Towards a Fine-Grained Access Control Mechanism for Privacy Protection and Policy Conflict Resolution

Ha Xuan Son[1]
FPT University
Can Tho city, Viet Nam

En Chen[2]
National Taiwan Normal University
Taipei, Taiwan

Abstract—**Access control is a security technique that specifies access rights to resources in a computing environment. As information systems nowadays become more complex, it plays an important role in authenticating and authorizing users and preventing an attacker from targeting sensitive information. However, no proper consideration has been fully investigated so far in privacy protection. While many studies have acknowledged this issue, recent studies have not provided a fine-grained access control system for data privacy protection. As the data set becomes larger, we have to confront more privacy challenges. For example, the access control mechanism must be able to guarantee fine-grained access control, privacy protection, conflicts and redundancies between rules of the same policy or between different policies. In this paper, we propose a comprehensive framework for enforcing attribute-based security policies stored in the JSON document together with the feature of data privacy protection and incorporates a policy structure based on the prioritization of functions to resolve conflicts at a fine-grained level called "Privacy aware access control model for policy conflict resolution". We also use Polish notation for modeling conditional expressions which are the combination of subject, action, resource, and environment attributes so that privacy policies are flexible, dynamic and fine-grained. Experiments are carried out to two aspects (i) illustrate the relationship between the processing time for access decision and the complexity of policies; (ii) illustrate the relationship between the processing time for the traditional approach (single policy, multi-policy without priority) and our approach (multi-policy with priority). Experimental results show that the evaluation performance satisfies the privacy requirements defined by the user.**

Keywords—ABAC; privacy; JSON; policy conflict resolving; document store; fine-grained access control

I. Introduction

The remarkable growth of Internet and social media applications over the past few decades lead to an exponential increase of data. By capturing and analyzing these data, enterprises obtain a better understanding about their customers, leading to better business decisions. However, with a vast amount of information available on the Web, it is required a database system capable of storing and retrieving of data in a well-structured way. Currently, NoSQL database is the most popular approach to handle those semi and unstructured data for a scalable application. As in other relational databases, security must be highly considered as it has to process large volumes of data. For the last decade, many models e.g.Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role Based Access Control (RBAC) has been proposed to handle security problems. These traditional approaches are effective in a small-scale system; however, in a large scale dynamic systems, they experience some serious problems such as role explosion, inflexibility in specifying dynamic policies and contexture conditions [1]. To overcome those issues, Attribute Based Access Control (ABAC) model has been investigated. The model grants access to a request only if it satisfies conditions on attributes of subject, resource and environment specified in policies [2]. With declarative mechanism to specify access permission, ABAC has proved its effectiveness on complex systems than RBAC with a fixed mechanism.

Although access control systems are successful in the prevention of unauthorized accesses and malicious users, they are ineffective in privacy protection for a large, decentralized system such as social network and Internet of Things. Our concentration in this work; therefore, aims at developing a system that is able to grant access control while providing effective privacy protection.

Notwithstanding the ABAC model has proved its effectiveness on complex systems with its declarative mechanism, it is worth to note that the model assumes that all policies can be trusted. In other words, the correctness of all attributes and policies must be guaranteed. Moreover, since a complexity system usually managed by several administrators, conflicts can occur between rules of the same policy or between different policies. Therefore, in the case of conflicts among policies, the ABAC encounters problems in effectively detecting and resolving them. In reality, the scale of policies with varying level of privacy protection has led to an increasing risk of policies conflicting with each other. Moreover, for a particular system, there might be more than one administrator. As a result, each one may define different rules that contradict with others. In the worst case scenario, the policy set may permit unauthorized access; furthermore, those collisions may cause a denial of service for legal access. Therefore, it is required to develop a system capable of detecting conflicts in a policy and between policies and mitigating their effect in order to preserve the privacy protection.

To investigate the problem of conflict resolution, we introduced an ABAC system that incorporates a policy model based on the prioritization of functions to resolve conflicts at a fine-grained level. It allows the user to prioritize different functions that presented on the same domain from the lowest storage unit (fields) to the highest storage unit (as collection or database). This is the advantage of the solution compared to normal approaches: instead of returning decisions as `Permit` or `Deny`, we create a smooth resolution mechanism that can show a portion of the requested data based on the priority level

of the requester. Furthermore, our model supports complex polices presenting in a hierarchical structure which may include several sub-policy elements.

To investigate the all issues addressed, we have proposed a flexible model structure for privacy protection supporting conflict policy resolution called *Attribute-based Access Control model for fine-grained privacy protection*. The model evaluates a request not only by its access purpose but also by subject, action, resource, environment attributes and function defined by users. To describe complex policies containing information of user, action, resource, environment, and driven policies, Polish notation is used for modeling conditional expressions. We also build an implementation based on MongoDB which stores the policy and database of the system. Generally, the requests and policies are defined in JSON format where administrators and users can easily define policies and requests. The contribution of this article is four-fold: (i) we proposed an attribute-based security policies definition formatting in JSON; (ii) we describe a mechanism for protecting sensitive data in fine-grained level; (iii) we presented a dynamic solution for fine-grained policy conflict; and (iv) we used Polish notation for modeling conditional expressions.

This paper is organized as follows. In the second section, we briefly review related works. Section 2 describes our proposed model and how it handles both access control and privacy protection. Section 3 presents the privacy-aware access control policies including policy structure and policy decision mechanism. Section 4 illustrates our sample scenario and how our proposed model handles conflict in policy levels. Section 5 then describes our experimental designs and discusses the results. Finally, Section 6 presents our conclusions and future works.

