ACaaS: Access Control as a Service for IaaS Cloud

Ruoyu Wu[*], Xinwen Zhang[†], Gail-Joon Ahn[*], Hadi Sharifi[*], Haiyong Xie[†]

[*]Arizona State University

[†]Huawei Research Center

SEFCOM

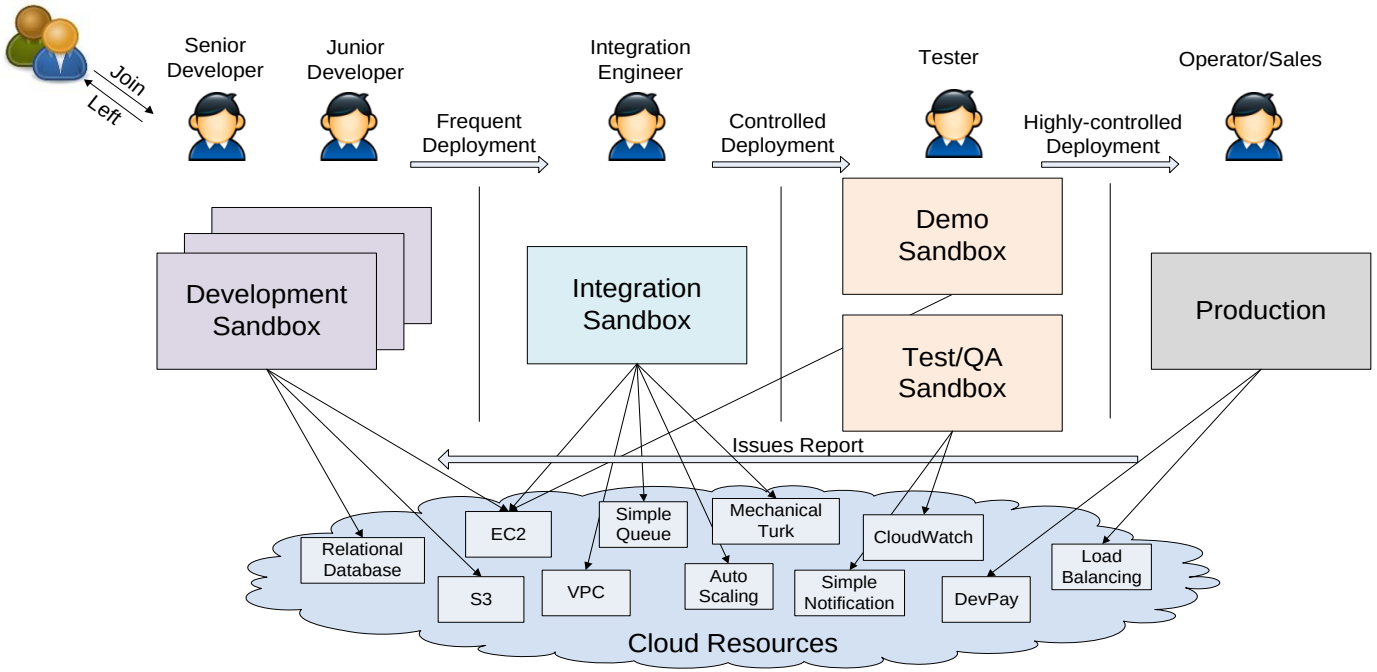
Security Engineering for Future Computing

Outline

- Introduction
  - Motivation Scenarios
  - ACaaS Overview
  - AWS IAM and Its Limitations
- $ACaaS_{RBAC}$ for AWS
  - Why RBAC
  - Challenges of Supporting RBAC for AWS
  - $ACaaS_{RBAC}$ model
- System Design
- Implementation and Evaluation
- Related Work
- Conclusion and Future Work

