Access control for on-demand provisioned cloud infrastructure services
Ngo, C.T.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 3
Multi-tenant Access Control for Intercloud

This chapter is based on the following publications:


3.1 Introduction

NIST’s cloud computing reference architectures [5] provide a basis for cooperation between providers to bring integrated cloud services to customers, defined as and referred thereafter as Intercloud [90, 91]. In the general Intercloud architecture, collaborations between providers form a hierarchical multi-level stack of cloud services where each service can compose from lower-level services and also be integrated into upper level ones. In Figure 3.1, IaaS cloud providers can aggregate individual VRs from different PIPs to build up VIs consisting of virtual computing nodes, virtual storage, reserved network links [10]. PaaS and SaaS providers can utilize the outsourced IaaS services to build up their systems. In turn, the end-user work flow systems may be composed from a set of different cloud resources, which requires interactions between cloud providers, even they do not have direct subscription contracts defined by Service Level Agreements (SLAs).
Such cloud collaborations bring challenges on distrusted authorization across multi-domains between providers. The cloud resources while are directly or indirectly managed at a provider domain, can be accessed by unknown entities based on the cloud resource owners consents. In Figure 3.1, the software running on \( PaaS_3 \) can access to the storage service managed by \( IaaS_1 \), but located at \( PIP_1 \) site. Most current authorization frameworks rely on known identifies of entities or using federated identity management systems with setting up manually when a member want to join to the federation, while the Intercloud requires the dynamic relationship establishment at runtime via third-parties. When collecting distributed decisions, the inconsistency of policies at different domains may lead to high rejected cases, i.e., in a chain of decisions, if the final result is denied, for efficiency, it'd better to be decided at the local domain rather than at a remote one.

We propose a token-based exchanging approach between providers combining with the attribute-based multi-tenant access control model that guarantees the harmonization of distributed authorization policies, thus reduces the denied decision rate to have low system overhead.

### 3.2 Problem Statement

The model in chapter 2 solves multi-tenant access control problems in a single security domain with a cloud provider. For Intercloud scenarios, a provider could play as a tenant of another provider to utilize its cloud resources. This paradigm can be illustrated in [10] and [49] where the VIP can collect VRs from set of PIPs to compose the VI. Our model in chapter 2 can be extended to support Intercloud
scenarios by arrange in hierarchy as follows:

- There are set of cloud providers: \( p_i \in P \), each of them runs the model in chapter 2.
- When a provider \( p_a \) subscribes cloud resources from set of providers \( p_b = \{p_{b_1}, p_{b_2}, \ldots, p_{b_k}\} \), \( p_a \) becomes the tenant of \( p_{b_i} \), thus can manage these resources with its own policies.
- The \( p_a \) can also transfer permissions to its tenants. Permissions either targets to local \( p_a \) resources, or the remote resources at a provider \( p_{b_i} \in p_b \).
- Any \( p_{b_i} \) may be a tenant of other providers, so the chain of providers can be extended.

In such scenarios, we need to solve the challenge of distributed authorization in multiple domains. A request from \( p_a \) domain may need to be authorized at two domains, first at \( p_a \) domain with relevant \( p_a \) tenants’ policies, then at \( p_{b_i} \) domain with the policy issued by \( p_{b_i} \) to \( p_a \). The possible approaches to synchronize and collect decisions are either exchanging tokens or exposing policies between domains. The exchanging token approach needs to deal with token management issues, including storing, synchronization, revocation and overhead of using tokens [68]. In Intercloud paradigm, exchanging policies approach may disclose tenants’ SLAs out of the provider’s domain while still has similar issues with token management [68]. This section proposes a token mechanism that solves token management problems with low overhead on the system performance.

3.3 Extended Model for Multiple Providers

3.3.1 Constraints in Distributed Authorizations

The potential problem in distribute authorization is the conflicting decisions between domains, resulting to the high rejection requests rate at remote domains and increasing system overhead. Preferably, denied requests should be answered as soon as possible at their local domains, rather than at a remote domain in the chain of distributed authorization. It can be solved by establishing grant constraints between the policies at tenant side \( p_a \) with policies at the \( p_b \) provider sides.

In the above scenario, the provider \( p_{b_i} \) issues a policy with the context \( c_{a_i} \) for the provider \( p_a \). At the \( p_a \), instead of contexts created by \( p_a \), contexts \( c_{a_i} | i = 1 \ldots k \) become the root context. All contexts created by \( p_a \) must be confined by the root contexts. It can be excepted for local resources \( X_r \) physically owned by \( p_a \). To synchronize contexts \( c_{a_i} \) between \( p_{b_i} \) and \( p_a \), we base on the SLA describing subscribing resources between them. According to the information model [79], the SLA request is described by INDL semantic concepts and synchronized upon provisioning and re-planning. We then can use SPARQL and policy generation techniques to extract constrained contexts and update to the trusted root list, which is similar to policy generation in Section 2.5.
3.3.2 Token Exchange in Intercloud

The distributed authorization workflow can be the push sequence as in Figure 3.2. It requires that the user needs to have an access token to verify it’s allowed to access the resources at remote provider $p_{bi}$’s domain. The grant token is initially issued by the first provider in the chain $p_a$ as the consent by $p_a$ to subsequent provider $p_{bi}$. The $p_{bi}$ must validate the token issuer $p_a$, then evaluate the request attribute embedded inside the token against its policies. If the decision is positive and the target resource is located in its local domain, $p_{bi}$ issues an access token allowing the user to access it. Otherwise, if the target resource is located at another domain, $p_{bi}$ issues another grant token to user for further distributed authorization process. In this sequence, the communication between authorization services at providers is relayed through the user via exchanging grant tokens and access token.