II. Related Works

A. Privacy Protection in Access Control Model

Most of the works in the literature focus on two directions: (i) constructing a whole new privacy-aware access control system based on ABAC model; and (ii) adding a level of privacy protection to a popular existence standard. Following the first trend, Hua Wang et al.[3] proposed a purpose-based framework for supporting privacy preserving access control policies and mechanisms. In this framework, the key component is a set of purpose-based access control policies that provide privacy protection by taking into account some important features (purposes and conditions). In addition, conflicting algorithm is also developed to detect and analyze conflicts between policies. However, the way to model conditional expression is not clearly described; moreover, the conflicting algorithm only focused on simple attributes which are not properly evaluated with conditional expressions on them. Prosunjit Biswas et al. presented an attribute based protection model for JSON elements documents in [4]. To perform security protection, each JSON element is assigned a new attribute called “security label” which is used to define the access control policies. A benefit of this separation of labeling and authorization policies procedure is that each element can be specified and administered independently and possibly by different level of administrators. As a result, the privacy protection is done for each element of the database management systems (DBMSs). A drawback of this method might come from a huge number of labels needed to be assigned since the total number is growing exponentially.As a consequence, the process is time-consuming while requiring a large space storage when the system is expanded.

In the second research direction, most of the studies focus on improving the privacy protection for the popular ABAC standards, `eXtensible Access Control Markup Language` (XACML). Claudio A. Ardagna et al.in [5] proposed a system that extend the traditional XACML architecture with a combination with PRIME, a solution supporting privacy-aware access control. As a result, the system provides a flexible access control functionality of XACML with the data governance and privacy features of PRIME. In detail, the system has two main blocks: (i) PrimeLife XACML Engine is responsible for granting access control and (ii) Data Handling Decision Function (DHDF) is in charged of privacy and data handling functionalities. When an access request is needed to be considered, the request is forwarded to both blocks. The final decision is taken by combining the access control process and the DHDF data handling evaluation process. Only if a request comes from an authorized users that satisfied both evaluation procedure, it will be granted access to the required data.

Another study based on XACML is presented in [6], [7], [8], [9]. In this work, a system which inserts privacy policies in access control solution to NoSQL database is developed and tested on MongoDB. The main component responsible for privacy protection is called `Access control as a service solution` (ACCAAS).Administrators can store access control policies in ACCAAS solution for each element. When a request is forwarded to the ACCAAS system, it decides whether or not that user is authorized. If yes, then it sends a request to the MongoDB system asking for the required data. If not, it would return a “Deny” to the requester.

The biggest advantage of these solutions is that they support privacy protection on each element of the DBMSs. Moreover, they are easily integrated with XACML policies which is a widely-used standard in real world and considered many parameters for granted access at the same time (e.g.purpose, obligation, user information, etc.). However, for each request, since it is processed paralelly with the access control procedure and privacy-aware procedure, the combined results can be only “Permit” or “Deny”. In this paper, we would like to extend the ability of the system of evaluating the request and granting permissions according to the level of authorized users. In detail, while processing a request, based on the policies and credential restrictions defined before, the system replies with three statuses: (i) Permit; (ii) Deny; and (iii) Partially Permit. The level of permissions depends on the level of privacy protection that the administrator sets up at the beginning. By this way, we can ensure the privacy protection for fine grained element of the DBMSs. In the next section, our architecture is described in detailed.

B. Policy Conflict Resolution

Two policies conflict with each other if they protect the same data area but granting different rights to users, whether **Access** or **Deny**. Policy conflicting affect the systems' security

as malicious users can easily exploit the vulnerability to access the system. In literature, many studies have addressed the problem of policy conflicting [9], [10], [11], [12], [13], [14]. These solutions include: using expert system [10], modifying (edit, insert, revoke) policy/rule at the collision area [9], [14], using algebraic solutions [11], using Bayesian Network [12], [13]. Furthermore, XACML 3.0-based approaches rely on the combining algorithm between policies and rules as in [15], [16], [17], [18], [19].

To detect conflicts between rules in a given policy and evaluate access request, Fan Deng et al. [20] presented an engine called $form\,conflict$. In detail, it detects two types of conflict to be resolved: (i) common resource conflict; and (ii) dependent resource conflict. In the $form\,conflict$ engine, a `Resource Index Tree` is built based on the resource attribute of a policy's target attribute to convert the rules in policy defined by XACML to the node information in the `Resource Index Tree`. The algorithm compares a rule with those with which it is likely to conflict to avoid unnecessary comparisons; thus, saving a lot of time, leading to an effective performance of the `Policy Decision Points`.

Martin et al. [21] used the model checking method to detect XACML policy conflicts and verified its correctness in Coq Proof Assistant. A rule defined in Coq includes two fields, including (i) $effect_type$ and (ii) $srac_type$ containing four elements of XACML attributes namely Subject, Action, Environment, and Resource. The rules are conflict if they shared the same $srac_type$ with different `effects`.

Mohan et al. [22] proposed a framework capable of dynamically add and remove specialized policies while providing a mechanism to reduce potential conflicts. This can be done by using dynamic attributes to determine applicable policy sets at runtime.