Motivation Scenarios

- Cloud Sandbox for Software Development

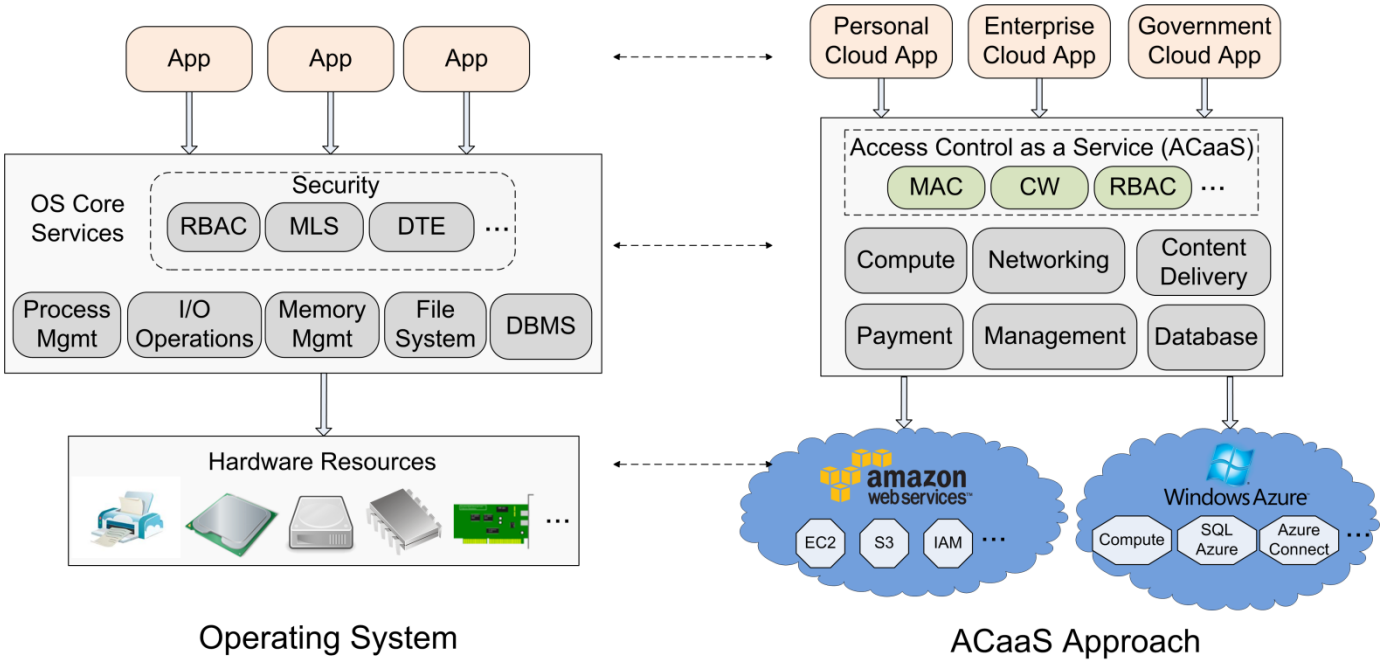


- Government cloud platform
  - Regulation compliance requirement (e.g., HIPAA, SOX)
  - Multi-level security requirement

SEFCOM

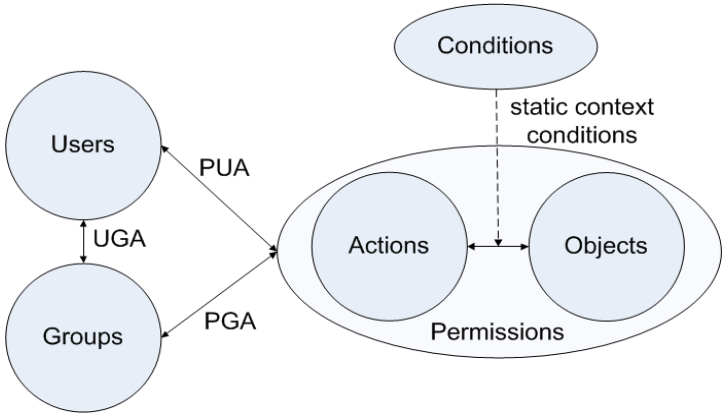
ACaaS Overview

- The diversity of cloud services across a wide range of orgs and domains requires various security models and policies
- However, major cloud providers all fail to accommodate this
- Hence, we need a comprehensive and adaptive access control mechanism in cloud



AWS IAM Service

- IAM model
  - Users, Groups, Action, Objects, Permissions, Constraints, UGA, PUA and PGA.



- Policy
  - 4-tuple P = (user, permission, constrain effect)

```
{"Statement":[{
  "Effect":"Allow",
  "Action":["iam:CreateAccessKey","iam:ListAccessKeys"],
  "Resource":"arn:aws:iam::123456789012:user/*",
  "Condition":{
    "DateGreaterThan":{
      "aws:CurrentTime":"2010-07-01T00:00Z"}}},
  {"Effect":"Allow",
  "Action":["sdb:CreateDomain","sdb:DeleteDomain"],
  "Resource":"arn:aws:sdb:*:123456789012:domain/*"}]
}
```

SEFCOM

AWS IAM Service - Cont'd

- Limitations
  - A very fine-grained approach
  - Groups are organized in a flat structure
  - Only static constraints, SoD is missing
  - No session management
  - Does not separate administrators and regular users clearly

- IAM Role
  - It only supports assigning IAM roles to an EC2 instance level but not to user level
  - An IAM role does not support session management
  - EC2 role does not support other important RBAC features such as role hierarchy and delegation

$ACaaS_{RBAC}$ for AWS

- In this work, we present $ACaaS_{RBAC}$, a reference architecture of ACaaS that supports RBAC for Amazon AWS cloud platform

- Why RBAC
  - It has been widely adopted in enterprise applications
  - It is a very generic access control model
  - It can address all the identified limitations of AWS IAM service.