3.3.2.1 Grant Token

The grant token needs to have the following information:

- Request content approved by the issuer, who allows the request to act on behalf of the issuer: it usually is the vector of attributes including issuer’s subject attributes.

- The approval proof of the issuer: this proof can be enforced by the digital signature mechanism of the issuer, either based on a digital signature using public cryptography or a message authentication code algorithm using symmetric cryptography.
• The lifetime limitation.

• The proof-of-procession of the user, so the issued access token is not a bearer token and only targets for the user. It’s either the user’s public key, or the session shared secret key generated by the user.

For the public key cryptography approach, we propose the grant token issued by $p_a$ and returned to the user $u$ as follows:

$$X := \{X_{pa}, X_r, X_e\}$$

$$m := X | t | pk_u$$

$$\text{granttoken} := SK(sk_{pa}, m)$$

(3.1)

with $SK(sk_{pa}, m)$ is the annotation that the message $m$ is signed by secret key $sk_{pa}$ of the provider $p_a$. The $pk_u$ is the user’s public key, $t$ is the lifetime and $X$ is the vector of attribute request containing $p_a$’s attributes. This grant token allows user to request on behalf of $p_a$ to the remote domain at $p_b$.

For the symmetric key cryptography approach, the grant token has the following information:

$$m := X | t | k_u$$

$$hmac := MAC(K_{pa,pbi}, m)$$

$$ek := E(K_{pa,pbi}, k_u)$$

$$\text{granttoken} := \{X | t | ek | hmac\}$$

(3.2)

with $K_{pa,pbi}$ is the shared secret key between the provider $p_a$ and $p_b$; $MAC$ is a message authentication code algorithm; $k_u$ is the session key of the user; $ek$ is the encryption of $k_u$ by the $K_{pa,pbi}$.

### 3.3.2.2 Access Token

According to the public key approach in Formula (3.1), the access token issued by $p_{bi}$ to the user $u$ is constructed as follows:

$$\text{accesstoken} := SK(sk_{p_{bi}}, tid | t)$$

(3.3)

with $t$ is the issuing timestamp, $tid$ is the identifier to the cached authorization session stored at $p_{bi}$, which contains access token lifetime, user’s associated key $pk_u$ and the involved attributes $X$.

With symmetric key approach in Formula (3.2), the access token contains following:

$$\text{accesstoken} := E(k_u, tid | stoken)$$

$$k_{u,p_{bi}} = k_u | stoken$$

(3.4)

(3.5)

with $stoken$ is the secret generated value by $p_{bi}$ shared to the user. The consequent requests from the user to $p_{bi}$ are signed with the session key $k_{u,p_{bi}}$.

After having the access token, user accesses the protected resource at $p_{bi}$. Upon receiving user’s request with access token, the $p_{bi}$’s resource service validates the access token with either $pk_{p_{bi}}$ for public key scheme, or $k_{u,p_{bi}}$ for symmetric key scheme. If comparison between the request with involved attributes $X$ is positive, the service will serve the request.
3.4 Implementation and Evaluation

3.4.1 Implementation Overview

Our exchanging token approach is implemented in the TokenService of the DACI. The service has a public/private key-pair used for issuing and validating tokens. Upon registration, each tenant is bound with a separate public/private key-pair used for Intercloud communication scenario as described in Section 3.1. In our key management implementation, we choose the RSA algorithm with 2048 bits key length. For digital signature used in issuing tokens, we define the token structure in XML schema and use the XML digital signature standard [92] implemented in the Apache XML security library [93]. We choose RSASSA-PKCS1-v1.5 signature scheme with SHA-1 algorithm [94]. DACI uses Bouncy Castle v1.49 [95] as the Java cryptographic provider.

We deploy DACI instances with TokenService in separate VMs having two virtual cores and 4096 MB RAM. Each VM represents a cloud provider running DACI with the sample datasets in Table 2.4 as in Chapter 2.

In our inter-provider test scenarios, we have two DACIs for $P_a$ and $P_b$ providers, in which $P_a$ subscribes resources of the $P_b$ as described in Section 3.2. PEPs at the user side of the $P_a$ send requests to access to the resource at $P_b$, so DACI of the $P_a$ needs to evaluate its local policies prior issuing grant-tokens for further authorization at $P_b$. Compared to the intra-provider scenario in the last chapter, the token issues and validations increase overhead of the original DACI system.

3.4.2 Evaluation Results

From our experiments, the performance tests show that on average, the response time for an authorization request with issuing grant-token is 320 ms, which is significantly slower than the response time in the intra-provider scenario. The overhead here mostly comes from the digital signing tokens with RSA 2048 bits key-length for every issued token and the XML messages serialization/deserialization. Therefore, we are developing a hybrid key management scheme in which tenants and providers use shared secret keys in communications, which are established and refreshed periodically based on the public/private key-pairs. The symmetric key scheme using message authentication code could improve the system performance significantly compared to the public key scheme.

3.5 Conclusions

In this chapter, we extend the MT-ABAC for distributed, multiple collaborative cloud providers in hierarchy to support Intercloud scenarios with exchanging tokens approach. In future work, we will improve key management model for Intercloud using combining public-key and symmetric cryptography, which could improve the system performance in the Intercloud communications using tokens. We are planning to develop adapter layers between our DACI system using INDL
with popular cloud management systems like OpenStack, CloudStack or Eucalyptus, thus could integrate the DACI with these systems. Regarding authorization policy language, beside XACML in XML profile, we plan to support others as well as supporting our DACI with legacy on-premise authorization systems.