UnitelmaSapienza
Università degli Studi di Roma

Sapienza
Università di Roma
Dipartimento di Informatica

Access Control Matrix

and its implementations

F. Parisi Presicce

Access Control Matrix

Overview

Access Control Matrix Model

- Boolean Expression Evaluation
- History

Protection State Transitions

- Commands
- Conditional Commands

Special Rights

- Principle of Attenuation of Privilege

Overview

Protection state of system

- Part of the system state (contents of memory locations, registers, etc.)
- Describes current settings, values of system relevant to protection

- State transitions change the system and therefore the protection state
- From all the states P of system, only those in Q considered safe (acceptable)
- **Security Policy** characterizes the states in Q
- **Security Mechanisms** prevent system from entering states in P-Q

State Transitions

- Change the protection state of system
- |– represents a transition
  - X_i |– $_\tau$ X_{i+1}: command τ moves system from state X_i to X_{i+1}
  - X_i |– * Y: a sequence of commands moves system from state X_i to Y
- Commands often called *transformation procedures* built from elementary operations

Protection state

- Access control matrix
  - Describes protection state precisely
  - Matrix describing rights of subjects
  - State transitions change elements of matrix
- Used to indicate **who** is allowed to do **what** to/with **whom** on the system.

Access Control Matrix

- Who is **who** ?
  - **Subject** is what we call active entities (processes, users, other computers) that want to "do something"
- The **what** the subject does with the object can be just about anything, and it may be multi-part.
  - The set of rights in a cell specify the access of the subject (row) to the object (column)
  - Typical manipulations include
    - READ, MODIFY, CREATE, CHANGE, DELETE

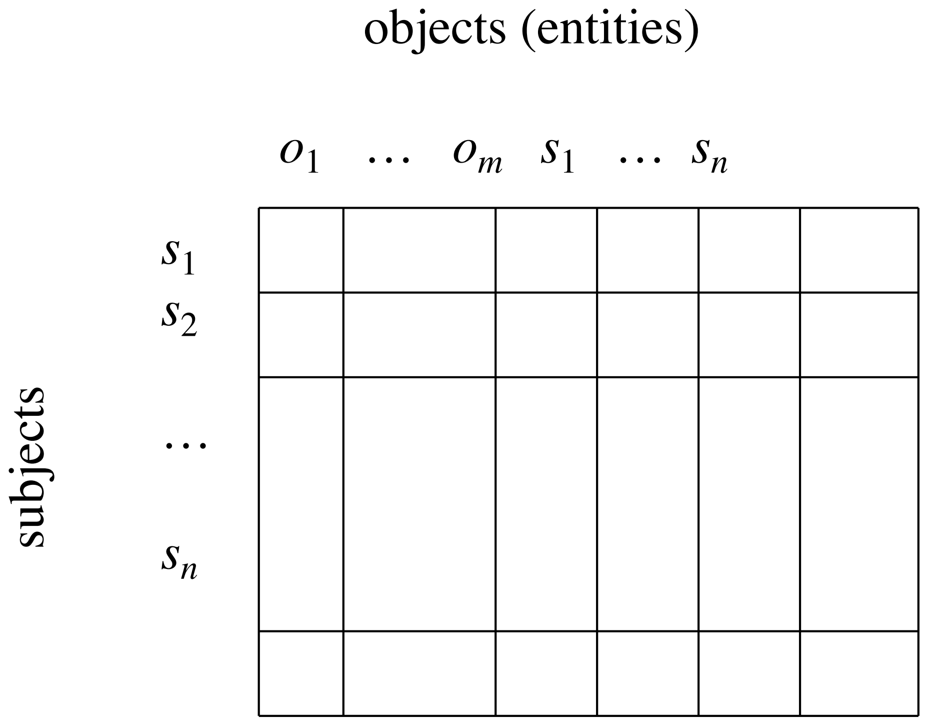
Object

- **Object** is what we call the broad category of items which can be manipulated or accessed
- Usually objects are passive, for example:
    - File
    - Directory (or Folder)
    - Memory segment
- An entity might be considered a **subject** in some circumstances but in other ones be an **object** on which it is possible to perform operations such as
    - kill
    - suspend
    - resume

Operation may require multiple access rights

- Example: User *parisi* (subject) wishes to copy (manipulate) the file */etc/passwd* (object1) to his home *directory* (object2)
- On many systems it requires access checks as in the following:
  - *parisi* needs permission to read OBJECT1
  - *parisi* needs permission to create a new file (OBJECT3) in OBJECT2
  - *parisi* needs permission to append to (or write to) OBJECT3

