DAC vs. MAC

- **Most people familiar with discretionary access control (DAC)**
  - Example: Unix user-group-other permission bits
  - Might set a file `private` so only group `friends` can read it
- **Discretionary means anyone with access can propagate information:**
  - `Mail sigint@enemy.gov < private`
- **Mandatory access control**
  - Security administrator can restrict propagation
  - Abbreviated MAC (NOT a message authentication code)

Bell-Lapadula model

- **View the system as subjects accessing objects**
  - The system input is requests, the output is decisions
  - Objects can be organized in one or more hierarchies, H (a tree enforcing the type of decendents)
- **Four modes of access are possible:**
  - execute – no observation or alteration
  - read – observation
  - append – alteration
  - write – both observation and modification
- **The current access set, b, is (subj, obj, attr) tripples**
- **An access matrix M encodes permissible access types (subjects are rows, objects columns)**

Security levels

- **A *security level* is a (c, s) pair:**
  - $c =$ classification – E.g., unclassified, secret, top secret
  - $s =$ category-set – E.g., Nuclear, Crypto
- (c_1, s_1) ***dominates*** (c_2, s_2) **iff** $c_1 \geq c_2$ **and** $s_2 \subseteq s_1$
  - L_1 *dominates* L_2 sometimes written $L_1 \sqsupseteq L_2$ or $L_2 \sqsubseteq L_1$
- **Subjects and objects are assigned security levels**
  - level(S), level(O) – security level of subject/object
  - current-level(S) – subject may operate at lower level
  - level(S) bounds current-level(S) (current-level(S) $\sqsubseteq$ level(S))
  - Since level(S) is max, sometimes called S's *clearance*

Label lattice

- **A *lattice* is a set and a partial order such that any two elements have a least upper bound**
  - I.e., given any x and y, there exists a unique z such that
    - $x \sqsubseteq z$ and $y \sqsubseteq z$ (z is an upper bound)
    - For any z' such that $x \sqsubseteq z'$ and $y \sqsubseteq z'$, $z \sqsubseteq z'$ (z is minimal)
  - Least upper bound (lub) z of x and y usually written $z = x \sqcup y$
- **Security levels form a lattice under $\sqsubseteq$**
- **What's lub of Bell-Lapadula labels (c_1, s_1) and (c_2, s_2)?**