Jebbaoui et al. [23] provided a semantic-based policy analysis scheme to detect flaws, conflicts, and redundancies between the rules of large-size and complex XACML policies. In detail, the detection algorithm analyzes the meaning of policy rules through semantics verification by inference rule structure and deductive logic.

As we can see, most of the existing studies in the literature focus on analyzing common problems of conflict in XACML policy and conflict detection. An intuitive means to resolve policy conflicts is to remove and/or edit all conflicts by revoking and/or modifying the conflicting rules [9], [17]. However, changing the conflicting rules is significantly difficult in practicing in many aspects. First, the policy may consists of thousands of rules, which are often logically entangled with each other. Furthermore, the policy conflicts are often very complicated. Most of the case, a particular rule may conflict with multiple rules; on the other hand, it may be associated with several rules. Modifying the rule, therefore, may lead to a defective policy set and greatly reduce the effectiveness of the access control model of the system. Finally, since policies deployed on a network are often maintained by more than one administrator, conflict detection and elimination requires an approval of all administrators of the system with a careful consideration of its impact to the policy set. Therefore, the key issue in resolving the conflict is how to work with them instead of modifying and/or eliminating them. Our approach in this paper assigns different priority levels to the protected data area. If there is no conflict between rules, those functions are executed sequentially. On the other hand, if there is a collision on the same domain, priority levels are executed in descending order of priority, i.e. level 1 will be given priority over level 2. Hence, the solution will be flexible for the large-scale information system in which multiple administrators participate in management.

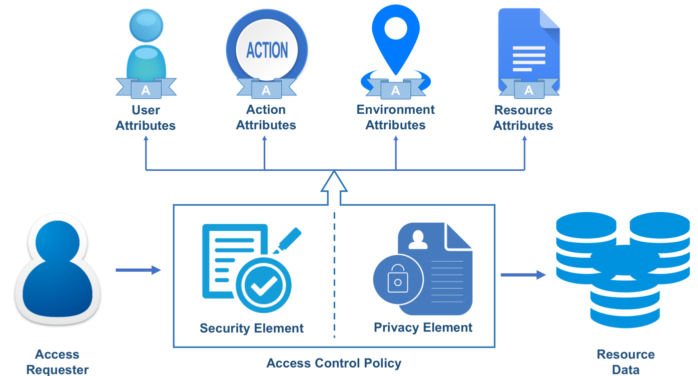

Fig. 1. Two levels of protection in the attribute-based access control model

III. Access Control System Supporting Privacy Protection

A. Privacy-Aware Access Control Policies

The key to ensuring privacy protection access control is identifying how policies can be defined. As we mentioned before, a fundamental requirement of privacy policies is policies having to support fine-grained access control. Fig. 1 illustrates the structure of our policies: when a request is forwarded, the authorization process is carried out through two stages called as 2-stage authorization (i) security stage; and (ii) privacy stage. In security stage, the authorization verifies that the request is legitimate with rights for the access requester to access data based on security elements. In privacy stage, the request is transferred to this stage for checking privacy compliance based on privacy elements.

B. Privacy-Aware Access Control Model

As our model based on ABAC, the model controls access by 4 main attribute types: (i) user attributes; (ii) action attribute; (iii) attributes associated with the resource to be accessed; and (iv) current environment conditions. Fig. 1 illustrates the architecture of the model and the flow of an access control evaluation including conflict resolution. The architecture contains the following main components.

- **Policy Enforcement Points (PEP):** responsible for receiving requests from users. Moreover, it performs access control by making decision requests and enforcing authorization decisions.
- **Repository Interface:** interactive interface between DBMSs. Other components can send request to *Repository Interface* whenever they need more information or data.

- **Policy Information Points (PIP):** serves as the source of attribute values, or the data required for policy evaluation.
- **Policy Decision Points (PDP):** responsible for receiving and examining requests. It retrieves and evaluates applicable policies. After the evaluation processes, it returns the authorization decision to PEP. It is the core component of the model.
- **Policy Administrator Points (PAP):** responsible for creating security policies and storing them in the repository.

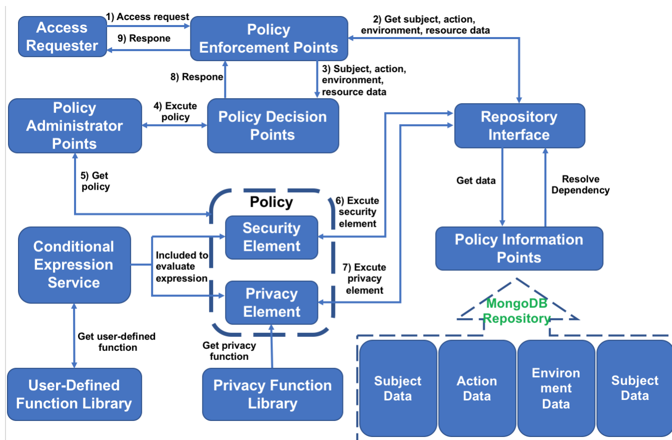

Fig. 2. Proposed privacy-aware access control model resolving conflict security and privacy element

