Cryptographically Enforced RBAC

Georg Fuchsbauer (IST Austria)

27 June 2013, CSF

joint work with **Anna Lisa Ferrara** and **Bogdan Warinschi**
(University of Bristol)

Overview

Cryptographically enforced access control

concentrate on

Role-Based Access Control for file system

- Formal **model**, security definitions
- Soundness theorem for enforcing **policies**
- **Implementation** using Attribute-Based Encryption

Access Control

- Access control by
  - **low-level** mechanisms
    - enforcement by design
    - absolute semantics

Access Control

- Access control by
  - **low-level** mechanisms
    - enforcement by design
    - absolute semantics

  - using **cryptography**
    - key management
    - only ”probabilistic” guarantees

Previous Work

- Cryptographic sealing [Gifford'82]
- Cryptographic model (UC) for access control [Halevi,Karger,Naor'05]
- Access control for XML documents [Abadi,Warinschi'08]

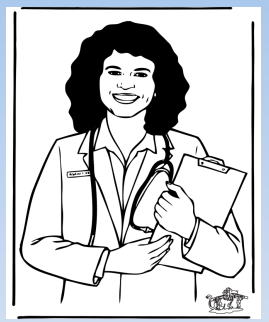
Previous Work

- Cryptographic sealing [Gifford'82]
- Cryptographic model (UC) for access control [Halevi,Karger,Naor'05]
- Access control for XML documents [Abadi,Warinschi'08]

- GAP between semantics associated to polices and cryptographic guarantees

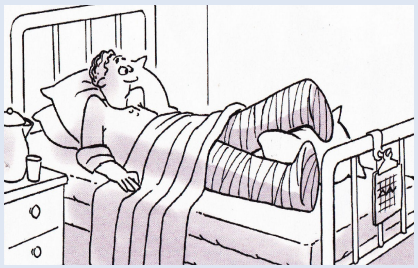
(previously: simple polices, simple primitives)

RBAC for Hospital



Doctor

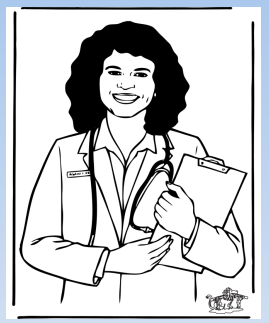
Nurse



Patient

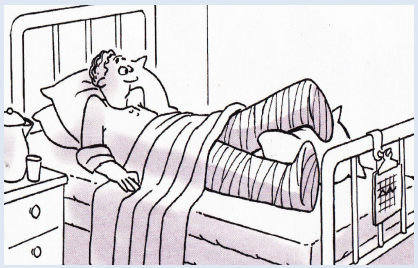
Receptionist

RBAC for Hospital



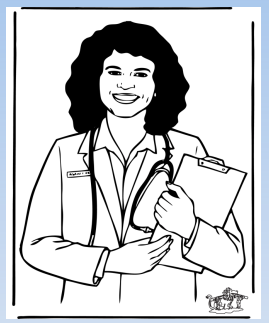
Medical records

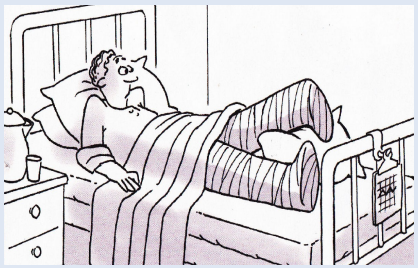
List of doctors



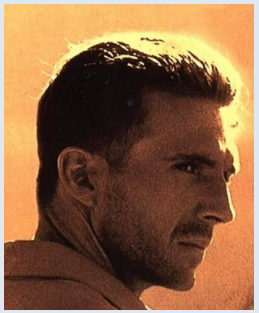
List of appointments

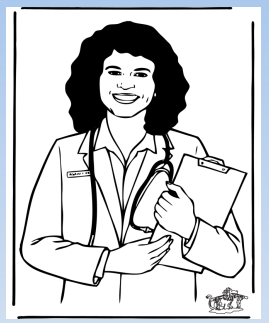
RBAC for Hospital

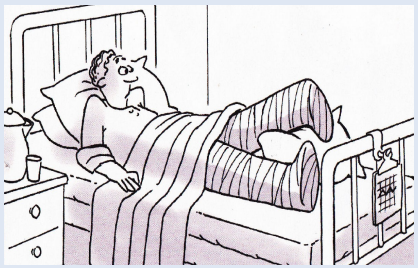




RBAC for Hospital







RBAC - The Model

R ... set of **roles** (fixed)

