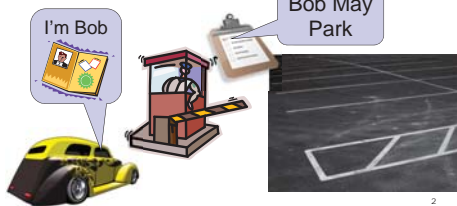


5 Access Control and Digital Rights Management

5.1 Introduction

Controlling access to resources

- Who is trusted to do *what* with a *resource*
 - Subject, Action, Object



2

Recall Security Policies

- Specify "allowed" / "disallowed"
 - Context; applies to ..., approved/imposed by ...
 - Usage; required enforcement, dealing with breaches
- Different notions of 'security policy':
 - from general intention statement
 - "Data shall only be available to those with a 'need-to-know'"
 - to formal, detailed specification
 - "drwxr-xr-x", access control list, XACML policy, etc.
- Security Model
 - Framework to express & interpret policies.
 - E.g. relations on Users - Objects - Permissions - Groups.
 - Example Multi level security



3

As mentioned in the introduction a security policy specifies the 'what and when of security; what are the security attributes that should be achieved and when/for what resources. A high level policy may be 'the lecturer maintains a gradelist that the students can read and a paper submission program that student can run'. Such a policy may be translated into system level *access control policies* which specify who (subject) is trusted with to do what (action) with which resource. E.g. the lecturer has read and write and the student read permission on the resource 'gradelist'.

Recall that the meaning of a security policy is given by interpretation in a mathematical model, the security model. A basic interpretation for access control policies can be a relation on subjects-resources and rights.

Access Control Matrix

Policy:
 Students may read grade list and read and run submitPaper
 Teacher may read and write grade list and submitPaper

User	GradeList	SubmitPaper
Jerry	rw	rw
Joris	r	rx
Tim	r	rx

- So we are done ?

4

Controlling access to resources

- Enforcement, Implementation
 - Captures intended policy (how to check?)
 - Dynamicity; rights not constant
- Maintenance, Consistency
 - Authority on the resource; Who *decides*?
 - Decentralized systems, delegation.
 - Conditions, Obligation, Purpose
- Privacy
 - Anonymity, attribute based AC



A basic format in which one can give an access control policy is the access control matrix. The rows are the subjects, the columns the resources and a field, e.g. (Jerry, Gradelist) is filled with the rights that the subject (Jerry) has on the resource (Gradelist); in this case read and write permission. The interpretation of an access control matrix is straight forward; a matrix is a way of representing a relation on subjects-resources and rights.

Eventhough the AC matrix gives us a way to specify rights we also need to implement the enforcement of the rights; ensure the rights subjects actually get the rights and only the rights specified

in the AC matrix. A single huge matrix for the entire system (or even network) may not be the best option to achieve this.

We also need to maintain the AC policy; recall that it implements the high level policy 'the lecturer maintains a gradelist that the students can read and a paper submission program that student can run'. If the students in the class change we need to update the matrix. If Joris leaves the class we may need to revoke his rights but how do we know that Joris had the read right (only) because he was a student in the class? For that matter, are we sure the AC matrix is a good implementation of the highlevel policy to begin with?

Finally, to use his read right Joris will have to identify himself to the system. This should not be needed; Joris should only have to prove that he is a student not reveal who he is.

Access Control Lists

(Enforcement & Maintenance)

User	GradeList	SubmitPaper
Jerry	rw	rw
Joris	r	rx
Tim	r	rx

U	User	SubmitPaper
J _e	Jerry	rw
J _o	Joris	rx
T _i	Tim	rx

Role base access control

(Maintenance, Consistency)

- Role (Similar to 'group')
 - Teacher
 - Student
- Assign access rights to Roles and Roles to users
- Added indirection makes for easier maintenance

Role	GradeList
Teacher	rw
Student	r

Role	Users
Teacher	Jerry
Student	Joris, Tim

Access control lists are a way to deal with the problems that a single centralized access control matrix would give. An AC list is basically a column from an access control list, stating all the rights that different subjects have on a single resource. As such it has a natural place to store it; together with the resource. Ofcourse a problem is if rights changes (e.g. a student is added or removed from the class) all relevant AC lists have to found and updated. I.e. this gives a viable implementation, however the maintainability only gets worse.

Role based access control (RBAC) tries to improve maintainability through an adding level of indirection. Instead of assigning rights to subjects (Jerry, Joris, etc.) they are assigned to roles (teacher, student) and the subjects are assigned to roles. If Joris leaves the class all we now have to change is the role-subject table. If the high level policy changes and students are no longer allowed to submit papers only this entry in the role-resource table needs to be changed.

Role dependency (Role Hierarchies)

```

    graph TD
      Staff[Staff] --> Scientific[Scientific]
      Staff --> Financial[Financial]
      Staff --> Legal[Legal]
      Scientific --> Prof[Prof]
      Scientific --> Lecturer[Lecturer]
      Financial -.-> Dots1[...]
      Legal -.-> Dots2[...]
    
```

- Staff may Enter Building
 - Staff rights also granted to Professors

Decentralized AC

(Specification & Authority)

- Different authorities at different locations
 - UT admin does not control TU/e resources
- Different Hierarchies for different locations
 - In NL PhD student is subrole of Employee
 - in US PhD student is subrole of Student
- Access control for distributed resources?
 - TU/e student list, US student discount.


Roles can provide a huge improvement in maintainability, however, especially in large organizations, the number of roles can quickly grow to unmanageable proportions. Role hierarchies help bring structure to the roles. A university can define different categories of staff, each with their own rights and responsibilities. By using a role hierarchy we only have to specify shared rights once; e.g. we may assign the right to supervise students to professors and the right to enter buildings to staff. The professors will inherit the rights from staff so will also be allowed to enter the building.

Distributed access control So far we have considered a single authority; all access rights, roles assignments etc. are specified (and enforced) by a single entity. When considering multiple authorities we need to consider the communication between entities and issues such as trust and delegation.

Sharing resources

- Multiple policy from different authorities


Authority Rose



Alice may Read

Alice may do what Beryl says

Authority Beryl



Bob mag Lezen

Alice may NOT read (anymore)

Talking the same language

Standardization <Subject>X</Subject>
<Action>Y</Action>

Ontology
R.Read is an action
B.Lezen is an action
R.Read = B.Lezen

Trust Management
Delegation of (access) decisions

Negative information
revocation, conflict resolution, monotonicity

10

Delegation

- Define your roles based on roles of other users:
 - Jerry.StudentsInMyClass = EducationOffice.RegisteredStudents[2IS05]
- Trust Management Issue:
 - trust education office to define registered student role
 - Education office in turn may trust registration office

EducationOffice.RegisteredStudents[2IS05] =
RegistrationOffice.Student and
StudyWeb.subscribed[2IS05]

11

Trust and delegation play key roles in distributed AC. If I trust another authority I can rely on the statements this authority makes, for example on the identity or roles of users. The trust language we saw in the previous chapter is an example of this; I can define my roles (e.g. 'StudentInMy-Class') in terms of the roles of others (e.g. the education office student registry on OASE). Here a decision is delegated. Delegation of rights is also possible (see Logic in access control example below).

