Policy Engine as a Service (PEaaS): An Approach to a Reliable Policy Management Framework in Cloud Computing Environments

Faraz Fatemi Moghaddam*,†, Philipp Wieder*, Ramin Yahyapour*,†
*Gesellschaft fur wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), Göttingen, Germany
†Institute of Informatics, Georg-August-Universität Göttingen, Göttingen, Germany
Email: {faraz.fatemi-moghaddam, ramin.yahyapour, philipp.wieder}@gwdg.de

***Abstract*—Security challenges are the most important obstacle for advancement of IT-based on-demand services and cloud computing as an emerging technology. In this paper, an attribute based policy management engine has been introduced to enhance the reliability of managing different policies in clouds and to provide standard and also dedicated security levels (rings) based on capabilities of the cloud provider and requirements of cloud customers. Accordingly, policy database has been designed based on capabilities and policy engine establishes appropriate relations between policy database and SLA engine to provide security terms as a service. Furthermore, policy match maker and reasoning engine have been designed for syntactic and semantic analysis of security requests based on three-levels of protection ontology to enhance the process of policy management in clouds.**

Keywords—cloud computing, security, policy management, policy engine, semantic policy analysis, protection ontology.

I. Introduction

Cloud computing technology is an emerging paradigm that uses the concepts of virtualization, processing power, distribution and connectivity to store and share IT-resources over a broad network. Despite the considerable benefits of this technology [1], there are some information policy concerns such as security, privacy and access control that have affected the reliability of cloud-based environments [2]. One of the most challenging issues regarding to the information policy concerns is to provide an appropriate level of security for the stored data in cloud storages. In fact, each individual customer needs to be granted reliable security level(s) based on defined details in SLA [3].

These security levels might be common for all customers or independent based on the data sensitivity. Applying a single security level for all stored data is not efficient and takes considerable processing power to manipulate sensitive and also non-sensitive data. On the other hand, managing multiple security levels is the most challenging concern in multi-level policy models and needs an appropriate and efficient algorithm. The most popular approach to express high-level security constraints is based on the usage of metadata and languages for the specification of security policies [4]. The main aim of this paper is to propose an attribute-based policy management model to enhance the process of managing security policies in cloud computing environments. Accordingly, several security specifications and policies are defined to match and manage the most appropriate security level based on data sensitivity.

II. Related Works

There are several policy management models that use metadata and language for specification of security policies.

IETF [5] is a defined framework includes definition language, policy model, sets of policy technologies and a policy architecture meta model to represent, manage, share and reuse policies and policy information in a vendor-independent, interoperable and scalable manner. IETF model mostly focuses on network policies to control IP-Security and Quality of Service (QoS) [6]. One of the most challenging efforts in this model is to map IETF models into specified implementation schema [7] such as Web Based Enterprise Management (WBEM) [8] or Directory Enabled Network (DEN) [9].

Imperial College developed an object oriented policy management framework (Ponder) [10] that contains general architecture and policy deployment model and various extensions for access control and QoS management. Ponder allows users to define events, constraints, constants and other reusable elements that can be part of many policies and allows the instantiation of typed policy specification to support parameterization of policies. However, the lack of generality is the main drawback of Ponder [7]. In fact, Ponder uses several basic policy types and composite policy types each with different syntax.

KAoS is an ontology language uses the Web Ontology Language (OWL) [11] to allow users to define policies to grant predictability and controllability of agents and distributed systems such as grid computing and multi-agent systems [12]. Four main types of policies are defined in KAoS: *Positive-Authorization*, *Negative-Authorization*, *Positive-Obligation* and *Negative-Obligation* and similar to IETF, each policy is

associated with management properties. KAoS provides two sets of services: Domain Services (that enable the hierarchical grouping of users and computational entities to administrate policies easier) and Policy Services (that is based on specification, conflict resolution, management and policy enforcement) [13].

In 2002, HP labs proposed a policy framework (Rei) [14] to provide an independent domain policy specification based on deontic constructs and F-OWL reasoned [15] and to allow several specification policies (*i.e.* right, prohibition, dispensations and obligations). Rei supports two main meta policies: Default Meta Policy (to describe the default behavior of the policy) and Meta-Meta Policy (to allow the setting of precedencies based on default meta policies).

Choi et al. [16] proposed Ontology-Based Access Control Model (Onto-ACM) by using Context-aware Role-Based Access Control (C-RBAC) concepts [17] to provide a semantic analysis model that can address the difference in the permitted access control between service providers and users. Onto-ACM uses context analysis process, user authentication analysis, access context analysis, Ontology handling and security process interface as the main requirements of a fine-granted access to enhance the efficiency policy management and grant a role inheritance by both system administrator and user (*i.e.* data owner) and protect malicious information leakage. However, the lack of simultaneous syntactic and semantic matching of security policies based of cloud computing characteristics is the most important drawback of this model.

Di Modica and Tomarchio (2015) [18] suggested an approach that leverages on the semantic technology to enrich standardized security policies with an ad-hoc content and to enable machine reasoning which is then used for both the discovery and the composition of security-enabled services. In this model, requirements and capabilities for cloud customers and providers are defined within policies which are adopted to policy intersection mechanism provided by WS-Policy [19].

