Towards a Fine-Grained Privacy-Enabled Attribute-Based Access Control Mechanism

Que Nguyet Tran Thi(✉) and Tran Khanh Dang

Ho Chi Minh City University of Technology, Ho Chi Minh City, Vietnam
{ttqnguyet,khanh}@hcmut.edu.vn

Abstract. Due to the rapid development of large scale and big data systems, attribute-based access control (ABAC) model has inaugurated a new wave in the research field of access control. In this paper, we propose a novel and comprehensive mechanism for enforcing attribute-based security policies stored in JSON documents. We build a lightweight grammar for conditional expressions that are the combination of subject, resource, and environment attributes so that the policies are flexible, dynamic and fine grained. Besides, we also present an extension from the ABAC model for privacy protection with the approach of purpose usage. The notion of purpose is associated with levels of data disclosure and constraints to support more fine-grained privacy policies. A prototype built for the proposed model using Java and MongoDB has also presented in the paper. The experiment is carried out to illustrate the relationship between the processing time for access decision and the complexity of policies.

Keywords: Attribute based access control model · Purpose based access control model · Privacy protection · Privacy preserving

1 Introduction

Since the rapid development of large scale, open and dynamic systems, the shortcomings of traditional access control models (e.g. Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role based Access Control (RBAC) [1]) have gradually revealed, for example, applied for only closed systems, role explosion, complexity in compulsory assignments between users, roles, and permissions, and inflexibility in specifying dynamic policies and contextual conditions. Attribute based access control models (ABAC) have been recently investigated [2–4] and considered as one of three mandatory features for future access control systems [5].

Extensible Access Control Markup Language (XACML) 3.0 is an industrial OASIS standard[1] for enforcing access control policies based on attributes, considered as a predecessor of ABAC. In XACML policies, every operation on attributes even trivial conditions such as comparison requires function and data type definitions. This has caused the verbosity and difficulty in the specification of policies. Moreover, XACML is based on XML, which is not well-suited for Web 2.0 applications. Meanwhile,

[1] https://www.oasis-open.org/committees/xacml/.

JavaScript Object Notation (JSON) language[2] is the fat-free alternative of XML. In [27], the experimental results indicate that JSON is remarkably faster and uses fewer resources than XML. Thus, JSON is currently a light weight and widely used data interchange format in the Web of Things. Moreover, since JSON is a subset of JavaScript, it is easier to parse components of a policy into programming objects for further processing. Besides, JSON has been used in many NoSQL databases for storage and retrieval with the high performance. Such advantages of JSON have brought the motivation for our work when using it to model attribute based policies.

Furthermore, our mechanism is built on the principle of the NIST Standard ABAC model that an access decision is *permitted* only if the request satisfies conditions on attributes of subject, resource and environment specified in policies. We also propose a light-weight grammar for conditional expressions, which are human readable text and enough robust to describe complex policies such as user, data, environment driven policies. Besides, we also build an additional module by extending the ABAC model for data privacy protection.

Privacy is a major concern in both of research and industrial fields due to dissemination of personal and sensitive data without user control, especially in mobile and ubiquitous computing applications and systems. In [7], privacy is defined as the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others. Most previous studies have considered privacy protection in access control models as constraints on purpose of data usage. The research on purpose based access control (PBAC) model has recently drawn many interests, although it has developed since 2000s. However, to the best of our knowledge, no research has integrated PBAC into ABAC. The novel contribution of our work includes three main aspects: (1) using JSON to specify attribute based policies, (2) integrating PBAC model into ABAC model and (3) developing the prototype by Java and MongoDB database for demonstrating privacy preserving attribute based policy evaluation mechanism.

The rest of the paper is organized as follows. Section 2 gives a brief survey of related works. Section 3 presents the overview of our approach. In Sect. 4, we introduce the structure of policies and main components in our proposed model. Section 5 indicates the mechanism of the proposed access control model in details. The experiment for evaluating the processing time is shown in Sect. 6. Concluding remarks and future work are discussed in Sect. 7.

2 Related Work

The development of Information Technology, especially in the age of Big Data and Internet of Things, causes the role explosion problem and increases the complexity in permission management in RBAC models which have been dominant for a long time [20, 21]. An emerging interest in addressing these problems is ABAC models, which can be adaptable with large, open and dynamic environments [2, 3].

[2] www.json.org.

In the common approach of ABAC, according to the NIST standard [2], authorization decision is based on rules that simultaneously specify a set of conditions on numerous attributes such as subject, object, action and environment for a certain valid permission. There are many research works on ABAC. In [28], the authors have presented a taxonomy of ABAC research which is demonstrated in Fig. 1. According to the classification, our work focuses on the branches as follows: *ABAC models, policy language* and *confidentiality of attributes*. In this survey, ABAC research about confidentiality of attributes means that how to ensure the privacy of attributes in the model. It has been also recognized that no research work in the survey have used JSON for policy language and addressed data privacy protection in ABAC.

There are two fields for researching about ABAC models, pure ABAC models and hybrid models. Several papers have provided various approaches for a general model. In [4], the authors took a first consideration about formal connections between traditional access control models (DAC, MAC, RBAC) and ABACα which consists of users, subjects, objects and their attributes. In this ABACα, the authors did not mention the environment component in policies as well as the enforcement mechanisms. In [2, 3], the authors provided a definition of ABAC and considered about using ABAC in organizations according to NIST standard. However, the implementation has not been discussed yet in these papers. In another paper [29], Next Generation Access Control (NGAC) takes advantages of graphs illustrating assignment and association between attributes and values to perform access rights. It has provided benefits for policy review and management. However, the cost for building such graphs and the complexity of the policy evaluation algorithm increase significantly when the size of attribute domains and the variety of data structure grow up. In hybrid models, majority of research works have integrated traditional access control models (DAC, MAC, and RBAC) with ABAC such as [21–23]. About *policy language* to express policy enforcement mechanism in the above papers, several approaches such as XACML, logic programming languages or UML have been proposed. For the last problem, the *confidentiality of attributes*, access control models need to provide a mechanism which can protect data attributes with the highest fine-grained level as possible. Attribute based access control models can allow or not allow to access to each data attribute in various context through attribute based policies. Purpose based access control (PBAC) model is another approach to protect data privacy based on the concept of purpose of data usage. A purpose compliance check in PBAC models depends on the relationship between access purposes and intended purposes of data objects ranging from the level of tables to the data cells [7–10]. In the beginning, Byun et al. [7] proposed the model with two types of allowable and prohibited intended purposes. It was then extended with an additional purpose, i.e. conditional intended purpose [9]. Several works have been conducted on enhancing this model by combining with role based access control (e.g., [11–14]), implementing with relational database management systems (DBMSs) with the technique of SQL query rewriting [15] and integrating with MongoDB [16]. Recently, action-aware with indirect access and direct access has also been considered in policies [17]. Nevertheless, research works about PBAC have not expressed privacy policies with the approach of attribute based policies yet except of the paper in [30]. However, such PBAC models have not provided the fine granularity for privacy policies. In such approaches, there are three setting levels for disclosing values of