Typically monotonicity is desired in distributed AC systems; additional credentials only increase rights, not decrease them. This is important for safety; an attacker Malory will not be able to increase her rights by preventing (negative) certificates from reaching the decision point. It also prevents conflicts in policies; where one policy allows and another denies an action. With monotonic system adding policies will only add rights, never remove them.

Negative information, however, can be practically useful e.g. to enable revocation of rights. AC system such as XACML (discussed below) allow for 'negative' information, e.g. allowing a policy of the form 'students may NOT enter the building'. If another policy says that 'employees may enter the building' this will lead to a conflict for student assistant Alice who is both a student and an employee. XACML allows specifying how such conflicts should be resolved. Another problem is obtaining all relevant attributes; if Bob hides his student card we may not know he is a student and should be denied access. Thus a negative rule should only be used if we can guarantee that we will get the required information.

Access Control and Logic Above we have seen how we can use roles in AC and also how roles can be shared between authorities. The roles reduce the maintenance effort and capture more of the *why* a right is granted instead of only *what* right is granted. The role based approach, however, may not be sufficient to capture our AC needs. If the 'why' is more complex than role membership, the policy cannot be captured well in RBAC. Consider for example a hospital where each doctor is allowed to access the documents related to their own patients but not those of other patients. This 'why' is a simple (logical) rule that does not fit well in the role system. It is not sufficient to have a role 'doctor' and a role 'patient'; as then each doctor could access all patient records. One would already need either a role 'patient-of-dr-x' for each doctor x or a role 'treating-doctor-of-y' for each patient y. Both benefits of adding roles are thus mostly lost.

(*)¹ When we assign resource permissions to a role we are actually describing a block of permissions in an AC matrix; each user in the role (row) gets the permissions for those resources (column). Thus if the AC rights in the policy we wish to describe do not form blocks then RBAC does not help us. Still, the structure in the AC matrix for the hospital policy above is very regular; a single right exists in each column - at the treating doctor of a patient. As the treating doctor relation

¹This paragraph forms a more technical side note.

is undoubtedly already present in the system it would good to express the policy in terms of this relation. A logical specification allows us to describe other structures in the matrix to efficiently represent it.

Logic in Access Control

(Specification, Policies)

Express AC rules with logical formulas

- Rights expressed by predicates
 - $\text{may-access}(p,o,r)$
 - principle p has access right r to object o
- Basic rules can also be expressed
 - $\text{may-access}(p,o,Wr) \rightarrow \text{may-access}(p,o,Rd)$
 - write access implies read access
- Different ways to generalize this principle

12

Logic in Access Control (2)

(Decentralized systems, Delegation)

SAYS: delegation construct:

- for stating requests
- for delegation, e.g. p says $\text{may-access}(q,o,r)$

```
p says may-access(q,o,r)
=>
( may-access(p,o,r)
=> may-access(q,o,r)
)
```

13

One can describe the AC rights using logical facts and rules. A basic predicate captures single entries; e.g. $\text{may-access}(p,o,r)$ can be used to express that principle p has right r on object o . While stating a logical fact is just the same as having an entry in an AC matrix, the predicate formulation allows the use of logical connectives to build rules. For example, often read access is granted to anyone with write access. This can be expressed as an implication²: $\text{may-access}(p,o,Wr) \rightarrow \text{may-access}(p,o,Rd)$

So far we have only considered rights in our logical formulas. However, the true power of the logical system becomes clear when we also allow the use of other relations and information in the system: For example, a hospital system will typically have a list of doctors which we can use in our formulas by introducing a predicate $Dr(x)$. Any role can be expressed as a predicate and role dependencies as rules (implications); RBAC functionality is also offered by a logical system. The hospital will also record which doctor is treating which patient. We import this relation in our logical system as a binary predicate, e.g. $DrOf(x,y)$ where x is a doctor and y a patient being treated by x .

Advanced LiAC issues

- Examples:
 - $\text{SAY Pay}(1\$) \rightarrow \text{CanPlay}(\text{movie}) \text{ TO OtherUser}$
 - $\text{DrOf}(D,x) \rightarrow D \text{ SAYS } (\text{Nurse}(y) \text{ or } \text{Doctor}(y)) \rightarrow \text{canTreat}(y,x) \ \& \ \text{orderDrugsFor}(y,x) \ \text{TO } y$
 - $\text{DrOf}(D,x) \ \& \ (\text{Nurse}(y) \ \text{or } \ \text{Doctor}(y)) \rightarrow D \text{ SAYS } \text{canTreat}(y,x) \ \& \ \text{orderDrugsFor}(y,x) \ \text{TO } y$
- Expressiveness
 - Typically first order; no `all policies such that ...`
- Performance and decidability
 - Verifiability, Provability
- Comparing, combining, refining of policies

14

Sandboxing

(Enforcement)



- Run untrusted code in limited environment
- Only predetermined access to resources
- E.g. run on a virtual machine
 - JVM (Java)
 - Type safety, Memory safety

15

Our example of before, where doctors are allowed to read the record of patients they are treating can now be expressed as a simple implication: $DrOf(x,y) \rightarrow \text{may-access}(x, \text{Record}(y), \text{read})$ ³.

That x has to be the treating doctor of y is an example of an AC *condition*; a fact that has to be true before the access right is granted. We can also consider *Obligations*. Obligations are actions that need to be taken if the access is granted (or if it is denied-see section on XACML below). For example, grant access to the medical record but inform the patient, allow playing a video but only if 1€ is payed, allow making a copying of a file but only if copy is deleted within 5 days, etc.

The 'SAY(S)' construct is introduced for systems with multiple entities/authorities. For example,

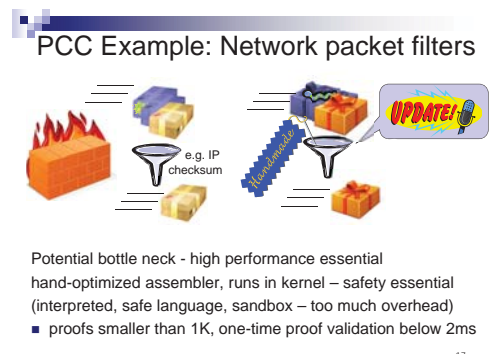
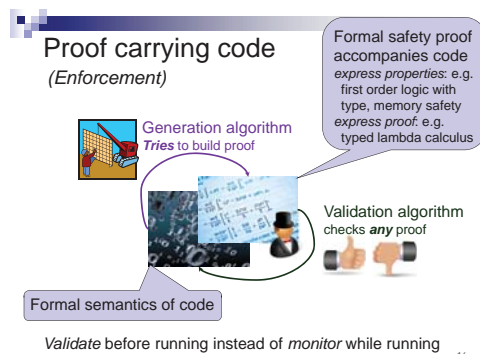
²Note that this is actually an implicit universal quantification for p and o while Wr and Rd are constants.