Description of ACM



- Subjects $S = \{ s_1,…,s_n \}$
- Objects $O = \{ o_1,…,o_m \}$
- Rights $R = \{ r_1,…,r_k \}$
- Entries $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, …, r_y \}$ means subject s_i has rights $r_x, …, r_y$ over object o_j

EXAMPLE 1

- Processes *p*, *q*
- Files *f*, *g*
- Rights *r*, *w*, *x*, *a*, *o*

	f	*g*	*p*	*q*
p	*rwo*	*r*	*rwxo*	*w*
q	*a*	*ro*	*r*	*rwxo*

EXAMPLE 2

- Procedures (subjects) *inc_ctr*, *dec_ctr*, *manage*
- Variable (object) *counter*
- Rights *+*, *–*, *call*

	counter	*inc_ctr*	*dec_ctr*	*manage*
inc_ctr	+			
dec_ctr	–			
manage		*call*	*call*	*call*

Primitive Operations

- **create subject** *s*; **create object** *o*
  - Creates new row, column in ACM; creates new column in ACM
- **destroy subject** *s*; **destroy object** *o*
  - Deletes row, column from ACM; deletes column from ACM
- **enter** *r* **into** ***A***[*s*, *o*]
  - Adds *r* rights for subject *s* over object *o*
- **delete** *r* **from** ***A***[*s*, *o*]
  - Removes *r* rights from subject *s* over object *o*

Create Subject

- Precondition: $s \notin S$
- Primitive command: **create subject** *s*
- Postconditions:
  - $S' = S \cup \{ s \}, O' = O \cup \{ s \}$
  - $(\forall y \in O')[a'[s, y] = \varnothing]$,
  - $(\forall x \in S')[a'[x, s] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

Create Subject

- Precondition: $s \notin S$
- Primitive command: **create subject** *s*

	p	q
x	r_1	r_1, r_2
y		r_1

$\Rightarrow$

	p	q	**s**
x	r_1	r_1, r_2	
y		r_1	
s			

Create Object

- Precondition: $o \notin O$
- Primitive command: **create object** *o*
- Postconditions:
  - $S' = S, O' = O \cup \{ o \}$
  - $(\forall x \in S')[a'[x, o] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

Create Object

- Precondition: $o \notin O$
- Primitive command: **create object** *o*

	p	q
x	r_1	r_1,r_2
y		r_1

$\Rightarrow$

	p	q	**o**
x	r_1	r_1,r_2	
y		r_1	

Add Right

- Precondition: $s \in S$, $o \in O$
- Primitive command: enter r into $a[s, o]$
- Postconditions:
  - $S' = S$, $O' = O$
  - $a'[s, o] = a[s, o] \cup \{ r \}$
  - $(\forall x \in S')(\forall y \in O' - \{ o \})\ [a'[x, y] = a[x, y]]$
  - $(\forall x \in S' - \{ s \})(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Add Right

- Precondition: $y \in S$, $p \in O$
- Primitive command: **enter** *r* **into** *a*[*y*, *p*]

	p	q
x	r_1	r_1,r_2
y		r_1

→

	p	q
x	r_1	r_1,r_2
y	**r**	r_1

Delete Right

- Precondition: $s \in S$, $o \in O$
- Primitive command: **delete** r **from** $a[s, o]$
- Postconditions:
  - $S' = S$, $O' = O$
  - $a'[s, o] = a[s, o] - \{ r \}$
  - $(\forall x \in S')(\forall y \in O' - \{ o \})\ [a'[x, y] = a[x, y]]$
  - $(\forall x \in S' - \{ s \})(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Delete Right

- Precondition: $y \in S$, $p \in O$
- Primitive command: **delete** *r* **from** *a*[*y*, *p*]

	p	q
x	r_1	r_1,r_2
y	r	r_1

$\Longrightarrow$

	p	q
x	r_1	r_1,r_2
y	🟨	r_1

Destroy Subject

- Precondition: $s \in S$
- Primitive command: **destroy subject** s
- Postconditions:
  - $S' = S - \{ s \}$, $O' = O - \{ s \}$
  - ~~$(\forall y \in O')[a'[s, y] = \varnothing]$,~~
  - ~~$(\forall x \in S')[a'[x, s] = \varnothing]$~~
  - $(\forall x \in S')(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Destroy Subject

