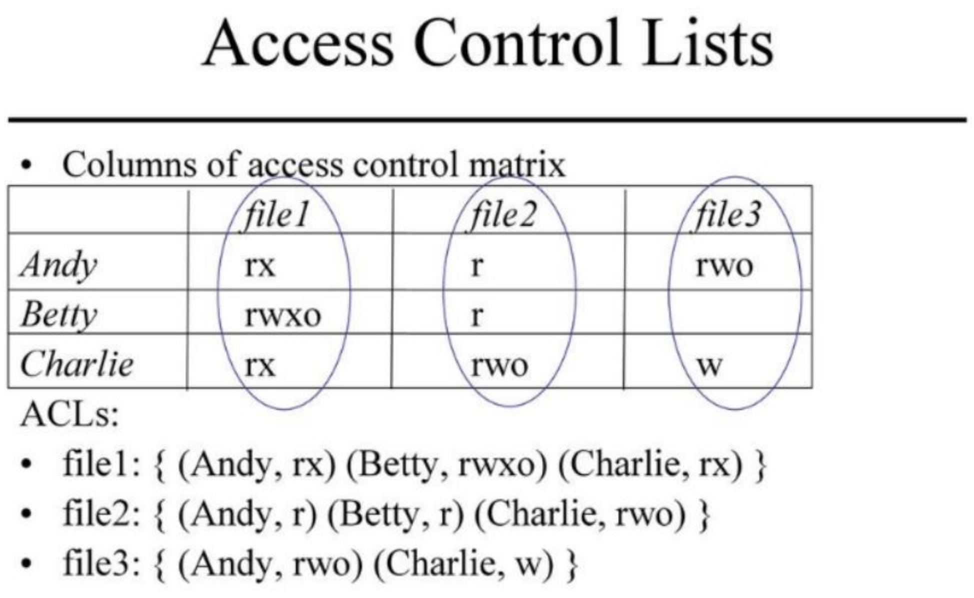
Access Control Mechanisms

- Access Control Lists
- Capability Lists
- Locks and Keys
- Rings-based Access Control
- Propagated Access Control Lists

Access Control Lists

An obvious variant of the access control matrix is to store each column with the object it represents. Thus, each object has associated with it a set of pairs, with each pair containing a subject and a set of rights. The named subject can access the associated object using any of those rights.

Let S be the set of subjects, and R the set of rights, of a system. An *access control list* (ACL) l is a set of pairs $l = \{ (s, r) : s \in S, r \subseteq R \}$. Let *acl* be a function that determines the access control list l associated with a particular object o. The interpretation of the access control list $acl(o) = \{ (s_i, r_i) : 1 \leq i \leq n \}$ is that subject s_i may access o using any right in r_i.



Access Control Lists

- Columns of access control matrix

	file1	*file2*	*file3*
Andy	rx	r	rwo
Betty	rwxo	r	
Charlie	rx	rwo	w

ACLs:

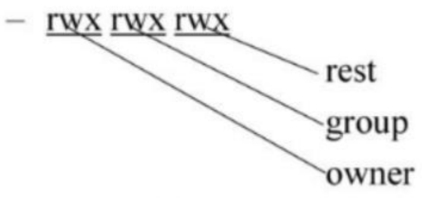
- file1: { (Andy, rx) (Betty, rwxo) (Charlie, rx) }
- file2: { (Andy, r) (Betty, r) (Charlie, rwo) }
- file3: { (Andy, rwo) (Charlie, w) }

One issue is the matter of default permission. If a subject is not named in the ACL, it has no rights over the associated object. On a system with many subjects, the ACL may be very large. If many subjects have the same right over the file, one could define a “wildcard” to match any unnamed subjects, and give them default rights.

- If many subjects, may use groups or wildcards in ACL
  - UNICOS: entries are (*user*, *group*, *rights*)
    - If *user* is in *group*, has rights over file
    - ‘*’ is wildcard for *user*, *group*
      - (holly, *, r): holly can read file regardless of her group
      - (*, gleep, w): anyone in group gleep can write file