³See also Exercise 3

we can state that p has the right to delegate the rights (on o) that he has to q . Subtle differences in our logical policies can express important distinctions. If we write $DrOf(D,x) \rightarrow D \text{ SAYS } (Nurse(y) \vee Doctor(y)) \rightarrow canTreat(y,x) \text{ TO } y$ this gives doctor D the right to delegate permission 'if you are a nurse or doctor you may treat patient x ' and this right can be delegated to anyone. If we write $DrOf(D,x) \wedge (Nurse(y) \vee Doctor(y)) \rightarrow D \text{ SAYS } canTreat(y,x) \text{ TO } y$ this gives doctor D the right to delegate permission 'you may treat patient x ' but only to doctors and nurses. The key difference here is by whom (where) the check whether y is a nurse or doctor happens. In the first case this is done by (happens on the system of) y while in the second case it is done by (on the system of) D . If D and y are on different systems run by different authorities, this becomes an important distinction.

We see that in a logical system are quite expressive but this does come at a cost; from an access control matrix we directly see whom gets what right on which object. For a logical policy, it is much less clear which rights entities actually get based on these logical rules. This is true for a person looking at a policy but also for the AC system that has to implement the policy; when a request comes in it will need to logically derive whether or not it should be granted which is obviously a lot more work than simply looking it up in a table or matrix (or checking two tables with RBAC). Because of this trade-off logical system typically use first order languages; i.e. we may have statements like 'for all objects ...' and 'for all subjects ...' but not 'for all policies ...'. (With delegation (says) it would be useful to be able to reason about policies; for instance if I have two projects for different clients that need to be kept separate (see Chinese wall below) I may want to state a delegation of the form I'll accept all policies from Bob that do not grant read rights to both projects 1 and 2 to the same entity.) The reason is that non-trivial higher order languages are usually undecidable; thus we may get requests for which we cannot even determine whether they should be granted or not. This is clearly not acceptable in an AC system. Even with first order languages we have to be careful; making the language too expressive will quickly lead to a very complex decision procedure impacting performance or even to undecidability.

Note we have moved the discussion from specification of access rights to enforcement of the rights. Typically some enforcement point will intercept request for a resource and check access rights before allowing the request to proceed. If requests come from other machines we can run the (relevant) network traffic through the enforcement point so it can intercept and check requests. If the requests come from code that is being run locally (e.g. an app running on a phone or in a browser or a program we have downloaded) we can use sandboxing. The code will not have direct access to the underlying system but instead will get to 'play in a sandbox', cut off from the rest of the system. Within the sandbox it will only get access to predetermined resources and requests can be checked by our AC system.



Proof Carrying Code (PCC) Instead of checking and enforcing policies at the point/time of access we could also check the code that will be doing the access and make sure that it only uses resource in the intended way. We can then execute the code without runtime overhead for access control enforcement. Using a formal semantics (meaning) of the code we can verify that it satisfied a given (logical) policy. One application areas for PCC is performance critical applications, where

the overhead of other access control enforcement mechanisms and/or the use 'safe' languages is unacceptable. Another application area for PCC is mobile agents. In the mobile agent setting, a user sends out an (autonomous) agent which moves between different hosts, gathering data and computing results which eventually will be returned to the user. As this requires hosts to accept untrusted agents, safety of the agent's code is essential. PCC is way for the host to establish trust in the agent; by checking the proof it can be sure the agent will adhere to the policies.


PCC Example (2): Mobile Agents

Site has Safety policy e.g.

- Memory safety
- Resource usage bounds

PCC to ensure untrusted agents are safe

- Access database of airfares.
 - assigned access level when received
 - may only look at lower level database records
- Agents must prove:
 - terminate within given number of instructions
 - not exceed preset bandwidth
 - etc.



18

Discretionary – Mandatory AC

- Who decides the access rights ?
 - Owner of the file
 - Owner of the system (system security policy)
- Owner file not always 'owner' information
 - Classified information.
 - Programs acting on behalf of the user
 - Personal information

20

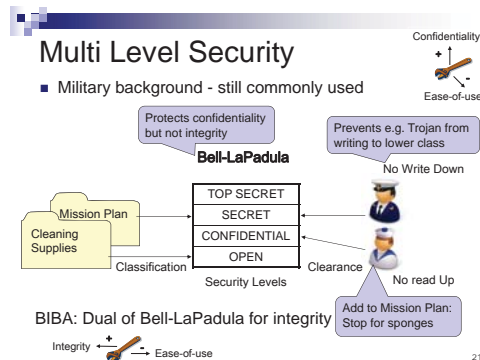
5.2 Security Models

Recall the discussion on security models in the first lecture. To be able to understand a security policy we need a security model. The AC matrix security model gives the meaning of policies in the term of relations on subjects-objects and rights. With RBAC we have the addition of the notion of roles and permissions assigned to roles (Anderson [3, Ch. 7]) refers to this as a Security Policy Model). With the RBAC security (policy) model we can find the meaning of policies using these extensions; i.e. what relation on subject-objects and rights these policies express. Here we will see other security (policy) models that, like RBAC, introduce new mechanisms to express and interpret security policies.

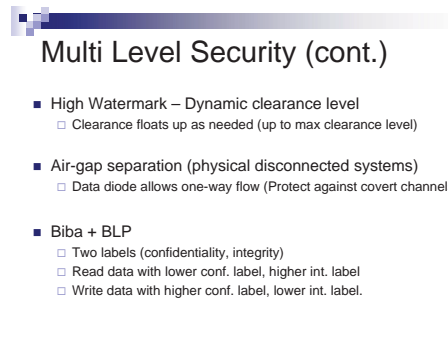
When considering AC and security models it is important to consider who is the authority; who is entitled to set/change a policy on a resource (some AC frameworks (security models) manage this aspect as well e.g. administrative policies in RBAC). A key distinction here is that between *discretionary* and *mandatory* access control. In discretionary access control the users are in control of the resources; e.g. if I create a file I can set the permissions on this file. This is fine if the information in the file is mine, however, this is not always the case. I may be allowed to read confidential company data (and write my conclusions in a file) but should of course not distribute the file outside the company. In mandatory access control, the system is in charge of the resources. E.g. if I create a file the system will set its permissions based on the information contained in the file. A combination of the two types of AC is also possible; e.g. the system sets my file to 'within the company only' but I get to decide whom within the company gets to read it.

In a military setting confidentiality of certain information is essential. Clearly a form of mandatory access control is needed. Multi level security models such as the *Bell-LaPadula (BLP)* model have been introduced and are still being used for this purpose in this setting. In BLP, the resources are assigned a classification (security label) which is a level indicating how confidential the information is. Basically the levels form a simple increasing sequence e.g. public-confidential-secret-top secret. (Though BLP also supports compartmentalization using code words, see lattices based security below.) A user (and a process running on behalf of the user) is also assigned a security label; the clearance level. The simple security property of BLP is that users are not allowed to read document above their clearance level (No read up). This is likely what you would have expected. The special property of the BLP model is that also a user is not allowed to *write* to a *lower* level (this is called the 'no write down' or *-property). The goal of this property is to prevent a

high level user or process (e.g. a trojan, see also Section 3.3) from leaking information to a lower level.