- Precondition: $s \in S$
- Primitive command: **destroy subject** *s*

	p	q	**s**
x	r_1	r_1,r_2	
y			
s		r	

	p	q
x	r_1	r_1,r_2
y		

Destroy Object

- Precondition: $o \in O$
- Primitive command: **destroy object** *o*
- Postconditions:
  - $S' = S, O' = O - \{ o \}$
  - $(\forall x \in S')[a'[x, o] = \varnothing]$
  - $(\forall x \in S')(\forall y \in O')\ [a'[x, y] = a[x, y]]$

Destroy Object

- Precondition: $o \in O$
- Primitive command: **destroy object** *o*

	p	q	**o**
x	r_1	r_1,r_2	r_2
y		r_1	

$\Rightarrow$

	p	q
x	r_1	r_1,r_2
y		r_1

Creating File

- Process *p* creates file *f* with *r* and *w* permission

```
command create_file(p, f)
    create object f;
    enter own into A[p, f];
    enter r into A[p, f];
    enter w into A[p, f];
end
```

Mono-Operational Commands

- Make process *p* the owner of file *g*

```
command make_owner(p, g)
   enter own into A[p, g];
end
```

- Mono-operational command
  - Single primitive operation in this command

Conditional Commands

- Give *q* the right *r* over *f*, if *p* owns *f*

```
command grant_read_file_1(p, f, q)
    if own in A[p, f]
    then
        enter r into A[q, f];
end
```

- Mono-conditional command
  - Single condition in this command

Multiple Conditions

- Let *p* give *q* the rights *r* and *w* over *f*, if *p* owns *f* and *p* has *c* rights over *q*

```
command grant_read_file_2(p, f, q)
    if own in A[p, f] and c in A[p,q]
    then
        enter r into A[q, f];
        enter w into A[q, f];
end
```

NOTE: `or` and `not` are not allowed in conditions.

(specific) Copy Right

- Allows possessor to give rights to another (sometimes called grant right)
- Copier may or may not surrender the right, depending on system
- Often attached to a right, so only applies to that right
  - *r* is read right that cannot be copied
  - *rc* is read right that can be copied
- Is copy flag copied when giving *r* rights?
  - Depends on model, instantiation of model

(specific) Own Right

- Usually allows possessor of this right over an object to change entries in corresponding ACM column
  - So owner of object can add, delete rights for others
  - May depend on what system allows
    - Cannot give rights to specific (set of) users
    - Cannot pass copy flag to specific (set of) users

Attenuation of Privilege

- Principle says you cannot give rights you do not possess
  - Restricts addition of rights within a system
  - Usually *ignored* for owner
    - Why? Owner gives herself rights, gives them to others, deletes her rights.
- Meaningful in original systems, not in more recent ones (distinction between owner and administrator, e.g.; military)

What’s Wrong with ACM?

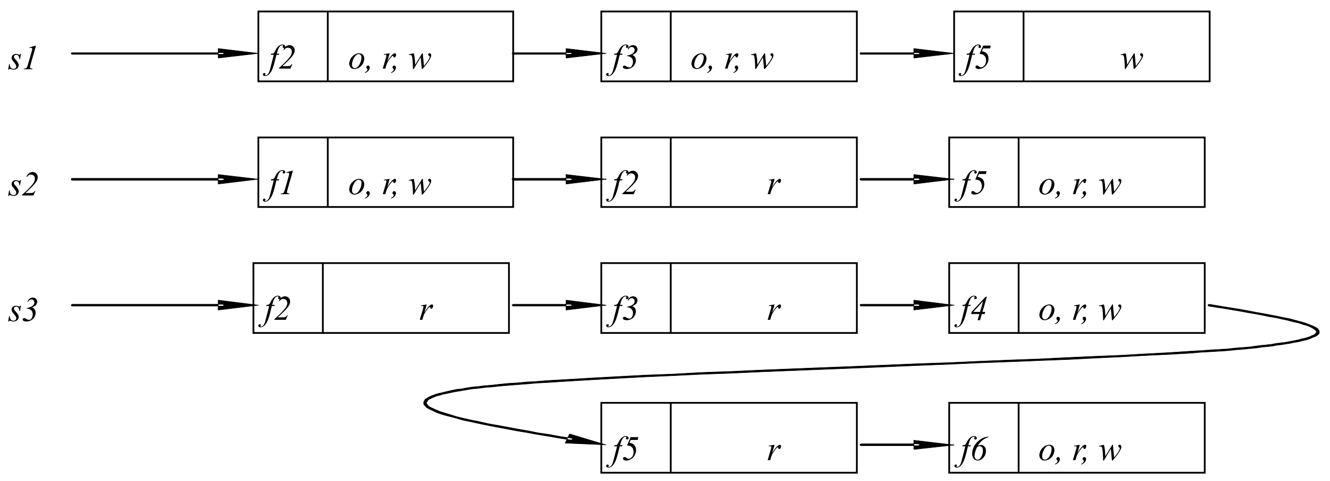
- Nothing if viewed as a model, but suppose we have 1k ‘users’ and 100k ‘files’ and users should only read/write their own files
  - The ACM will have 101k columns and 1k rows
  - Most of the 101M elements are either empty or identical
- Good for theoretical study but inappropriate for implementation
- Why bother with the empty elements?