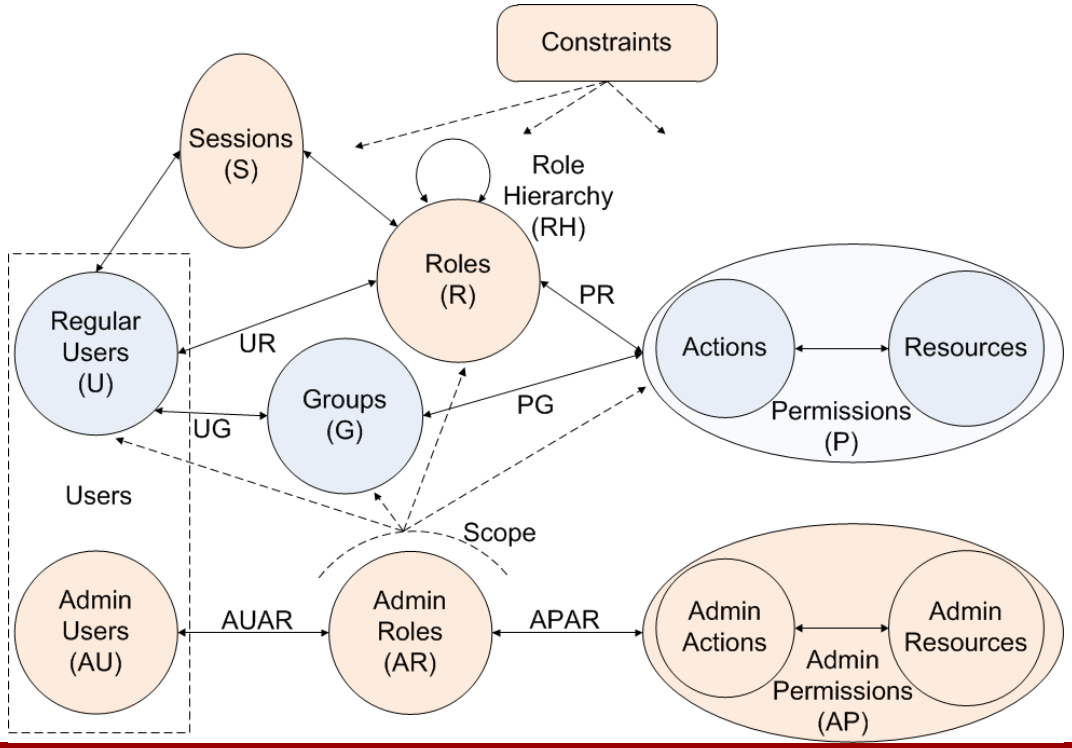
$ACaaS_{RBAC}$ for AWS - Cont'd

- Challenges to support RBAC for AWS
  - Efficient role hierarchy management
  - Session management
  - SoD support and management of privileged account
  - System integration and minimal overhead

- $ACaaS_{RBAC}$ addresses these challenges with a service-oriented RBAC for AWS cloud resources

$ACaaS_{RBAC}$ Model

- Beyond the existing RBAC96 and NIST RBAC model, $ACaaS_{RBAC}$ integrate the *Group* concept of IAM into RBAC
  - Group can be a good counterpart to role for better managing user sets and relatively static permission sets



$ACaaS_{RBAC}$ Model - Cont'd

- Core Model
  - Sets: *U, AU, G, R, AR, P, AP, S*
  - Relations: *UR, PR, RH, UG, PG, AUAR, APAR*
  - Functions: *Sessions: U -> 2^S; Roles: S -> 2^R; Permissions: S -> 2P*
- Role-based Administrative Model
  - Admin resource $RES_A \subseteq U \cup G \cup P \cup R$
  - Admin action ACT_A= {*Create, Delete, Assign, Revoke*}
  - Admin scope $SCO \subseteq 2^U \times 2^G \times 2^P \times 2^R$
  - Admin policy 4-tuple p_a = <*ar, act, res_a, aco* >, *where ar* $\in$ *AR, act* $\in ACT_A$, $res_a \in Get_RES_A(sco)$, *and sco* $\in$ *Scopes(ar)*