21

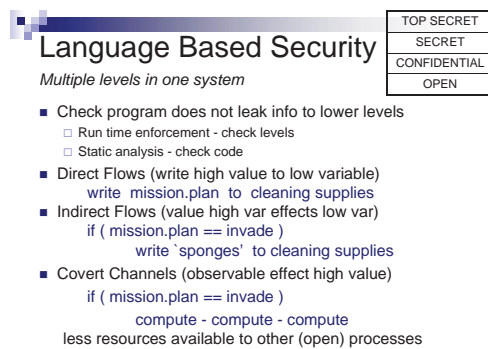


22

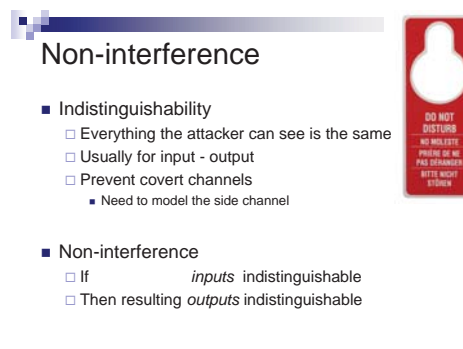
To help protect sensitive data, the *principle of least privilege* is often used. Users work at the lowest clearance level that allows them to access the resources that they need, which may be lower than their (maximum) clearance level. Within BLP this also has the advantage of being able to write to documents at that lower level. This can be done dynamically; the clearance level increases (upto the maximum level for that user) as more sensitive documents are read. Note that this affects the files that the user can write to. For example, the security label of any temporary files being used to be increased as well otherwise we cannot write to them anymore. Related to this is the problem label creep; when reading several files any file written after this will have to have a security label that is a upper bound of all read files. Documents can quickly become (too) highly classified.

The label creep and the fact that one cannot write to a lower file or in other way send information to entities with a lower clearance level is a big practical problem. Yet it is exactly the goal of the system to stop such information flow, to prevent disclosing confidential information. Still one would like to have the possibility to communicate 'harmless' information. Declassification is a technique where the restrictions are relaxed and specific, well-defined information may be assigned a lower classification.

If you look at the BLP rules we see that only the confidentiality of data is protected; a sailor cannot see what the mission plan is but can overwrite it. The BIBA model is the dual of BLP for integrity. BIBA can also be combined with BLP (assigning two separate security labels, one for confidentiality and one for integrity). Though different classifications are possible one may end up allowing only read-write in single level (which removes any benefit from creating an order of security levels; they become the same as isolated, unrelated labels.)



23



24

Implementation To achieve BLP military systems often use physically separated systems for the different level (air gap separation. These system are connected with 'data-diodes' which allow data to only flow only one way (from the low to high level). On normal networks one could hide

information in e.g. the timing of acknowledgements of received packages. The data-diodes try to prevent information from being sent by such covert channels (as well as directly, obviously). Timing is an example of a side channel; another (unintended) channel of obtaining information about a running process/system other than the input given to and the output generated by the process/system. (E.g. the response time of the system, temperature of the CPU, etc. See also the section on side channel attacks ??.)

Using physically separated systems is often not feasible. To implement multiple levels in a single system one needs to check that programs cannot leak data from a higher level to a lower one, either directly or indirectly. The property *non-interference* is used to capture ‘no information leaks’; the high level activities and data do not interfere in any observable way with the low level activities and data. This guarantees that anyone with only access to low level information cannot discern anything about the high level information. Of course, what it means to ‘not interfere in an observable way’ depends on what is observable i.e. on our attacker model. To precisely, formally define non-interference we thus rely on a notion of *indistinguishability* to capture the observation capabilities of an attacker. Indistinguishability gives whether two situations are the same from an attacker’s point of view. Note that ‘being able to tell situations apart’ is just a way of capturing/describing knowledge. E.g. Consider a program that outputs two booleans b_1 and b_2 and an attacker that knows the output b_1 but not output b_2 . We express this by saying that the attacker can distinguish between the situations in which b_1 is true and any other situation in which b_2 but not between e.g. $b_1 = \text{true}, b_2 = \text{true}$ and $b_1 = \text{true}, b_2 = \text{false}$. (See also Exercise ??). A common assumption is that the attacker can observe all input and output at the lower level, but not those at the high level. (Typically only two levels are considered in the analysis; all levels at or below the attacker’s level are grouped in the ‘low’ level and all other levels in the ‘high’ level.) If we also consider certain covert channels then the related side channels needs to be included (modelled) in the indistinguishability notion as well.

Given a notion of indistinguishability we define non-interference as; if the inputs given to the system are indistinguishable for the attacker then the output (including side channels where relevant) is also indistinguishable. In other words the attacker learns nothing from the system running; all knowledge she has afterwards she already had before hand.

Multi Lateral Security

- Compartmentation
 - Non-hierarchical levels
 - Lattice structure
- Chinese wall
 - Prevent conflict of interest
 - Non-interference

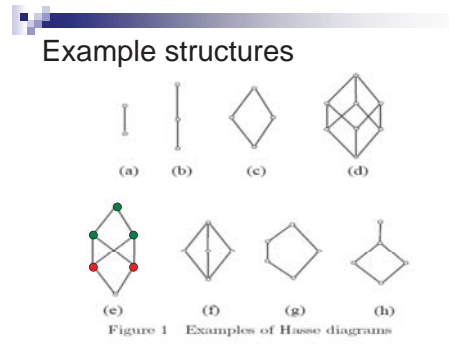
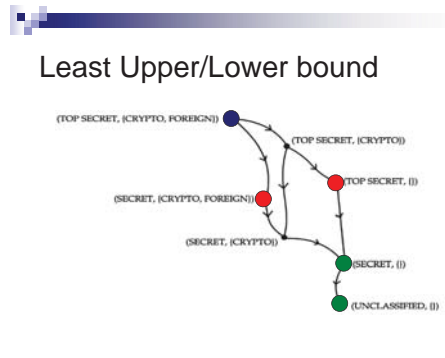
source: Security Engineering, Anderson

Lattice based security models

- Set of security labels
- partial order
 - some elements may be incomparable
- A least upper bound for each set
- A greatest lower bound for each set