ACM implementation 1

o: own
r: read
w:write

	f1	*f2*	*f3*	*f4*	*f5*	*f6*
s1		*o, r, w*	*o, r, w*		*w*	
s2	*o, r, w*	*r*			*o, r, w*	
s3		*r*	*r*	*o, r, w*	*r*	*o, r, w*

Access Matrix



Capabilities

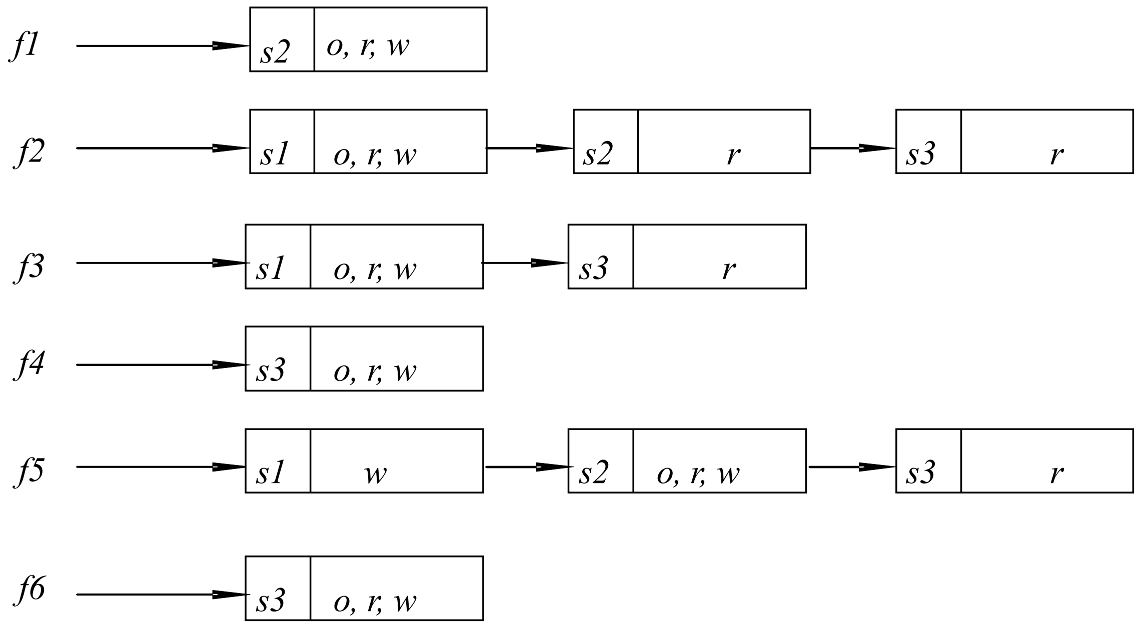
ACM implementation 2

o: own
r: read
w:write

	f1	*f2*	*f3*	*f4*	*f5*	*f6*
s1		*o, r, w*	*o, r, w*		*w*	
s2	*o, r, w*	*r*			*o, r, w*	
s3		*r*	*r*	*o, r, w*	*r*	*o, r, w*

Access Matrix

Access Control List


f1 → s2 | o, r, w
f2 → s1 | o, r, w → s2 | r → s3 | r
f3 → s1 | o, r, w → s3 | r
f4 → s3 | o, r, w
f5 → s1 | w → s2 | o, r, w → s3 | r
f6 → s3 | o, r, w

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) }

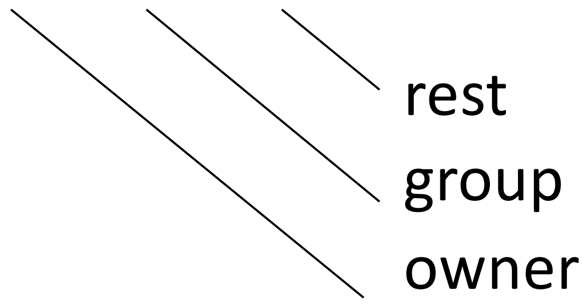
Default Permissions

- Normal: if not named, *no* rights over file
  - Principle of Fail-Safe Defaults
- 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