$\mathcal{U}$ ... universe of **users**

$U \subseteq \mathcal{U}$ ... set of **users**

$\mathcal{P}$ ... universe of **permissions**

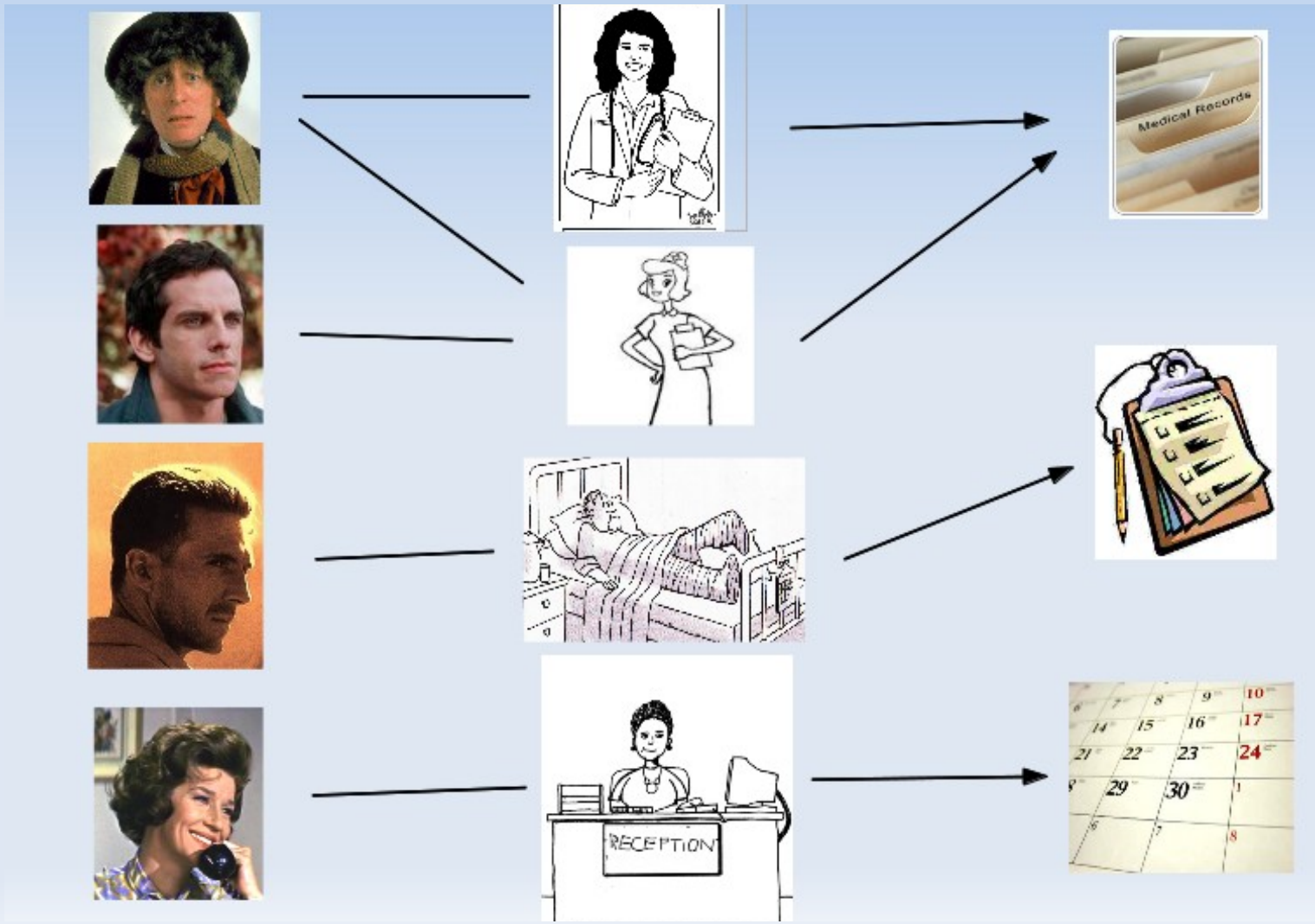
$P \subseteq \mathcal{P}$ ... set of **permissions**

$UA \subseteq U \times R$ ... **user-role** assignment relation

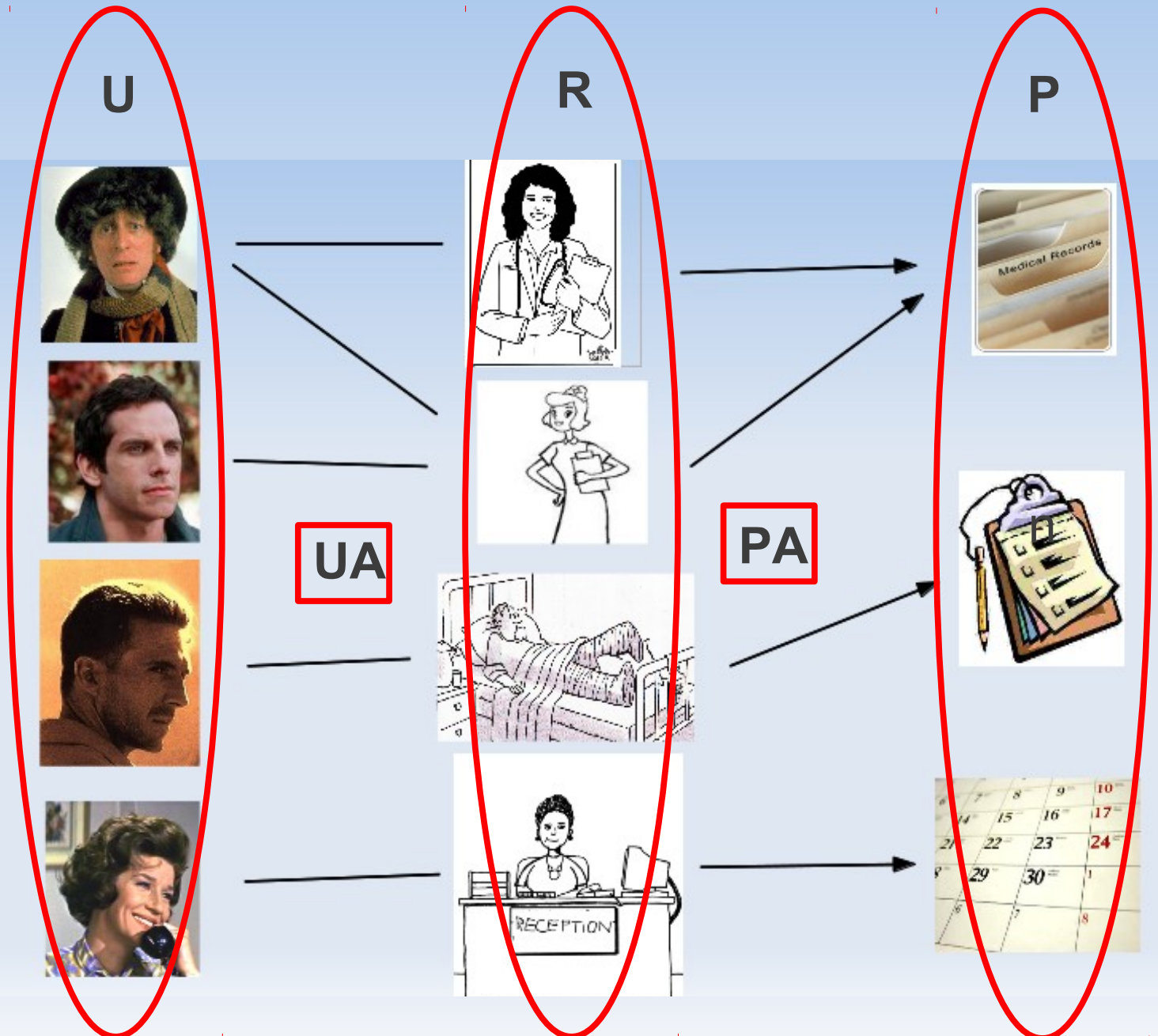
$PA \subseteq P \times R$ ... **permission-role** assignment relation