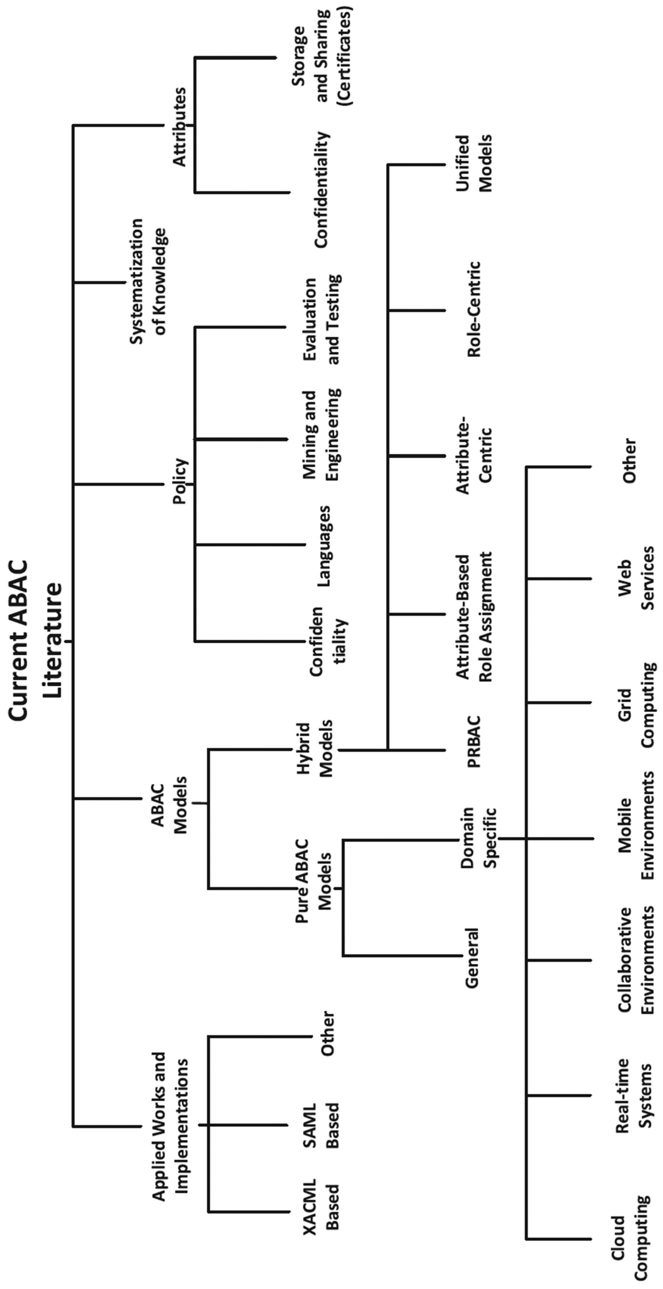


Fig. 1. A taxonomy of attribute based access control research [28]

attributes, that is, *show, hide* and *partial*. All data items of the same attribute have the same disclosure level for the same purpose. Besides, such privacy policies have also not considered about contextual conditions.

With the different approach compared to related works, our research takes a new mechanism for enforcing attribute based policies using the JSON language and proposes an additional module for data privacy protection based on the principle of PBAC and its enhancement.

In summary, our work contributes a novel and comprehensive attribute based access control mechanism which can preserve data privacy. In this mechanism, we use JSON which is more light-weight than XML and widely used in Web 2.0 applications as the language for policy specification. The additional module, *PurJABAC*, is integrated into the model based on the purpose concept to describe attribute based data privacy policies. Our approach can support fine grained policies with contextual and attribute based constraints and protect data attributes with various disclosure levels according to the tree of data generalization.

3 The Overview

In this paper, we propose a model, called as the JABAC model which is an integration between ABAC, PBAC and using JSON to express attribute based security policies to regulate data accesses and protect data privacy. We also provide a mechanism to execute this model. In this section, we briefly describe the access control mechanism of our proposed model. In Fig. 2, requests from applications are sent to the *Policy Enforcement Point* module. For each request, all necessary data is retrieved. Both of data and the request are processed and converted to the *JSON based request context.* After that, it is sent to the *JABAC* module to be decided whether it is permitted or denied according to the predefined attribute based policies stored in *JSON document store database*. In addition, before returning data to the requester, *JABAC* calls the *PurJABAC* module to filter data according to privacy policies. We achieve privacy awareness through the *PurJABAC* module which enforces privacy policies to show, hide and generalize data before the requester receives them.

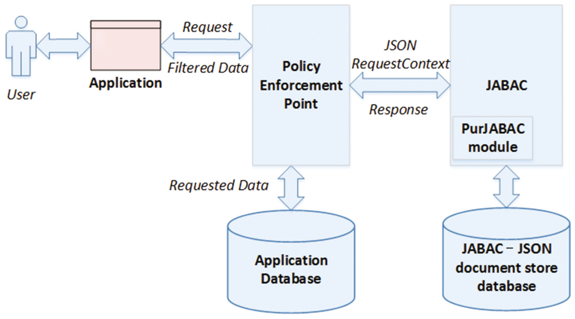


Fig. 2. Overview of the approach

A *JSON based request context* contains information of a request string and the context at the time of requesting. Any request issued from the application is converted to the structure of a *request context* in the JSON format by *Policy Enforcement Point*. If there exists at least one rule requiring requested data during the process of policy evaluation, the data queried from the application database will be converted into the JSON format and filled in the *request context* for processing.

After evaluating necessary policies, *JABAC* returns a response to *Policy Enforcement Point* for the decision result. The response also contains the final data if they are filtered according to related privacy policies by the *PurJABAC* module. *Policy Enforcement Point* has the responsibility to convert data into the format of application.

The communication between the application, the application database and *JABAC* uses the JSON format to exchange data. Therefore, *JABAC* is independent from the technology of application and application database.

Unlike the traditional purpose based access control models, in the approach, all degrees of privacy policies from the table one to the cell one are expressed in the same way under the form of attribute based policies through generalization functions. Furthermore, we take attribute based conditions into consideration for privacy policies.