* UNICOS (now version 8.0) is operating system of Cray, based on Unix V

Abbreviations

- ACLs can be long … so combine users
  - UNIX: 3 classes of users: owner, group, rest
  - rwx rwx rwx



  - Ownership assigned based on creating process
  - Limited granularity
  - Cannot “exclude”

ACLs and Abbreviations

- Augment abbreviated lists with ACLs
  - Intent is to shorten ACL
- ACLs override abbreviations
  - Exact method varies
- Example: IBM AIX
  - Base permissions are abbreviations, extended permissions are ACLs with user, group
  - ACL entries can add rights, but on deny, access is denied

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
```

AIXC ACL example

The following is an example of an AIXC ACL:

```
attributes: SUID
base permissions:
  owner(frank): rw-
  group(system): r-x
  others: ---
extended permissions:
enabled
  permit rw- u:dhs
  deny r-- u:chas, g:system
  specify r-- u:john, g:gateway, g:mail
  permit rw- g:account, g:finance
```

AIXC ACL example

The ACL entries are described as follows:

- The first line indicates that the **setuid** bit is turned on.
- The next line, which introduces the base permissions, is optional.
- The next three lines specify the base permissions. The owner and group names in parentheses are for information only. Changing these names does not alter the file owner or file group. Only the chown command and the chgrp command can change these file attributes.
- The next line, which introduces the extended permissions, is optional.
- The next line indicates that the extended permissions that follow are enabled.
- The last four lines are the extended entries. The first extended entry grants user *dhs* read (r) and write (w) permission on the file.
- The second extended entry denies read (r) access to user *chas* only when he is a member of the *system* group.
- The third extended entry specifies that as long as user *john* is a member of both the *gateway* group and the *mail* group, he has read (r) access. If user *john* is not a member of both groups, this extended permission does not apply.
- The last extended entry grants any user in *both* the *account* group and the *finance* group read (r) and write (w) permission.

Note: More than one extended entry can apply to a process that is requesting access to a controlled object, with restrictive entries taking precedence over permissive modes.

ACL Modification

- Who can do this?
  - Creator is (typically) given *own* right that allows this
  - System R provides a *grant* modifier (like a copy flag) allowing a right to be transferred, so ownership not needed
    - Transferring right to another modifies ACL

Privileged Users

- Do ACLs apply to privileged users (*root* in UNIX or *administrator* in Windows)?
  - Solaris UNIX: abbreviated lists are ignored for root subjects, but full-blown ACL entries are not
  - Other vendors: varies

Conflicts: three possibilities

- Allow access if any entry would allow access
- Deny access if any entry would deny access
  - AIX: if any entry denies access, *regardless of where the entry is*, access is denied
- Apply first entry matching subject
  - Cisco routers: run packet through access control rules (ACL entries) in order; on a match, stop, and forward the packet; if no matches, deny
    - Note default is deny so honors principle of fail-safe defaults

Revocation Question

- How do you remove subject’s rights to a file?
  - Owner deletes subject’s entries from ACL, or rights from subject’s entry in ACL
- What if ownership not involved?
  - Depends on system
  - System R: restore protection state to what it was before right was given
    - May mean deleting descendent rights too …

Capability Lists

- 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) }

Semantics

- Like a bus ticket
  - Mere possession indicates rights that subject has over object