WS-Policy is a recommended framework from W3C for policy specification of Web Services that includes policies that are defined as a collection of alternatives contain assertions to specify well-established characteristics for using selection of various services (*e.g.* requirements, capabilities or behaviors).

In overall, the main concern of described models is the discovery, interoperability and compatibility of security requirements based on characteristics of current distributed networks and cloud-based environments. Hence, this paper uses the concepts of security ontology to define reliable and interoperable security policies in virtualized infrastructure and to grant syntactic and also semantic matching of security policies.

III. Proposed Architecture

The architecture of our proposed framework is based on four main components that have been shown in figure 1: Policy Engine (PE), Policy Database (PD), Policy Match Maker (PMM) and Reasoning Engine (RE). In the suggested framework, a cloud service provider offers two types of security rings to PD based on defined or potential policies:

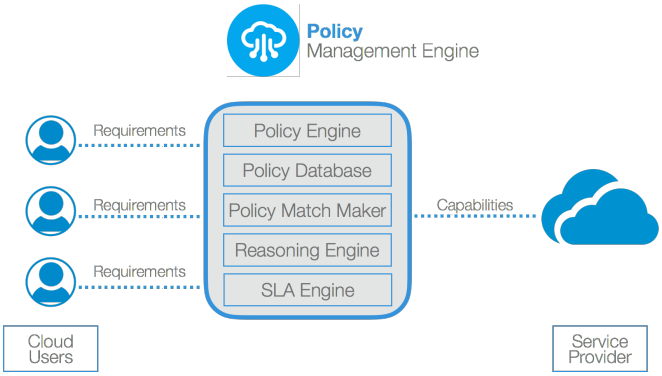

Fig. 1. Basic Architecture of our Proposed Model (icon by [20])

- The first type includes sets of standard policies in package form. In fact, the service provider defines some standard security rings and offers them to customer to select based on desired security level and data sensitivity.
- The second type includes sets of potential security policies based on capabilities of service provider. In fact, several security policies (*e.g.* authentication, obligation, cryptography algorithms, etc.) are introduced to let the customer to create a dedicated custom security ring based on the requirements.

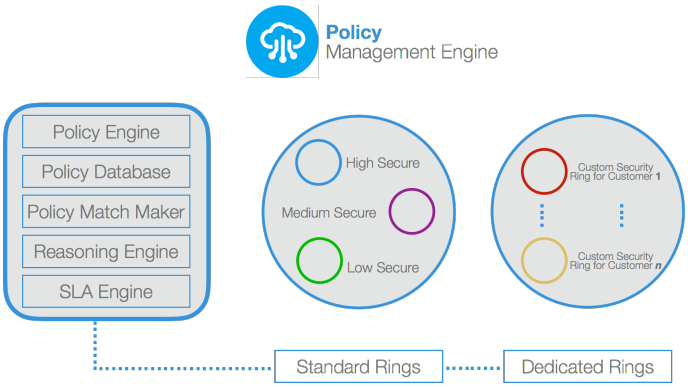

Fig. 2. Security Rings Types in our Proposed Model

PD would be updated periodically based on new or revoked capabilities of service provider and has mutual interaction with SLA engine regarding to these updates.

The main component that is responsible for policy matching and other policy-based procedures in this framework is Policy Engine. PE receives service requests and requirements from cloud customer by API. These requests are received as a selection of existing policy packages or customize parameters based on users' requirements. In the first case (Algorithm 1), the request is processed in PE and is sent to PD for checking the updates, availability, and other specifications in PD.

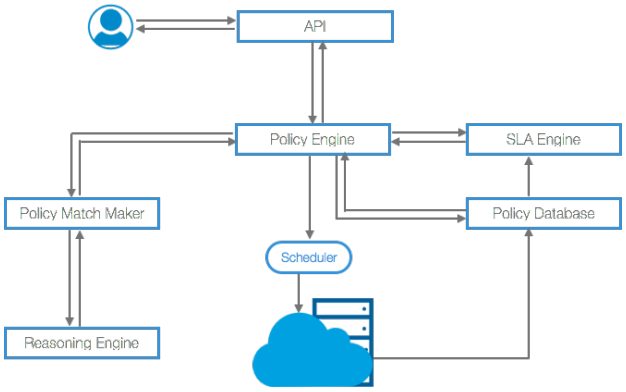

Fig. 3. Relations between Defined Components in Proposed Model

After receiving the confirmation from PD, PE applies for agreement details (including pricing options, security algorithms, other required items) from SLA engine and sends the response to user for final confirmation (Fig. 3). After this confirmation, policy application is sent to scheduler for processing current data or uploading new data. In the second case, cloud customer requests a dedicated security ring based on potential security policies according to the capabilities of the service provider. Typically, the service provider updates list of potential security policies in PD according to the new or revoked capabilities and based on these updates, the SLA details are changed. The process of requesting dedicated security rings is done based selection of potential security mechanism by cloud customer according to the requirements, syntactic checking and match making by PMM and updating necessary changes, and also semantic analysis by RE for simultaneous syntactic and semantic analysis. Algorithm 2 shows the process of creating dedicated rings in details.