The structure of access control and privacy policies is identical. Therefore, in general, we provide a simple but sound and comprehensive solution. The details of the JABAC model and its mechanism will be provided in Sects. 4 and 5.

4 The JABAC Model

We describe the proposed access control model in this section before describing the mechanism to enforce this model. When a subject s accesses an object o, the authorization process is carried out through two stages called as 2-stage authorization: (1) access policy authorization and (2) privacy policy authorization. The first step using access policies verifies that the request is legitimate with rights for the subject to access data. After that, the request is transferred to the second stage for checking privacy compliance based on privacy policies.

Access policies describe access rights of subjects on resources, and conditions compositing of attributes of subject, action, resource, and environment as well as obligations that are instructions from *Policy Decision Point* to *Policy Enforcement Point* to be performed before or after data results is returned to the requester.

Privacy policies describe access restriction on data objects which need to be preserved privacy. The structure of privacy policies is similar to access policies. It also includes subject, action, resource, environment, obligations and attribute based condition. However, each component contains the slightly different meaning. The components of subject, action, resource, environment in privacy policies indicate attribute based conditions and the component of obligation contains *generalization functions* applied on values of data objects for privacy protection.

In privacy preserving access control, the purpose concept plays an important role in privacy policies to describe a valid reason for data usage. When users send a request to query data, they must provide their access purpose to the system. The access purpose is then verified whether the subject is permitted for using it in the access policy authorization stage. In our model, this value, access purpose, is considered as an attribute of the environment entity. However intended purpose is not implicitly mentioned, it is described through a conditional expression based on the access purpose attribute identifying which values are valid for data usage and generalization (Fig. 3).

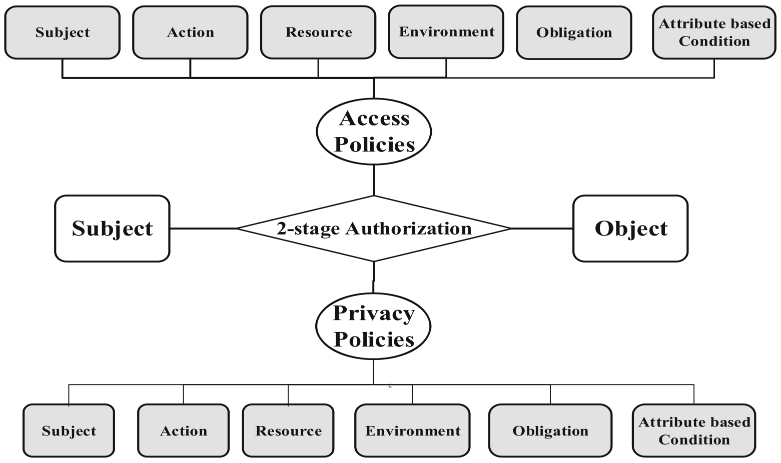


Fig. 3. The components of ABAC model for privacy protection

4.1 Policy Structure

In this section, we present a general structure used for both of access and privacy policies. However, for each use case, we will describe how to specify policies through examples. The mechanism of processing policies works slightly differently.

General Structure. In our model, a *policy set* includes *policies*. Each *policy* includes *rules*. Each *rule* defines a conditional expression that is a critical component in the policy. The rule returns a value specified in *Effect* if the condition is true. The *target* component, including three sub components *Subject*, *Action* and *Resource*, is used to pre-select applicable policies for access decision. To avoid conflicts between policies and rules, the policy and rule combining algorithms such as *permit override*, *deny override*, etc. are applied into the policy set and policies. The implementation for rule combining algorithm is inherited from XACML [26]. The final component in a rule is *obligations* indicating actions which will be performed before or after a final response is established by *Policy Enforcement Point*. The relationship diagram between policy set, policies and rules are illustrated in Fig. 4. The structure of a policy in the JSON format is illustrated in Fig. 5. Its properties are described in details in Table 1.

In the above structure, the *condition* component <*conditionalExpr*> is written according to the below grammar (Fig. 6):

It can be seen that the operands in the condition expression are attributes from *Subject, Action, Resource, ResourceContent* and *Environment* or specific values. The values of attributes are loaded from the *request context*. For missing values, *JABAC*

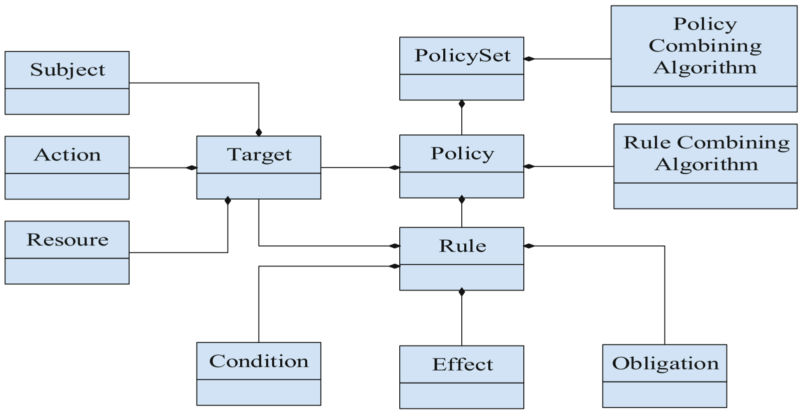


Fig. 4. The relationship diagram of components in policy set

```
PolicySet {
  PolicyCombiningAlgID: <CombiningPolicyAlgID>,
  Policies: [{
      PolicyID: <policyID>,
        RuleCombiningAlgID: <CcombiningRuleAlgID>,
        Target: {
          Subject: <subjectID>,
          Action: <actionID>,
          Resource: <resourceName>},
      Rules: [{
          RuleID: <ruleID>,
          Target: {
            Subject: <subjectID>,
            Action: <actionID>,
            Resource: <resourceName>},
          Condition: <conditionalExpr>,
          Effect: <"permit" or "deny">,
          Obligations: [{
            FunctionID: <functionName>,
            Parameters: [{
               ParaValue: <attributeName>|<specificValue>,
               SourceType: <entityName>|Null,
               DataType: <dataType>}],
              FullFillOn: <"permit" or "deny">,
            Directive: <"before" or "after">}]
           }]}]}
```