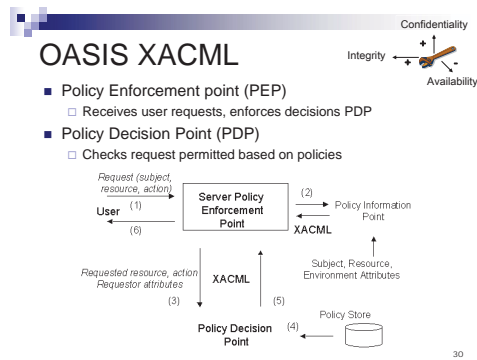
Sometimes the confidentiality of data is determined e.g. by the topic it treats and a should be restricted to those involved in that topic; e.g. data about foreign countries should be available to a low ranking diplomat but not to minister of the interior, eventhough the minister has a higher clearance level in general. Compartmentalization addresses this introducing security levels that can be incomparable; e.g. Secret-Foreign is lower than Top Secret-Foreign but incomparable to Top Secret-Internal. (Related to this are Chinese Wall policies. Consider a consultancy company that works for multiple competing companies. The companies will not want their information to get to a competing company. A Chinese Wall policy can be applied; people working on project X are not allowed to work on projects for competing companies.) The security levels are not totally ordered but form a *lattice*. In a lattice not all levels can be compared but given a set of levels there is always a least upper bound; a level that is at least as big as all levels in the set and is the smallest of all

levels that satisfy this property. In this way, if we combine different documents, we know the level the new document should have. Similarly a lattice has a greatest lower bound; if several people are working together than the greatest lower bound of their clearance levels is used to determine the documents they may access.



If we consider the red levels in slide 27 above we see that the blue level is the only one which is above all of the red levels. Thus it is automatically the smallest of these, i.e. the least upper bound. Both green levels are below all the red levels (lower bounds). The 'Secret' level is the biggest of the green levels; it is the greatest lower bound. If we look at the Hasse diagrams of slide 28 we see that all are lattices except figure e. The green levels are the upper bounds for the set of red levels. However, there is no smallest amongst the green levels.

5.3 XACML



- ### XACML Policy (sets):
- Policy set
 - Combining Algorithm
 - Set of Policies
 - Policy
 - Target (applicable to what)
 - Attributes of Subject, Resource, Action, Environment
 - Rule (when applicable)
 - Attributes as above and conditions.
 - Obligations

The eXtensible Access Control Markup Language (XACML) standard is a popular access control language and enforcement system. The system defined several components involved in the AC enforcement. The Policy Enforcement Point (PEP) is responsible for intercepting requests and ensuring that users only get access to the granted resources. The PEP uses a Policy Decision Point (PDP) to determine which requests are allowed (should be granted). The PEP gathers information about the subject and the resource and the context in which the request is being issued (the environment) from a Policy Information Point (PIP) and incorporates this in the request sent to the PDP. The PDP tests the request against the set of policies that it has in its policy store to make a granted/denied decision. The PDP can also return values indicating it is unable to make a decision: indeterminate (some error occurred) or not-applicable (this PDP has no policies related to the request).

A policy set contains, in addition to a list of policies (hence policy list would be a better name than policy set-the order can matter), a combination algorithm that is used to combine the different

decisions of the policies in the set. Examples are 'first applicable'; the first policy to return a decision gets selected, 'DENY overrides'; if a single policy says to deny the request this is the end decision, even if others allow the request and 'PERMIT overrides' where a single permit decision overrides any other decisions.

A policy has a target which determines which requests it is (or at least might be) applicable to. For those requests the rules of the policy will be evaluated. The rules, which each also have a target, check conditions (e.g. the issuer of the requester is a student) to either PERMIT or DENY a request. To combine the answers of different rules, the policy also has a combining algorithm. Target and conditions are expressed in attributes which can be many things; the ID or role of the requester, a name or label of the file, time of day, etc. Finally obligations may be associated with a decision; e.g. PERMIT the read operation but with some action that should be executed; e.g. notify the data owner, delete any copy of the data within a week, etc.

A policy in XACML

```
<Policy PolicyId="ExamplePolicy"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:permit-overrides">
  <Target>
  <Subjects> <AnySubject/> </Subjects>
  <Resources> <Resource>
  <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
  http://server.example.com/code/docs/developer-guide.html
  </AttributeValue>
  <ResourceAttributeDesignator
  DataType="http://www.w3.org/2001/XMLSchema#anyURI"
  AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
  </ResourceMatch>
  </Resources> </Resource>
  </Target>
  <Actions> <AnyAction/> </Actions>
  </Policy>
  ...
  source: sunxacml.sourceforge.net/guide.html 33
```

A policy in XACML (cont.)

```
...
  <Rule RuleId="ReadRule" Effect="Permit">
  <Target> <Subjects> <AnySubject/> </Subjects>
  <Resources> <AnyResource/> </Resources>
  <Actions> <Action>
  <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
  <AttributeValue DataType="...#string">read</AttributeValue>
  <ActionAttributeDesignator
  DataType="...#string" AttributeId="urn:...action-id"/>
  </ActionMatch> </Action> </Actions>
  </Target>
  <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
  <Apply FunctionId="urn:...function:string-one-and-only">
  <SubjectAttributeDesignator DataType="...#string" AttributeId="group"/>
  </Apply>
  <AttributeValue DataType="...#string">developers</AttributeValue>
  </Condition>
  </Rule>
  </Policy>
  source: sunxacml.sourceforge.net/guide.html 34
```

Policies are written in XML making them human and machine readable. (Though XML and XACML is human readable it is not very human friendly; interpreting a policy can be hindered by the large amount of textual overhead.) The example on the slide shows part of a policy which allows a developer to read the developers guide document. Both the generality and power of the system as well as the fact that it is cumbersome to specify policies by hand already become clear from this simple example.

A Request in XACML

```
<Request>
  <Subject>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
  DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
  <AttributeValue>seth@users.example.com</AttributeValue>
  </Attribute>
  <Attribute AttributeId="group" DataType="http://www.w3.org/2001/XMLSchema#string"
  Issue="admin@users.example.com"> <AttributeValue>developers</AttributeValue>
  </Attributes>
  </Subject>
  <Resource>
  <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
  DataType="http://www.w3.org/2001/XMLSchema#string"> <AttributeValue>
  http://server.example.com/code/docs/developer-guide.html
  </AttributeValue> </Attribute>
  </Resource>
  <Actions> <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
  DataType="http://www.w3.org/2001/XMLSchema#string">
  <AttributeValue>read</AttributeValue> </Attributes>
  </Actions>
  </Request>
  source: sunxacml.sourceforge.net/guide.html 32
```

PDP response in XACML

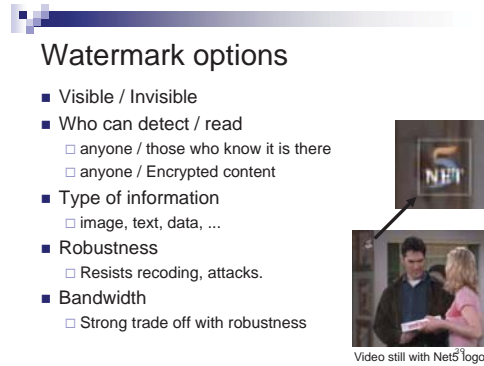
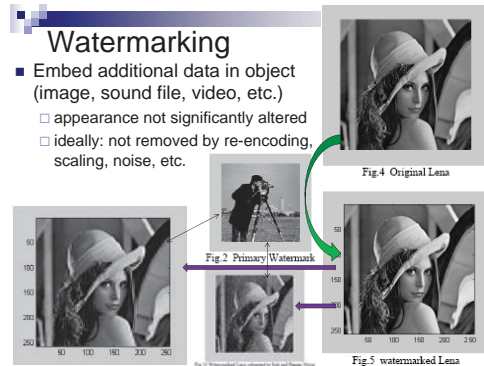
```
<Response>
  <Result>
  <Decision>Permit</Decision>
  <Status>
  <StatusCode
  Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
  </StatusCode>
  </Result>
  </Response>
  source: sunxacml.sourceforge.net/guide.html 35
```

The requests and responses are also formulated in XML. The request basically describes who (user and attributes; Seth from group developers) wants to do what (action; read) with which resource (developer guide). It may also contain information about the environment; e.g. the request was made from within the office and during working hours. The response contains the decision and a status code.

