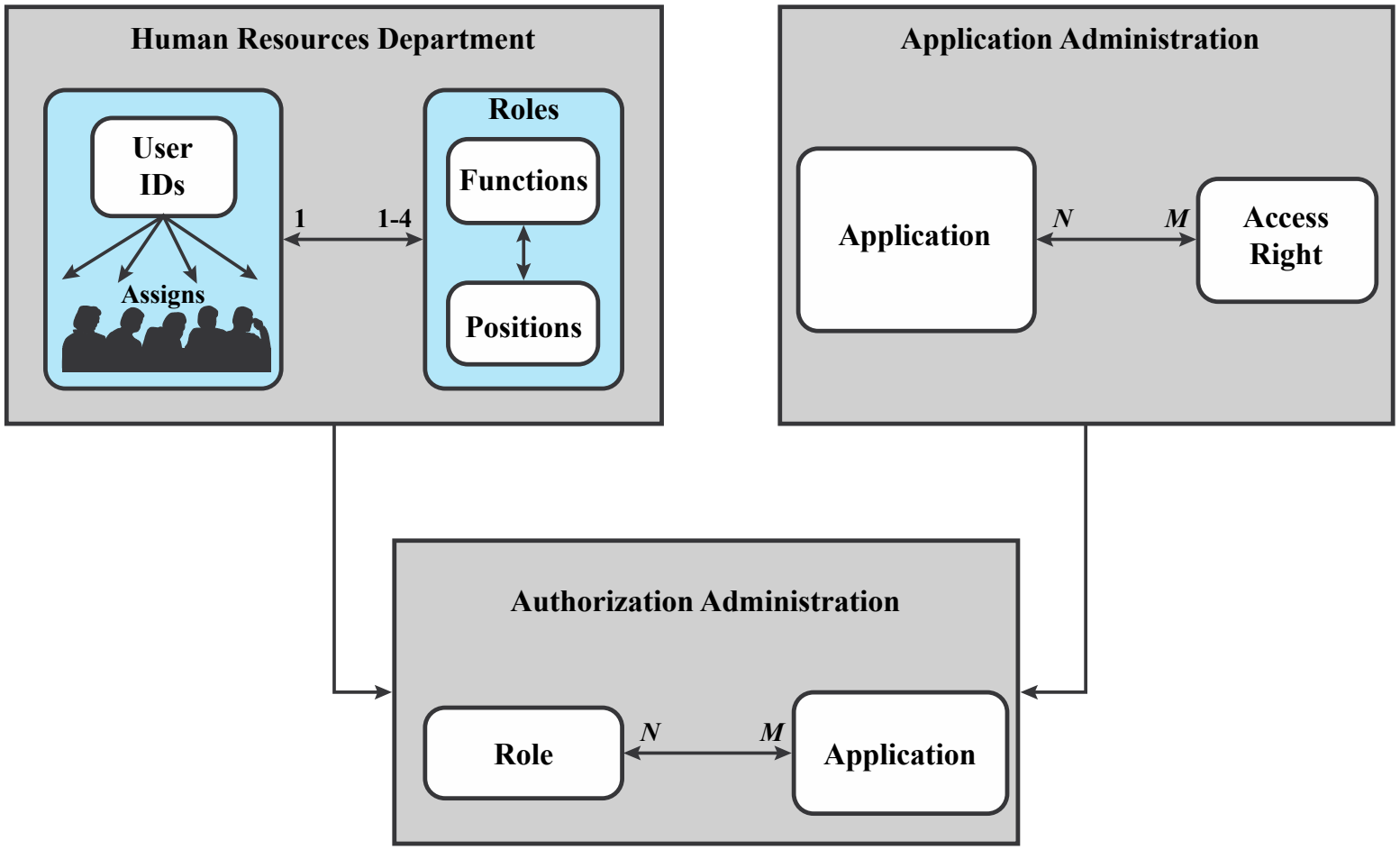


Figure 4.14 Example of Access Control Administration

Authorization Administration → Linking Roles to Applications

- Authorization Admin assigns permissions to roles.
- Role A, B, C inherit rights (e.g., B inherits A).
- Defines which roles access which banking systems.
- Matches 'Role ↔ Application' in diagram.

Mapping Bank RBAC Case Study

- Figure 4.14 Mapping:
  - 1. HR: User → Function → Position → Role (e.g., Sarah → Senior Analyst → Role E).
  - 2. Applications define rights (e.g., Derivatives Trading → Rights 1,2,3,7,10,12).
  - 3. Authorization Admin ties Role E to apps and rights.
  - 4. User login: System checks role, permissions, SoD, and least privilege.

Summary

- Access control principles
  - Access control context
  - Access control policies
- Subjects, objects, and access rights
- Discretionary access control
  - Access control model
  - Protection domains
- UNIX file access control
  - Traditional UNIX file access control
  - Access control lists in UNIX
- Role-based access control
  - RBAC reference models
- Attribute-based access control
  - Attributes
  - ABAC logical architecture
  - ABAC policies
- Identity, credential, and access management
  - Identity management
  - Credential management
  - Access management
  - Identity federation
- Trust frameworks
  - Traditional identity exchange approach
  - Open identity trust framework
- Bank RBAC system