2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering

Efficient Binary-Encoding Access Control Policy Combination for Large-Scale Collaborative Scenarios

Chunhua Li*, Weirui Xie, Ke Zhou

Wuhan National Lab for Optoelectronics
Huazhong University of Science and Technology
Wuhan, China
li.chunhua@hust.edu.cn

Abstract—**Collaboration among organizations is an important part in an e-business or e-health service. In this case, it is necessary for each organization to share their data with each other in order to provide better services. However, each organization may have specified a set of secure access control policies for shared data, resulting in policy conflicts or differences in a collaboration scenario. Therefore, it is critical to generate an integrated global access control policy for shared common data. Due to low efficiency queries and large memory space, current approaches in security policy integration cannot be applied to large-scale collaboration scenarios. In this paper, we present a binary-encoding policy integration method that can quickly seek the combinable rule sets and easily merge single access control policy into a global policy set. The main idea is that first each policy is converted to a triple, namely <attribute, action, constraint>, then to encode all attributes and actions in each rule in a binary form, after that to shift and weighted code so as to get combined attribute code, finally to sort code and merge rules. Compared with similar researches, our work can reduce the integration time by 34%, and the integration time is stable as the number of attributes increases, hence quite suitable for the large-scale collaborative scenarios.**

Keywords—access control, security policy integration, conflict reconciliation, organization collaboration, binary encoding

I. INTRODUCTION

With the rapid development of e-business and e-health, both internal and external organizations need more and more interactive requirements [1]. Thus it can be seen that cooperation among organizations becomes an indispensable part. In order to improve the efficiency of cooperation and provide better service, it is necessary for each organization to share common resources. However, these shared resources may have been assigned a set of access policies by organizations based on their respective security requirements, resulting in policy conflicts or differences in collaboration situations. For example, in an e-health service, access policy of organization A requires that the nurse can read the patients' medical records, while policy of organization B describes that only the doctor can access it. Therefore, it is very important to integrate the access policies from different organizations into a global conflict-free one to govern the interactions among collaborating organizations during the collaboration.

The problem of policy integration with constraints condition was first introduced by Bonatti et al. [2]. They proposed an algebra of security policies together with its formal semantics and illustrated how to formulate complex policies in the algebra and how to translate policy expressions into equivalent logic programs. Backes proposed an algebra providing various types of operators for composing and restricting enterprise privacy policies together with its formal semantics [3]. Yau presented a similarity-based security policy integration and conflict reconciliation for collaborations among organizations in ubiquitous computing environments [4], in which if one party can't negotiate with others, the resources can never be reached. Mazzoleni et al. presented an algebra system for combining fine-grained authorization policies for multiple anticipant organizations [5], they used rule similarity to merge multiple policies into a single one. Later, Jagadeesan et al. proposed a triple policy algebra process utilizing the timed concurrent constraint programming paradigm and defined another Boolean operators [6], which has similar formulation to the work described in [7]. Duan et al. proposed an automated policy combination for data sharing using attribute compression [8-9], but in which directly searched intersection rules using string matching thereby leading to a low efficient query.

In this paper, we introduce a binary-encoding rule compacting and policy integration based on Duan's work [9]. Our method encodes all attributes in each rule in a binary code after dividing policies expressed in XACML into some triples. The binary code helps seek the rules of intersection quickly, which accelerates the process of policy combining. We adopt the similar compacting approach described in the literature [10], which has much more restrictive to the combined policies, i.e. a request only meeting all policies' constraints is permitted to access shared data, any one of denied constraint could hold back a request. we use bottom-up approach to separate policies into a series of rules and then to classify rules into different intersections, rules in the same class have identical attributes and actions. In the end, all classes are merged into two sets according to the permitting or denying action. If a rule is not contained in any policy, it will be abandoned as the conflicted rule.

Our contributions can be summarized as follows:

(1) We propose a binary-based policy coding approach, which transforms the XACML policy into a triple expressed in binary code instead of semantic description. With the help of binary encoding and sorting, corresponding rules can be quickly sought in each policy from different organizations but associated with the same attribute constraints.

2324-9013/18/31.00 ©2018 IEEE
DOI 10.1109/TrustCom/BigDataSE.2018.00085

(2) We present an efficient policy integration algorithm, which can be able to automatically merge multiple local policies into a global one. When decision of every policy applying to a request is allowed, the integrated policy is the intersection of these policies, and the opposite scenario is the same. Our algorithm can quickly search for the intersection of policies to be integrated. In addition, we also handle the contradictory situation with conflict regulation.

(3) We implement our method on the OpenStack Object Storage (Swift). Experimental results show that our proposed method is stable and applicable to the large-scale collaborative scenarios.

The rest paper is organized as follows. Section II firstly reviews the XACML policy modeling, then gives some definition related to our policy, and furtherly describes our system model. Section III elaborates our policy integration approach. Section IV reports our experiments and analysis. Section V concludes our works.

II. PRELIMINARIES

A. An overview of XACML

When addressing policy integration problem, the policies are usually expressed with XACML modeling, which is based on the requirement of each organization. XACML (an eXtensible Access Control Markup Language) is an OASIS standard policy description language[11], which defines rule combining algorithms and are used to resolve conflicts within a policy.

An XACML policy consists of four main components: *a data target, a rule set, conflict algorithm, and other information*. Data target is the object that a policy intends to protect for, such as file, which decides whether a request is suitable for policy rules or not. A rule is a basic element of a policy, a rule set contains all rules. Conflict algorithm is used to deal with conflicts among available rules when different results are come up with. Other information describes some detail operations when the policy enforces. During the process of policy integration, a rule set should be departed into different parts and a conflict algorithm is utilized to handle the contradictory circumstances.