The system is very general and by using certain formats it can be used in combination with other identification and access control systems and standards e.g. 'SAML profile for XACML'. Note that XACML is an 'attribute based' access control system; the policies use attributes (e.g. developer) (of the requester, resource and/or environment) to make its access control decision.

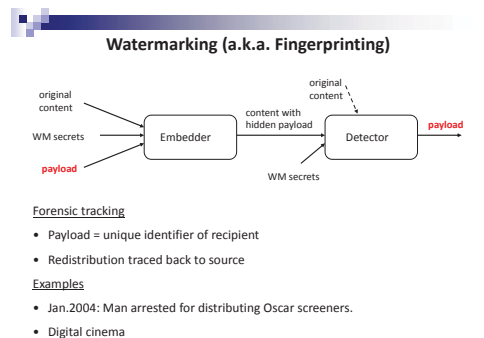
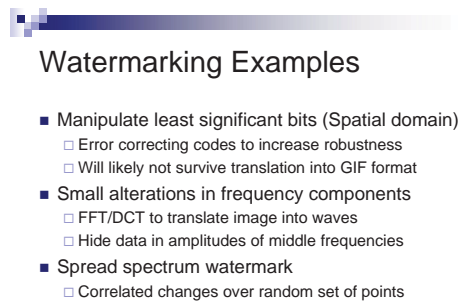
5.4 Watermarking

So far in the discussion we have focussed on preventative access control for data being controlled by some authority. Distributed access control addressed the issue of sharing data and rights with other, trusted, authorities. But what if the data leaves the premises of the authority? We would still want our policies to be adhered to. One way is trying to enforce policies on remote data, e.g. with digital rights management techniques discussed below. The other option is to deter rather than prevent misuse. For this second approach we will, of course, need to recognize our data being misused and determine where the policy violation happened.



Watermarking is a technique to embed extra information into an (typically media) object. The extra information should not make the object unusable/reduce its value; e.g. no significantly altering of appearance. A good watermark is robust; it will survive different operations the object may be subjected to; e.g. changing the size or encoding format of an image, added noise, etc.

The embedded information can be used to identify the content and/or its origin. For example the 'Net 5' logo on the image is an example of a visible watermark that allows anyone to recognize that the image is taken from a broadcast by that company. Other watermarks may encode different information, be invisible (or at least; not apparent) and may only be detectable by those that know how to look for it.



One method to watermarking an image is to manipulate the least significant bits; this results in minute color changes which likely will not be visible to the user. A problem is that some encoding techniques reduce the size of the image by removing exactly this type and for the same reason; the user will likely not notice the difference between the original and the compressed image. A second method uses the frequency instead of the spatial domain; instead of as a grid (bitmap) an image (color values) can be seen as the sum of a number of waves with different amplitudes and frequencies. Slightly altering amplitudes will also have only small visual effects but spread over the picture rather than a single bit making it more robust.

The amount of information that can be added with a specific embedding method, its bandwidth, needs to be balanced with its robustness, as increasing robustness and carrying more information both lead to more extensive changes to the object.

One way to use watermarking is for so called traitor tracing. By embedding a unique value into each copy of the object that you give out, you can find the source of the leak if a leaked document is encountered. An example is the movie screeners of Oscar candidates sent to the reviewers.

Clearly a potential ‘traitor’ would want to remove or distort the watermark in the object so it can no longer be recognized as being his copy. Again the attacker model is important to consider when evaluating (attacks) against watermarks; what knowledge does the attacker have.

Spread spectrum is a way to try to hide the watermark; the ‘locations’ where the watermark is embedded are chosen randomly making it much harder for an attacker to find and distort the watermark.

Collusion attacks

“Coalition of pirates”

- Users pool their content
- Differences point to watermark
- Attackers remove watermark

■ = “detectable positions”

pirate #1	1	1	1	0	1	0	1	0	0	0	0	1
#2	1	0	1	0	1	0	1	0	1	0	0	1
#3	1	0	1	0	1	0	1	0	0	0	0	1
#4	1	1	1	0	0	0	1	1	0	0	0	1
Attacked Content	1	0/1	1	0	0/1	0	1	0/1	0/1	0	0/1	1

43

Tardos scheme: Creation

n users
m: watermark bits

Step 1: choose $p_1, \dots, p_m \in (0,1)$; according to:

Step 2: Generate matrix X with $\text{Prob}[X_{ji}=1] = p_i$.

	p_1	p_2						p_i												p_m
user j								1												1
								1												1
								0												0
								0												0
								1												1

bit i of watermark

Step 3: Give user j content with X_j embedded (keep p and X)

44

If multiple ‘traitors’ cooperate, it makes it easier for them to find the watermark; difference between their copies of the same content point at locations where the watermark is embedded. By creating a combined version which distorts these locations the traitors may try to hide that they are involved in the leaking.

The Tardos scheme: Detection

$$S_j = \sum_{i=1}^m y_i g(X_{ji}, p_i)$$

Observed (y)	1	1	1	0	1	0	1	0	0	0	0	1
User j (X_j)	1	0	1	0	1	0	1	0	1	0	1	1
Evidence	-	-	+	+	-	-	+	+	-	-	+	+

- Image with payload y is observed.
- Compare 1’s in y to stored payload X_j
 - If X_j also has 1 then evidence guilt of user j
 - If X_j has 0 then evidence innocence of user j
- Strength of evidence (g):
 - for guilt: big if p is small $g(1,p) = \sqrt{\frac{1-p}{p}}$
 - for innocence: big if 1-p small $g(0,p) = -\sqrt{\frac{p}{1-p}}$
- User #j gets accused if $S_j >$ threshold

45

The Tardos scheme: Detection

Proven properties:

With sufficiently long code m:

$$m = 100c_0^2 \ln(1/\epsilon_1), \quad \epsilon_2 = \epsilon_1^{c_0/4}$$

n = #users m = code length
 ϵ_1 = Prob[accuse specific innocent]
 ϵ_2 = False Negative prob.