Fig. 5. The JSON structure of policy

Table 1. The description of the fields in a policy

Level	Field name	Type	Description
PolicySet	PolicyCombiningAlgID	String	Indicate which combining algorithm is used to combine the results of evaluating policies in the policyset
PolicySet. Policy	Policies	Array of documents	Contain a list of policies
PolicySet. Policy	PolicyID	String	Indicate policy code
PolicySet. Policy	RuleCombiningAlgID	String	Indicate which combining algorithm is used to combine results of rule evaluation in a policy
PolicySet. Policy	Target	Embedded document	Indicate which subject, action, resource is applied for the policy
PolicySet. Policy.Target	Subject	String	Indicate ID of subject or ANY
PolicySet. Policy.Target	Action	String	Indicate ID of action or ANY
PolicySet. Policy.Target	Resource	String	Indicate name of resource or ANY
PolicySet. Policy	Rules	Array of embedded documents	Contain a list of rules in each policy
PolicySet. Policy.Rules	RuleID	String	Indicate rule code
PolicySet. Policy.Rules	Target	Embedded document	Indicate which subject, action, resource is applied for a rule. Similar to PolicySet.Policy. Target
PolicySet. Policy.Rules	Condition	String	Contain a Boolean expression. If the condition is evaluated as true, the rule returns the value of the effect property
PolicySet. Policy.Rule	Effect	String	Indicate whether the rule returns permit or deny if the condition is evaluated as true
PolicySet. Policy.Rule	Obligations	Array of embedded documents	Contain a list about obligations which will be executed after evaluating policies
PolicySet. Policy.Rule. Obligations	FunctionID	String	Contain the function name which will be called to execute for an obligation

(*continued*)

Table 1. (*continued*)

Level	Field name	Type	Description
PolicySet. Policy.Rule. Obligations	Parameters	Array of embedded documents	Contain a list of parameters of the function defined in FunctionID, including ParaValue, SourceType, and ParaType
PolicySet. Policy.Rule. Obligations. Parameters	ParaValue	Object	Contain the value for the parameter
PolicySet. Policy.Rule. Obligations. Parameters	SourceType	String	If the ParaValue is a certain attribute name, SourceType contains the name of source of attribute to get value for that attribute. Example, ParaValue is "subjectName" and SourceType is "Subject". If SourceType has Null value, the ParaValue field contains a specific and direct value
PolicySet. Policy.Rule. Obligations. Parameters	ParaType	String	The name of data type of ParaValue. This is used to convert into the data type of the corresponding argument declared in the function
PolicySet. Policy.Rule. Obligations	FullFillOn	String	Contain the value of *permit* or *deny* which indicates the case of executing an obligation. The *Permit* value means that the obligation will be executed when the final result is Permit and vice versa for the Deny value
PolicySet. Policy.Rule. Obligations	Directive	String	Indicate when the obligation is executed, namely, before or after data results are returned to the subject

will look up from database to fulfill the request context. More details will be presented in Sect. 5.

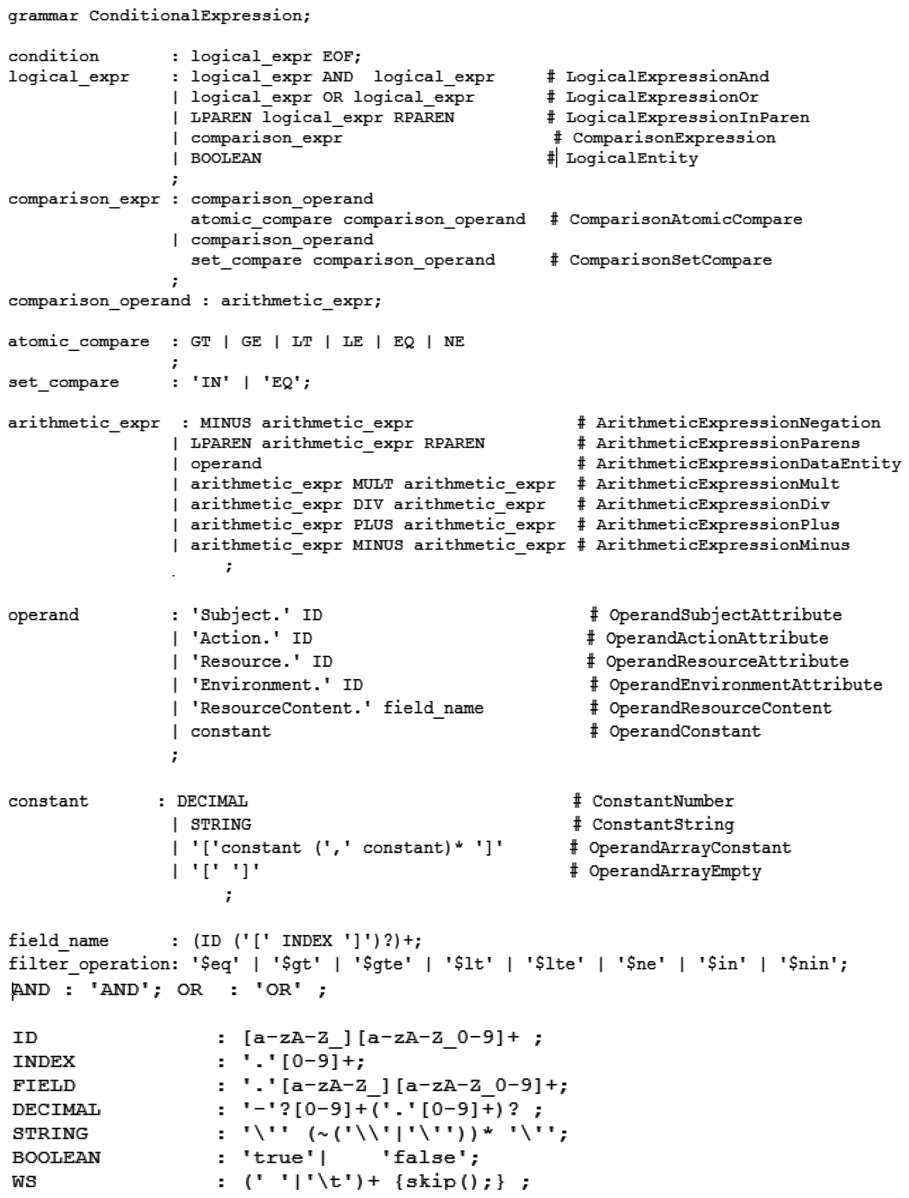
A below example for a rule demonstrates the policy structure and the grammar of conditional expression:

Doctors can read their patient records with the purpose of treatment. If the request is denied, the system will email to the administrator John about the subjectID and the access purpose of subject.

```
RuleID: "ARU001",
Target: {
    Subject: "ANY",
    Action: "read",
    Resource: "patients"},
Condition: "Subject.role = 'doctor' AND
           ResourceContent.doctorID = Subject.subjectID
          AND Environment.accessPurpose = 'treatment'",
Effect: permit,
Obligations:[{
    FunctionID: "email",
    Parameters: [{
         ParaValue: "john@gmail.com",
         SourceType: Null,
         DataType: "String"},
        {ParaValue: "subjectID",
         SourceType: "Subject",
         DataType: "String"}],
    FullFillOn: "deny",
    Directive: "after"}]
```

