Lecture 2, April 1

ECS 235B, Foundations of Computer and Information Security

Spring Quarter 2026

Attributes

- *attribute*: variable of a specific data type associated with an entity
- *att*(*o*): set of attribute values associated with *o*, called the *attribute value tuple* of *o*
  - Each attribute is written $o.a_i$, with value v drawn from set Va_i
- *attribute predicate*: boolean expression built from attributes and constants with appropriate operation and relation symbols
  - Unary predicate: built from one attribute
  - Binary predicate: built from two attributes
  - Can have as many attributes in a predicate as needed
  - Example: *Alice.credit* ≥ $100.00

Attribute Based Access Control Matrix (ABAM)

- Change access control matrix so rows correspond to subjects and their attributes, and columns correspond to objects and their attributes
- Note access control matrix discussed previously is special case
  - Just make the attribute sets be empty

Primitive Operations

- **enter**, **delete** as before
- **create subject** *s* **with attribute tuple** *att*(*s*): create subject *s* with given attribute tuple; additionally, add an identity attribute with a unique value
- **create object** *o* **with attribute tuple** *att*(*o*): create object *o* with given attribute tuple; additionally, add an identity attribute with a unique value
- **destroy** as before except it also deletes. the associated attribute tuple
- **update attribute** $o.a_i$: update $att(o) = (v_1, ..., v_i, ..., v_n)$ to

$$att(o)' = (v_1, ..., v_i', ..., v_n), \text{ where } v_i, v_i' \in Va_i, \text{ and } v_i \neq v_i'$$

Commands

- Like previous commands, except that conditions may include attribute predicates
- Let *p* give *q r* rights over *f*, if *p* owns *f* and value of *p*'s attribute *jobcode* is between 3 and 5 inclusive

```
command grant•read•file•attribute•3to5(p, f, q)
  if own in A[p, f] and 3 ≤ p.jobcode and p.jobcode ≤ 5
  then
    enter r into A[q, f];
end
```

What Is "Secure"?

- Adding a generic right *r* where there was not one is "leaking"
  - In what follows, a right leaks if it was not present *initially*
  - Alternately: not present *in the previous state* (not discussed here)
- If a system *S*, beginning in initial state s_0, cannot leak right *r*, it is *safe with respect to the right r*
  - Otherwise it is called *unsafe with respect to the right r*

Safety Question

- Is there an algorithm for determining whether a protection system *S* with initial state s_0 is safe with respect to a generic right *r*?
  - Here, “safe” = “secure” for an abstract model

Mono-Operational Commands

- Answer: *yes*
- Sketch of proof:

  Consider minimal sequence of commands c_1, …, c_k to leak the right.

  - Can omit **delete**, **destroy** (with some rewriting)
  - Can merge all **create**s into one

  Worst case: insert every right into every entry; with s subjects and o objects initially, and n rights, upper bound is $k \leq n(s+1)(o+1)+1$

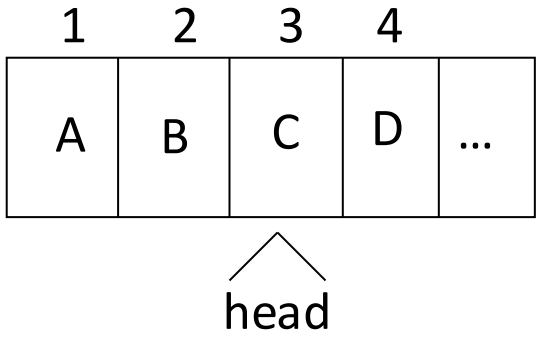
General Case

- Answer: *no*
- Sketch of proof:

  Reduce halting problem to safety problem

  Turing Machine review:
  - Infinite tape in one direction
  - States *K*, symbols *M*; distinguished blank *b*
  - Transition function $\delta(k, m) = (k', m', \text{L})$ means in state *k*, symbol *m* on tape location replaced by symbol *m′*, head moves to left one square, and enters state *k′*
  - Halting state is q_f; TM halts when it enters this state