- Resistance against up to c_0 colluders independent of coalition strategy
- If $c > c_0$, then
 - no false accusations!
 - high prob. that nobody gets accused

46

Watermark encodings such as the Tardos scheme aim to protect against colluding traitors. It assumes that some method exists to embed and extract watermark bits in the content and focuses on what codes to encode in the watermarks. For each bit i to be embedded a probability p_i is chosen. Then, for user j the code to be embedded is chosen randomly according to the probabilities p_i . The resulting code X_j is embedded in the content and stored.



If later a leaked copy of the content is observed, its watermark y is extracted and compared against the stored codes. Each bit y_i which is 1 in the observed code and also 1 in stored code j is an indication that X_j was involved in creating the leaked content. If the likelihood p_i of getting a one at location i is low then this evidence is stronger; only few people beside j will also have a 1 at this position. If instead X_j has a zero at this location it is an indication that j was innocent. Similarly

the strength of this evidence depends on the probability p_j . (The given formulas have been shown to be the best choice for 'evidence strength'.) We add the evidence for guilt and subtract the evidence for innocence to obtain an accusation value. If this is larger than a set threshold user j gets accused. An obvious improvement to the above scheme, is to also consider the zero's in the observed code, considering whether the two bits are equal or different.



The analysis of the Tardos codes has shown that if the code is long enough (quadratic the number of traitors) the code is resistant against collusion attacks.

5.5 Digital Rights Management (DRM)

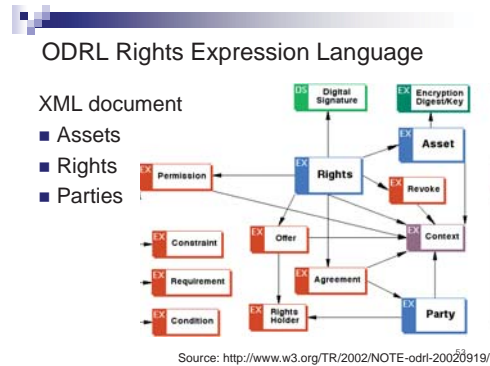
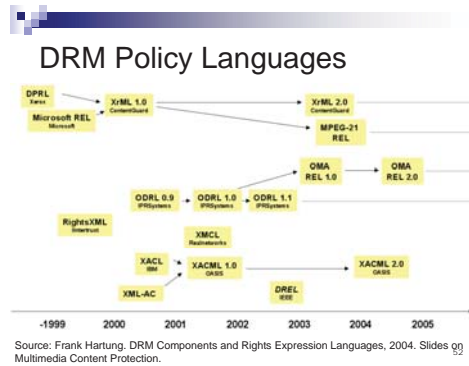
Watermarking allows *detection* of misuse of content. Digital rights management tries to *enforce* policies even though the content leaves the premises of the trusted authority. The basic idea is to encrypt the content and allow content to only be decrypted in a 'trusted environment'.

 <h3>Digital Rights Management</h3> <ul style="list-style-type: none"> ■ Usage rules <ul style="list-style-type: none"> □ Expressed by a digital license □ Grants rights to a user, e.g. <ul style="list-style-type: none"> ■ Frequency of access ■ Expiration date ■ Copy and transfer rights □ Match business model <ul style="list-style-type: none"> ■ rental, subscription, try-before-buy, pay-per-use, ... 	 <h3>DRM Trust Model</h3> <ul style="list-style-type: none"> ■ DRM is not only technology ■ Trust Model specifies <ul style="list-style-type: none"> □ Legal agreements □ Compliance rules □ Robustness rules □ Security related services
51	49

DRM is not just a technology; an effective technology is not possible without some assumptions about its use. Thus a DRM system also defines an environment in which the technology is deployed to ensure these assumptions can be met. A trust model specifies (legal) responsibilities of parties involved, and rules they have to comply with.

 <h3>DRM Examples</h3> <ul style="list-style-type: none"> ■ HDCP (hdmi) ■ iTunes (FairPlay), Windows Media DRM ■ Video on demand (set top boxes with smartcard) ■ Content Scrambling System (CSS) <ul style="list-style-type: none"> □ used on DVD, quickly broken ■ non skipable tracks on DVDs <ul style="list-style-type: none"> □ Early players had 'bugs' which allowed skipping ■ Online Free Record Shop ■ Sony's 'rootkit' (XCP) ■ OMA, Marlin, OMArlin 	 <h3>Digital Rights Management</h3> <ul style="list-style-type: none"> ■ Usage rules <ul style="list-style-type: none"> □ Expressed by a digital license □ Grants rights to a user, e.g. <ul style="list-style-type: none"> ■ Frequency of access ■ Expiration date ■ Copy and transfer rights □ Match business model <ul style="list-style-type: none"> ■ rental, subscription, try-before-buy, pay-per-use, ...
50	51

DRM enforces policies. It thus has a policy language with which to express policies; see the discussion on access control policies before. By using different types of policies the same system can support different business models for use of the content. Note that AC policy languages such as XACML can also be suitable as DRM policy languages; both aim to specify usage rights of resources. Permissions/Rights/actions, objects/resources/assets and subjects/users/parties, different names for the same key components.



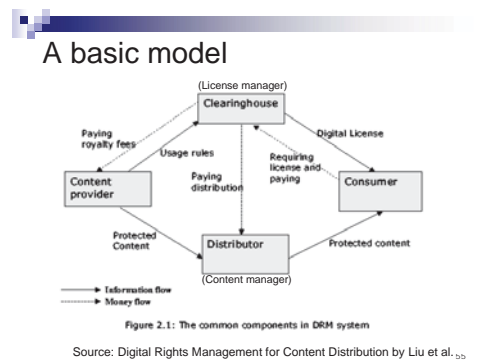
The management part of DRM involves a distribution method and enforcing the specified rights (within a compliant environment, see discussion above)

Digital Rights Management

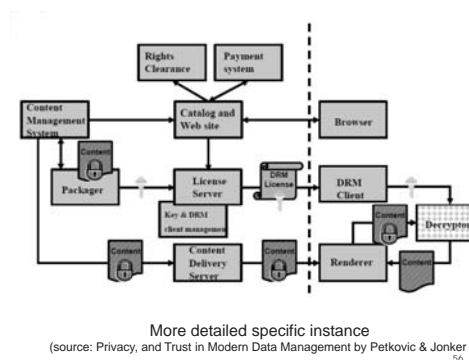
- Specification of usage rules
- Distribution of licenses
- Enforce or check compliance with licenses
 - e.g. encrypt content, require trusted environment for decryption (e.g. requires specific viewer)
 - dedicated hardware, e.g. set top box with smartcards, DIVX, USB dongles
 - Self destructing media (EZ-Ds, Self-Destructing USB Stick)

Inspector gadget reading a 'single use' message

54



A basic model for a DRM based distribution system considers besides the providers and consumer of content, also distributors which provide protected (e.g. encrypted) content and a rights seller that sells the rights (licenses that contain decryption keys, usable by compliant hardware) to use the content.