$ACaaS_{RBAC}$ Model - Cont'd

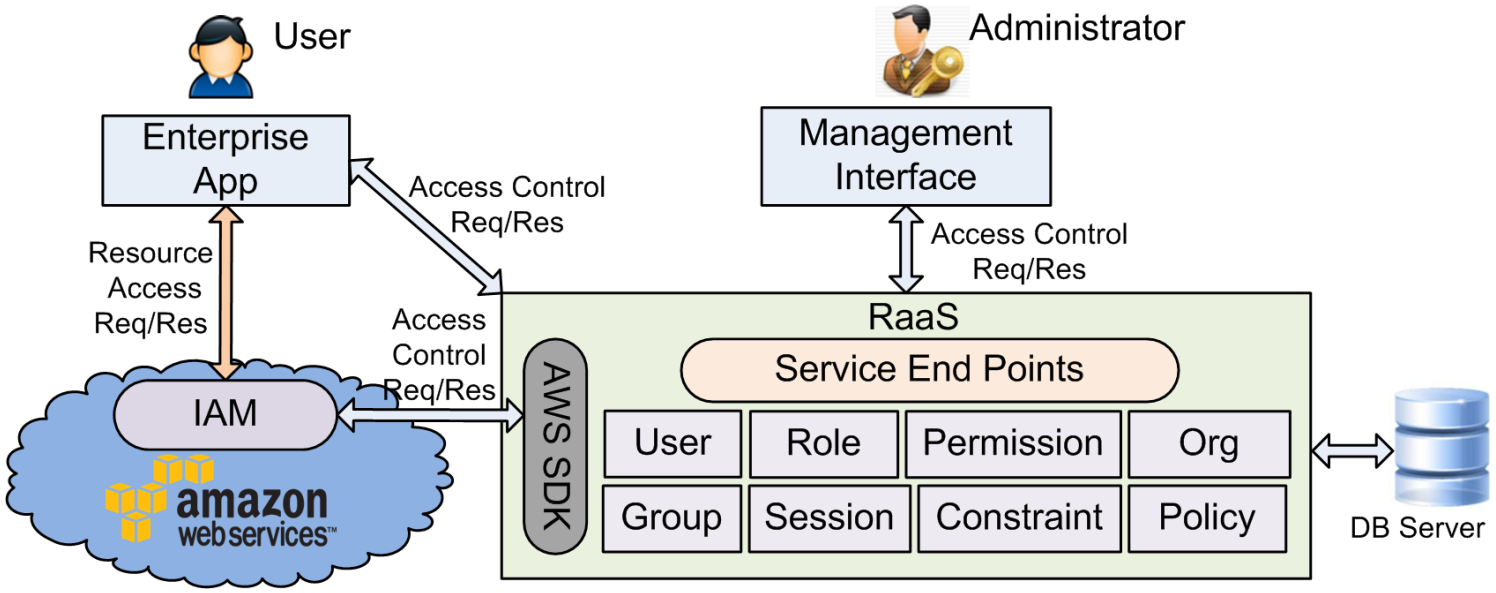
- Administrative action table

Action	Preconditions	Postconditions
Create(u)	N/A	$u \in U$, $u \in Scope_U(ar)$
Delete(u)	$u \in Scope_U(ar)$	$u \notin U$, $u \notin Scope_U(ar)$
Create(g)	N/A	$g \in G$, $g \in Scope_G(ar)$
Delete(g)	$g \in Scope_G(ar)$	$g \notin G$, $g \notin Scope_G(ar)$
Create(p)	N/A	$p \in P$, $p \in Scope_P(ar)$
Delete(p)	$p \in Scope_P(ar)$	$p \notin P$, $p \notin Scope_P(ar)$
Create(r)	N/A	$r \in R$, $r \in Scope_R(ar)$
Delete(r)	$r \in Scope_R(ar)$	$r \notin R$, $r \notin Scope_R(ar)$
Assign(r_i, r_j)	r_i, $r_j \in Scope_R(ar)$	$(r_i, r_j) \in RH$
Revoke(r_i, r_j)	r_i, $r_j \in Scope_R(ar)$	$(r_i, r_j) \notin RH$
Assign(u, g)	$u \in Scope_U(ar)$, $g \in Scope_G(ar)$	$(u, g) \in UG$
Revoke(u, g)	$u \in Scope_U(ar)$, $g \in Scope_G(ar)$	$(u, g) \notin UG$
Assign(u, r)	$u \in Scope_U(ar)$, $r \in Scope_R(ar)$	$(u, r) \in UR$
Revoke(u, r)	$u \in Scope_U(ar)$, $r \in Scope_R(ar)$	$(u, r) \notin UR$
Assign(p, r)	$p \in Scope_P(ar)$, $r \in Scope_R(ar)$	$(p, r) \in PR$
Revoke(p, r)	$p \in Scope_P(ar)$, $r \in Scope_R(ar)$	$(p, r) \notin PR$
Assign(p, g)	$p \in Scope_P(ar)$, $g \in Scope_G(ar)$	$(p, g) \in PG$
Revoke(p, g)	$p \in Scope_P(ar)$, $g \in Scope_G(ar)$	$(p, g) \notin PG$