IV. Protection Ontology

The protection ontology is a defined object oriented framework that includes potential security concepts such as protocols, mechanism, algorithms and established connections between them to provide an appropriate and reliable policy management model. There are two main super classes that are defined in this ontology: Policy Matrix and Policy Packages.

A. Policy Matrix Super Class (PMSC)

Policy Matrix Super Class is based on 3 levels of sub-classes according to the concepts of inheritance in object oriented designs (Fig. 4):

- Level 1 (Protocol Level): This level includes 6 main security protocol sub-classes: Access Control, Cryptography, Key Management, Transport, Authentication and Signature.

- Level 2 (Mechanism Level): There are several security mechanisms for each protocol that are defined to categorize security algorithms and provide appropriate relation between the highest level and the lowest level of architecture.

- Level 3 (Algorithm Level): This level is the lowest level the protection ontology that includes different security algorithms that are provided be cloud service provider. In fact, the capabilities of service provider are updated regarding to availability or un-availability of security algorithms in this level.

Algorithm 1
Selection of Standard Rings

Input: Set P: Let $P = \{p_1, p_2, \dots, p_n\}$ represents all standard policy packages.
Output: $F(s, PP1)$

1 $\exists\, p_i \in P : p_i = \{sp_1, sp_2, \dots, sp_m\}$
//where sp_j $(j \in \{1,2,\dots,m\})$ donates a defined Security Protocol (e.g. access management protocol, cryptography protocol, authentication protocol, *etc.*) in standard policy package i.

2 $Send\ (p_i, User, PE);$
//The selected policy package from user is sent to PE.

3 $Policy\_Package\ PP1 = new\ Policy\_Package(User_u, req, sp_1, sp_2, \dots, sp_m)$
//PP1 is created as an object from Policy Package class based on selected package by user. $User_u$ and req represent user ID and status of request respectively.

4 $Send\ (PP1, PE, PD);$
//Object PP1 is sent to PD for checking availability, updates and other specifications.

5 $For\ (int\ x = 1;\ x \leq\ m;\ ++x)\ \{$
$bool\ SPCA = Check\_Availability(PP1.sp_x);$
$if\ (SPCA\ ==\ False) then\ \{$
$PP1.req\ =\ False;$
$Break;\}$
$bool\ SPCU = Check\_Updates(PP1.sp_x);$
$if\ (SPCU\ ==\ True) then\ Update(PP1.sp_x)\ ;\}$

6 $if\ (PP1.req\ ==\ true) then\ Send\ (REQ, PE, SLA);$
//If all the policies and protocols are available a request is sent to SLA Engine for generating agreement and pricing details.

7 $Send\ (SLA_{Details}, PE, User);$
$if\ SLA_{Details}.Agreement\ ==\ true)\ then\ F(s, PP1);$
//If the SLA details (includes security services and billing) are confirmed by user, the requested policies are applied to user's data.

Algorithm 2
Selection of Dedicated Rings

Input: Matrix P: Donate a $M \times N \times K$ three-dimensional matrix P to be a feasible and potential security policies with each element p_{mnk} represents a security policy ($m \in \{1,2, \dots, M\}$, $n \in \{1,2, \dots N\}$ and $k \in \{1,2, \dots K\}$) where M, N and K are total types of security protocols (e.g. access, cryptography, authentication protocol, *etc.*), security mechanisms and total choices for each mechanism respectively.
Output: $F(s, PP1)$

1 $for\ (int\ x\ =\ 1\ ;\ x\ <=\ M\ ;\ ++x)\ \{$
$for\ (int\ y\ =\ 1\ ;\ y\ <=\ N\ ;\ ++y)\ \{$
$for\ (int\ z\ =\ 1\ ;\ z\ <=\ K\ ;\ ++z)\ \{$
$if\ \left(p_{xyz}.update_{request}\ ==\ true\right)$
$then\ update\left(p_{xyz}\right) and\ update(SLA.p_{xy})\}\}\}$

//Cloud service provider periodically updates list of potential policies, supported security protocol, mechanisms and algorithms. Moreover, the SLA details are changed regarding to the updated policy lists.

2 $if\ (user.dedreq\ ==\ true)\ then$
$policy\_matrix\ PM1\ =\ new\ policy\_matrix(M, N, K);$

//If cloud customer requests for a dedicated security ring, a policy matrix object is created based on all security protocols and total choices for each protocol. The default value of all items is equal to 0.

3 $for\ (int\ x\ =\ 1\ ;\ x\ <=\ M\ ;\ ++x)\ \{$
$for\ (int\ y\ =\ 1\ ;\ y\ <=\ N\ ;\ ++y)\ \{$
$for\ (int\ z\ =\ 1\ ;\ z\ <=\ K\ ;\ ++z)\ \{$
$if\ (p_{xyz}.availability\ ==\ true\ \&\&\ user.req\ ==\ true)\ then\ \{PM1.x.y.z\ =\ 1;\}\}\}\}$

//The cloud customer can select desired security algorithms based on the requirements and also the availability of the algorithm. It should be noted that each customer can select more than one choice for each security mechanism.