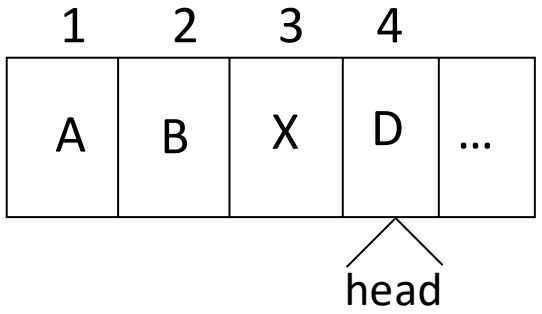
Mapping

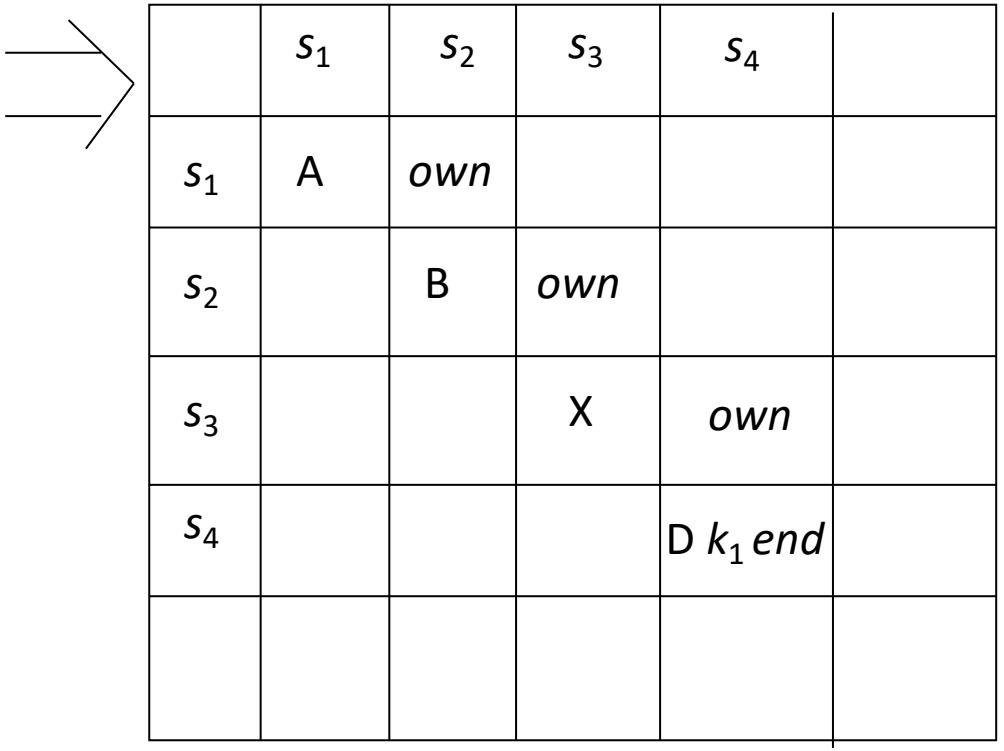


Current state is *k*

	s_1	s_2	s_3	s_4	
s_1	A	*own*			
s_2		B	*own*		
s_3			C *k*	*own*	
s_4				D *end*	

Mapping





	s_1	s_2	s_3	s_4	
s_1	A	*own*			
s_2		B	*own*		
s_3			X	*own*	
s_4				D k_1 *end*	

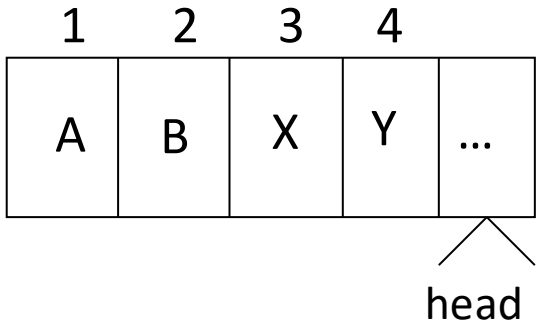
After $\delta(k, C) = (k_1, X, R)$ where k is the current state and k_1 the next state

Command Mapping

- $\delta(k, C) = (k_1, X, R)$ at intermediate becomes

```
command c_{k,C}(s_3, s_4)
if own in A[s_3, s_4] and k in A[s_3, s_3] and C in A[s_3, s_3]
then
  delete k from A[s_3, s_3];
  delete C from A[s_3, s_3];
  enter X into A[s_3, s_3];
  enter k_1 into A[s_4, s_4];
end
```

Mapping



After $\delta(k_1, D) = (k_2, Y, R)$ where k_1 is the current state and k_2 the next state

	s_1	s_2	s_3	s_4	s_5
s_1	A	*own*			
s_2		B	*own*		
s_3			X	*own*	
s_4				Y	*own*
s_5					*b* k_2 end

Command Mapping

- $\delta(k_1, \text{D}) = (k_2, \text{Y}, \text{R})$ at end becomes

```
command crightmost_{k,C}(s_4,s_5)
if end in A[s_4,s_4] and k_1 in A[s_4,s_4]
        and D in A[s_4,s_4]
then
  delete end from A[s_4,s_4];
  delete k_1 from A[s_4,s_4];
  delete D from A[s_4,s_4];
  enter Y into A[s_4,s_4];
  create subject s_5;
  enter own into A[s_4,s_5];
  enter end into A[s_5,s_5];
  enter k_2 into A[s_5,s_5];
end
```

Rest of Proof

- Protection system exactly simulates a TM
  - Exactly 1 *end* right in ACM
  - 1 right in entries corresponds to state
  - Thus, at most 1 applicable command
- If TM enters state q_f, then right has leaked
- If safety question decidable, then represent TM as above and determine if q_f leaks
  - Implies halting problem decidable, which we know is false
- Conclusion: safety question undecidable

Other Results

- Set of unsafe systems is recursively enumerable
- Without **create** primitive, safety question is complete in **P-SPACE**
- Without **destroy**, **delete** primitives, then safety question is undecidable
  - Systems are called “monotonic”
- Safety question for biconditional protection systems is decidable
- Safety question for monoconditional, monotonic protection systems is decidable
- Safety question for monoconditional protection systems with **create**, **enter**, **delete** (and no **destroy**) is decidable.

Take-Grant Protection Model

- A specific (not generic) system
  - Set of rules for state transitions
- Safety decidable, and in time linear with the size of the system
- Goal: find conditions under which rights can be transferred from one entity to another in the system

System

- ○ objects (files, …)
- ● subjects (users, processes, …)
- ⊗ don't care (either a subject or an object)

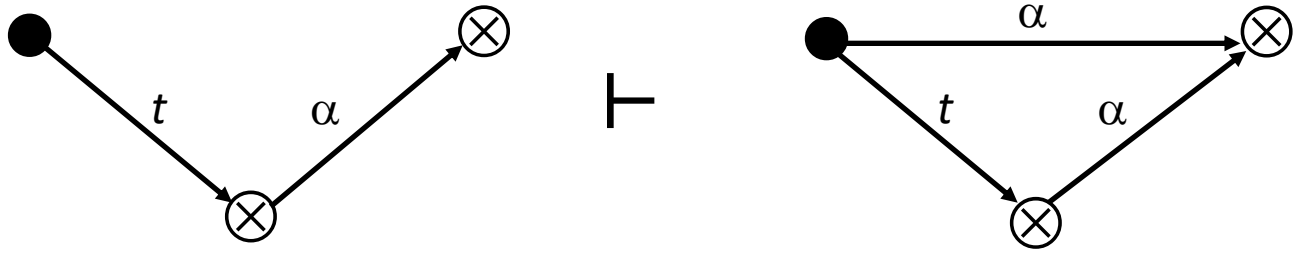
$G \vdash_x G'$ apply a rewriting rule x (witness) to G to get G'