As an access request is sent to the PEP module (step 1), the module queries the `Repository Interface` to get the value of important attributes about the requester including subject, action, environment and resource data (step 2). After retrieving these attributes, the PEP forwards them with the access request to the PDP in which the access request will be evaluated for granting access. The PDP identifies applicable policies that are stored in the PAP module for the evaluation process. These policies contain two levels of protection, security elements, and privacy elements. If a single condition of security stage is not satisfied, the system returns a `Deny` and the request is not granted access to the data. In case all security elements are satisfied, the requested data is loaded from the storing database (in this case, the MongoDB database is used). Then the privacy checking procedure is enabled as privacy function is loaded from the `Privacy Function Library`. Depending on the number of satisfied privacy elements, a portion of the data corresponding to the satisfied conditions will be returned to the access request. As different rules/policies may present conflicts between each other, the `Conditional Expression Service` module is responsible for changing the string conditions to the condition tree as illustrated in `Algorithm 1`. On the overlapped domain, privacy functions are selected based on their priority levels. In general, priorities are applied whenever conflicts between rules in an inside policy or inter-policy occur. Depending on the function selected, the requested data can be showed in three statuses: (i) Show; (ii) Partially show; or (iii) Hide to the requester.

C. Policy Structure

A policy set includes one or more policies. Policy structure contains one or multiple rules which can be created from several elements. There are two main elements namely `security` and `privacy`. On the one hand, the `security` element is responsible for allowing or not allowing to execute users' requirements. On the other hand, the `privacy` element is responsible for determining whether access data should be shown, hidden or generalized. A typical policy can be specified as follows:

- **policy_id:** identifier of policy
- **collection_name:** name of collection or table containing resource data
- **rule_combining:** responsible for solving the conflict of rules
- **is_attribute_resource_required:** a derived field used to determine whether the policy needs attribute resource to evaluate conditions of target or rules.
- **target:** conditional expression specifies when the policy should be applied to.
- **security:** an array field with each element in it is a rule which contains `id` field, `effect` field (value of this field can either `Permit` or `Deny`) and condition.
- **privacy:** the privacy protection engine is also based on rules which are the Boolean expressions evaluated by user's defined function, subject, resource, environment attribute.

Each rule of security element defines a conditional expression that is modeled as a function tree structure. They return a value specified in the element `Effect` if the condition is `true`.

As an example, consider the following `Policy 1`.

```
{   "policy_id" : "Policy 1",
    "collection_name" : "Department",
    "action" : "read",
    "rule_combining" : "permit-overrides",
    "is_attribute_resource_required": true,
    "target" : {
//Equal(Subject.active, true)
        "function_name" : "Equal",
        "parameters" : [{
                "value" : "active",
                "resource_id" : "Subject"
            }, {
                "value" : "true",
                "resource_id" : null
          }]
        },
    "security" : [{
        "id" : "rule 1",
        "effect" : "Permit",
        "condition" : {
//Equal(Resource.dept_name, department)
            "function_name" : "Equal",
            "parameters" : [{
                "value" : "dept_name",
```

```
                "resource_id" : "Resource"
                }, {
                 "value" : "department",
                 "resource_id" : "null"
              }]
      } } ]
}
```

For the privacy element, each rule is an array field with each element is similar to an obligation (in XACML) containing `id` field, `field_effect` field and condition. It is worth to note that the `field_effect` field which has an array type describes the list of data disclosure levels for each field of JSON data constrained by these rules. Each element in `field_effects` has two components: (i) component `name` storing the path to the field; and (ii) component `effect_function` containing `X.Y` value where `X` denotes the privacy domain and `Y` denotes the name of privacy functions on that domain. In the normal situation, the default values of privacy functions are `PrivacyDom.Show`. These elements are Boolean expressions evaluating by user's defined function, subject, resource, and environment attribute. Here, as a constraint, a field of the resource can only belong to at most two domains. The first one is default domain containing two basic privacy functions to represent the status of the data, i.e. `Hide` or `Show`. The other one is configured by the administrator. Below, an example of privacy structure for `Policy 1` is illustrated:

```
{ <...>
"privacy" : {
  "rule_id" : "rule 1",
  "condition" : {
     "function_name" : "Equal",
     "parameters" : [{
          "value" : "dept_name",
          "resource_id" : "Resource"
      }, {
          "value" : "OPERATIONS",
          "resource_id" : null
      }]
  },
 "field_effects" : [{
    "name" : "dept_id",
    "effect_function" : "PrivacyDom.Hide"
  },{
    "name" : "dept_no",
    "effect_function" : "PrivacyDom.Show"
  }, {
    "name" : "dept_name",
    "effect_function" : "PrivacyDom.Show"
  }]
}
```

As shown in the code, we assumed that `PrivacyDom` is the protected area. When a request is made, depending on the evaluation of the model, the data has two statuses `Hide` or `Show`.

D. Algorithms

Algorithm 1 Algorithm for parsing conditional expression

Input: $rawExpression$: String
Output: $function$: Function class
Let $listTok$: List<String> $\leftarrow getToks(rawExpression)$
Let $stackTok$: Stack<String>
Let $queueTok$: Queue<String>
Let $queueFun$: Queue<Function>