4 $Send\ (PM, PE, PMM);$

//The created and updated object is sent to Policy Match Maker to syntactic matching.

5 $if\ (PM1.Syn\ ==\ ture)\ then\ Send(PM1, PMM, RE) else\ \{$
$for\ (int\ x\ =\ 1\ ;\ x\ <=\ M\ ;\ ++x)\ \{$
$for\ (int\ y\ =\ 1\ ;\ y\ <=\ N\ ;\ ++y)\ \{$
$for\ (int\ z\ =\ 1\ ;\ z\ <=\ K\ ;\ ++z)\ update\ (PM1.x.y.z);\}\}\}$
$Send(PM1, PMM, PE);$

//If the syntactic match making is done successfully the object is sent to Reasoning Engine. Else, PMM tries to update the matrix and send it back to PE for user's review. The process of syntactic match making and updating the matrix is explained in section V.

6 $Send\ (PM1, PMM, RE);$

//The checked object is sent to Reasoning Engine for semantic matching.

7 $if\ (PM1.Sem\ ==\ ture)\ then\ Send(PM1, RE, PE) else\ \{$
$for\ (int\ x\ =\ 1\ ;\ x\ <=\ M\ ;\ ++x)\ \{$
$for\ (int\ y\ =\ 1\ ;\ y\ <=\ N\ ;\ ++y)\ \{$
$for\ (int\ z\ =\ 1\ ;\ z\ <=\ K\ ;\ ++z)\ update\ (PM1.x.y.z);\}\}\}$
$Send(PM1, RE, PE);$

////If the semantic match making is done successfully the object is sent to PE for creation of policy package. Else, RE tries to update the matrix and send it back to PE for user's review. The process of semantic match making and updating matrix is explained in section VI.

8 $Policy\_Package\ PP1\ =\ new\ Policy\_Package(User_u, req)$
$for\ (int\ x\ =\ 1\ ;\ x\ <=\ M\ ;\ ++x)\ \{$
$for\ (int\ y\ =\ 1\ ;\ y\ <=\ N\ ;\ ++y)\ \{$
$for\ (int\ z\ =\ 1\ ;\ z\ <=\ K\ ;\ ++z)\ \{$
$if\ (PM1.x.y.z\ ==\ true)\ then\ PP1.sp_x\ =\ p_{xyz};\}\}\}$

//PP1 is created as an object from Policy Package class based on dedicated package by user that is approved by PMM and RE.

9 $Send\ (PP1, PE, PD);$

//Object PP1 is sent to PD for checking availability, updates and other specifications.

10 $For\ (int\ x = 1;\ x \leq\ m;\ ++x)\ \{$
$bool\ SPCA = Check\_Availability(PP1.sp_x);$
$if\ (SPCA\ ==\ False) then\ \{$
$PP1.req\ =\ False;$
$Break;\}$
$bool\ SPCU = Check\_Updates(PP1.sp_x);$
$if\ (SPCU\ ==\ True) then\ Update(PP1.sp_x)\ ;\}$

11 $if\ (PP1.req\ ==\ true) then\ Send\ (REQ, PE, SLA);$

//If all the policies and protocols are available a request is sent to SLA Engine for generating agreement and pricing details.

12 $Send\ (SLA_{Details}, PE, User);$
$if\ SLA_{Details}.Agreement\ ==\ true)\ then\ F(s, PP1);$

//If the SLA details (includes security services and billing) are confirmed by user, the requested policies are applied to user's data.