$G \vdash^* G'$ apply a sequence of rewriting rules (witness) to G to get G'

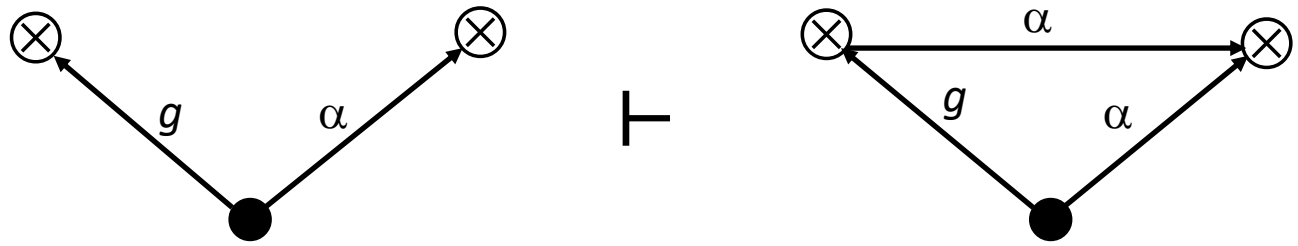
$R = \{ t, g, r, w, \ldots \}$ set of rights

Rules

take

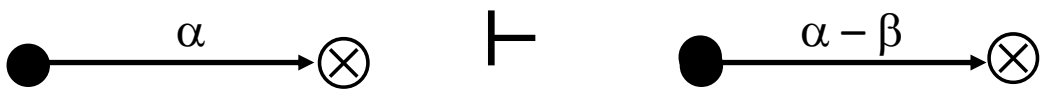


grant



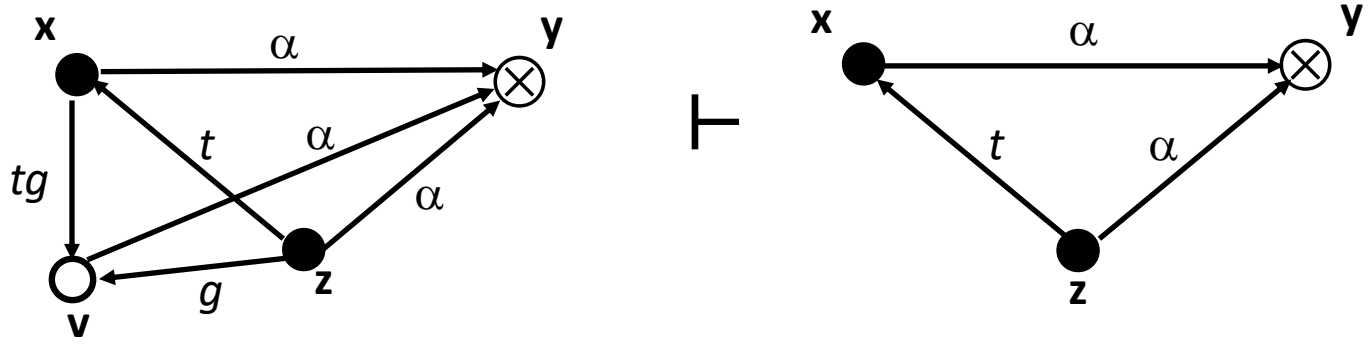
More Rules

create 

remove 

These four rules are called the *de jure* rules

Symmetry



1. ***x*** creates (*tg* to new) ***v***
2. ***z*** takes (*g* to ***v***) from ***x***
3. ***z*** grants (α to ***y***) to ***v***
4. ***x*** takes (α to ***y***) from ***v***

Similar result for grant

Islands

- *tg*-path: path of distinct vertices connected by edges labeled *t* or *g*
  - Call them “tg-connected”
- island: maximal *tg*-connected subject-only subgraph
  - Any right one vertex has can be shared with any other vertex

Initial, Terminal Spans

- *initial span* from **x** to **y**
  - **x** subject
  - *tg*-path between **x**, **y** with word in $\{ \vec{t}^*\vec{g} \} \cup \{ \nu \}$
  - Means **x** can give rights it has to **y**
- *terminal span* from **x** to **y**
  - **x** subject
  - *tg*-path between **x**, **y** with word in $\{ \vec{t}^* \} \cup \{ \nu \}$
  - Means **x** can acquire any rights **y** has