State of the system: $\mathcal{S} = (U, P, UA, PA)$

Hospital Example



Hospital Example

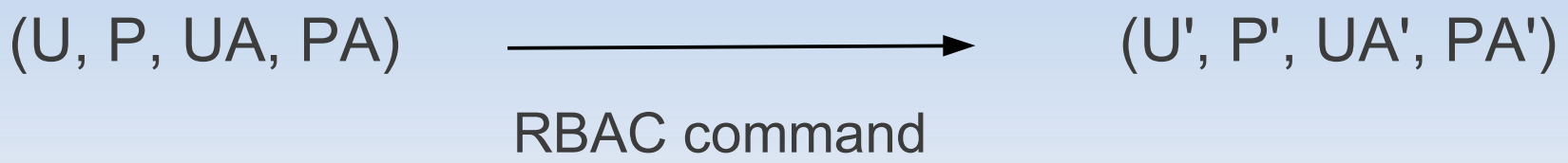


RBAC Commands

- AddUser(u) $U \rightarrow U \cup \{u\}$
- DeleteUser(u) $U \rightarrow U \setminus \{u\},\ UA \rightarrow UA \setminus \{(u,r) \in UA \mid r \in R\}$
- AddObject(p) $P \rightarrow P \cup \{p\}$
- DeleteObject(p) $P \rightarrow P \setminus \{p\},\ PA \rightarrow PA \setminus \{(p,r) \in PA \mid r \in R\}$
- AssignUser(u,r) $UA \rightarrow UA \cup \{(u,r)\}$
- DeassignUser(u,r) $UA \rightarrow UA \setminus \{(u,r)\}$
- GrantPermission(p,r) $PA \rightarrow PA \cup \{(p,r)\}$
- RevokePermission(p,r) $PA \rightarrow PA \setminus \{(p,r)\}$

RBAC Semantics

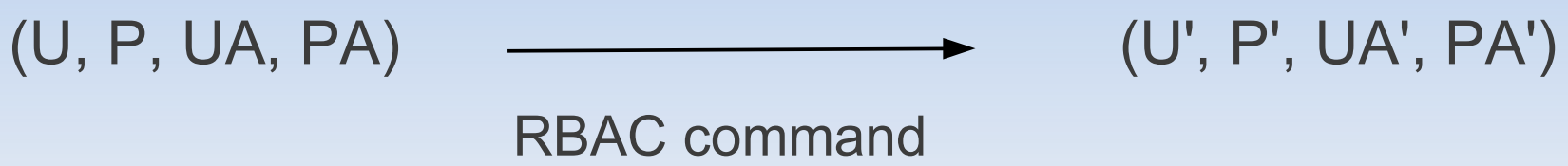
- System evolves via commands



- **Trace**: sequence of commands: → → ... →

RBAC Semantics

- System evolves via commands



- **Trace**: sequence of commands: → → ... →

- **Semantic**: User u has access to p:

$$\text{HasAccess}(u,p) \Leftrightarrow \exists\, r \in R: (u,r) \in UA \ \wedge\ (p,r) \in PA$$

Security Policies

... express limits on access

- **Separation of Duty:**

  ... models conflict of interest between r, r' : no user can hold both roles

- **Privilege Escalation:**

  ... lower-rank roles cannot access resources for higer-rank roles

Security Policies

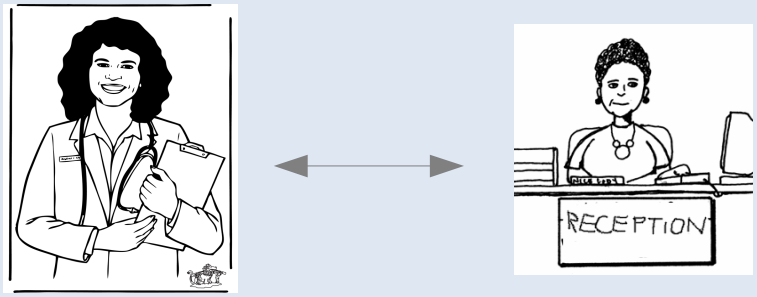
... express limits on access

- **Separation of Duty:**

  ... models conflict of interest between r, r' : no user can hold both roles

- **Privilege Escalation:**

  ... lower-rank roles cannot access resources for higer-rank roles



Example: Doctor Receptionist

Security Policies

- **Security policy** for RBAC is a formula

$$\forall u \in U \;\; \forall p \in P: \;\; Cond(u,p) \;\; \Rightarrow \;\; \neg \, HasAccess(u,p)$$

Security Policies

- **Security policy** for RBAC is a formula

$$\forall u \in U \;\; \forall p \in P: \;\; Cond(u,p) \;\; \Rightarrow \;\; \neg \, HasAccess(u,p)$$

interpreted over a state $\mathcal{S} = (U, P, UA, PA)$

- A policy should hold over any execution trace.