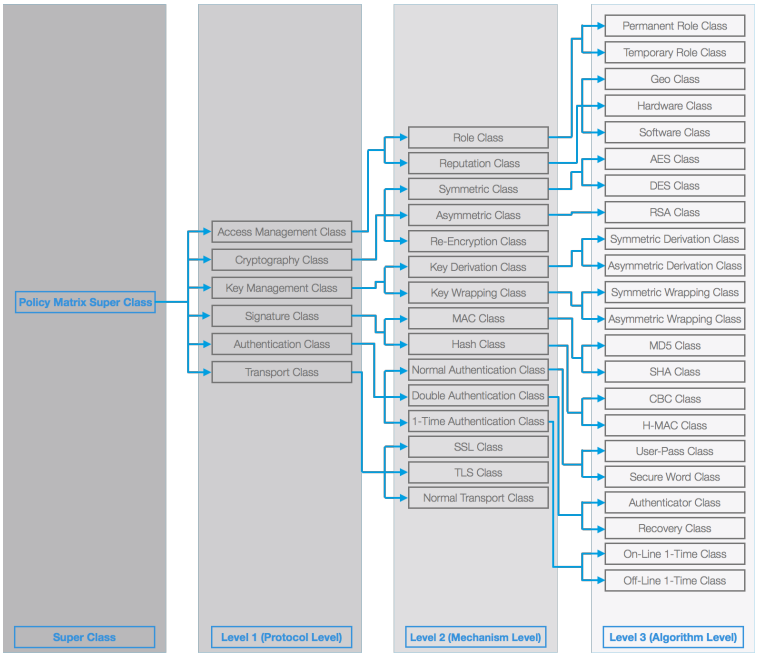

Fig. 4. Protection Ontology Class Levels

Based on security levels, an example of the protection ontology has been written as follows according to WS-Policy structure as a standard policy structure. In this example, the cloud customer requests a permanent role class (*e.g.* Manager, Employee, *etc.*) and geo class (*e.g.* IP address from US) as access management protocol, AES-256 [21] for encryption associated with manual re-encryption as cryptography protocol, CBC hash function as signature protocol, symmetric key derivation and key wrapping as key management protocol, second-password authenticator as authentication class, and TLS as transport class.

```
<wsp:Policy>
   <wsp:ExactlyOne>
      <wsp:all>
         <security:AccessManagementProtocol rdf:ID=”AccessManagementRequirement”>
            <security:RoleMechanism rdf:ID=”RoleClassRequirement”>
               <security:PermanentAlgorithm rdf:resource=”PermanentRoleClass”/>
            </security:RoleMechanism>
            <security:ReputationMechanism rdf:ID=ReputationClassRequirement”>
               <security:GeoAlgorithm rdf:resource=”GeoClass”/>
            </security:ReoutationMechanism>
         </security:AccessManagementProtocol>
         <security:CryptographyProtocol rdf:ID=”CryptographyRequirement”>
            <security:SymmetricMechanism rdf:ID=”SymmetricRequirement”>
               <security:AESClass rdf:resource=”AESClass”, key:”256”/>
            </security:SymmetricMechanism>
            <security:ReEncryptionMechanism rdf:ID=REEncryptionRequirement”>
               <security:ManualReEncryption rdf:resource=”ManualReClass”/>
            </security:ReEncryptionMechanism>
         </security:CryptographyProtocol>
         <security:SignatureProtocol rdf:ID=”SignatureRequirement”>
            <security:HashMechanism rdf:ID=”HashClassRequirement”>
               <security:CBCAlgorithm rdf:resource=”CBCClass”/>
            </security: HashMechanism>
         </security: SignatureProtocol>
         <security:KeyManagementProtocol rdf:ID=”KeyManagementRequirement”>
            <security:KeyWrappingMechanism rdf:ID=”KeyWrappingClassRequirement”>
               <security:SymmetricAlgorithm rdf:resource=”SymmetricWrClass”/>
            </security: KeyWrappingMechanism>
            <security:KeyDerivationMechanism rdf:ID=KeyDerivationClassRequirement”>
               <security:SymmetricAlgorithm rdf:resource=”SymmetricDerClass”/>
            </security: KeyDerivationMechanism>
         </security:KeyManagementProtocol>
         <security:AuthenticationProtocol rdf:ID=”AuthenticationRequirement”>
            <security:DoubleMechanism rdf:ID=”DoubleClassRequirement”>
               <security:AuthenticatorAlgorithm rdf:resource=”AuthenticatorClass”/>
            </security: DoubleMechanism>
         </security: AuthenticationProtocol>
         <security:TransportProtocol rdf:ID=”TransportRequirement”>
            <security:TLSMechanism rdf:ID=”TLSClassRequirement”/>
         </security: TransportProtocol>
      </wsp:all>
   </wsp:ExactlyOne>
</wsp:Policy>
```

Regarding to the protection ontology, PMSC is capable of directing various security algorithms and classifying them based on the security mechanisms in the first level and security protocols in the second level of classification. If fact, the cloud service provider only needs to update the algorithm level based on new or revoked features and capabilities, and to categorize them according to the 1st and 2nd level of protection ontology.

The process of mapping policies is done based on defined resources in each protocol, mechanism or algorithm. Hence, PMSC is created by establishment of appropriate mapping between security terms and semantic concepts. Table 1 shows an example of this establishment in details. Each security term is mapped to a semantic resource based on a protocol (row), a mechanism (column) and an algorithm (leaf).

B. Policy Package Super Class (PPSC)

PPSC is a super class that provides a policy object for each stored data in cloud storage to apply set of existing policies as standard rings or to create an individual and dedicated security ring based on the requirements. In the first case, PPSC object is created after checking the availability and applied updates. However, in the second scenario, an approved PMSC object after syntactic and semantic analysis is required for creating a PPSC object. There are several data and security parameters and methods that are defined in PPSC for applying security terms to data based on the capabilities and requirements

V. Syntactic Analysis

Policy Match Maker is the responsible component for syntactic matching of requested policies based on the capabilities and requirements. If fact, the policy matrix object syntactically analysed in PMM to find out potential errors regarding to the relations between protocols, mechanisms and algorithms. There are several syntactic functions that are possible to examine in match maker process such as the minimum selections in each protocol, the priority sort based on requirements in individual protocols and the established limitation of selected algorithm based on mechanism and protocol. Algorithm 3 shows two syntactic analysis functions based on selected algorithms in details.

VI. Semantic Analysis

RE is the responsible component for the process of semantic analysis to check whether the selected policies are semantically matched according to the capabilities and requirements. The process of semantic analysis is based on *n* rounds.