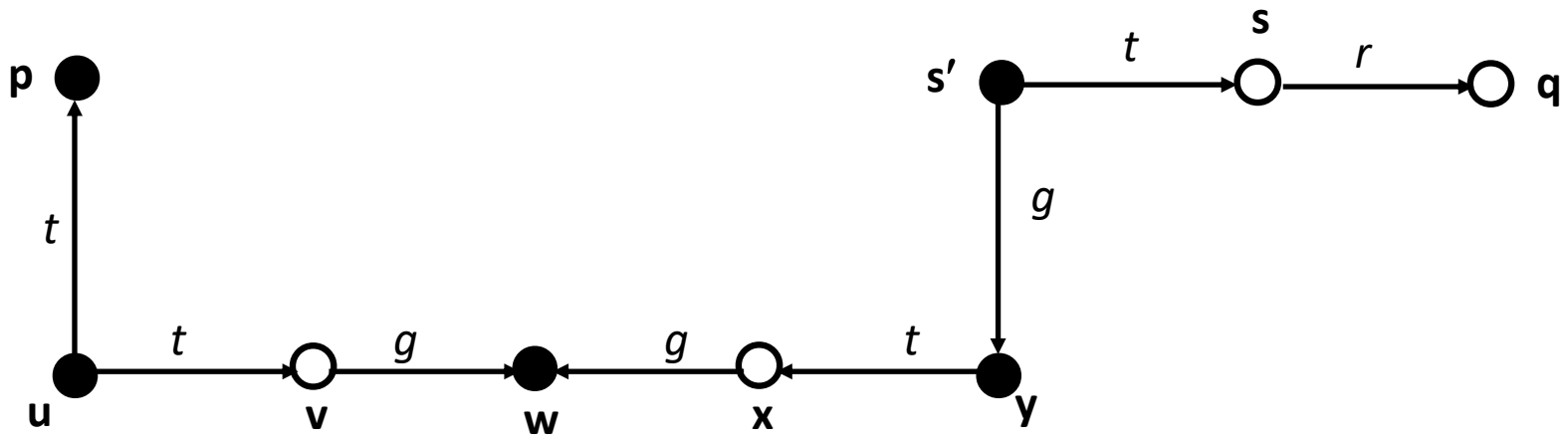
Bridges

- *bridge*: *tg*-path between subjects **x**, **y**, with associated word in

$$\{ \vec{t}^*, \overleftarrow{t}^*, \vec{t}^* g \overleftarrow{t}^*, \vec{t}^* g \overrightarrow{t}^* \}$$

  - Rights can be transferred between the two endpoints
  - *Not an island* as intermediate vertices must be objects
  - Note bridges are initial spans, terminal spans, or a combination of the two going to the same object

Example



- islands { **p**, **u** } { **w** } { **y**, **s′** }
- bridges **uvw**; **wxy**
- initial span **p** (associated word ν)
- terminal span **s′s** (associated word $\vec{t}$)

can•share Predicate

Definition:

- *can•share*(*r*, **x**, **y**, G_0) if, and only if, there is a sequence of protection graphs G_0, …, G_n such that $G_0 \vdash^* G_n$ using only *de jure* rules and in G_n there is an edge from **x** to **y** labeled *r*.

can•share Theorem

- *can•share*(*r*, **x**, **y**, G_0) if, and only if, there is an edge from **x** to **y** labeled *r* in G_0, or the following hold simultaneously:
  - There is an **s** in G_0 with an **s**-to-**y** edge labeled *r*
  - There is a subject **x′** = **x** or initially spans to **x**
  - There is a subject **s′** = **s** or terminally spans to **s**
  - There are islands I_1,…, I_k connected by bridges, and **x′** in I_1 and **s′** in I_k

Outline of Proof

- **s** has *r* rights over **y**
- **s′** acquires *r* rights over **y** from **s**
  - Definition of terminal span
- **x′** acquires *r* rights over **y** from **s′**
  - Repeated application of sharing among vertices in islands, passing rights along bridges
- **x′** gives *r* rights over **y** to **x**
  - Definition of initial span

Example Interpretation

- ACM is generic
  - Can be applied in any situation
- Take-Grant has specific rules, rights
  - Can be applied in situations matching rules, rights
- Question: what states can evolve from a system that is modeled using the Take-Grant Model?

Take-Grant Generated Systems

- Theorem: G_0 protection graph with 1 vertex, no edges; R set of rights. Then $G_0 \vdash^* G$ iff:
  - *G* finite directed graph consisting of subjects, objects, edges
  - Edges labeled with nonempty subsets of *R*
  - At least one vertex in *G* has no incoming edges

Outline of Proof

$\Rightarrow$: By construction; *G* final graph in theorem

- Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be subjects in *G*
- Let $\mathbf{x}_1$ have no incoming edges

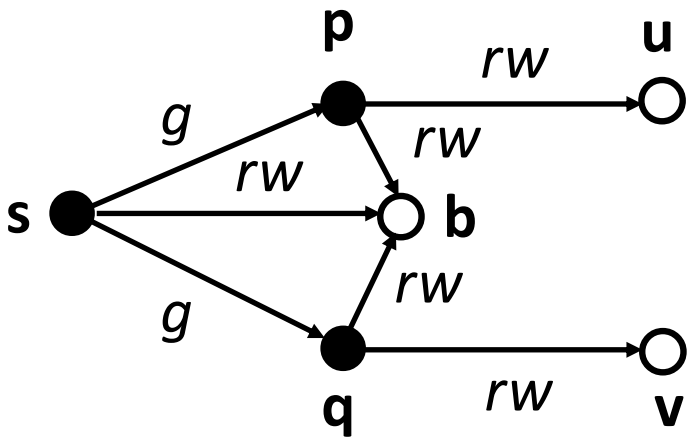
- Now construct *G´* as follows:
  1. Do "$\mathbf{x}_1$ creates ($\alpha \cup \{ g \}$ to) new subject $\mathbf{x}_i$"
  2. For all ($\mathbf{x}_i$, $\mathbf{x}_j$) where $\mathbf{x}_i$ has a rights over $\mathbf{x}_j$, do
     "$\mathbf{x}_1$ grants (α to $\mathbf{x}_j$) to $\mathbf{x}_i$"
  3. Let β be rights $\mathbf{x}_i$ has over $\mathbf{x}_j$ in *G*. Do
     "$\mathbf{x}_1$ removes (($\alpha \cup \{ g \} - \beta$ to) $\mathbf{x}_j$"
- Now *G´* is desired *G*

Outline of Proof

⇐: Let **v** be initial subject, and $G_0 \vdash^* G$

- Inspection of rules gives:
  - *G* is finite
  - *G* is a directed graph
  - Subjects and objects only
  - All edges labeled with nonempty subsets of *R*
- Limits of rules:
  - None allow vertices to be deleted so **v** in *G*
  - None add incoming edges to vertices without incoming edges, so **v** has no incoming edges