SEFCOM

$ACaaS_{RBAC}$ Design

- Architecture
  - $ACaaS_{RBAC}$ introduces RBAC as a service (RaaS), which is an RBAC module can be hosted by AWS or any third party service provider



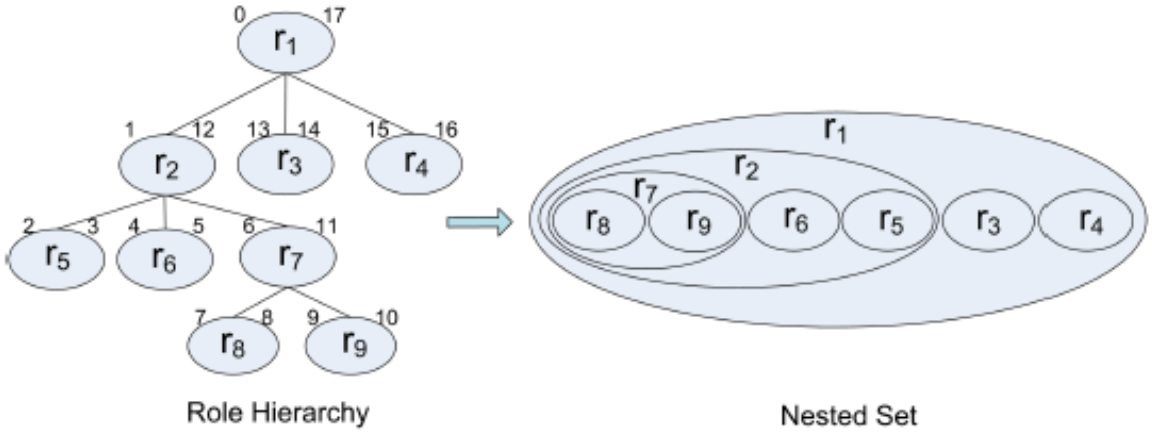
$ACaaS_{RBAC}$ Design – Cont'd

- RaaS consist of 8 sub-modules, each of which is exposed as a web service
  - *Organization module*: manage (e.g., list, register, and delete) organizations to support the multi-tenant feature
  - *Group module*: manage user groups *G* of a single organization for administrative users *AU*
  - *Permission module*: manage permissions *P, AP*, and their relations with roles *R, AR*
  - *User module*: manage (e.g., create, delete, activate, deactivate) users *U, AU, and their relations with roles R, AR*

Note: both user and permission info are directly provisioned by AWS for existing AWS customers.

$ACaaS_{RBAC}$ Design – Cont'd

- *Role module*: manage (e.g., create, delete) *roles R, AR*, and *RH*
  - Towards efficient role-related operations, we adopt the Nested Set Model
  - It assign assigns left and right values to represent a scope of each role in a role hierarchy



- *Session module*:
  - Support efficient role activation and deactivation operations

SEFCOM

ACaaS$_{RBAC}$ Design – Cont'd

- role activation and deactivation algorithms

Algorithm 1: *ComputeActivatePermissions*$(u, r_a) \rightarrow P$

Input: A user u wants to activate a role r_a
Output: A permission set P, of which corresponding IAM policies need to be enforced

```
1  P ← ∅;
2  P_all ← ∅;
3  r_ts ← getImmediateSeniorRole(r_a);
4  if hasRole(u, r_ts) = TRUE AND active(u, r_ts) = TRUE
   then
5      return ∅;
6  else
7      ComputeP(u, r_a);
8      foreach p ∈ Permissions(r_a) do
9          if p ∉ P_all then
10             add p into P;
11         end
12     end
13     return P;
14 end
15 ComputeP(User u, Role r_a)
16 begin
17     R ← getSiblingRoles(r_a) ⋃
       getImmediateJuniorRoles(r_a);
18     if R = ∅ then return;
19     else foreach r ∈ R do
20         if active(u, r) = TRUE then
21             foreach p ∈ Permissions(r) do
22                 if p ∉ P_all then
23                     add p into P_all;
24                 end
25             end
26         else
27             ComputeP(u, r);
28         end
29     end
30 end
```