A. 1st Priority Analysis Round (1PA)

In the first steps all of the security algorithms with value of 1 are considered and analysed. The simplest scenario is happened when there are one or more than one un-conflicted algorithms with value of 1 in each security protocols. If fact, each algorithm should not be conflicted with another one in same or different protocol levels. Table 3, shows four examples of conflicted algorithms in after checking syntactically in PMM. In the first example, there are 2 encryption algorithm with value of 1 in symmetric and asymmetric cryptography mechanisms that are adverse but syntactically acceptable by PMM. This confliction happens in the protocol level of encryption and key management according to the second example and happens in authentication protocol in the third example. The semantic analysis confirms the first priorities in example 4 without any contradiction but in the other cases the second round of semantic analysis is called.

Table 1
Mapping Between Security Terms and Semantic Concepts Based on PMSC

Security Term	Semantic Resource	Protocol	Mechanism	Algorithm	Row	Column	Leaf
Permanent Role Access Control	`PermanentRoleClass`	Access	Role	Permanent	1	1	1
Temporary Role Access Control	`TemporaryRoleClass`	Access	Role	Temporary	1	1	2
Geographical Access Control (IP-Based)	`GeoClass`	Access	Repudiation	Geo	1	2	1
Hardware Access Control (MAC)	`HardwareAccessClass`	Access	Repudiation	Hardware	1	2	2
Software Access Control (OS, Browser)	`SoftwareAccessClass`	Access	Repudiation	Software	1	2	3
AES Encryption	`AESClass`	Cryptography	Symmetric	AES	2	1	1
DES Encryption	`DESClass`	Cryptography	Symmetric	DES	2	1	2
RSA Encryption	`RSAClass`	Cryptography	Asymmetric	RSA	2	2	1
Manual Re-Encryption	`ManualREClass`	Cryptography	Re-Encrypt	Manual	2	3	1
Time-Based Re-Encryption	`TimeREClass`	Cryptography	Re-Encrypt	Periodically	2	3	2
Symmetric Key Wrapping	`SymmetricWrClass`	Key Manage	Wrapping	Symmetric	3	1	1
Symmetric Key Wrapping	`AsymmetricWrClass`	Key Manage	Wrapping	Asymmetric	3	1	2
Symmetric Key Derivation	`SymmetricDerClass`	Key Manage	Derivation	Symmetric	3	2	1
Asymmetric Key Derivation	`AsymmetricDerClass`	Key Manage	Derivation	Asymmetric	3	2	2
MD5 Signature Algorithm	`MD5Class`	Signature	MAC	MD5	4	1	1
SHA Signature Algorithm	`SHAClass`	Signature	MAC	SHA	4	1	2
CBC Signature Algorithm	`CBCClass`	Signature	Hash	CBC	4	2	1
H-MAC Signature Algorithm	`HMACClass`	Signature	Hash	H-MAC	4	2	2
Asymmetric Digital Signature Algorithm	`DSSClass`	Signature	Digital	DSS	4	3	1
User-Pass Authentication	`UserPassClass`	Authentication	Normal	User-Pass	5	1	1
User-Pass and Secure Word	`SecureWordClass`	Authentication	Normal	S-Word	5	1	2
Using Authenticator Component	`AuthenticatorClass`	Authentication	Double	Auth-App	5	2	1
Using Recovery Email/Number	`RecoveryClass`	Authentication	Double	Recovery	5	2	2
Online One-Time Password	`OnlineOneTimeClass`	Authentication	Double	Online	5	3	1
Offline One-Time Password	`OfflineOneTimeClass`	Authentication	Double	Offline	5	3	2
TLS Connection	`TLSClass`	Transport	TLS	-	6	1	-
Normal Connection	`NormalConClass`	Transport	Normal	-	6	2	-

Algorithm 3
Syntactic Analysis of Policy Matrix Object

Input: PM1: a policy matrix object form PMSC that has been created with desired inputs by cloud customer.

Output: $\begin{cases} Send\ (PM1, PMM, RE) & PM1.Syn == true \\ Send\ (PM1, PMM, PE) & PM1.Syn == falsa \end{cases}$

//PM1 is sent to RE for semantic analysis if the syntactic analysis is confirmed.

1 $for\ (int\ x\ =\ 1\ ;\ x\ <=\ M\ ;\ ++x)\ \{$
$for\ (int\ y\ =\ 1\ ;\ y\ <=\ N\ ;\ ++y)\ \{$
$for\ (int\ z\ =\ 1\ ;\ z\ <=\ K\ ;\ ++z)\ \{syntemp_{xy} = syntemp_{xy} + PM1.x.y.z;\}$
$if\ (syntemp_{xy} == 1\ ||\ syntemp_{xy} == 0) then\ break;$
$else\ for\ (int\ z\ =\ 1\ ;\ z\ \leq\ K\ ;\ ++z) \quad \{SynPeriority(PM1.x.y, 1,2, \dots, syntemp_{xy}); \}\}\}$

//In the first syntactic analysis, all capable security algorithms in each security mechanism are analysed. If the user does not apply for any of algorithms in a security mechanism, the first analysis is stopped. Also, if the user selects only one security algorithm, the selected algorithm achieves the highest priority. However, if the user selects more than one algorithm in one security mechanism column, the selected items should be prioritized based on the capabilities and requirements with the highest priority of 1 (perfect match), low priorities of 2 and 3 (close and possible match) and no-match priority with value of 0.