1: **for** tok **in** $listTok$ **do**
2:   **if** $IsFunctionName(tok)$ **or** tok == "(" **or** $IsLogicalOperator(tok)$ **then**
3:     $stackTok$.**push**(tok)
4:   **else if** tok == ")" **then**
5:     **while** $stackTok$.length ¿ 0 **do**
6:       $temp$ = $stackTok$.pop()
7:       **if** $temp$ == "(" **then**
8:         $queueTok$.enqueue($stackTok$.pop())
9:         **break**
10:       **else**
11:         $queueTok$.enqueue($temp$)
12:       **end if**
13:     **end while**
14:   **else**
15:     $queueTok$.enqueue (tok)
16:   **end if**
17: **end for**
18: **while** $stackTok$.length > 0 **do**
19:   $queueTok$.enqueue ($stackTok$.pop())
20: **end while**
21: **while** $queueTok$.length > 0 **do**
22:   tok = $queueTok$.dequeue()
23:   **if** $IsFunctionName(token)$ **then**
24:     $function$ = Function.CreateFunction (tok)
25:     **for** 1 **to** $GetNumberParameters(function)$ **do**
26:       $function$.Parameter.Add($queueFun$.dequeue())
27:     **end for**
28:     $queueFun$.enqueue($function$)
29:   **else**
30:     $queueFun$.enqueue(Function.CreateConstValue(tok))
31:   **end if**
32: **end while**
33: **return** $queueFun$.enqueue

---

1) Algorithm for parsing conditional expression: Algorithm 1 converts the conditional expression in text format to the $Function$ structure. Firstly, the $rawExpression$ is split into tokens ($listTok$). We assume that $stackTok$ is a stack storing names of functions, $queueTok$ is a queue storing tokens in Reverse Polish Notation form, and $queueFun$ is a stack storing functions. Then the tokens queue is parsed into $Function$ structure (**for loop** line 1 – 17). The process is built as an expression tree with bottom–up approach. After dequeuing the token queue until it is empty, it is parsed to $Function$ structure and enqueued to $queueFun$ in the **while loop** between line 18 and 20. Then we dequeue the $queueFun$ based on the number of parameters and add those elements to `parameters` field in the **for loop** from line 25 to 27. After that, these new elements are enqueued to $queueFun$ in line 28 and line 30. We continue with the remaining elements and return the value of $queueFun$ in line 33.

Algorithm 2 Algorithm for evaluating policy and request

Input: List<Policy>, Request: JSON
Output: $respone$ Respone class
Let $listPolicy$: List<Policy>
Let $request$: Request $\leftarrow getValue(Request)$

```
1:  for policy in listPolicy do
2:    if GetSubject (policy, request) and GetCollection
      (policy, request) and GetAction (policy, request)
      then
3:      if is_sub_policy then
4:        if Overlap_Domain then
5:          //Execute the function with lower priority
6:        else
7:          response = PolicyCombining(sub_policy)
8:        end if
9:      else if Target(policy, request) then
10:       listSecRule = policy.GetSecurityRule(policy)
11:       listPriRule = policy.GetPrivacyRule(policy)
12:       flag = true
13:       while secRule in listSecRule and flag do
14:         if !Condition(secRule,request) then
15:           flag = false
16:         end if
17:       end while
18:       response = RuleCombining(secRule.GetEffect())
19:       while priRule in listPriRule and flag do
20:         if Condition(priRule,request) then
21:           //Choose field_effects by name
22:           //Execute effect_function
23:         end if
24:       end while
25:     else
26:       //Continue with next policy in listPolicy
27:       response = Response(policy,Request)
28:     end if
29:   else
30:     //Continue with next policy in listPolicy
31:     response = Response(policy,Request)
32:   end if
33: end for
34: return response
```

2) Algorithm for evaluating policy and request: Algorithm 2 describes the evaluation between the list of policy and the request. The **Input** of this algorithm is the list of the policy stored in PAP and the request sent from an access requester. We assume that $listPolicy$ is a list storing the policies in PAP and $request$ is a variable storing the value of subject, action, environment, resource. First, we find the best policy which allows the request to access the data resource. If the subject value, resource value, and the action value between $policy$ and $request$ does not equal, the system will consider the next policy (line 2). Next, the value of is_sub_policy is checked. If returns **true**, we check the value of $Overlap_Domain$. In this case **true**, the conflict occurs in the evaluation process, and the function with lower priority is executed (line 4 - 5). Otherwise, the value of $response$ is the value of the function of `PolicyCombining`(sub_policy) (line 6 - 7). If the value of is_sub_policy is **false** compared to the value of $request$ to the `Target` element. If the $request$ can fulfill all target constraints, the `Security` and `Privacy` elements is evaluated. Otherwise, we move to the next policy (line 9). $listSecRule$ and $listPriRule$ are the variable storing the `rule` of `Security` and `Privacy` respectively. Apparently, if a single condition is not satisfied, the returned value is **false** and user's request is not granted access (**while loop** from line 13 to line 17). We only execute the `Privacy` element if and only if the access request in `Security` element returns `Permit` (line 19). According to the `name` of `filed_effect`, the `effect_function` is executed. Finally, the algorithm continues with the remaining policy (between line 26 and line 30) and returns the value of $response$ in line 34.

IV. Policy Conflict Resolving

A. Policy Conflict

In an authorization system, a particular policy set often contains multiple policies while a policy generated by many rules. For each rule, its policy evaluates to different decisions (e.g. Permit, Deny). To avoid conflicts between policies and rules, traditional approaches applied a set of combining rules to the policy set. Those solutions are inherited from XACML [24]. In general, the combining algorithm is represented by a structure called "PolicyCombining" described by two components as below:

- **policies_id:** An array of policy identifiers
- **combining_algorithm:** The name of algorithm is used to solve conflict when multiple policies are contained in `policies_id` field.