With the above rule, a doctor can read only patient records if he is a doctor and his *subjectID* equals to the *doctorID* in each patient record. By obligations, if the request is denied, the information of *subjectID* will be sent to *john@gmail.com* by calling the function *email* after the response is returned to the requester. The attributes *role, subjectID* of subject, the attribute *accessPurpose* of environment, and the attribute *doctorID* of resource content keep values in the *request context*. If they cannot be found, the system will call *Policy Information Point* to look up them in the *application database* and the *JABAC database*. If an error occurs or some values are missing, the rule will return *Indeterminant*. If the target component does not match with the information in the request context, the rule will return *NotApplicable*. To produce the final result, the mechanism will take advantage of the rule and policy combing algorithms specified in the policy to make a decision when evaluating rules and policies in the loop. The structure of *request context* and details of the mechanism will be presented in Sects. 4.2 and 5 respectively.

When a request is evaluated as permit, it does not ensure that all data corresponding to the request will be accessed by the subject. For example, Alice with the role of doctor can read the records of her patients according to the above rule. However, due to privacy, Bob, one of her patients, only wants his information about address, social security number, and birthdate to be partially shown to his doctor. Thus, in our proposed model, Bob can define *privacy policies* to protect his data. However, it can appear special cases defined by the highest security administrative to bypass his privacy policies such as standard regulations of the organization. In this paper, such delegation problem has not been mentioned yet.



```
grammar ConditionalExpression;

condition       : logical_expr EOF;
logical_expr    : logical_expr AND  logical_expr     # LogicalExpressionAnd
                | logical_expr OR logical_expr       # LogicalExpressionOr
                | LPAREN logical_expr RPAREN         # LogicalExpressionInParen
                | comparison_expr                     # ComparisonExpression
                | BOOLEAN                            # LogicalEntity
                ;
comparison_expr : comparison_operand
                  atomic_compare comparison_operand  # ComparisonAtomicCompare
                | comparison_operand
                  set_compare comparison_operand     # ComparisonSetCompare
                ;
comparison_operand : arithmetic_expr;

atomic_compare  : GT | GE | LT | LE | EQ | NE
                ;
set_compare     : 'IN' | 'EQ';

arithmetic_expr  : MINUS arithmetic_expr               # ArithmeticExpressionNegation
                | LPAREN arithmetic_expr RPAREN        # ArithmeticExpressionParens
                | operand                              # ArithmeticExpressionDataEntity
                | arithmetic_expr MULT arithmetic_expr  # ArithmeticExpressionMult
                | arithmetic_expr DIV arithmetic_expr   # ArithmeticExpressionDiv
                | arithmetic_expr PLUS arithmetic_expr  # ArithmeticExpressionPlus
                | arithmetic_expr MINUS arithmetic_expr # ArithmeticExpressionMinus
                    ;

operand         : 'Subject.' ID                          # OperandSubjectAttribute
                | 'Action.' ID                           # OperandActionAttribute
                | 'Resource.' ID                         # OperandResourceAttribute
                | 'Environment.' ID                      # OperandEnvironmentAttribute
                | 'ResourceContent.' field_name          # OperandResourceContent
                | constant                               # OperandConstant
                ;

constant       : DECIMAL                                 # ConstantNumber
                | STRING                                 # ConstantString
                | '['constant (',' constant)* ']'       # OperandArrayConstant
                | '[' ']'                                # OperandArrayEmpty
                    ;

field_name      : (ID ('[' INDEX ']')?)+;
filter_operation: '$eq' | '$gt' | '$gte' | '$lt' | '$lte' | '$ne' | '$in' | '$nin';
AND : 'AND'; OR  : 'OR' ;

ID                  : [a-zA-Z_][a-zA-Z_0-9]+ ;
INDEX               : '.'[0-9]+;
FIELD               : '.'[a-zA-Z_][a-zA-Z_0-9]+;
DECIMAL             : '-'?[0-9]+('.'[0-9]+)? ;
STRING              : '\'' (~('\\'|'\''))* '\'';
BOOLEAN             : 'true'|    'false';
WS                  : (' '|'\t')+ {skip();} ;
```

Fig. 6. The grammar of conditional expression

Privacy Policies. They have the same structure with the general one but it is slightly different about the use of components. The obligations in privacy policy play the role of expressing *how to generalize* the value of data item to protect privacy. In our mechanism, an obligation in privacy policies associates with the special function *MakeGeneralization (field name, data disclosure level)* for hiding details of data.

It takes two parameters; *the name of the field* which needs to be protected and *the level of data disclosure* for privacy preserving. The definition of *data disclosure level* is presented as below.

Data Disclosure Level (DL). DL of data item represents the level of data generalized in the *Domain Generalization Hierarchy* (DGH). Based on DL, data are generalized into a new value according to *Value Generalization Hierarchy* (VGH) generated from DGH. The concepts of VGH and DGH are explained as follows: Each attribute has a range of values designated by a domain. For data privacy preserving, the generalization process is applied to the value domain of the attribute and establishes a *DGH tree*. Each value in the domain contains many generalized value in generalized domains. A set of these values with the order of generalization process establishes a *VGH tree*. The formal definitions for DGH and VGH tree can be seen in [24].

Figure 7 describes a *DGH tree* and *VGH tree* for the attribute domain *Birthdate*. In this example, the number of data disclosure levels of birthdate is five, in which the smallest number (DL0) indicates the data does not require any privacy protection, whereas the highest number indicates the data will be hidden with the keyword "*".