2 $for\ (int\ x\ =\ 1\ ;\ x\ <=\ M\ ;\ ++x)\ \{$
$for\ (int\ y\ =\ 1\ ;\ y\ <=\ N\ ;\ ++y)\ \{$
$for\ (int\ z\ =\ 1\ ;\ z\ <=\ K\ ;\ ++z)\ \{syntemp_{xy} = syntemp_{xy} + PM1.x.y.z;\}$
$syntemp = syntemp + syntemp_{xy};\}$
$if\ (syntemp == 0) then\ \{$
$PM1.Syn == false;$
$Break;\}\}$

//The second process of syntactic analysis ensures about minimum selection of security policies in each security protocol.

3 $if\ (PM1.Syn == ture)\ then\ Send(PM1, PMM, RE) else\ \{$
$for\ (int\ x\ =\ 1\ ;\ x\ <=\ M\ ;\ ++x)\ \{$
$for\ (int\ y\ =\ 1\ ;\ y\ <=\ N\ ;\ ++y)\ \{$
$for\ (int\ z\ =\ 1\ ;\ z\ <=\ K\ ;\ ++z)\ update\ (PM1.x.y.z);\}\}\}$
$Send(PM1, PMM, PE);$

B. 2nd Priority Analysis Round (2-PA)

The second round of semantic analysis uses three main functions to match all desired policies with un-conflicted security terms: Elimination, Substitution and Finalization. The first method eliminates one of the conflicted terms without any significant effect to users' security requirements and the second on tries to substitute one of the conflicted terms with other priorities. For instance, 2-PA considers two encryption algorithms with different mechanism in example 1 (Table 3) with same priority. In this case, one of the encryption algorithms will be eliminated based on the requirements (*i.e.* symmetric or asymmetric based on data privacy that can be private, public or unlisted). In the second example, asymmetric derivation conflicts with AES encryption and symmetric wrapping. Hence, 2-PA uses substitution function to use the second priority of key derivation mechanism and change it to symmetric derivation. If all of conflicted terms are modified and the current policy matrix object does not meet any confliction in all algorithms with value of 1, the object is finalized and other priority values (*i.e.* 2 and 3) are changed to 0. Else, the next semantic analysis round is called.

C. nth Priority Analysis Round (n-PA)

The process of semantic analysis needs *n* rounds to check conflicted security terms with same functions. If the confliction is solved by each of these modification functions, the finalization function of each round transfers the object to the previous round. However, if all of the possibilities in the n^{th} round are checked without any appropriate and logical results, the value of semantic analysis will be changed to false and will be sent back to PE for further actions. Figure 5 shows the semantic analysis rounds in details.

Table 3.
Example of Conflicted Algorithms in Same or Different Protocol Levels.

Protocol	Example 1	Example 2	Example 3	Example 4
Access	Permanent Role	Permanent Role	Permanent Role	Permanent Role
Access	Geo	Geo	Geo	Geo
Cryptography	AES !	AES !	AES	AES
Cryptography	RSA !	-	-	-
Key Manage	Symmetric Wrap	Symmetric Wrap !	Symmetric Wrap	Symmetric Wrap
Key Manage	Symmetric Derivation	Asymmetric Derivation !	Symmetric Derivation	Symmetric Derivation
Authentication	User-Pass	User-Pass	User-Pass !	User-Pass
Authentication	-	-	Authenticator !	-
Signature	MD5	MD5	MD5	MD5
Transport	TLS	TLS	TLS	TLS