An example of the "PolicyCombining" is illustrated as follow:

```
{   "_id" : "58f24565de2b68f43464287a",
    "policies_id" : [
        "Policy 1", "Policy 2"
    ],
    "algorithm" : "deny-overrides"
}
```

B. Privacy Conflict

In privacy stage, a conflict can be created as multiple privacy rules from the same policy simultaneously satisfied a condition. As a result, several privacy functions can be applied to a particular field of the object. To handle this situation, we added a structure called `PrivacyDomain`. It contains four elements including:

- **domain_name:** The name of domain.
- **fields:** The names of fields in resource which are belong to this domain.
- **is_sub_policy:** To check whether this is domain for privacy function or sub-privacy policy.
- **hierarchy:** To configurate the priority for each privacy function. It contains two sub-elements, namely, **name** describe the name of function, **priority** describe the value of priority.

An example of `PrivacyDom` is showed below:

```
{   "domain_name" : "PrivacyDom",
    "fields" : [],
    "is_sub_policy" : false,
    "hierarchy" : [{
         "name" : "Hide",
         "priority" : 1
          },{
         "name" : "Show",
         "priority" : 2
     }
] }
```

C. Scenario

This section presents the sample of policy conflict resolution on privacy element and we will use as a running example through the article. Information of an employee is showed as below:

```
{ "name": "John",
  "personal_info": {
    "birth_date": "15/01/1994",
    "ssn": "457-55-5462"
} }
```

The rule element of `policy 1` is assumed as:

```
{  "policy_id": "policy 1",
    <...>
   "privacy" : {
   "rules" : [{
   "rule_id" : "rule 1",
   "condition" : {
//assume that this condition is satisfied}
   "field_effects" : [{
      "name" : "name",
      "effect_function" : "Optional"
      },{
      "name" : "personal_info.birth_date",
      "effect_function":"Date.ShowYear"
      }, {
      "name" : "personal_info.ssn",
      "effect_function":"Ssn.SerialNumber"
     }],
   }]}
}
```

The rule element of `policy 2` is assumed as:

```
{  "policy_id": "policy 2",
  <...>
  "privacy" : {
  "rules" : [{
  "rule_id" : "rule 1",
  "condition" : {
//assume that this condition is satisfied}
   "field_effects" : [{
     "name" : "name",
     "effect_function" : "PrivacyDom.Show"
     },{
      "name" : "personal_info.birth_date",
      "effect_function":"Date.ShowMonthYear"
      },{
      "name" : "personal_info.ssn",
      "effect_function":"Ssn.AreaNumber"
   } ],
   },{
    "rule_id" : "rule 2",
    "condition" : {
//assume that this condition is satisfied}
     "field_effects" : [{
       "name" : "name",
       "effect_function" : "PrivacyDom.Show"
       },{
       "name" : "personal_info.birth_date",
       "effect_function" : "Date.Show"
       },{
       "name" : "personal_info.ssn",
       "effect_function" : "Optional"
   } ]
  }]}
}
```

We explain more detail about the `field_effects` field in the privacy structure. It is an array field with the number of elements in each field is equal to the number of the single value field in the resource. Each element has the following structure:

- **name:** is the path to the single value field.
- **effect_function:** This field has only 2 value patterns. First is “Optional” value, second is “X.Y” value where X is privacy domain, and Y is the name of privacy function in that domain.

We have the conflicting privacy showing in Table 1:

TABLE I. THE EXAMPLE OF CONFLICT PRIVACY FUNCTIONS

Fields	Conflict Privacy Functions
name	Optional, PrivacyDom.Show
personal_info.birth_date	Date.ShowMonthYear, Date.ShowYear, Date.Show
personal_info.ssn	Ssn.AreaNumber, Ssn.SerialNumber, Optional

We assume the `Privacy Domain` below:

```
{
  "domain_name" : "Date",
  "fields" : ["Employee.personal_info.
                birth_date"],
  "is_sub_policy" : false,
  "hierarchy" : [{
     "name" : "ShowYear",
     "priority" : 1
  },{
     "name" : "ShowMonthAndYear",
     "priority" : 2
  }
 ] },{
  "domain_name" : "Ssn",
  "fields":["Employee.personal_info.ssn"],
  "is_sub_policy" : false,
  "hierarchy" : [{
```

```
      "name" : "AreaNumber",
      "priority" : 1
    }, {
      "name" : "GroupNumber",
      "priority" : 2
    }, {
      "name" : "SerialNumber",
      "priority" : 3
    }]
}
```

The privacy function will be chosen by the following rule:

P(”Optional”) < P(”PrivacyDom.Show”) < P(X.Y1) < ... < P(X.Yn) < P (”PrivacyDom.Hide”)

where P(X.Y) denotes for priority of privacy function Y in domain X. The priority is configured by administrator in `PrivacyDom` structure.

Applying this rule to the above conflict table, the result is described in Table 2:

TABLE II. RESULT OF SOLVING CONFLICT BETWEEN PRIVACY FUNCTIONS

Fields	Conflict Privacy Functions
name	PrivacyDom.Show
personal_info.birth_date	Date.ShowYear
personal_info.ssn	Ssn.AreaNumber

Applying the chosen privacy functions, the result of data is showed as below:

```
{
  "name": "John",
  "personal_info": {
    "birth_date": "1994",
    "ssn": "457"
  }
}
```

V. EXPERIMENT

A. Environment and Sample Dataset

The system configuration for the experiments is a 64-bit machine with 8GB of RAM and 2.8 GHz Intel Core i5 CPU running macOS High Sierra. The prototype is implemented by C#, .NET Core[1] and MongoDB v4.0 for storing policies and data. We used `mockaroo tool`[2] to generate sample dataset.