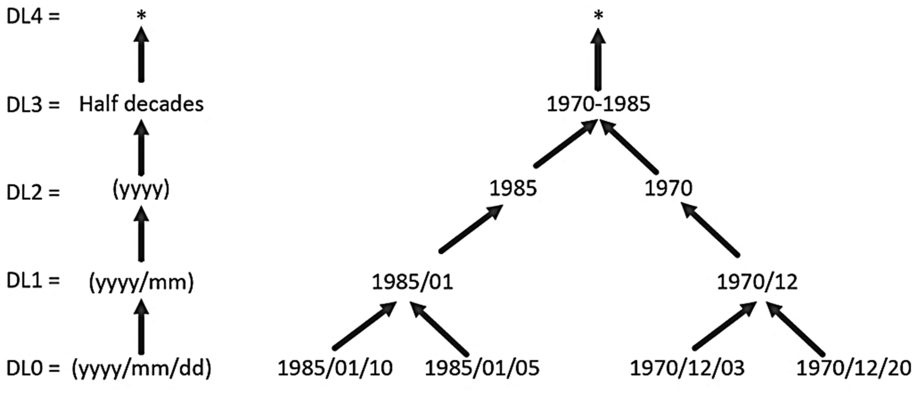


Fig. 7. Domain generalization hierarchy tree

In summary, to specify policies for privacy protection, *access purpose* is defined as an attribute of environment; obligations is modeled as an indicator for data generalizations. For example, the below privacy rule PRU001 indicates "The information about birthdate and social security number of Bob who has the *patientID* 10001 are generalized according to the following levels: 2 (only displaying the year of birthdate), and 1 (not displaying any information of the string) respectively when any subject read the patient record of Bob with the access purpose of treatment". Thus, the rule PRU001 is specified as follows:

The components such as *Effect* and *FullFillOn* will be discarded because they have no meaning in privacy policies. The *PurJABAC* module retrieves privacy policies and applies the *MakeGeneralization* function for related fields in each document in ResourceContent.

```
  RuleID: "PRU001",
  Target: {
      Subject: ANY,
      Action: read,
      Resource: patients},
  Obligations: [
     {FunctionID: "MakeGeneralization",
      Parameters: [
         {ParaValue: "Birthdate",
          SourceType: "ResourceContent",
          DataType: "String"},
         {ParaValue: 2,
          DataType: "Integer"}],
      FullFillOn:  Null},
     {FunctionID: "MakeGeneralization",
      Parameters: [
         {ParaValue: "SSN",
          SourceType: "ResourceContent",
          DataType: "String"},
         {ParaValue: 1,
          SourceType: Null,
          DataType: "Integer"}],
      FullFillOn:  Null}],
  Condition: "ResourceContent.patientID = '10001' AND
Environment.accessPurpose = 'treatment'",
  Effect: null
```

Take an example that Alice, who has the subjectID *20001*, sends a request to database to read her patient records with the access purpose "treatment". Alice is allowed but information about birthdate and social security number of Bob is generalized by PRU001. For instance, Table 2 shows patient records in the database. There are only two records of Bob and Kitty which Alice can see due to the rule *ARU001*. Besides, the information of Bob displays only the year of his birthdate and his ssn with the special character "*" due to the rule *PRU001* (ref. Table 3). In the next section, the structure of the request context is presented in details.

Table 2. An example of patient records

patientID	patientName	birthdate	ssn	doctorID
10001	Bob	1/13/1990	12345789	20001
10002	Paul	12/10/1980	12345999	20002
10003	Kitty	3/13/1970	12345777	20001

Table 3. An example of patient records returned to alice

patientID	patientName	birthdate	ssn	doctorID
10001	Bob	*1990*	*	20001
10003	Kitty	3/13/1970	12345777	20001

4.2 The Request Context Structure

The structure of a *request context* is very important for the *JAPAC* mechanism. A *request context* contains the components such as *Subject, Resource, Action* and *Environment*. The below example demonstrates the structure of the *request context*. Each

```
Subject: {
    SubjectID: "Alice",
    Attributes: [{
                AttributeID: "Role",
                AttributeType: "String",
                Value: "doctor"}]},
Resource: {
    ResourceID: "db.Patients",
    ResourceRequest: "db.Patients.find({doctorID:
                     '20001'})",
    ResourceContent: [
        { patientID : "10001",
          patientName: "Bob",
          birthDate:"1/13/1990",
          ssn:"12345789",
          doctorID:"20001"},
        { patientID : "10003",
          patientName: "Kitty",
          birthDate:"3/13/1970",
          ssn:"12345777",
          doctorID:"20001"}],
    Attributes: [{
                AttributeID: "SelectedFields",
                AttributeType: "Array",
                Value: "['patientID',
     'patientName', 'birthDate' 'ssn', 'doctorID']"}]},
Action: {
    ActionID: "READ",
    Attributes: []}
Environment: {
    EnvironmentID: Null,
    Attributes: [{
                AttributeID: "WorkingTime",
                AttributeType: "DateTime",
                Value: "14:30"}]}
```

part contains ID and its attributes. Especially, the *Resource* component includes *ResourceContent* which contains data queried from the original request and inserted by *Policy Enforcement Point* into the request context.

In our model, *JABAC* receives a request which must comply with the designed structure in the JSON format and contains information about the request such as subject, action, resource and environment. The system then analyzes the request and evaluates whether it is permitted or denied. After evaluating, *JABAC* calls the *PurJABAC* model to filter data for privacy protection.

In this section, the structure of components has been introduced in details. However, the mechanism for enforcing such policies has not been mentioned yet. In the next section, we present the mechanism of evaluating access control policies and processing data based on privacy policies.

5 The JABAC Mechanism

In this work, we utilize the concepts in XACML such as *Policy Enforcement Point* (PEP), *Policy Decision Point* (PDP), *Policy Information Point* (PIP), *Policy Administration Point* (PAP) and *Obligations*. PEP takes responsibility for receiving requests from applications, converting into request contexts in the JSON format, sending them to PDP for evaluating whether they are permitted or not, filtering and hiding data according to privacy policies and finally executing obligations of related access control policies. PDP receives the JSON request contexts to check access rights with access control policies. PIP has the feature of collecting information of attributes if PDP cannot find them in the request contexts. PAP represents the module of specifying policies. Obligation is the module of executing obligation functions of access control policies. With obligations specified in privacy policy, the PurJABAC module has a different mechanism to enforce. The below section will describe the data flow of our model.

The main processes in the data flow of our model depicted in Fig. 8 are described as follows:

1. **PEP** receives the access request consisting of the components: *subject, action*, and *resource*.
2. **PEP** creates another request, called *request context,* for policy decision from the access request fulfilled with the attributes of subject, action, resource, and environment and then sends to **PDP** for access authorization.
3. **PDP** retrieves the list of access policies from database.
4. For each policy, **PDP** checks whether the *target* element of the policy (i.e. *subject, action, resource*) matches with the corresponding components of the request context by the *Target Matching* module. If it returns "*successfully matching*", all rules of this policy are examined. Rules satisfying the *target* component will be processed in the next step.