Security Policies

- **Security policy** for RBAC is a formula

$$\forall u \in U \;\; \forall p \in P: \;\; Cond(u,p) \;\; \Rightarrow \;\; \neg \, HasAccess(u,p)$$

interpreted over a state $\mathcal{S} = (U, P, UA, PA)$

- A policy should hold over any execution trace.

- Example: Privilege escalation from role r to role r':

$$Cond(u,p) \;\; \equiv \;\; [(u,r) \in UA \;\wedge\; (p,r') \in PA]$$

Security Policies

- **Security policy** for RBAC is a formula

$$\forall u \in U \;\; \forall p \in P: \;\; Cond(u,p) \;\; \Rightarrow \;\; \neg \, HasAccess(u,p)$$

interpreted over a state $\mathcal{S} = (U, P, UA, PA)$

- A policy should hold over any execution trace.

- Example: Privilege escalation from role **r** to role **r'**:

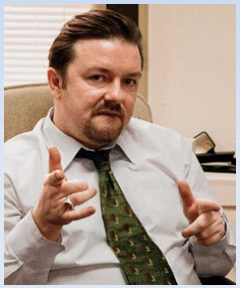
$$Cond(u,p) \;\; \equiv \;\; [(u,r) \in UA \;\wedge\; (p,r') \in PA]$$

- Multiple policies: $C_1, \ldots, C_n \longrightarrow C_1 \vee \ldots \vee C_n$

Cryptographic RBAC

Cryptographic RBAC

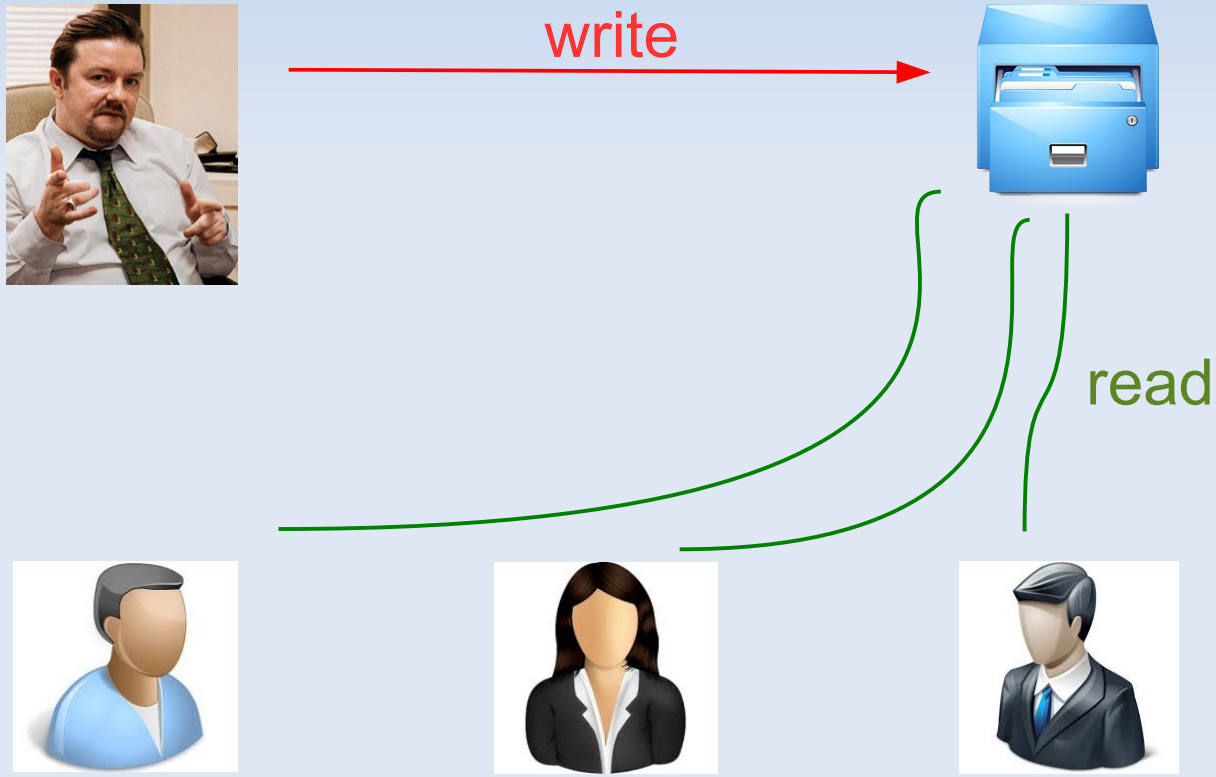
- **Permission** = read access for file p in file system **FS**
- **Manager**