Example: Shared Buffer



- Goal: **p**, **q** to communicate through shared buffer **b** controlled by trusted entity **s**
    1. **s** creates ({*r*, *w*} to new object) **b**
    2. **s** grants ({*r*, *w*} to **b**) to **p**
    3. **s** grants ({*r*, *w*} to **b**) to **q**

can•steal Predicate

Definition:

- *can•steal*(*r*, **x**, **y**, G_0) if, and only if, there is no edge from **x** to **y** labeled *r* in G_0, and there exists a sequence of protection graphs G_0, G_1, …, G_n for which the following hold simultaneously:
  a) There is an edge from **x** to **y** labeled *r* in G_n
  b) There is a sequence of rule applications ρ_1, …, ρ_n such that $G_{i-1} \vdash G_i$ using ρ_i
  c) For all vertices **v** and **w** in G_{i-1}, $1 \leq i < n$, if there is an edge from **v** to **y** labeled *r* in G_0, then ρ_i is ***not*** of the form "**v** grants (*r* to **y**) to **w**"

can•steal Theorem

can•steal(α, **x**, **y**, G_0) if, and only if, the following hold simultaneously:

a) There is no edge from **x** to **y** labeled α in G_0

b) There exists a subject **x**′ such that **x**′ = **x** or **x**′ initially spans to **x**

c) There exists a vertex **s** with an edge labeled α to **y** in G_0

d) can•share(*t*, **x**′, **s**, G_0) holds

Outline of Proof

$\Rightarrow$: Assume conditions hold

- **x** subject
  - **x** gets *t* rights to **s**, then takes α to **y** from **s**
- **x** object
  - *can•share*(*t*, **x′**, **s**, G_0) holds
  - If **x′** has no α edge to **y** in G_0, **x′** takes (α to **y**) from **s** and grants it to **x**
  - If **x′** has α edge to **y** in G_0, **x′** creates surrogate **x′′**, gives it (*t* to **s**) and (*g* to **x′′**); then **x′′** takes (α to **y**) and grants it to **x**

Outline of Proof

$\Leftarrow$: Assume *can•steal*(α, **x**, **y**, G_0) holds

- First two conditions immediate from definition of *can•steal*, *can•share*
- Third condition immediate from theorem of conditions for *can•share*
- Fourth condition: ρ minimal length sequence of rule applications deriving G_n from G_0; i smallest index such that $G_{i-1} \vdash G_i$ by rule ρ_i and adding α from some **p** to **y** in G_i
  - What is ρ_i?

Outline of Proof

- Not remove or create rule
  - **y** exists already
- Not grant rule
  - G_i first graph in which edge labeled α to **y** is added, so by definition of *can•share*, cannot be grant
- take rule: so *can•share*(*t*, **p**, **s**, G_0) holds
  - So is subject **s′** such that **s′** = **s** or terminally spans to **s**
  - Sequence of islands with **x′** $\in I_1$ and **s′** $\in I_n$
- Derive witness to *can•share*(*t*, **x′**, **s**, G_0) that does not use

  "**s** grants (α to **y**) to" anyone

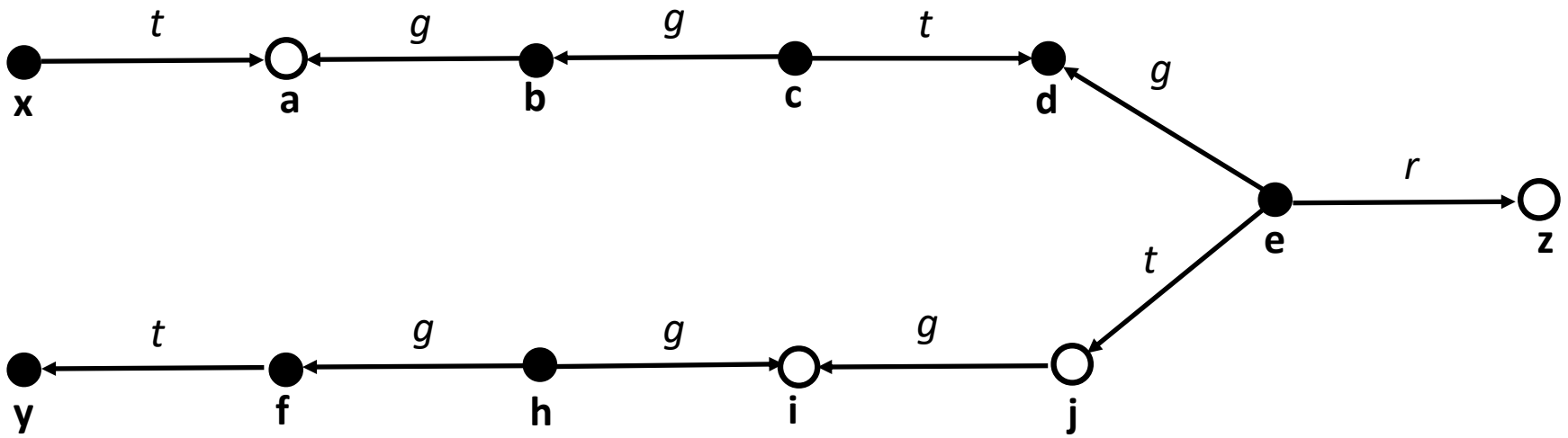
Conspiracy

- Minimum number of actors to generate a witness for

  $can\bullet share(\alpha, \mathbf{x}, \mathbf{y}, G_0)$

- Access set describes the "reach" of a subject
- Deletion set is set of vertices that cannot be involved in a transfer of rights
- Build *conspiracy graph* to capture how rights flow, and derive actors from it