Algorithm 2: *ComputeDeactivatePermissions*$(u, r_d) \rightarrow P$

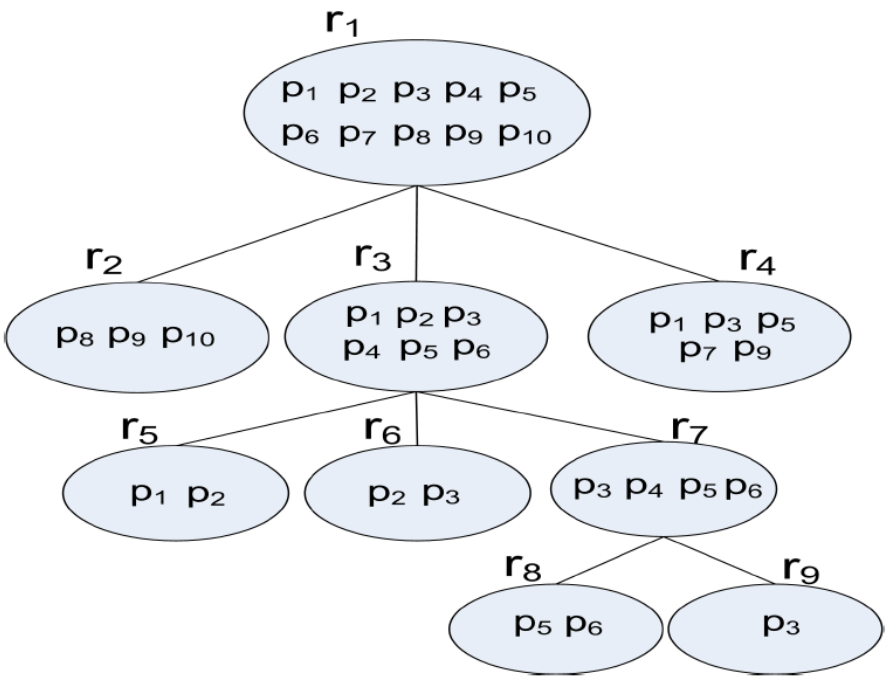
Input: A user u wants to deactivate a role r_d
Output: A permission set P, of which corresponding IAM policies need to be enforced

```
1  P ← ∅;
2  P_all ← ∅;
3  R_senior ← getSeniorRoles(r_d);
4  if R_senior ≠ ∅ then
5      foreach r ∈ R_senior do
6          if active(u, r) = TRUE then
7              return ∅;
8          end
9      end
10 end
11 R_sibling ← getActivatedSiblingRoles(r_d);
12 if R_sibling = ∅ then
13     return Permissions(r_d);
14 else
15     foreach r ∈ R_sibling do
16         if active(u, r) = TRUE then
17             foreach p ∈ Permissions(r) do
18                 if p ∉ P_all then
19                     add p into P_all;
20                 end
21             end
22         end
23     end
24     foreach p ∈ Permissions(r_d) do
25         if p ∉ P_all then
26             add p into P;
27         end
28     end
29     return P;
30 end
```

SEFCOM

$ACaaS_{RBAC}$ Design – Cont'd

- role activation and deactivation algorithms example



$ACaaS_{RBAC}$ Design – Cont'd

- *Constraint module*: provides constraints management services including creating, deleting, updating static constraints as well as SoD
  - Static constrains are specified based on IAM syntax, and enforced by IAM
  - SoD is enforced by constraint module
- *Policy module*: provides Amazon IAM policy generation and pushing services to ensure RBAC configurations of an enterprise can be reflected in AWS cloud platform
  - For example, for each permission in the permission set computed by Algorithm 1 or Algorithm 2, a corresponding IAM policy is constructed and sent to IAM for the enforcement