Abbreviations of Access Control Lists

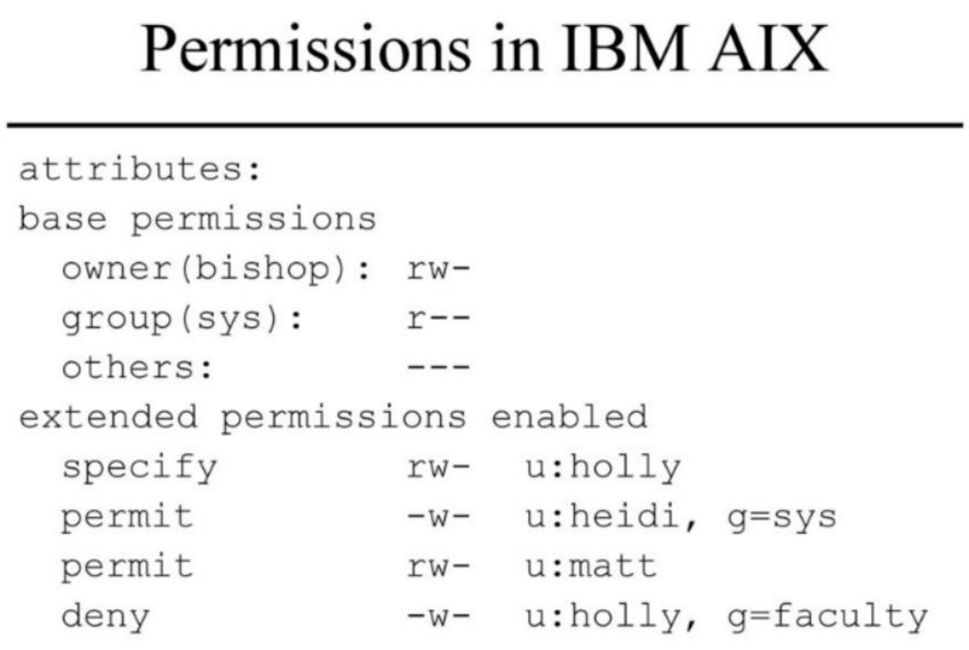
Some systems abbreviate access control lists. The basis for file access control in the UNIX operating system is of this variety. UNIX systems divide the set of users into three classes: the owner of the file, the group owner of the file, and all other users. Each class has a separate set of rights.



- Ownership assigned based on creating process
  - Some systems: if directory has setgid permission, file group owned by group of directory (SunOS, Solaris)

Abbreviations of access control lists, such as those supported by the UNIX operating system, suffer from a loss of granularity.
Many systems augment abbreviations of ACLs with full-blown ACLs. This scheme uses the abbreviations of ACLs as the default permission controls; the explicit ACL overrides the defaults as needed. The exact method varies.

EXAMPLE:
IBM’s version of the UNIX operating system, called AIX, uses an ACL (called “extended permissions”) to augment the traditional UNIX abbreviations of ACL (called “base permissions”)


Permissions in IBM AIX

```
attributes:
base permissions
  owner(bishop): rw-
  group(sys):    r--
  others:        ---
extended permissions enabled
  specify        rw-  u:holly
  permit         -w-  u:heidi, g=sys
  permit         rw-  u:matt
  deny           -w-  u:holly, g=faculty
```

Creation and Maintenance of Access Control Lists

Specific implementations of ACLs differ in details. Some of the issues are as follows.

1. Which subjects can modify an object's ACL?
2. If there is a privileged user (such as root in the UNIX system or administrator in Windows NT), do the ACLs apply to that user?
3. Does the ACL support groups or wildcards (that is, can users be grouped into sets based on a system notion of "group" or on pattern matching)?
4. How are contradictory access control permissions handled? If one entry grants read privileges only and another grants write privileges only, which right does the subject have over the object?
5. If a default setting is allowed, do the ACL permissions modify it, or is the default used only when the subject is not explicitly mentioned in the ACL?

❖ Which Subjects Can Modify an Object's ACL?

When an ACL is created, rights are instantiated. Chief among these rights is the one we will call own. Possessors of the own right can modify the ACL. Creating an object also creates its ACL, with some initial value.

By convention, the subject with own rights is allowed to modify the ACL. However, some systems allow anyone with access to manipulate the rights.

❖ Do the ACLs Apply to a Privileged User?

Many systems have users with extra privileges. The two best known are the root superuser on UNIX systems and the administrator user on Windows NT and 2000 systems. Typically, ACLs (or their degenerate forms) are applied in a limited fashion to such users.

❖ Do the ACLs Apply to a Privileged User?

Many systems have users with extra privileges. The two best known are the root superuser on UNIX systems and the administrator user on Windows NT and 2000 systems. Typically, ACLs (or their degenerate forms) are applied in a limited fashion to such users.

❖ Does the ACL Support Groups and Wildcards?

In its classic form, ACLs do not support groups or wildcards. In practice, systems support one or the other (or both) to limit the size of the ACL and to make manipulation of the lists easier. A group can either refine the characteristics of the processes to be allowed access or be a synonym for a set of users (the members of the group).

- AIX: base perms gave group sys read only

```
permit    -w-    u:heidi, g=sys
```

  line adds write permission for heidi when in that group
- UNICOS:
  - holly : gleep : r
    - user holly in group gleep can read file
  - holly : * : r
    - user holly in any group can read file
  - * : gleep : r
    - any user in group gleep can read file

❖ **Conflicts**

A conflict arises when two access control list entries in the same ACL give different permissions to the subject. The system can allow access if any entry would give access, deny access if any entry would deny access, or apply the first entry that matches the subject.

❖ **ACLs and Default Permissions**

When ACLs and abbreviations of access control lists or default access rights coexist (as on many UNIX systems), there are two ways to determine access rights. The first is to apply the appropriate ACL entry, if one exists, and to apply the default permissions or abbreviations of access control lists otherwise. The second way is to augment the default permissions or abbreviations of access control lists with those in the appropriate ACL entry.