B. Privacy Protection Testbest

The proposed architecture was implemented for two cases: (i) with simple data structure; and (ii) with complex data structure. For the first scenario, structure of each resource consists of ten fields (key – value) and one document. On the other hand, the second one contains and array of embedded documents. Each record has an array of embedded documents field containing at least five elements inside. In general, all experiments are included in total five policies. Moreover, to observe the difference between the performances of policy with single security element (traditional solution) and policy with security and privacy elements (our solution), the processing time of each case is recorded.

Table 3 compares the performances of both policies on the two cases simple and complex data structure. On analyzing the table, it can be observed that as the number of records increases, the gap difference between processing time of both policies expands sharply. Considering the case of simple structure, when the number of records is 2000, this gap is only 0.328 seconds; however, it increases to 1.863 seconds as the number of records reaches 12000. A similar situation happens in the case of complex structure as this difference rises from 0.613 seconds to 2.514 seconds. For a database of up to 12000 records, the difference of approximately 2 second is acceptable. It is worth to note that as the complexity of the data structure increases, the time needed to process a record increases.

In order to analyze in detail the performance of each case, Table 3 also presents the average processing time for each record. While the traditional solution needs around 1 millisecond to process a record, the proposed model requires 1.14 and 1.36 milliseconds depends on the complexity of the data structure. Again, with the development of computer system nowadays, this difference is acceptable.

TABLE III. PROCESSING TIME (MEASURE IN MILLISECOND) FOR THE MODEL WITH AND WITHOUT PRIVACY POLICY ON DIFFERENT DATA STRUCTURE

Number of records	Privacy element		No privacy element	
	Simple structure	Completely structure	Simple structure	Completely structure
2000	2264	2797	1936	2184
4000	4394	5471	3972	4237
6000	6734	8168	5555	6769
8000	8877	10897	6657	7867
10000	11751	13539	8963	9975
12000	13983	16550	12120	14036
Average for each record	**1.143**	**1.367**	**0.933**	**1.073**

C. Policy Conflict Resolution Testbed

The proposed architecture was implemented for three cases: (i) single policy; (ii) multi-policy without priority; and (iii) multi-policy without priority. The first structure of policy consists of a single policy. The second one is the multi-policy which consists of one main policy, ten sub-policy and being executed without priority. The last one had a similar policy structure but being executed with priority. In general, all experiments are included in total ten policies. Moreover, to observe the difference between the performances of different models with a single policy, multi-policy without priority (normal solution) and policy with priority (our solution), the processing time of each case is recorded.

Table 4 compares the performances of all models on the three cases single, multi-policy with(out) priority. On analyzing the table, it can be observed that as the number of records increases, the gap difference between the processing time of both policies expands. Considering the case of a single policy, when the number of records is 50000, this gap is only 0.539

[1]https://github.com/xuansonha17031991/privacy-aware-access-control-model
[2]https://www.mockaroo.com/

seconds; however, it increases to 47.508 seconds as the number of records reaches 500000. A similar situation happens in the case of the multi-policy with and without priority as this difference rises from 0.556 seconds to 48.994 seconds and from 0.565 seconds to 59.855 seconds, respectively. It is worth to note that as the complexity of the policy structure increases, the time needed to process record increases. For a database of up to 500000 records, the difference of approximately 10 seconds is acceptable. The time difference between our solution with a normal solution is spent on conflict resolution.

In order to analyze in detail the performance of each case, Table 4 also presents the average processing time for each record. While the normal solution needs nearly 0.09 millisecond to process a record, the proposed model requires 0.111 milliseconds depending on the policy structure having the priority or not.

TABLE IV. PROCESSING TIME (MEASURE IN MILLISECOND) FOR DIFFERENT POLICY STRUCTURES

Number of record	Single policy	Multi-policy without policy conflict resolution	Multi-policy with policy conflict resolution
50000	5390.6	5560.2	5654
100000	10496.6	11178.4	15380
200000	13285.6	13354.4	17062
300000	26190.8	26537.6	31212
400000	35366.6	36003.4	44648
500000	47508.8	48993.6	59855.2
Average for each record	0.0892	0.0914	0.1121

VI. CONCLUSIONS

In this paper, we have proposed an Attribute-based Access Control model for fine-grained privacy protection. The model defines two levels of protection on the policy structure namely security stage and privacy stage. The privacy element allows the system to show or hide the requested data based on credential restrictions defined before and reply to the requester with three statuses: (i) Permit; (ii) Deny; and (iii) Partially Permit. As system usually managed by several administrators, conflicts can occur between rules of the same policy or among different policies. In reality, the conflicts pose a massive security risk to the user's privacy as sensitive information can be accessed without authorized permission. Our approach provides a mechanism to define different priority levels for each privacy domain. In this way, instead of detecting whether there is a conflict or redundancy or not, the system executes privacy functions according to their priority. In addition, we introduced a fine-grained privacy protection by providing user-defined libraries. As a result, one can easily interact with and evaluate access control with the lowest storage unit, e.g. field to collections or databases. From the analysis of the experimental results obtained on two testbeds: (i) several data structure, (ii) policy conflict resolution, we can state that the proposed model is implemented successfully and the difference of processing time between our solution and the traditional one is acceptable. In future work, we aim to apply the model to healthcare system in which the requirements for privacy protection is at the highest level while supporting dynamic policy is needed. Moreover, we also plan to apply a new approach [25] to our scheme whereby the system will be greater flexibility, availability while ensuring security and privacy for system.