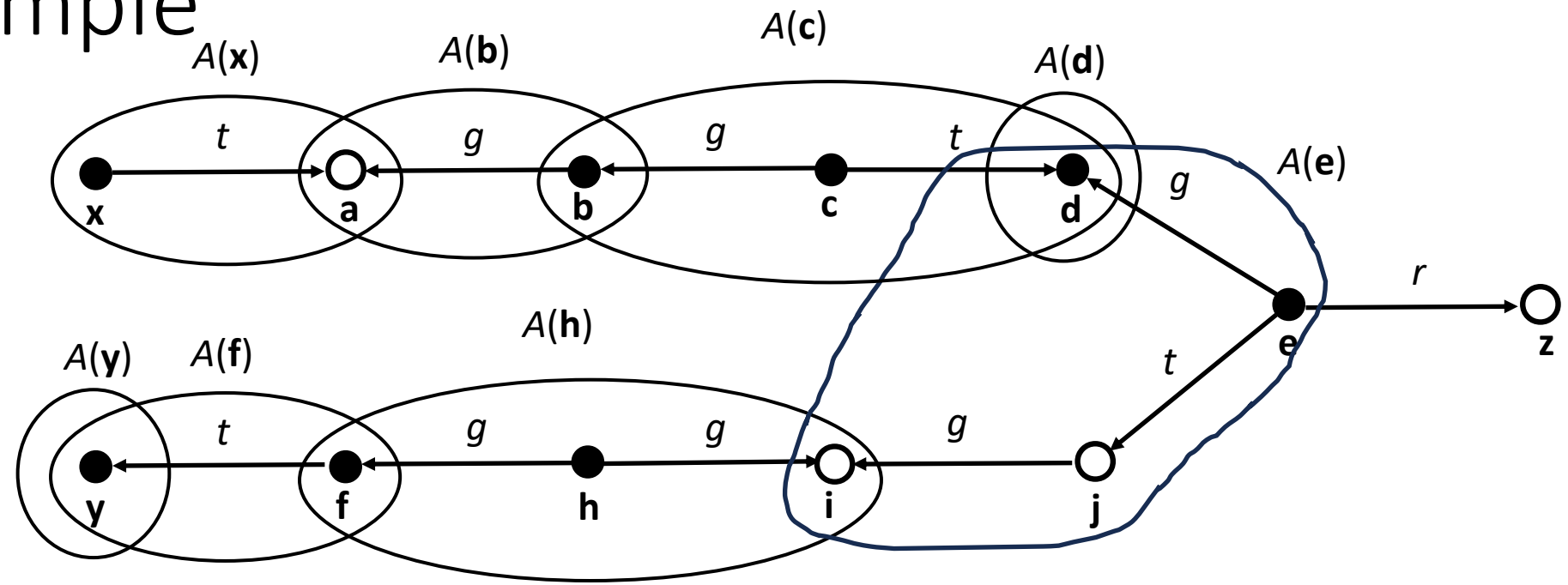
Example



Access Set

- *Access set A(***y***) with focus* **y**: set of vertices:
  - { **y** }
  - { **x** | **y** initially spans to **x** }
  - { **x′** | **y** terminally spans to **x′** }
- Idea is that focus can give rights to, or acquire rights from, a vertex in this set

Example

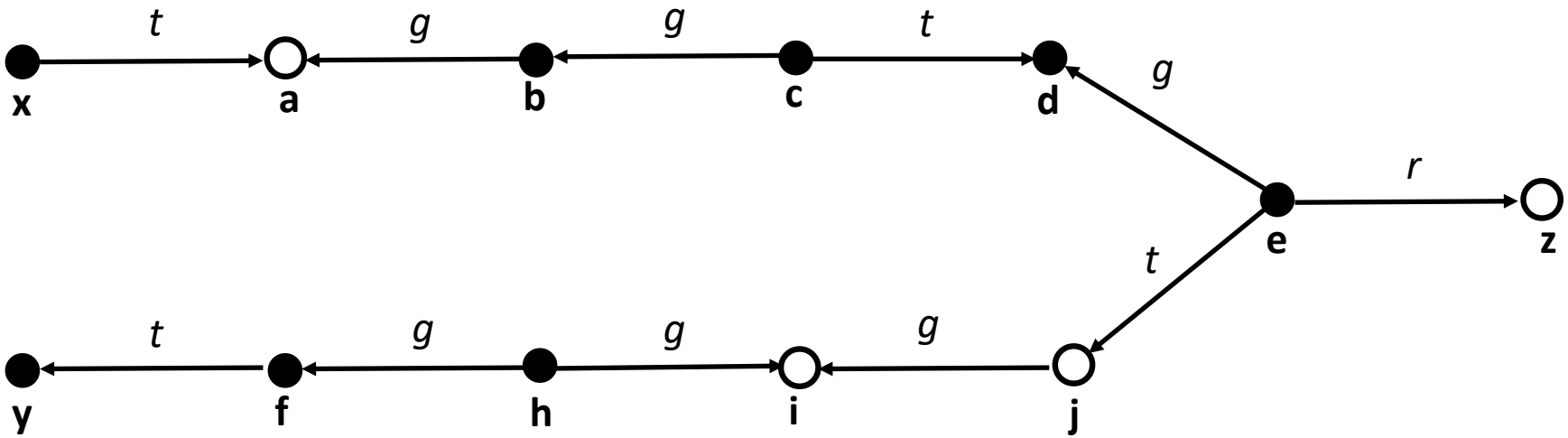


- $A(\mathbf{x}) = \{ \mathbf{x}, \mathbf{a} \}$
- $A(\mathbf{b}) = \{ \mathbf{b}, \mathbf{a} \}$
- $A(\mathbf{c}) = \{ \mathbf{c}, \mathbf{b}, \mathbf{d} \}$
- $A(\mathbf{d}) = \{ \mathbf{d} \}$
- $A(\mathbf{e}) = \{ \mathbf{e}, \mathbf{d}, \mathbf{i}, \mathbf{j} \}$
- $A(\mathbf{y}) = \{ \mathbf{y} \}$
- $A(\mathbf{f}) = \{ \mathbf{f}, \mathbf{y} \}$
- $A(\mathbf{h}) = \{ \mathbf{h}, \mathbf{f}, \mathbf{i} \}$