Implementation

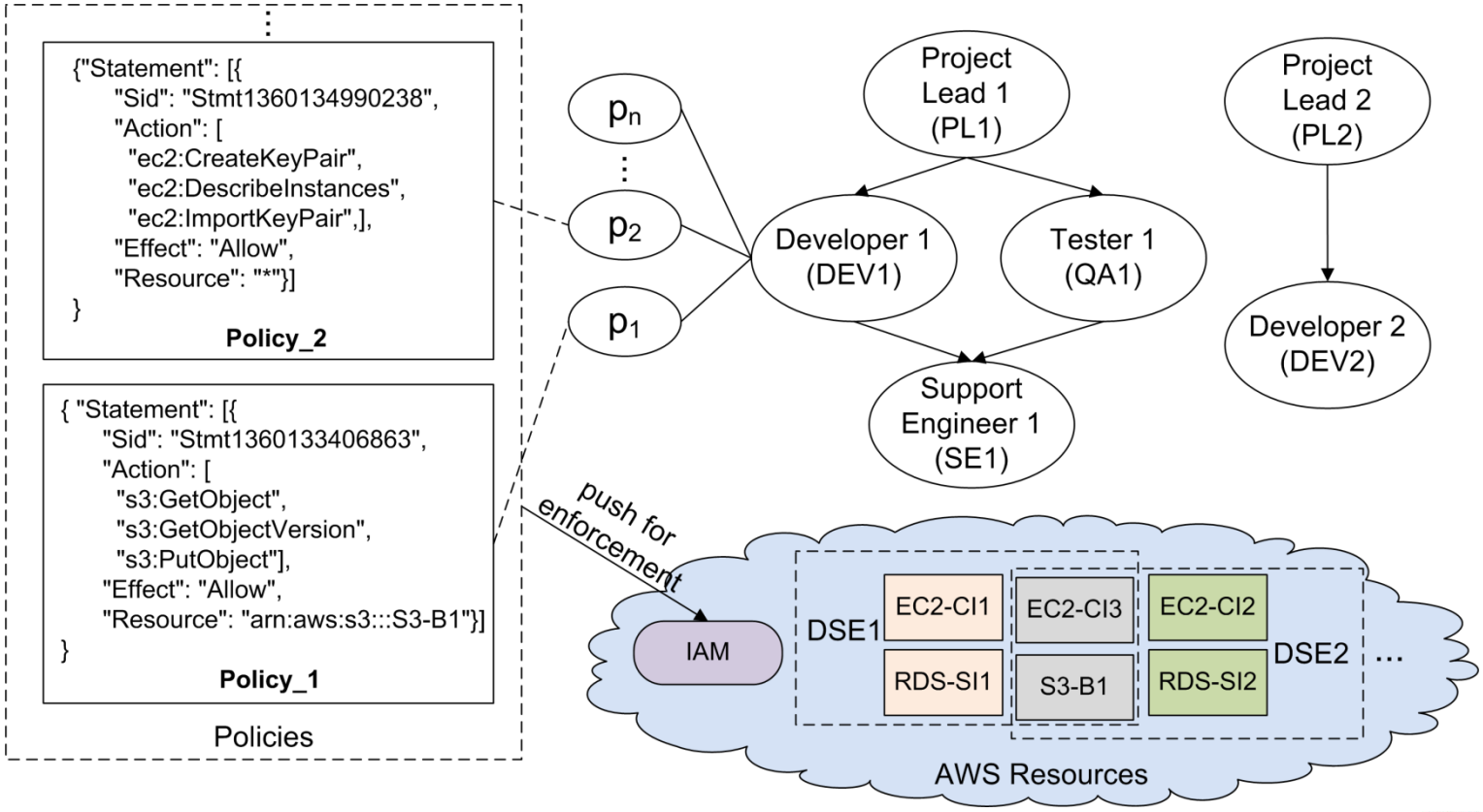
- Implemented a prototype system to provide RBAC services in AWS cloud platform through a web browser interface as well as web services
- The core services of the system are implemented in Java based on AWS SDK 1.3.0 and exposed as web services using GlassFish Metro 2.2
- The web-based management interface is developed by using JavaServer Pages (JSP)

Evaluation

- We evaluated our $ACaaS_{RBAC}$ system in terms *practicality*, *efficiency* and *scalability*
  - A scenario-based case study is presented to illustrate how services provided by $ACaaS_{RBAC}$ can be utilized by cloud applications
  - We then show the performance measured on role activation and deactivation services

Evaluation – Cont'd

- A Sandbox scenario for software development in AWS with $ACaaS_{RBAC}$



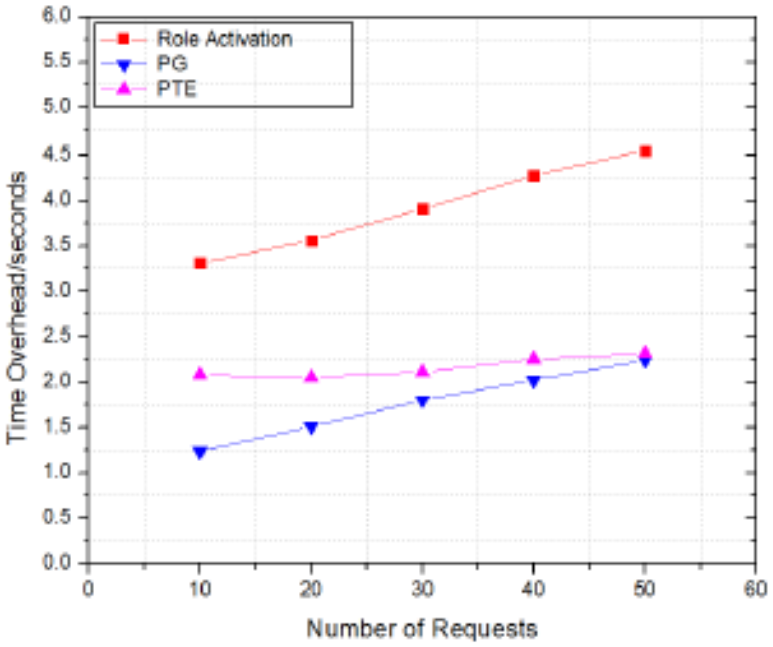
Evaluation – Cont'd

- Alice is a member of the role *Dev1* working on Project 1 and Bob is a member of the role *Dev2* working on Project 2
- In order to complete collaborative task, Alice need to be assigned to role *Dev2*
- We measure the *Dev2* role activation for Bob when he logs in and the *Dev2* role activation for Alice when she is assigned to the role *Dev2* in terms of performance overhead (T) and network traffic overhead (NT).
- Results:

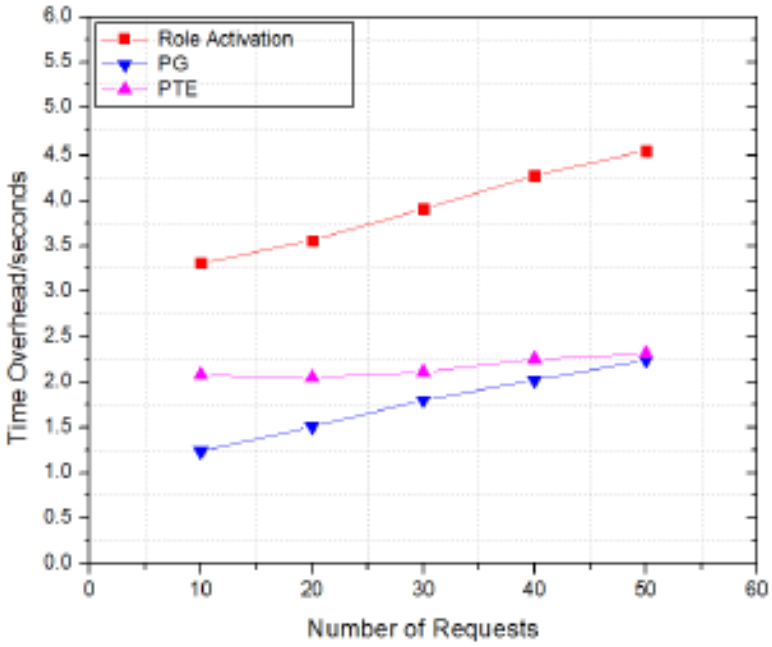
	T(ms)	PG(ms)	PTE(ms)	NT(kb)
Dev2 Activation for Bob	1898.9	91.6	1807.2	15.95
Dev2 Activation for Alice	1200.2	47.4	1152.4	14.652

Evaluation – Cont'd

- In order to measure scalability of $ACaaS_{RBAC}$, measure average performance overhead while increasing the numbers of simultaneous role activation and deactivation requests from users



(a) Activation Time



(b) Deactivation Time

SEFCOM

Related Work

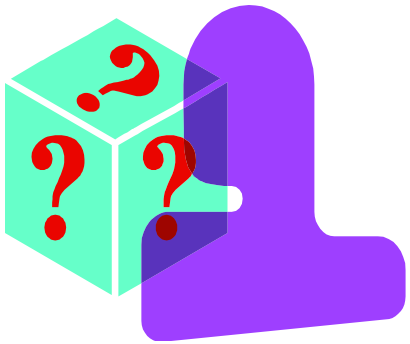
- Authorization and access control solutions in clouds
  - An authorization model that supports multi-tenancy, role-based access control, path-based object hierarchies, and federation [Calero'10]
  - Semantic web technologies to distributed role-based access control method and propose a new Semantic Access Control Policy Language (SACPL) for describing access control policies in clouds [Hu'09]
  - CloudPolice, a distributed access control mechanism implemented in hypervisors, to meet the access control needs of multi-tenancy, network-independence, and scalability in cloud computing environment [Zhu'12]
- Cryptographic techniques based access control in clouds
  - An effective solution for de-coupling access control from services that provide content by leveraging ABE [Echeverria'10]
  - Design of a trusted cloud computing platform (TCCP) [Santos'09]
  - Use proxy reencryption and broadcast revocation algorithms for flexible content access control in cloud [Zhang'12]
  - HASBE scheme which incorporates a hierarchical structure of system users by applying a delegation algorithm to ASBE [Wan'11]

SEFCOM

Conclusion

- We articulated the critical need of a comprehensive and fine-grained access control mechanism to meet dynamic, configurable, and extensible security requirements in public IaaS cloud computing environments
- To accommodate this need, we proposed a new modular architecture towards access control as a service (ACaaS) for supporting multiple access control models
- As a reference implementation, we designed and implemented ACaaSRBAC, a service architecture that supports configurations of RBAC as a service for AWS
- Our case study and system performance evaluation demonstrated the practicality and efficiency of our approach

Future Work

- We would evaluate our system with real world datasets. In addition, we would enhance our system with flexible delegation mechanisms and accommodate revocation requirements, and design a more generic architecture to support other access control policies such as multilevel and general mandatory access control policies

Thank you !!!