ACKNOWLEDGMENT

Sincerely thank to Luong Van Huy who supported in implementation and provided feedback on early revisions.

REFERENCES

[1] E. Bertino *et al.*, "Access control for databases: concepts and systems," *Foundations and Trendsin Databases*, vol. 3, no. 1–2, pp. 1–148, 2011.

[2] V. C. Hu *et al.*, "Guide to attribute based access control (abac) definition and considerations (draft)," *NIST special publication*, vol. 800, no. 162, 2013.

[3] H. Wang, L. Sun, and V. Varadharajan, "Purpose-based access control policies and conflicting analysis," in *IFIP International Information Security Conference*. Springer, 2010, pp. 217–228.

[4] P. Biswas, R. Sandhu, and R. Krishnan, "An attribute-based protection model for json documents," in *International Conference on Network and System Security*. Springer, 2016, pp. 303–317.

[5] C. A. Ardagna *et al.*, "An xacml-based privacy-centered access control system," in *Proceedings of the first ACM workshop on Information security governance*. ACM, 2009, pp. 49–58.

[6] M. E. Kabir, H. Wang, and E. Bertino, "A role-involved conditional purpose-based access control model," in *E-Government, E-Services and Global Processes*. Springer, 2010, pp. 167–180.

[7] M. E. Kabir *et al.*, "A conditional purpose-based access control model with dynamic roles," *Expert Systems with Applications*, vol. 38, no. 3, pp. 1482–1489, 2011.

[8] Q. Ni, E. Bertino, J. Lobo, C. Brodie, C.-M. Karat, J. Karat, and A. Trombeta, "Privacy-aware role-based access control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 3, p. 24, 2010.

[9] H. X. Son, L. K. Tran, T. K. Dang, and Y. N. Pham, "Rew-xac: an approach to rewriting request for elastic abac enforcement with dynamic policies," in *Advanced Computing and Applications (ACOMP), 2016 International Conference on*. IEEE, 2016, pp. 25–31.

[10] B. Stepien and A. Felty, "Using expert systems to statically detect" dynamic" conflicts in xacml," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*. IEEE, 2016, pp. 127–136.

[11] E. Karafili, S. Pipes, and E. C. Lupu, "Verification techniques for policy based systems." IEEE, 2017, pp. 1–6.

[12] B. Bahrak, "Ex ante approaches for security, privacy, and enforcement in spectrum sharing," Ph.D. dissertation, Virginia Tech, 2013.

[13] A. Al-Mutairi and S. Wolthusen, "Mpls policy target recognition network," in *International Conference on Risks and Security of Internet and Systems*. Springer, 2015, pp. 71–87.

[14] M. H. Nguyen and H. X. Son, "A dynamic solution for fine-grained policy conflict resolution," in *International Conference on Cryptography, Security and Privacy*. ACM, 2019.

[15] M. Ayache, M. Erradi, A. Khoumsi, and B. Freisleben, "Analysis and verification of xacml policies in a medical cloud environment," *Scalable Computing: Practice and Experience*, vol. 17, no. 3, pp. 189–206, 2016.

[16] A. Lunardelli, I. Matteucci, P. Mori, and M. Petrocchi, "A prototype for solving conflicts in xacml-based e-health policies," in *Computer-Based Medical Systems (CBMS), 2013 IEEE 26th International Symposium on*. IEEE, 2013, pp. 449–452.

[17] Q. N. T. Thi *et al.*, "Using json to specify privacy preserving-enabled attribute-based access control policies," in *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. Springer, 2017, pp. 561–570.

[18] H. X. Son and M. H. Nguyen, "A novel attribute-based access control system for fine-grained privacy protection," in *International Conference on Cryptography, Security and Privacy*. ACM, 2019.

[19] H. X. Son, T. K. Dang, and L. K. Tran, "Xacs–dypol: Towards an xacml–based access control model for dynamic security policy."

[20] F. Deng and L.-Y. Zhang, "Elimination of policy conflict to improve the pdp evaluation performance," *Journal of Network and Computer Applications*, vol. 80, pp. 45–57, 2017.

[21] M. St-Martin and A. P. Felty, “A verified algorithm for detecting conflicts in xacml access control rules,” in *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*. ACM, 2016, pp. 166–175.

[22] A. Mohan and D. M. Blough, “An attribute-based authorization policy framework with dynamic conflict resolution,” in *Proceedings of the 9th Symposium on Identity and Trust on the Internet*. ACM, 2010, pp. 37–50.

[23] H. Jebbaoui, A. Mourad, H. Otrok, and R. Haraty, “Semantics-based approach for detecting flaws, conflicts and redundancies in xacml policies,” *Computers & Electrical Engineering*, vol. 44, pp. 91–103, 2015.

[24] E. Rissanen *et al.*, “extensible access control markup language (xacml) version 3.0,” *OASIS standard*, vol. 22, 2013.

[25] H. X. Son, T. K. Dang, and F. Massacci, “Rew-smt: A new approach for rewriting xacml request with dynamic big data security policies,” in *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. Springer, 2017, pp. 501–515.