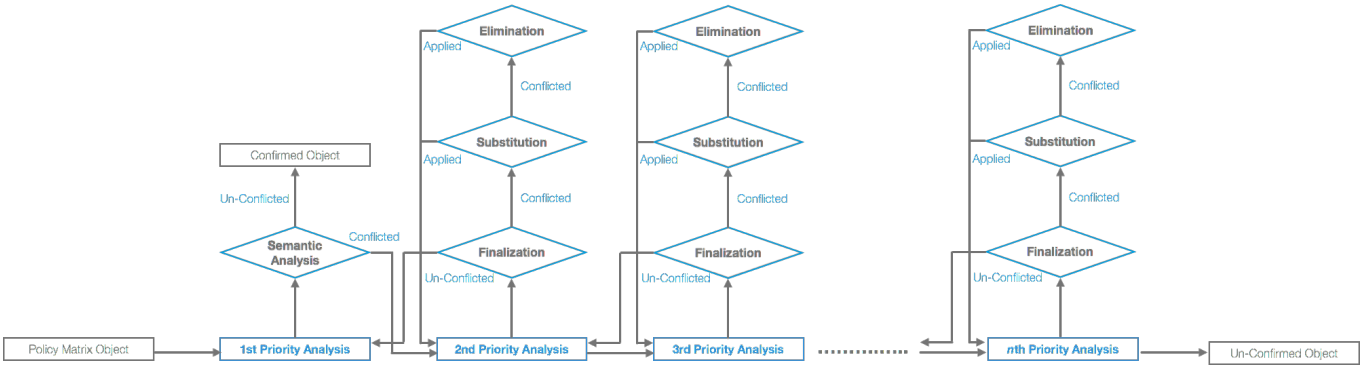

Fig. 5. Semantic Analysis Rounds

VII. Conclusion

Regarding to security challenges in cloud-based environments, an attributed-based policy engine management was introduced in this paper to provide standard and dedicated security levels based on the the capabilities of cloud provider and requirements of cloud customers. Accordingly, the policy engine establishes appropriate relations between policy database and SLA engine to provide security terms as a service in cloud computing models. Furthermore, a syntactic and semantic analysis of security requests have been designed based on three-levels of protection ontology to enhance the process of policy management in clouds.

Acknowledgment

This research has been supported by Clean Sky ITN project (607584 Grant No.) funded by the Marie-Curie-Actions within the 7th Framework Program of the European Union (EU FP7).

References

[1]. F. Fatemi Moghaddam, M. Ahmadi, S. Sarvari, M. Eslami, and A. Golkar, "Cloud Computing Challenges and Opportunities: A Survey," in *Proc. of 1st International Conference on Telematics and Future Generation Networks (IEEE TAFGEN),* 2015, pp. 34–38.

[2]. H. Takabi, J.B. Joshi, and G.J. Ahn, "Security and Privacy Challenges in Cloud Computing Environments," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 24–31, 2010.

[3]. S.A. de Chaves, C.B. Westphall, and F.R Lamin, "SLA Perspective in Security Management for Cloud Computing," in *Proc. of Sixth International Conference on Networking and Services (ICNS), 2010*, pp. 212-217.

[4]. T. Phan, J. Han, J. Schneider, T. Ebringer, and T. Rogers, "A Survey of Policy-Based Management Approaches for Service Oriented Systems," in *Proc. of 19th Australian Conference on Software Engineering (ASWEC)*, 2008, pp. 392–401.

[5]. Y.Snirand, Y.Rambergand, J.Strassnerand, R.Cohenand, and B. Moore, "Policy Quality of Service (QoS) Information Model," *Technical report, IETF*, 2003.

[6]. B. Moore, E. Ellesson, J. Strassner, and A. Westerinen, "Policy Core Information Model," *Version I Specification*, 2001.

[7]. T. Phan, J. Han, J.G. Schneider, T. Ebringer, and T. Rogers, "A Survey of Policy-Based Management Approaches for Service Oriented Systems," in *Proc. 19th Australian Conference on of Software Engineering (ASWEC)*, 2008, pp. 392-401.

[8]. IETF. Specification for the Representation of CIM in XML, Version 2.2. *Technical Report, IETF*, 2007.

[9]. J. Strassner, "Mapping the Policy Core Information Model to a Directory," *Technical Report, OASIS*, 2001.

[10]. N. Damianou, "A Policy Framework for Management of Distributed Systems," *Imperial College*, 2002.

[11]. A. Uszok, J. Bradshaw, M. Johnson, R. Jeffers, A. Tate, J. Dalton, and S. Aitken, "KAoS Policy Management for Semantic Web Services," *IEEE Intelligent Systems*, vol.19, no.4, pp. 32–41, 2004.

[12]. A. Uszok, J. Bradshaw, R. Jeffers, M. Johnson, A. Tate, J. Dalton, and S. Aitken, "KAoS Policies for Web Services," *Technical Report, W3C*, 2003.

[13]. A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott, "KAoS Policy and Domain Services: Toward a Description Logic Approach to Policy Representation, De-confliction, and Enforcement," *Policy*, 2003.

[14]. L. Kagal, "Rei: A Policy Language for the Me-Centric Project," *Technical Report, HP Labs*, 2002.

[15]. Y. Zou, T. Finin, and H. Chen, "F-OWL: An Inference Engine for the Semantic Web. In Formal Approaches to Agent-Based Systems," *Lecture Notes in Computer Science. Springer-Verlag*, vol. 3228, pp. 238-248, 2004.

[16]. C. Choi, J. Choi, and P. Kim, "Ontology-Based Access Control Model for Security Policy Reasoning in Cloud Computing," *The Journal of Supercomputing*, vol. 67, no. 3, pp. 711-722, March 2014.

[17]. M.N. Tahir M, "C-RBAC: Contextual Role-Based Access Control Model," *Ubiquitous Computer Communication Journal*, vol. 2, no. 3, pp. 67-74, 2007.

[18]. G. Di Modica and O. Tomarchio, "Matchmaking Semantic Security Policies in Heterogeneous Clouds," *Future Generation Computer Systems*, vol. 55, pp. 176–185, March 2015.

[19]. W3C, Web Services Policy 1.5 - Framework, W3C Recommendation, September 2007. Available at: http://www.w3.org/TR/ws-policy/.

[20]. Icon made by Freepik from www.flaticon.com.

[21]. J. Daemen, and V. Rijmen, "AES Proposal: Rijndael". *National Institute of Standards and Technology*, pp. 1-10. April 2001.