5. In this step, the *condition* component of applicable rules is evaluated by the module **ConditionalExpr Parser and Evaluator**. **ConditionalExpr Parser and Evaluator** uses the open source *ANTLR*[3] with the grammar presented in Sect. 4.1 to evaluate the expression.
6. The module **Condition Parser and Evaluation** sends requests to *Policy Information Point* (**PIP**) to retrieve values for operands.
7. **PIP** collects values from the request context and database; and then sends them to **ConditionalExpr Parser and Evaluator** for expression evaluation.
   Depending on the combining rule algorithm specified in the component PolicyCombiningAlgID in the current policy, PDP will continue to check the next access rule (e.g. permit overrides) or terminate with the result deny (e.g. deny overrides). Similarly, depending on the combining algorithms for policies specified in the component RuleCombiningAlgID of PolicySet, PDP will terminate together with the result of policy evaluation or keep on checking with other applicable policies.
8. After evaluating all applicable policies from step 4 to step 7, **PDP** returns the response to **PEP**. The response can be *permit* or *deny.*
9. In the case of permit, **PEP** asks the **PurJABAC** module for filtering and hiding data in *ResourceContent* of the request context.
   To generalize data for privacy protection, **PurJABAC** retrieves privacy policies which their *target* component matches with the request context and then read data documents in *ResourceContent*. For each document, **PurJABAC** selects the set of privacy policies which can be applied to the current document and then computes the highest disclosure level for each field name from those privacy policies. Take the example in Sect. 4; we assume that there have two privacy rules for Bob and one privacy rule for Kitty. The first one is PRU001 described above and the second one is PRU002 which allows showing the year and the month of birthdate (the lower level than PRU001) in the case of treatment purpose. Thus, PRU001 wants to generalize the birthdate of Bob into the year level meanwhile the birthdate of Kitty is not influented by PRU001. Therefore, the **PurJABAC** module chooses the highest disclosure value, DL02 for Bob and DL01 for Kitty to generalize their birthdate.
10. In the case *ResourceContent* has not been fulfilled in the request context due to no prior policy requires resource data, the **PurJABAC** module will send the request to **PIP** for querying resource content.
11. After checking all objects in the result set, PEP calls *Obligation Services* to perform obligations. The obligations are executed before or after PEP returns results to the requester. If any function in obligations returns an exception, PEP will not send data to the requester however the access is permitted. For example, an obligation requires the requester to accept terms and conditions. If she/he refuses to perform this obligation, her request is denied.
12. **PEP** returns data results to the requester.

[3] www.antlr.org.

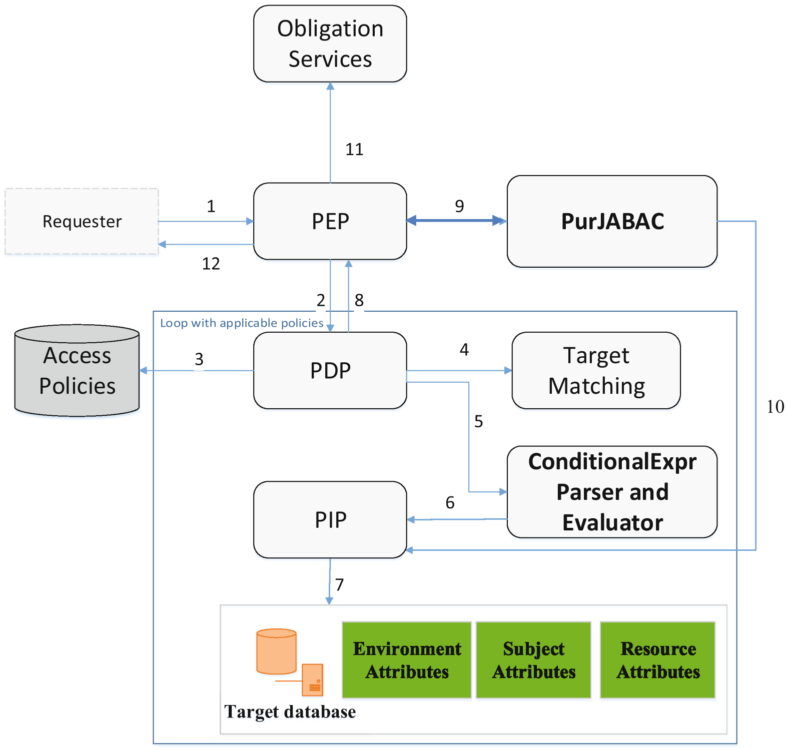


Fig. 8. The data flow diagram of the proposed access control model

In general, the components of our model and its mechanism are presented in this paper. With our best knowledge, no research work has integrated PBAC into ABAC. Besides, we extend the fine grained feature for access control policies through the conditional expression specified by our grammar and for privacy policies through the generalization function with various data disclosure levels.

6 Evaluation

We carried out experiments about the relevance between processing time of the PDP module and complexity of rules. The system configuration for the experiments is Dell Vostro 3650, 8 GB RAM, Intel core i5-3230M 2.60 GHz. The prototype is implemented by Java SDK, Spring Framework and MongoDB 3.0.7 for storing policies and data. The target database includes 20 collections with 20 attributes and 200 documents for each collection generated randomly. Each subject, action, resource and environment contains 20 attributes.

The following Table 4 indicates the results after five experiments. For each experiment, we measure processing time with three times (e.g. T1, T2 and T3). From the Table 1, it can be seen that the processing time increases with the complexity of

Table 4. The Results of Experiments

ID	Number of rules	Logical expressions in each rule	Arithmetic expressions in each rule	T1 (ms)	T2 (ms)	T3 (ms)
1	1	1	1	5	4	4
2	1	10	10	8	8	7
3	10	10	10	12	11	12
4	50	10	10	15	14	14
5	100	10	10	17	17	16

rules. However, when there was a fivefold and twofold increase in the number of rules from 10 to 50 and from 50 to 100, the processing time only increased by 2–3 ms, approximately 13–25%.

About the fine granularity, our attribute based access control model can describe various attribute based policies due to the flexibility of conditional expressions built under the proposed grammar. Compared to other approaches, our model can specify policies with the conditions based on the combination of attributes of subject, resource, and environment. Besides, by using JSON, our model can easily integrate with Web 2.0 applications as well as devices and systems in Internet of Things.

7 Conclusion and Future Work

In this paper, we have proposed the fine grained attribute and purpose based access control model *JABAC* for privacy protection with the mechanism of 2-stage authorization. A conditional expression based on attributes of subject, resource, action and environment are built on the ANTLR grammar, which is enough to describe various policies. Besides, an additional module *PurJABAC* is integrated into JABAC for privacy protection. In our approach, privacy policies are specified under the same structure of attribute based policies. Purposes are associated with the new concept, disclosure level, for indicating the degree of generalization for privacy protection. They are demonstrated as obligations in privacy policies which call *MakeGeneralization* functions to generalize data.