Deletion Set

- Deletion set $\delta(\mathbf{y}, \mathbf{y}')$: contains those vertices **z** in $A(\mathbf{y}) \cap A(\mathbf{y}')$ such that:
  - **y** initially spans to **z** and **y′** terminally spans to **z**; or
  - **y** terminally spans to **z** and **y′** initially spans to **z**; or
  - **z** = **y**; or
  - **z** = **y′**
- Idea is that rights can be transferred between **y** and **y′** if this set non-empty

Example

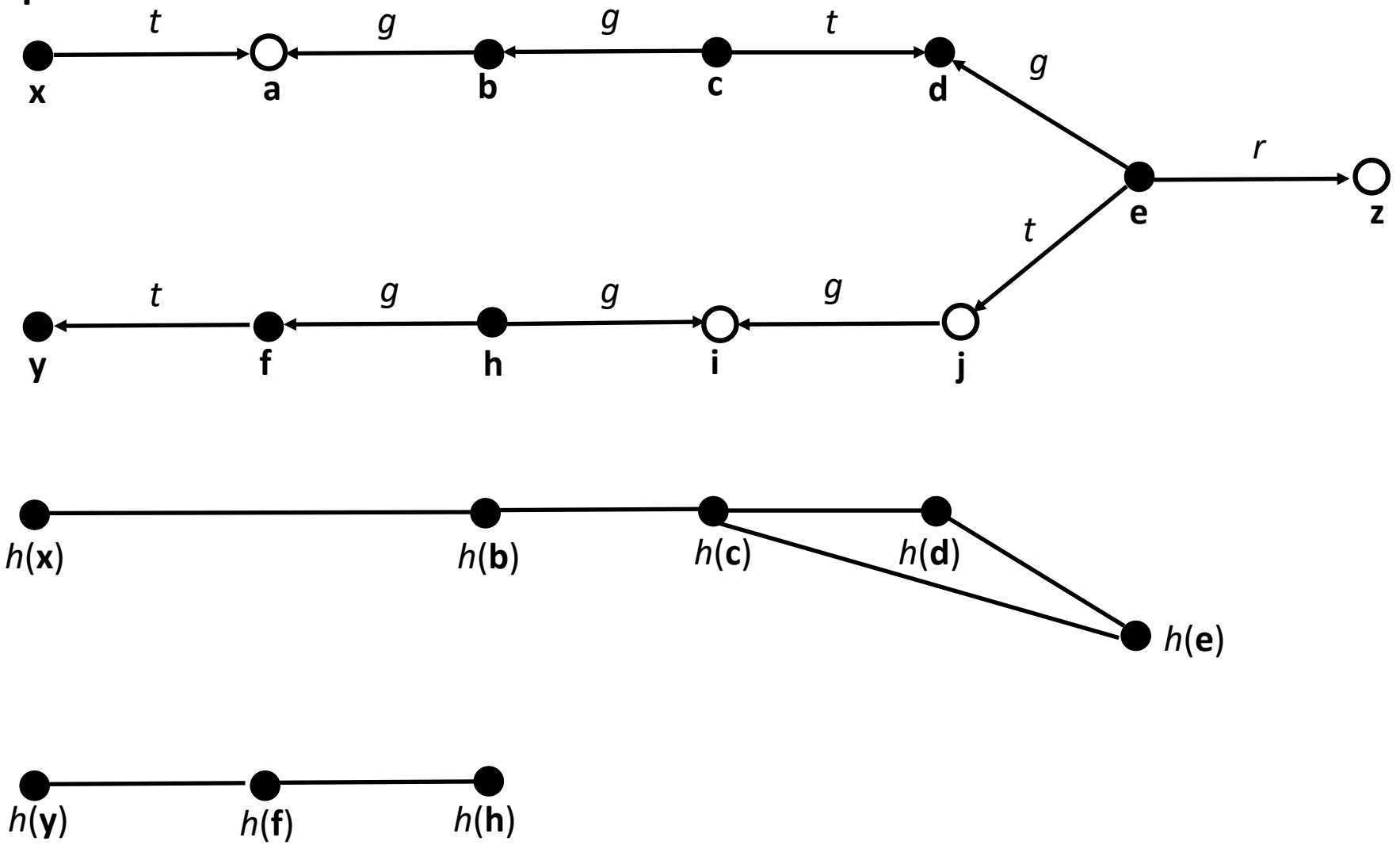


- δ(**x**, **b**) = { **a** }
- δ(**b**, **c**) = { **b** }
- δ(**c**, **d**) = { **d** }
- δ(**c**, **e**) = { **d** }
- δ(**d**, **e**) = { **d** }
- δ(**y**, **f**) = { **y** }
- δ(**h**, **f**) = { **f** }

Conspiracy Graph

- Abstracted graph H from G_0:
  - Each subject $\mathbf{x} \in G_0$ corresponds to a vertex $h(\mathbf{x}) \in H$
  - If $\delta(\mathbf{x}, \mathbf{y}) \neq \varnothing$, there is an edge between $h(\mathbf{x})$ and $h(\mathbf{y})$ in H
- Idea is that if $h(\mathbf{x})$, $h(\mathbf{y})$ are connected in H, then rights can be transferred between **x** and **y** in G_0