- **Users**

Cryptographic RBAC

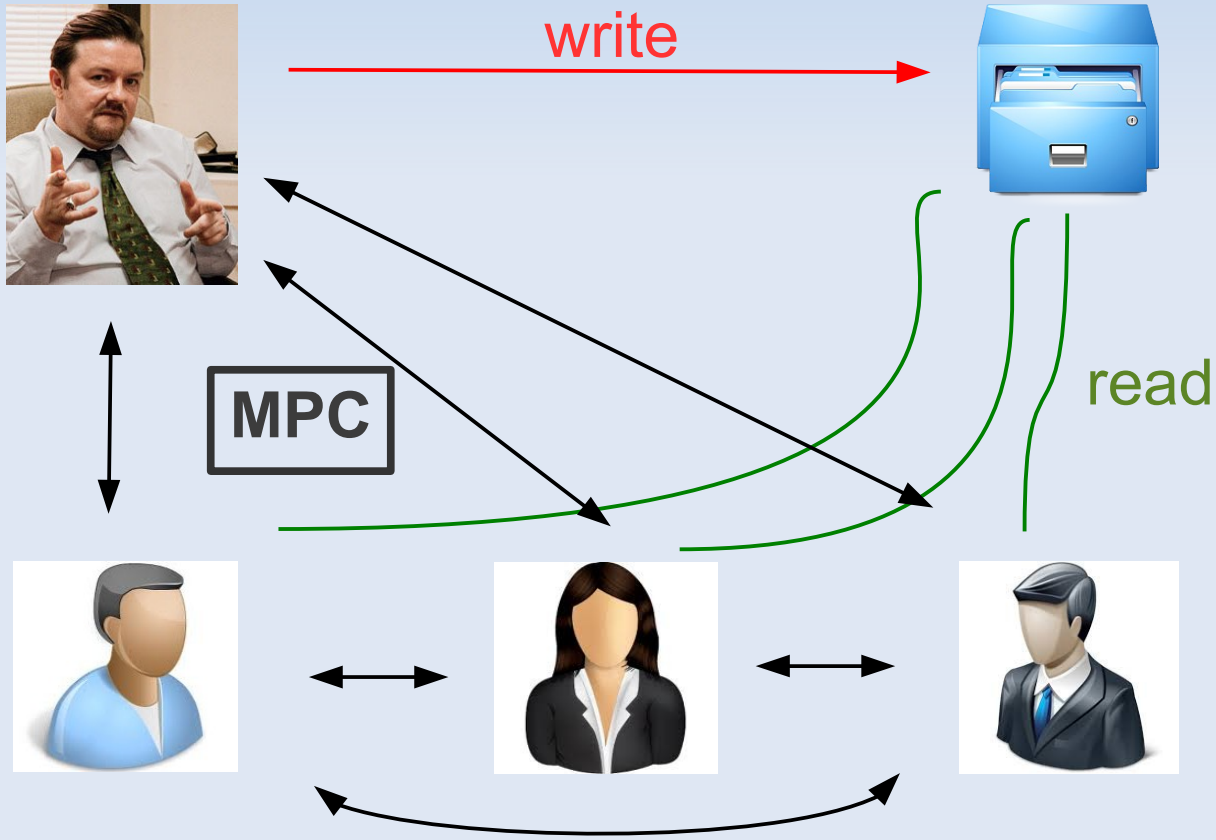
- **Permission** = read access for file p in file system **FS**
- **Manager**



- **Users** have unrestricted access to FS

Cryptographic RBAC

- **Permission** = read access for file p in file system **FS**
- **Manager**

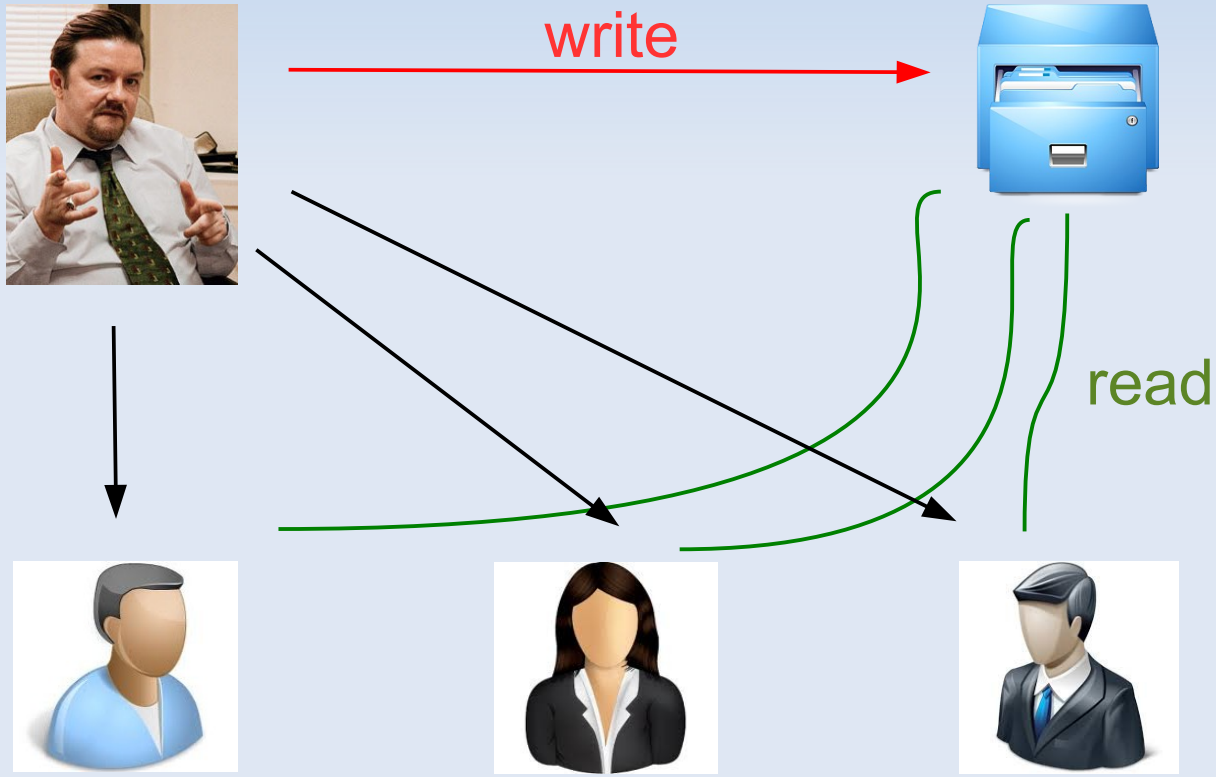


- **Users** have unrestricted access to FS

Cryptographic RBAC

Non-interactive protocol:

- **Manager**
  - runs command locally
  - writes to **FS**
  - sends update msgs to users

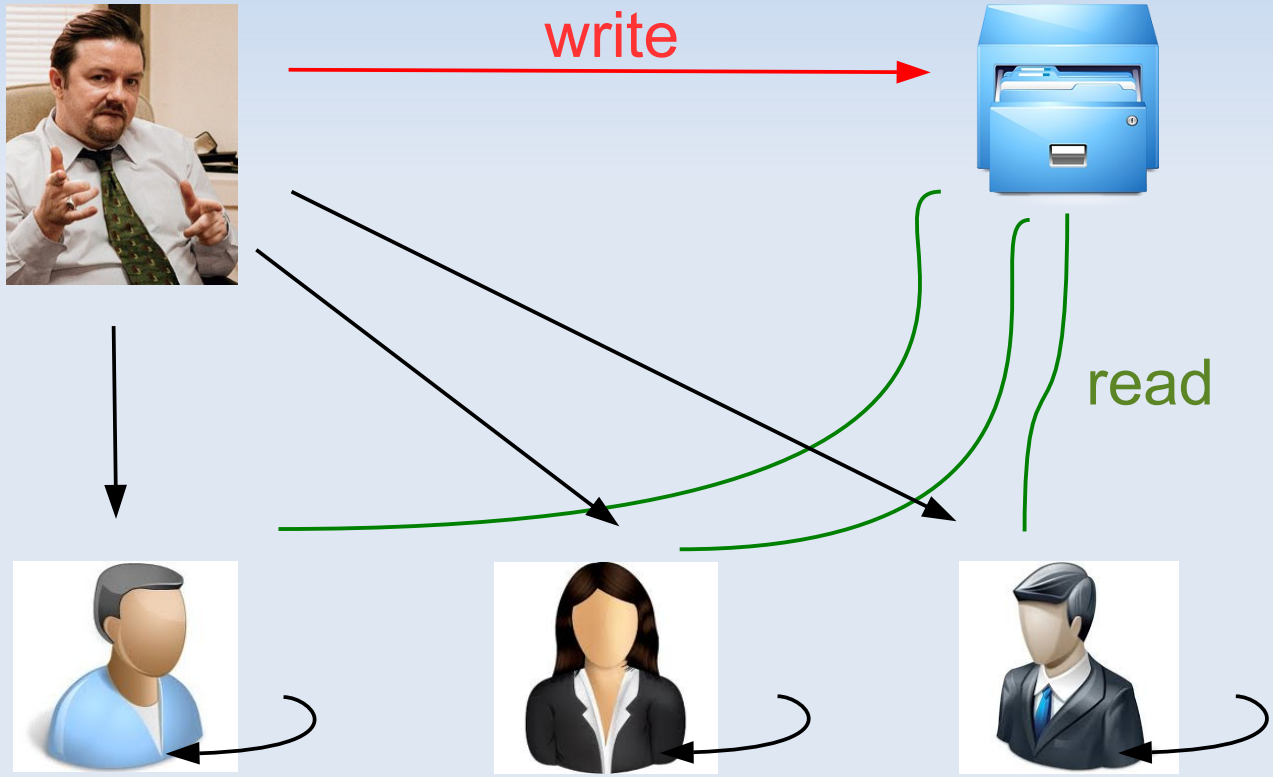


- **Users**

Cryptographic RBAC

Non-interactive protocol:

- **Manager**
  - runs command locally
  - writes to **FS**
  - sends update msgs to users

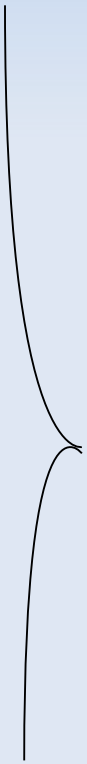


- **Users** update their states when receiving msg

cRBAC Algorithms

- Init

- AddUser
- DelUser
- AddObject
- DelObject
- AssignUser
- DeassignUser
- GrantPerm
- RevokePerm



RBAC commands,
run by manager

input: $state_M$, FS, args

output: $state'_M$, FS', $\{msg_u\}$

cRBAC Algorithms

- Init

- AddUser
- DelUser
- AddObject
- DelObject
- AssignUser
- DeassignUser
- GrantPerm
- RevokePerm

RBAC commands,
run by manager

input: $state_M$, FS, args
output: $state'_M$, FS', $\{msg_u\}$

- Write

run by manager,
input: p, m

- Read
- Update

run by users,
input: $state_u$

Properties of cRBAC

- **Correctness**:

  After any executions of commands,

  if HasAccess(u,p)

  then Read(st_u,p,FS) outputs content of p

Properties of cRBAC

- **Correctness**:

  After any executions of commands,

  if HasAccess(u,p)

  then Read(st_u,p,FS) outputs content of p

- **Security:**

  No information on content of p is leaked to u when

  $\neg$ HasAccess(u,p)

  Adversary can
  - make manager execute RBAC cmds
  - corrupt users (not manager)
  - ask for ”challenges”

Security

- **Security game**:

$\mathrm{Exp}^{\mathrm{ind}}(\lambda)$
$b \leftarrow_{\$} \{0,1\}$
$(st_M, FS, \{st[u]\}_{u \in U}) \leftarrow_{\$} \mathrm{Init}(1^{\lambda}, R)$
$b' \leftarrow_{\$} \mathcal{A}(1^{\lambda}, FS : O)$
Return $(b' = b)$

$$\mathrm{Adv}^{\mathrm{ind}}(\lambda) := \left| \Pr[\mathrm{Exp}^{\mathrm{ind}}(\lambda) \rightarrow \mathrm{true}] - {}^{1}/_{2} \right|$$

- **Oracles O**:
  - Oracles for RBAC commands
  - CorruptU(u), Challenge(p,m_0,m_1)

Security

- **Security game**:

$\mathrm{Exp}^{\mathrm{ind}}(\lambda)$
$b \leftarrow_{\$} \{0,1\}$; $Cr, Ch \leftarrow \varnothing$
$(st_M, FS, \{st[u]\}_{u \in U}) \leftarrow_{\$} \mathrm{Init}(1^{\lambda}, R)$
$b' \leftarrow_{\$} \mathcal{A}(1^{\lambda}, FS : \mathcal{O})$
Return $(b' = b)$

$$\mathrm{Adv}^{\mathrm{ind}}(\lambda) := \left| \Pr[\mathrm{Exp}^{\mathrm{ind}}(\lambda) \rightarrow \mathrm{true}] - {}^{1}/_{2} \right|$$

- **Oracles $\mathcal{O}$**:
  - Oracles for RBAC commands
  - CorruptU(u), Challenge(p, m_0, m_1)

  Game ensures that at all time:

  $$\forall u \in Cr\ \forall p \in Ch: \neg\, \mathrm{HasAccess}(u,p)$$

Secure Policy Enforcement

- Policy Φ defined by Cond s.t.

  whenever Cond(u,p) then $\neg$ HasAccess(u,p)

  (i.e. $\forall r \in R: (u,r) \notin UA \vee (p,r) \notin PA$)

Secure Policy Enforcement

- Policy Φ defined by Cond s.t.

  whenever Cond(u,p) then $\neg$ HasAccess(u,p)

  (i.e. $\forall r \in R: (u,r) \notin UA \vee (p,r) \notin PA$)

- cRBAC **enforces** Φ if

  whenever Cond(u,p) then

  u does not have access to p *"in reality"*

Secure Policy Enforcement

- Policy Φ defined by Cond s.t.

  whenever Cond(u,p) then $\neg$ HasAccess(u,p)

  (i.e. $\forall r \in R$: $(u,r) \notin UA \vee (p,r) \notin PA$)

- cRBAC **enforces** Φ if

  whenever Cond(u,p) then

  u does not have access to p *"in reality"*

- Ind-based **definition**:
  - Manager enforces Φ symbolically
  - $\mathcal{A}$ impersonates u, for Cond(u,p)
  - ... must distinguish content of p

Secure Policy Enforcement

Theorem: If $\mathcal{CRBAC}$ is *secure* then it enforces any policy

Computational soundness:

Policies that are satisfied symbolically are also satisfied computationally.

Implementation

Attribute-Based Encryption

Attribute-Based Encryption

- Encryption w.r.t. set of **attributes** from universe **A**
- Keys associated to **policies** over attributes
- ... decrypt ciphertexts whose attributes satisfy policy

Attribute-Based Encryption

Attribute-Based Encryption

- Encryption w.r.t. set of **attributes** from universe **A**
- Keys associated to **policies** over attributes
- ... decrypt ciphertexts whose attributes satisfy policy

Only need **weak** form of ABE:

- Policies:

  Disjunction of attributes

  $\psi = a_1 \vee a_2 \vee ... \vee a_n$

Attribute-Based Encryption

Attribute-Based Encryption

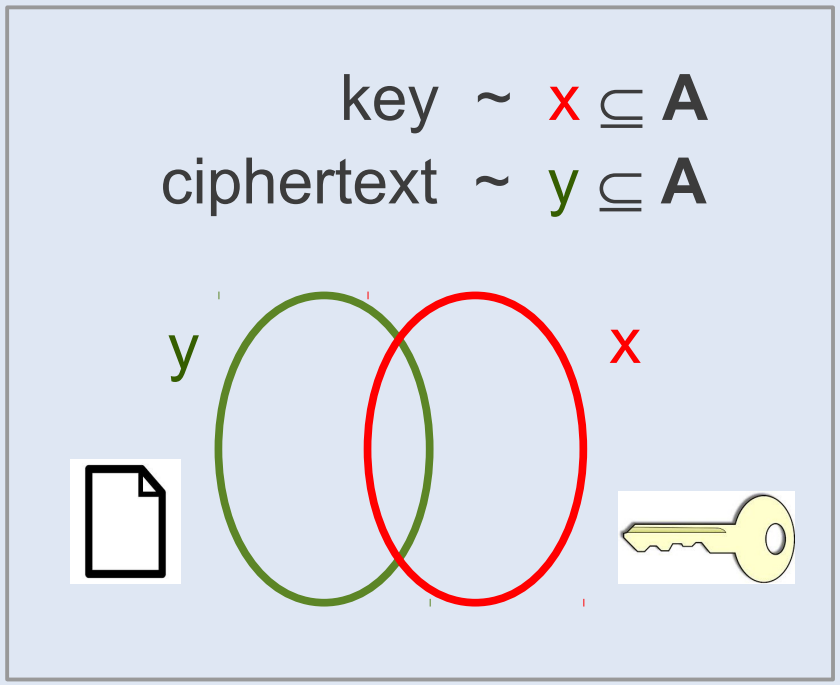
- Encryption w.r.t. set of **attributes** from universe **A**
- Keys associated to **policies** over attributes
- ... decrypt ciphertexts whose attributes satisfy policy

Only need **weak** form of ABE:

- Policies:

  Disjunction of attributes

  $\psi = a_1 \vee a_2 \vee \ldots \vee a_n$



Implementation of cRBAC

- **RBAC**
  - roles ⟷ attributes
  - objects ⟷ ciphertexts
- **ABE**

Implementation of cRBAC

RBAC		ABE
roles	⟷	attributes
objects	⟷	ciphertexts

- File p is encrypted w.r.t. roles (attributes) associated to it
  (according to PA) Write ⟷ Encrypt
- User u receive keys corresponding to their roles
  (according to UA) Read ⟷ Decrypt

Revocation and Deassignment

- AssignUser(u,r)

  ... **issue** new **key** for user u's extended set of roles

- GrantPerm(p,r)

  ... **re-encrypt** content of p w.r.t. extended set of roles

- RevokePerm(p,r)

  ... **re-encrypt** content of p w.r.t. reduced set of roles

Revocation and Deassignment

- AssignUser(u,r)

  ... **issue** new **key** for user u's extended set of roles

- GrantPerm(p,r)

  ... **re-encrypt** content of p w.r.t. extended set of roles

- RevokePerm(p,r)

  ... **re-encrypt** content of p w.r.t. reduced set of roles

- DeassignUser(u,r)

  ... **?**

Revocation and Deassignment

- AssignUser(u,r)

  ... **issue** new **key** for user u's extended set of roles

- GrantPerm(p,r)

  ... **re-encrypt** content of p w.r.t. extended set of roles

- RevokePerm(p,r)

  ... **re-encrypt** content of p w.r.t. reduced set of roles

- DeassignUser(u,r)
  - associate role r with a **new** attribute a
  - re-encrypt files p with $(p,r) \in PA$
  - re-issue keys to users u' with $(u',r) \in UA$

Security

- Not secure. Why? Consider:
  - $(u,r) \notin UA$, $(p,r) \in PA$ $\quad \neg\, HasAcc(u,p)$

Security

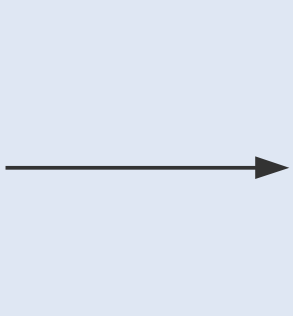
- Not secure. Why? Consider:
  - $(u,r) \notin UA$, $(p,r) \in PA$ $\neg$ HasAcc(u,p)
  - RevokePerm(p,r) $\neg$ HasAcc(u,p)

Security

- Not secure. Why? Consider:
  - $(u,r) \notin UA$, $(p,r) \in PA$ $\neg$ HasAcc(u,p)
  - RevokePerm(p,r) $\neg$ HasAcc(u,p)
  - AssignUser(u,r) $\neg$ HasAcc(u,p)

Security

- Not secure. Why? Consider:
  - $(u,r) \notin UA$, $(p,r) \in PA$ $\quad \neg$ HasAcc(u,p) $\quad$ c $\quad$ sk
  - RevokePerm(p,r) $\quad \neg$ HasAcc(u,p) $\quad$ c' $\quad$ sk
  - AssignUser(u,r) $\quad \neg$ HasAcc(u,p) $\quad$ c' $\quad$ sk'

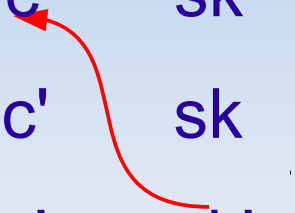


Security

- Not secure. Why? Consider:
  - $(u,r) \notin UA$, $(p,r) \in PA$ $\quad \neg$ HasAcc(u,p) $\quad$ c $\quad$ sk
  - RevokePerm(p,r) $\quad \neg$ HasAcc(u,p) $\quad$ c' $\quad$ sk
  - AssignUser(u,r) $\quad \neg$ HasAcc(u,p) $\quad$ c' $\quad$ sk'

Security

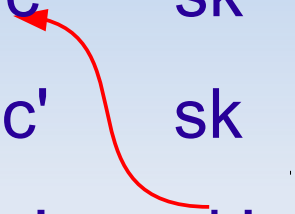
- Not secure. Why? Consider:
  - $(u,r) \notin UA$, $(p,r) \in PA$ $\quad \neg$ HasAcc(u,p) $\quad$ c $\quad$ sk
  - RevokePerm(p,r) $\quad \neg$ HasAcc(u,p) $\quad$ c' $\quad$ sk
  - AssignUser(u,r) $\quad \neg$ HasAcc(u,p) $\quad$ c' $\quad$ sk'



- New attribute for role r also when RevokePerm(p,r)!

Security

- Not secure. Why? Consider:
  - $(u,r) \notin UA$, $(p,r) \in PA$ $\quad \neg$ HasAcc(u,p) $\quad$ c $\quad$ sk
  - RevokePerm(p,r) $\quad \neg$ HasAcc(u,p) $\quad$ c' $\quad$ sk
  - AssignUser(u,r) $\quad \neg$ HasAcc(u,p) $\quad$ c' $\quad$ sk'



- New attribute for role r also when RevokePerm(p,r)!

- Theorem: If $\mathcal{ABE}$ satisfies indistinguishability

  then $\mathcal{CRBAC}[\mathcal{ABE}]$ is a secure implementation

Conclusion

- Defined syntax & security for cryptographic access control
- Soundness theorem for policies
- Provably secure implementation using weak ABE

Conclusion

- Defined syntax & security for cryptographic access control
- Soundness theorem for policies
- Provably secure implementation using weak ABE

Future work

- Hierarchical RBAC, attribute-based RBAC
- More general framework for abstract notions of
  symbolic & computational access-control enforcement