However our model takes a novel approach in the research field of attribute access control and purpose based access control, there have still some disadvantages. In future, we will improve the grammar for conditional expressions to describe more complex policies which can retrieve data from multiple sources not only from the request context. Furthermore, the solution currently supports the data which are not allowed many levels of embedded documents. Therefore, the complexity of data will be investigated as well. Besides, the problems such as policy review and administration have not been mentioned yet in this article. They will be promising research problems in the future.

Acknowledgements. This research is funded by Vietnam National University Ho Chi Minh City (VNU-HCM) under grant number C2017-20-11.

References

1. Bertino, E., Ghinita, G., Kamra, A.: Access Control for Databases: Concepts and Systems. Now Publishers, Hanover (2011)
2. Hu, V.C., Ferraiolo, D., Kuhn, R., Friedman, A.R., Lang, A.J., Cogdell, M.M., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K.: Guide to Attribute Based Access Control (ABAC) definition and considerations (draft). NIST Special Publication, 800, 162 (2013)
3. Hu, V.C., Kuhn, D.R., Ferraiolo, D.F.: Attribute-based access control. Computer **2**, 85–88 (2015)
4. Jin, X., Krishnan, R., Sandhu, R.: A Unified attribute-based access control model covering DAC, MAC and RBAC. In: Cuppens-Boulahia, N., Cuppens, F., Garcia-Alfaro, J. (eds.) DBSec 2012. LNCS, vol. 7371, pp. 41–55. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31540-4_4
5. Sandhu, R.: The Future of access control: attributes, automation, and adaptation. In: Krishnan, G.S.S., Anitha, R., Lekshmi, R.S., Kumar, M.S., Bonato, A., Graña, M. (eds.) Computational Intelligence, Cyber Security and Computational Models. AISC, vol. 246, p. 45. Springer, New Delhi (2014). https://doi.org/10.1007/978-81-322-1680-3_5
6. Westin, A.F.: Privacy and Freedom. Atheneum, New York (1967)
7. Byun, J.-W., Bertino, E., Li, N.: Purpose based access control of complex data for privacy protection. In: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies (2005)
8. Byun, J.W., Li, N.: Purpose based access control for privacy protection in relational database systems. VLDB J. **17**(4), 603–619 (2008)
9. Kabir, M.E., Wang, H.: Conditional purpose based access control model for privacy protection. In: Proceedings of the Twentieth Australasian Conference on Australasian Database, vol. 92, pp. 135–142. Australian Computer Society, Inc. (2009)
10. Wang, H., Sun, L., Bertino, E.: Building access control policy model for privacy preserving and testing policy conflicting problems. J. Comput. Syst. Sci. **80**(8), 1493–1503 (2014)
11. Kabir, M.E., Wang, H., Bertino, E.: A role-involved conditional purpose-based access control model. In: Janssen, M., Lamersdorf, W., Pries-Heje, J., Rosemann, M. (eds.) EGES/GISP 2010. IFIP AICT, vol. 334, pp. 167–180. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15346-4_13
12. Kabir, M.E., Wang, H., Bertino, E.: A conditional purpose-based access control model with dynamic roles. Expert Syst. Appl. **38**(3), 1482–1489 (2011)
13. Ni, Q., Lin, D., Bertino, E., Lobo, J.: Conditional privacy-aware role based access control. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 72–89. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74835-9_6
14. Ni, Q., Bertino, E., Lobo, J., Brodie, C., Karat, C.M., Karat, J., Trombeta, A.: Privacy-aware role-based access control. ACM Trans. Inf. Syst. Secur. (TISSEC) **13**(3), 24 (2010)
15. Colombo, P., Ferrari, E.: Enforcement of purpose based access control within relational database management systems. IEEE Trans. Knowl. Data Eng. **26**(11), 2703–2716 (2014)
16. Colombo, P., Ferrari, E.: Enhancing MongoDB with purpose based access control. IEEE Trans. Depend. Secur. Comput. (2015, will appear)
17. Colombo, P., Ferrari, E.: Efficient enforcement of action-aware purpose-based access control within relational database management systems. IEEE Trans. Knowl. Data Eng. **27**(8), 2134–2147 (2015)
18. Pervaiz, Z., Aref, W.G., Ghafoor, A., Prabhu, N.: Accuracy-constrained privacy-preserving access control mechanism for relational data. IEEE Trans. Knowl. Data Eng. **26**(4), 795–807 (2014)

19. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. (TISSEC) **4**(3), 224–274 (2001)
20. Fuchs, L., Pernul, G., Sandhu, R.: Roles in information security–a survey and classification of the research area. Comput. Secur. **30**(8), 748–769 (2011)
21. Kuhn, D.R., Coyne, E.J., Weil, T.R.: Adding attributes to role-based access control. IEEE Comput. **43**(6), 79–81 (2010)
22. Huang, J., Nicol, D.M., Bobba, R., Huh, J.H.: A framework integrating attribute-based policies into role-based access control. In: Proceedings of the 17th ACM symposium on Access Control Models and Technologies, pp. 187–196. ACM (2012)
23. Rajpoot, Q.M., Jensen, C.D., Krishnan, R.: Attributes enhanced role-based access control model. In: Fischer-Hübner, S., Lambrinoudakis, C., Lopez, J. (eds.) TrustBus 2015. LNCS, vol. 9264, pp. 3–17. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22906-5_1
24. Sweeney, L.: Achieving k-anonymity privacy protection using generalization and suppression. Int. J. Uncertain. Fuzziness Knowl. Based Syst. **10**(05), 571–588 (2002)
25. Ni, Q., Bertino, E., Lobo, J.: An obligation model bridging access control policies and privacy policies. In: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, pp. 133–142 (2008)
26. Rissanen, E.: eXtensible Access Control Markup Language (XACML) version 3.0 (committe specification 01). Technical report, OASIS (2010). http://docs.oasisopen.org/xacml/3.0/xacml-3.0-core-spec-cd-03-en.Pdf
27. Nurseitov, N., et al.: Comparison of JSON and XML data interchange formats: a case study. In: Caine 2009 (2009)
28. Servos, D., Osborn, S.L.: Current research and open problems in attribute-based access control. ACM Comput. Surv. (CSUR) **49**(4) (2017)
29. Ferraiolo, D., et al.: Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC). In: Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control (2016)
30. Thi, Q.N.T., Si, T.T., Dang, T.K.: Fine grained attribute based access control model for privacy protection. In: Dang, T.K., Wagner, R., Küng, J., Thoai, N., Takizawa, M., Neuhold, E. (eds.) FDSE 2016. LNCS, vol. 10018, pp. 305–316. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48057-2_21