  - Object identified by capability (as part of the token)
    - Name may be a reference, location, or something else
  - Architectural construct in capability-based addressing; this just focuses on protection aspects
- ACL controlled by OS, capabilities in part by process
  - Must prevent process from altering capabilities
    - Otherwise subject could change rights encoded in capability or object to which they refer

Implementation

Three mechanisms to protect capabilities

- Tagged architecture
  - Bits protect individual words (only read or change too)
- Paging/segmentation protections
  - Like tags, but put capabilities in a read-only segment or page
  - Programs must refer to them by pointers
    - Otherwise, program could use a copy of the capability—which it could modify

Implementation (cont.)

- Cryptography
  - Associate with each capability a cryptographic checksum enciphered using a key known to OS
  - When process presents capability, OS validates checksum
  - Example: Amoeba, a distributed capability-based system
    - Capability is (*name*, *creating_server*, *rights*, *check_field*) and is given to owner of object
    - *check_field* is 48-bit random number; also stored in table corresponding to *creating_server*
    - To validate, system compares *check_field* of capability with that stored in *creating_server* table
    - Makes forging a capability difficult
    - ***Vulnerable if capability disclosed to another process***

Revocation

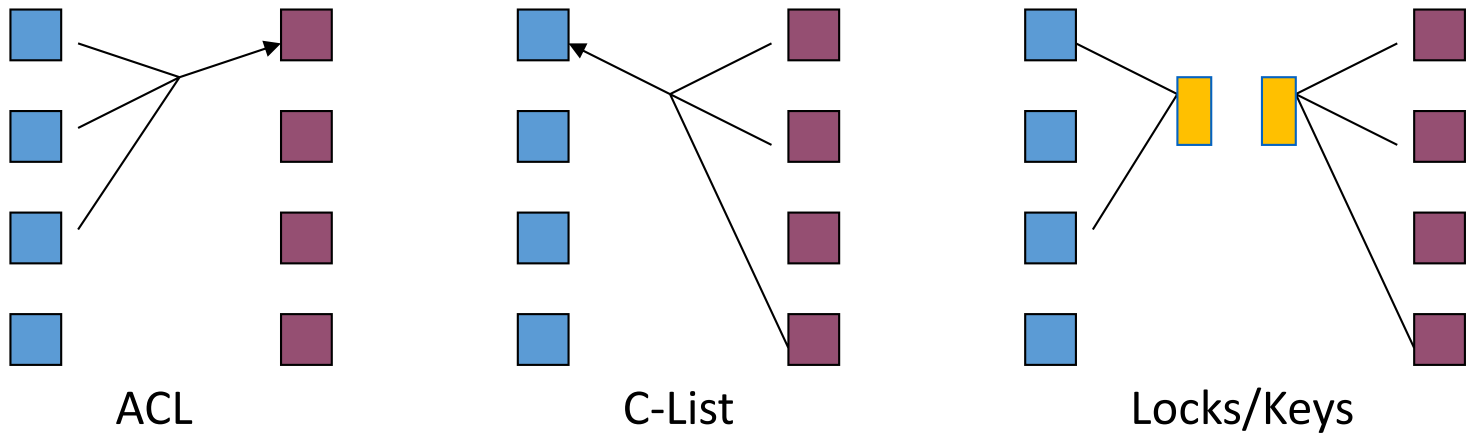
- Scan all C-lists, remove relevant capabilities
  - Far too expensive!
- Use indirection
  - Each object has entry in a global object table
  - Names in capabilities name the entry, not the object
    - To revoke, invalidate the entry in the table
    - Can have multiple entries for a single object to allow control of different sets of rights and/or groups of users for each object

ACLs vs. Capabilities

- Theoretically equivalent; consider 2 questions
  1. Given a subject, what objects can it access, and how?
  2. Given an object, what subjects can access it, and how?
  - ACLs answer second easily; C-Lists, first
- Probably the second question has been of most interest in the past, hence ACL-based systems more common than capability-based systems
  - As first question becomes more important (in incident response, for example), this may change

Locks and Keys

- Associate *lock* with object and *key* with subject
  - Latter controls what the subject can access and how
  - Subject presents key; if it corresponds to any of the locks on the object, access granted
- This is more flexible
  - Change either locks or keys



Cryptographic Implementation

- Enciphering key is lock; deciphering key is key
  - Encipher object *o*; store $E_k(o)$
  - Use subject's key k' to compute $D_{k'}(E_k(o))$
  - Any of *n* subjects can access *o* (**OR**-access): store

$$o' = (E_1(o), \ldots, E_n(o))$$

  - Requires consent of all *n* to access *o* (**AND**-access): store

$$o' = (E_1(E_2(\ldots(E_n(o))\ldots))$$

Type Checking

- Lock is type, key is operation
  - Example: UNIX system call *write* cannot work on directory object but does work on file
  - Example: distinguish Instruction and Data spaces of PDP-11
    - execute only on instructions,
    - read/write only on data
  - Example: countering buffer overflow attacks on the stack by putting stack on non-executable pages/segments
    - Then code uploaded to buffer will not execute
    - Does not stop other forms of this attack, though …

Key Points

- Access control matrix simplest abstraction mechanism for representing protection state
- Transitions alter protection state
- 6 primitive operations can alter matrix
    - Transitions can be expressed as commands composed of these operations and, possibly, conditions
- Implementations reflect frequency of operations