Revocation of Rights

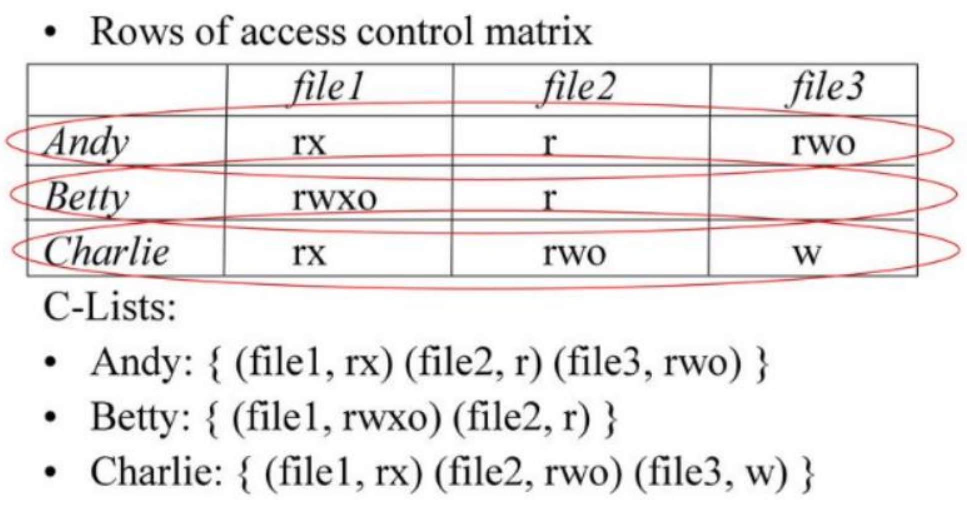
Revocation, or the prevention of a subject's accessing an object, requires that the subject's rights be deleted from the object's ACL. Preventing a subject from accessing an object is simple. The entry for the subject is deleted from the object's ACL. If only specific rights are to be deleted, they are removed from the relevant subject's entry in the ACL. If ownership does not control the giving of rights, revocation is more complex.

Capabilities

Conceptually, a capability is like the row of an access control matrix. Each subject has associated with it a set of pairs, with each pair containing an object and a set of rights. The subject associated with this list can access the named object in any of the ways indicated by the named rights.

Let O be the set of objects, and R the set of rights, of a system. A *capability list* c is a set of pairs $c = \{ (o, r) : o \in O, r \subseteq R \}$. Let *cap* be a function that determines the capability list c associated with a particular subject s. The interpretation of the capability list $cap(s) = \{ (o_i, r_i) : 1 \le i \le n \}$ is that subject s may access o_i using any right in r_i.

"Capability list" is abbreviated as C-List.



- Rows of access control matrix

	file1	*file2*	*file3*
Andy	rx	r	rwo
Betty	rwxo	r	
Charlie	rx	rwo	w

C-Lists:

- Andy: { (file1, rx) (file2, r) (file3, rwo) }
- Betty: { (file1, rwxo) (file2, r) }
- Charlie: { (file1, rx) (file2, rwo) (file3, w) }

Implementation of Capabilities

Three mechanisms are used to protect capabilities: tags, protected memory, and cryptography. A tagged architecture has a set of bits associated with each hardware word. The tag has two states: set and unset. If the tag is set, an ordinary process can read but not modify the word. If the tag is unset, an ordinary process can read and modify the word. Further, an ordinary process cannot change the state of the tag; the processor must be in a privileged mode to do so.

More common is to use the protection bits associated with paging or segmentation. All capabilities are stored in a page (segment) that the process can read but not alter. A third alternative is to use cryptography. The goal of tags and memory protection is to prevent the capabilities from being altered. This is akin to integrity checking.

Cryptographic checksums are another mechanism for checking the integrity of information. Each capability has a cryptographic checksum associated with it, and the checksum is digitally enciphered using a cryptosystem whose key is known to the operating system.

Copying and Amplifying Capabilities

The ability to copy capabilities implies the ability to give rights. To prevent processes from indiscriminately giving away rights, a copy flag is associated with capabilities. A process cannot copy a capability to another process unless the copy flag is set. If the process does copy the capability, the copy flag may be turned off.

Amplification is the increasing of privileges. The idea of modular programming, and especially of abstract data types, requires that the rights a process has over an object be amplified.

Revocation of Rights

In a capability system, revoking access to an object requires that all the capabilities granting access to that object be revoked. Conceptually, each process could be checked, and the capabilities deleted.

The simplest mechanism is indirection. Define one or more global object tables. In this scheme, each object has a corresponding entry in a table. Capabilities do not name the object directly; they name the entry in the table corresponding to the object.

This scheme has several advantages.

- First, to revoke capabilities, the entry in the global object table is invalidated. Then any references will obtain an invalid table entry and will be rejected.
- Second, if only some of the capabilities are to be revoked, the object can have multiple entries, each corresponding to a different set of rights or a different group of users.

An alternative revocation mechanism uses abstract data type managers. Included with each abstract data

type is a revocation procedure. When access is to be revoked, the type manager simply disallows further

accesses by the subject whose rights are being revoked.