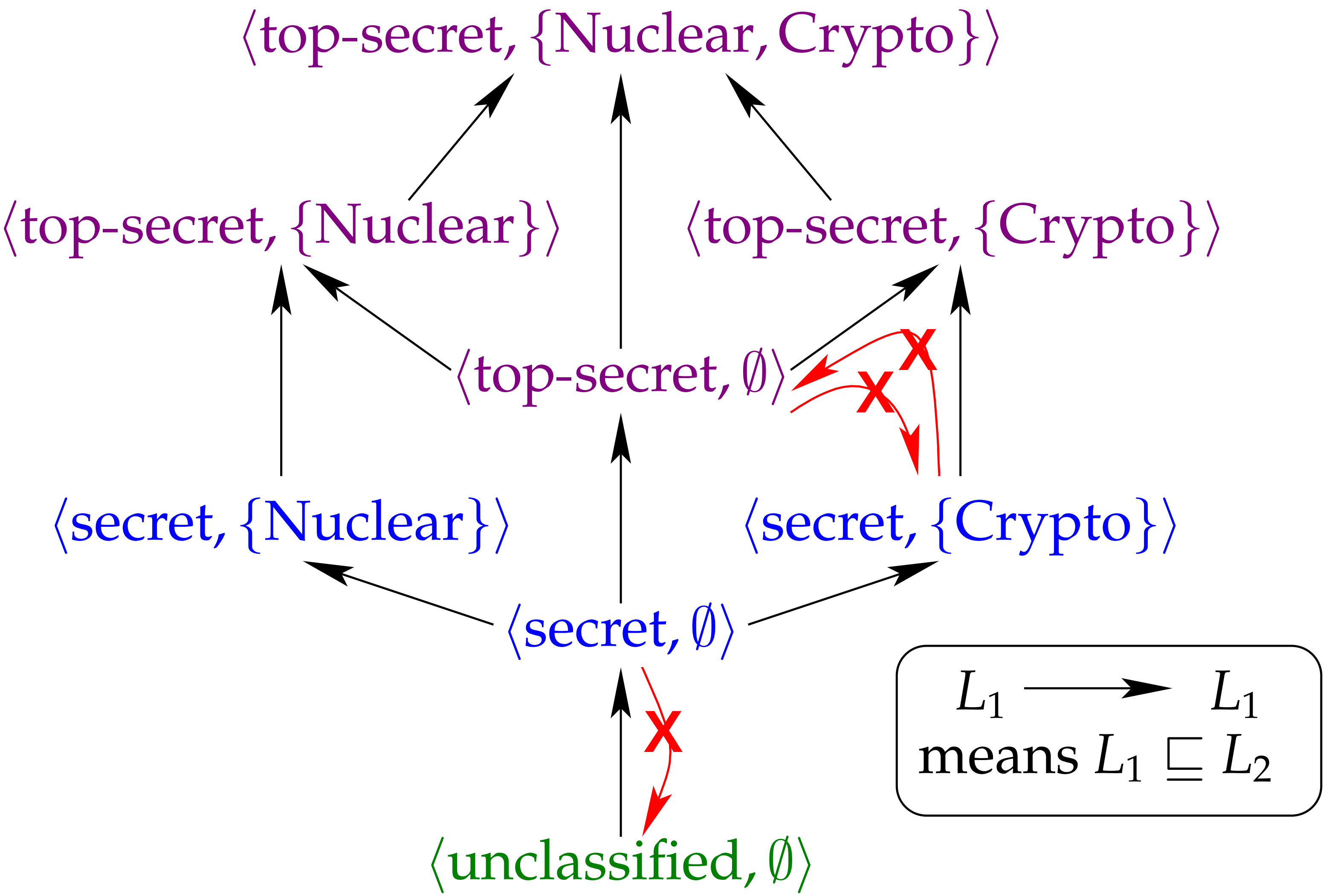
Label lattice

- **A *lattice* is a set and a partial order such that any two elements have a least upper bound**
  - I.e., given any x and y, there exists a unique z such that
    - $x \sqsubseteq z$ and $y \sqsubseteq z$ (z is an upper bound)
    - For any z' such that $x \sqsubseteq z'$ and $y \sqsubseteq z'$, $z \sqsubseteq z'$ (z is minimal)
  - Least upper bound (lub) z of x and y usually written $z = x \sqcup y$
- **Security levels form a lattice under $\sqsubseteq$**
- **What's lub of Bell-Lapadula labels (c_1, s_1) and (c_2, s_2)?**
  - $(\max(c_1, c_2), s_1 \cup s_2)$
  - I.e., higher of two classification levels, plus all categories in either label

Security properties

- **The simple security or *ss-property*:**
  - For any $(S, O, A) \in b$, if A includes observation, then level(S) must dominate level(O)
  - E.g., an unclassified user cannot read a top-secret document
- **The star security or **-property*:**
  - If a subject can observe O_1 and modify O_2, then level(O_2) dominates level(O_1)
  - E.g., cannot copy top secret file into secret file
  - More precisely, given $(S, O, A) \in b$:
    if $A = r$ then current-level(S) $\sqsupseteq$ level(O) ("no read up")
    if $A = a$ then current-level(S) $\sqsubseteq$ level(O) ("no write down")
    if $A = w$ then current-level(S) $=$ level (O)

Example lattice



- **Information can only flow up the lattice**
  - “No read up, no write down”

Straw man MAC implementation

- **Take an ordinary Unix system**
- **Put labels on all files and directories to track levels**
- **Each user U has a security clearance (level(U))**
- **Determine current security level dynamically**
  - When U logs in, start with lowest curent-level
  - Increase current-level as higher-level files are observed (sometimes called a *floating label* system)
  - If U's level does not dominate current, kill program
  - If program writes to file it doesn't dominate, kill it
- **Is this secure?**

No: Covert channels

- **System rife with *storage channels***
  - Low current-level process executes another program
  - New program reads sensitive file, gets high current-level
  - High program exploits covert channels to pass data to low
- **E.g., High program inherits file descriptor**
  - Can pass 4-bytes of information to low prog. in file offset
- **Labels themselves can be a storage channel**
  - Arrange to raise process p_i's label to communicate i
  - One reason why static analysis of programming languages is appealing (labels checked at compile time $\Rightarrow$ no covert channel)
- **Other storage channels:**
  - Exit value, signals, terminal escape codes, …
- **If we eliminate storage channels, is system secure?**

No: Timing channels

- **Example: CPU utilization**
  - To send a 0 bit, use 100% of CPU in a busy-loop
  - To send a 1 bit, sleep and relinquish CPU
  - Repeat to transfer more bits, maybe with error correction
- **Example: Resource exhaustion**
  - High prog. allocate all physical memory if bit is 1
  - If low prog. slow from paging, knows less memory available
- **More examples: Disk head position, processor cache/TLB polution, ...**
  - In fact, blurry line between storage & timing channels
  - E.g., might affect the order or two "low" FS operations

Reducing covert channels

- **Observation: Covert channels come from sharing**
  - If you have no shared resources, no covert channels
  - Extreme example: Just use two computers
- **Problem: Sharing needed**
  - E.g., read unclassified data when preparing classified
- **Approach: Strict partitioning of resources**
  - Strictly partition and schedule resources between levels
  - Occasionally reapportion resources based on usage
  - Do so infrequently to bound leaked information
  - In general, only hope to bound bandwidth of covert channels
  - Approach still not so good if many security levels possible

Declassification

- **Sometimes need to prepare unclassified report from classified data**
- **Declassification happens outside of system**
  - Present file to security officer for *downgrade*
- **Job of declassification often not trivial**
  - E.g., Microsoft word saves a lot of undo information
  - This might be all the secret stuff you cut from document

Biba integrity model

- **Problem: How to protect integrity**
  - Suppose text editor gets trojaned, subtly modifies files, might mess up attack plans
- **Observation: Integrity is the converse of secrecy**
  - In secrecy, want to avoid writing less secret files
  - In integrity, want to avoid writing higher-integrity files
- **Use integrity hierarchy parallel to secrecy one**
  - Now *security level* is a (c, s, i) triple, $i =$integrity
  - Only trusted users can operate at low integrity levels
  - If you read less authentic data, your current integrity level gets raised, and you can no longer write low files

Generalizing the lattice

- **Now say** $(c_1, s_1, i_1) \sqsubseteq (c_2, s_2, i_2)$ **iff:**
  - As before, $c_1 \le c_2$ and $s_1 \subseteq s_2$
  - In addition, require $i_1 \ge i_2$
- **In general, say** S_1 **is labeled** L_1**,** S_2 L_2**, and** $L_1 \sqsubseteq L_2$
  - Neither S_1 nor S_2 is more privileged than the other
  - S_1 can write more objects (including any S_2 can)
  - S_2 can read more objects (including any S_1 can)
  - Information *can flow from* S_1 *to* S_2, but not necessarily vice versa
- **Privilege comes from the ability to declassify**
  - I.e., read object labeled L_2, write object labeled L_1 when $L_2 \not\sqsubseteq L_1$