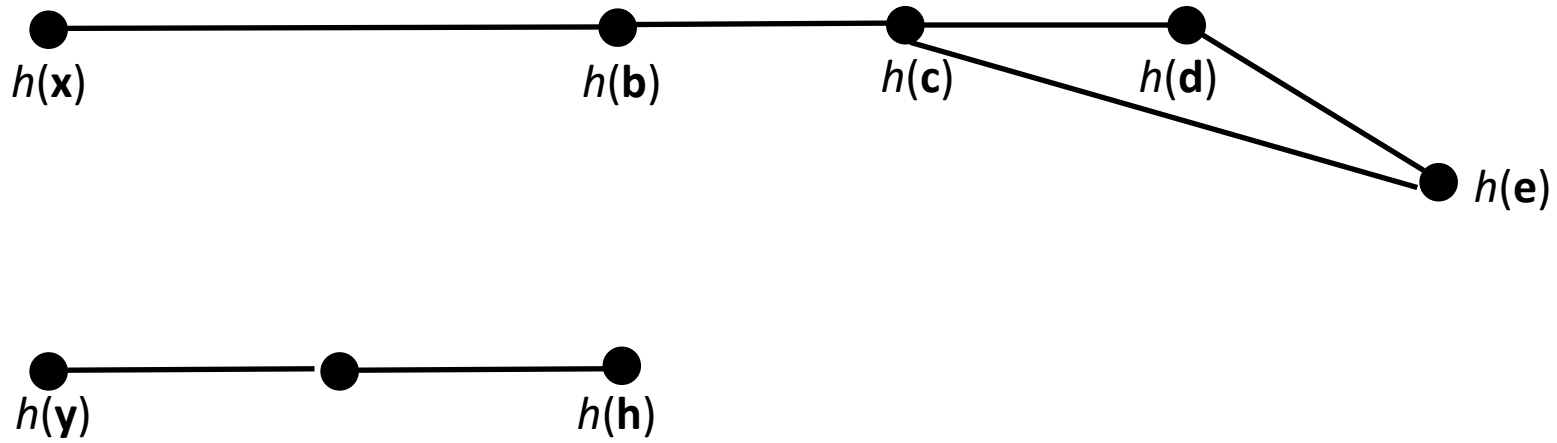
Example



Results

- $I(\mathbf{x})$: $h(\mathbf{x})$, all vertices $h(\mathbf{y})$ such that **y** initially spans to **x**
- $T(\mathbf{x})$: $h(\mathbf{x})$, all vertices $h(\mathbf{y})$ such that **y** terminally spans to **x**
- Theorem: *can•share*(α, **x**, **y**, G_0) iff there exists a path from some $h(\mathbf{p})$ in $I(\mathbf{x})$ to some $h(\mathbf{q})$ in $T(\mathbf{y})$
- Theorem: *n* vertices on shortest path between $h(\mathbf{p})$, $h(\mathbf{q})$ in above theorem; *n* conspirators necessary and sufficient to witness

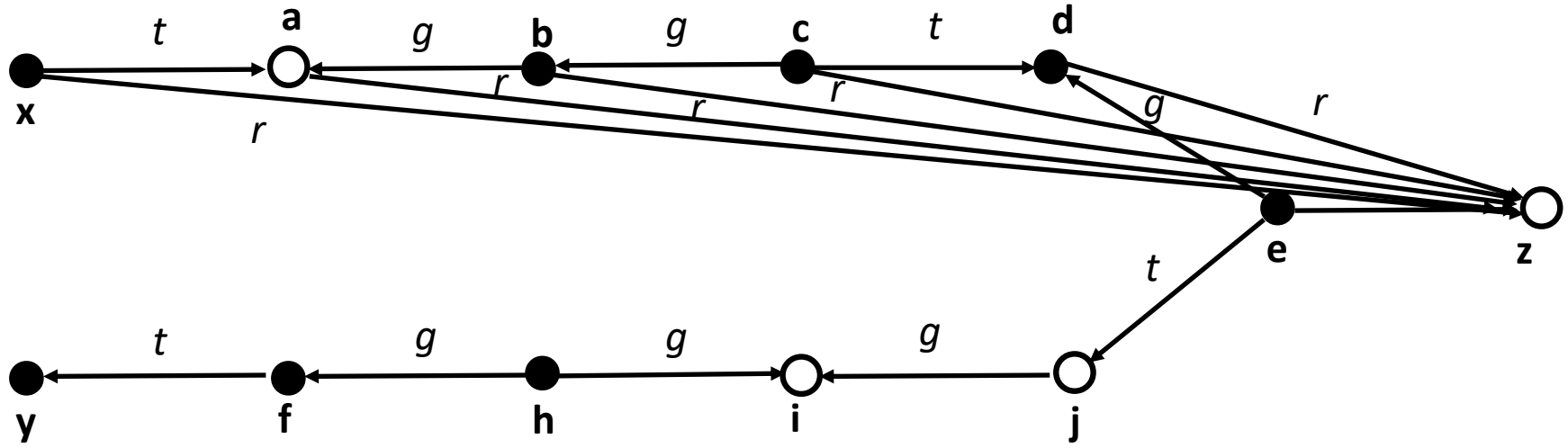
Example: Conspirators



- $I(\mathbf{x}) = \{ h(\mathbf{x}) \}$, $T(\mathbf{z}) = \{ h(\mathbf{e}) \}$
- Path between $h(\mathbf{x})$, $h(\mathbf{e})$ so $can\bullet share(r, \mathbf{x}, \mathbf{z}, G_0)$
- Shortest path between $h(\mathbf{x})$, $h(\mathbf{e})$ has 4 vertices

$\Rightarrow$ Conspirators are **e**, **c**, **b**, **x**

Example: Witness



1. **e** grants (*r* to **z**) to **d**
2. **c** takes (*r* to **z**) from **d**
3. **c** grants (*r* to **z**) to **b**
4. **b** grants (*r* to **z**) to **a**
5. **x** takes (*r* to **z**) from **a**