The Analogue hole

- Fundamental problem DRM
 - Content needs to be shown to user
- Record and digitalize analogue signal
 - Loss of quality
 - Amount of effort
- Trusted playback
 - DRM embedded in TV (e.g. HDMI)
- Trusted recording devices
 - VCR/Video camera does not record protected content

60

A more detailed view of a specific system shows the main components of the system the part on the right is local to the consumer. The lower components deal with decryption keys and decrypted content and thus need to be trusted by the content provider (thus will typically need to be compliant devices). Of course the final recipients of the content are not compliant devices; the consumer themselves. The content will have to be converted to analog form to be consumed. Once in analog form most protection scheme will no longer work; this is known as the analog hole. In theory one could e.g. watermark the content even in analog form and have the recording

device look for watermarks and refuse to record watermarked content. The technical problem is having to make each recording device compliant (and use the same system). An more important problem, which applies to all DRM methods is that of trade-offs; the DRM restricts the use of the content but some of this use is perfectly legitimate (e.g. fair use), even though the license does not specifically state the right. What falls under fair is different for different countries and is subject of ongoing legal debate; capturing it in a DRM policy is not possible.



DRM (cont.)

- Different goals but usable for DRM:
 - Mandatory access control
 - Access control policies set by administrator determine rights, not creator of the files
 - implemented in SELinux, SUSE AppArmor, TrustedBSD, Trusted Solaris, ...
 - Trusted computing
 - Trusted Platform module
 - Create required trusted component

57



Other applications of DRM

- Protection of private data within enterprises
 - Place usage policy on data entered in web form
 - Electronic health records
- Alternative to enforcing: a posteriori AC
 - Possible if users can be held accountable
 - Clear for examples above. More difficult in general
 - Observing actions
 - Tracing source of data (e.g. watermarking)

62

As DRM is a form of access control, some access control systems can also be used for DRM and DRM systems can also be applied for addressing access control requirements in other settings than multimedia content protection, in which case they are often referred to as 'sticky policies'. If users can be held accountable, one could use other sticky policies with monitoring instead of enforcement; data use is monitored and compliance to policies checked afterwards rather than enforced at access time. For settings where availability of data is even more important than confidentiality, like in a hospital, this is a preferable solution.

5.6 Conclusions

In this chapter we have looked at access control and content protection mechanisms. Expressing rights (policy) and enforcing them are the key goals. This has to be done in a convenient, usable and maintainable way. The need for convenience and ease of use apply for the specifying and maintaining of policies on the one hand but also for the users of the protected content/resources on the other hand.

If you wish to learn more about access control mechanisms there is the Master course on Distributed Trust Management (2IS25).

5.6.1 Literature

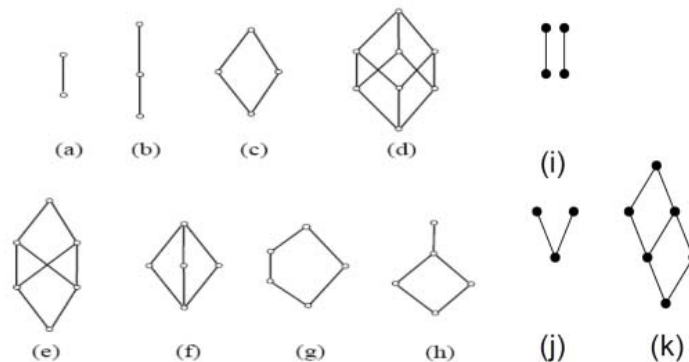
Suggested reading (check the course page [2] for the most up to date list of suggested reading materials):

- Security Engineering Introduction [3, Ch 4,7,8,20]; access control and related topics as well as copyrights (DRM and Watermarking).
- Logic in Access Control <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01210062>
- A Brief Introduction to XACML http://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html
- Applying DRM to E-health data <http://www.springerlink.com/content/rw850u12353q5u18>

- Privacy Preserving DRM <http://www.springerlink.com/content/w4x8j2vknv0yumgn>
- Digital Watermarking scheme <http://www.cs.ucla.edu/~miodrag/cs259-security/ip97.pdf>. Main ideas (e.g. what is a spread spectrum, collusion attack).
- Tardos fingerprinting is better than we thought https://venus.tue.nl/ep/cgi/ep_publ_detail.opl?rn=20088020&volgnr=217909. Sections 1 and 2.
- Patent on Digital Watermarking <http://www.google.nl/patents?hl=nl&lr=&vid=USPAT5915027&id=ao4XAAAAEBAJ&oi=fnd&printsec=abstract>. Main ideas of watermarking; what does a patent look like.

5.7 Exercises

- (a) A lattice is a partial order in which each set has a least upper and a greatest lower bound. Which of the orders in the hasse diagrams below are NOT lattices?
 - (b) A common requirement for multilevel access control systems is that the security labels form a lattice. Explain why.
 - (c) Permission systems for access control are often monotonic; additional credentials can only add permissions, never remove them. Why is this important in a distributed environment?



2. Consider again the online music store of the previous two chapters. Review your requirements analysis, adding AC considerations; threats and countermeasures where appropriate.
 3. (*) In the logical AC formula $DrOf(D, x) \rightarrow mayRead(D, Record(x), x)$ we are actually using a function *Record* that for any patient returns the object that is the record of that patient. Give an equivalent formulation using only predicates (you will need to introduce a new predicate).

4. Describe the following scenario and situations (as far as possible) using a logical system, a role based access control system and the resulting access control matrices.

A Hospital has a patient electronic health record (EHR) system. An EHR describes the medication history of a patient. There are two possible actions on the EHR; read the content and add a new prescription.

- The hospital has Doctors Daisy and Edward, Nurses Nancy and Mark and Patients Alice, Bob and Charlie.
- Doctors are allowed to read the health records of patients.
- The doctor treating a patient is allowed to add new prescriptions and may let a nurse read the health record of the patient.

- Daisy is treating Alice and Bob, Edward is treating Charlie.
- Nurse Nancy is assisting Daisy with the treatment of Alice.

Give a scenario in which Nancy reads the record of Alice; include the steps involved and what happens in/with the AC system.

5. Translate the following XACML Policy to text:

```
<Policy PolicyId="SamplePolicy" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
<Target>
<Subjects> <AnySubject/> </Subjects>
<Resources> <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">SampleServer</AttributeValue>
  <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
    AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
</ResourceMatch> </Resources>
<Actions> <AnyAction/> </Actions>
</Target>

<Rule RuleId="LoginRule" Effect="Permit">
<Target>
<Subjects> <AnySubject/> </Subjects>
<Resources> <AnyResource/> </Resources>
<Actions> <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">login</AttributeValue>
  <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
    AttributeId="ServerAction"/>
</ActionMatch> </Actions>
</Target>

<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
    <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
      AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
  </Apply>
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
</Apply>
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
    <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
      AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
  </Apply>
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</AttributeValue>
</Apply>
</Condition>
</Rule>

<Rule RuleId="FinalRule" Effect="Deny"/>
</Policy>
```