The conflict algorithm in XACML provides a kind of solution when multiple rules or policies conflict with each other's response to the same request. There are four types of conflict algorithms, which are Deny-unless-Permit(DP)，Permit-unless-Deny (PD) and Permit Override (PO)，Deny Override (DO).

Permit-unless-Deny (PD): The result is 'deny' when any policy in the policy set is denied.

Deny-unless-Permit (DP): The result is 'permit' when any policy in the policy set is permitted.

Deny Override (DO): The rule result is 'deny' when any policy in the policy is denied. Combined result is 'permitted' if no rule evaluates to deny and at least one policy evaluates to permit. The final result is 'NotApplicable' if all rules don't evaluate.

Permit Override (PO): The rule result is permitted when any policy in the policy is permitted. Combined result is 'deny' if no rule evaluates to permit and at least one policy evaluates to deny. The final result is 'NotApplicable' if all rules don't evaluate.

B. Policy Defintion

Policy model of supporting attribute-based operation has been proposed [12]. There are three responses for a request in an XACML policy, which are Permit, Deny and NotApplicable. To describe kinds of logical layer, the fundamental definitions of policy description should be given.

Definition 1 (Attribute Tuple). The subject attribute tuple, action attribute and constraint subject attribute tuple are defined as $<s,V(s)>,<a,V(a)>,<c,V(c)>$, where s, a, c represents subject vector, user action vector and constraints string respectively, $V(s)$, $V(a)$, $V(c)$ means the value set of vector s, a and string c respectively.

Definition 2 (Rule). The tuple $\{(s, s_i), (a, a_j), (c, c_t)\}$ is called as attributes authorization term, where $s_i \in V(s)$, $a_j \in V(a)$, $c_t \in V(c)$, it shows subject s_i can carry out the action a_j only when the user condition attributes satisfy c_t. This tuple also define a suitable rule.

Definition 3 (Request). A request is formed as req $=\{(s, v(s)), (a, v(a)), (c, v(c))\}$, where $v(s) \in dom(s)$, $v(a) \in dom(a)$, $v(c) \in dom(c)$, in this case the request req is well-formed request. The dom(s) is all user's subject values set, and the definitions of dom(a) and dom(c) is the same as the former. The set of all requests over Σ is denoted as R_Σ.

Definition 4 (Request satisfactory). A request r $=\{(s, v(s)), (a, v(a)), (c, v(c))\}$, for s, a, c, if $v(s) \in V(s)$,$v(a) \in V(a)$, $v(c) \in V(c)$, then the request is suitable for the policy, so in this case the request r can access the data.

Definition 5 (Policy). A policy P is a combination of rules and other information, including the policy conflict algorithms which is used to deal with conflicts when rules lead to different results. The policy can combine all rules following the conflict algorithm. In the end, the policy can be expressed as the triple <Ry, Rn, Rna>, where the tuple denotes the set of requests permitted, denied, and not applicable by the policy P respectively. So, $R_\Sigma = Ry \cup Rn \cup Rna$, $Ry \cup Rn = \varnothing$, $Ry \cup Rna = \varnothing$, $Rna \cup Rn = \varnothing$, which means the triple has no intersection with each other.

Definition 6 (Policy Set). A policy set is a combination of policies and other information, which also includes the policy conflict algorithms that works when policies lead to a different result.

Take an example of Health Information System. If a request suitable to policy P is that a doctor whose age is 47 intends to access the medical information, this request can be represented as follow:

req = {(s, doc),(a, access),(age, =47)}

The policy P can be expressed as:

< rule1: {(s, doc), (a, access), (c, age>45)}, rule2: {(s, doc), (a, access), (c, seniority >15)}, conflict algorithm: PO>
where the policy means a doc whose age is more than 45 or seniority is higher than 15 can access the data. So after the combination, the pairs <Ry, Rn> can be expressed as Ry={(s, doc),(a, access), (c, age>45 or seniority>15)}, Rn={other situation}.

Tab.1 shows the XACML of the above policy. We can form the triples of conflict algorithms and rule sets. Before combination, we need to find the intersections of subjects and actions. Rules with the same subject and action can be combined.

TABLE I An example of policy

PolicyExample.xml

```
<Policy PolicyId="Policy1"
RuleCombiningAlgId="permit-overrides">
  <Rule1_1 Effect= Permit >
    <Target>
      <Subjects Designation = {doc}>
      <Action Access Type = {access}>
    </Target>
    <Condition ="age>45">
    </Rule1_1>

  <Rule1_2 Effect= Permit >
    <Target>
      <Subjects Designation = {doc}>
      <Action Access Type = {access}>
    </Target>
    <Condition ="seniority>15">
  </Rule1_2>
</Policy>
```

C. System Model

Fig.1 describes our system model and interactions of each part. It involves data owner, data user, data storage, policy restore, identity authority, and organizations. The identity authority charges to identity entry based on users' attributes. The policy restore is the place where the combined policies are stored. Organizations can take part in the process of policy formulation.

First, the authority identifies a user based on his attributes, and delivers him a token so they can access the system. Then, data owner and organizations cooperate with each other to generate the access control policy with policy conflict detection and policy combination. After that, data owner uploads data to the cloud and then data user sends a request to get data. At last, system verifies the request and responses to the data user.

When a request asking for data, the PEP (Policy Enforcement Point) transforms it to PDP (Policy Decision Point) to handle it. PDP evaluates requests for authorization policies before issuing a decision.

Owing to there is a policy combination processor and coding rules in our system, we can directly analyze whether a request is acceptable or not.

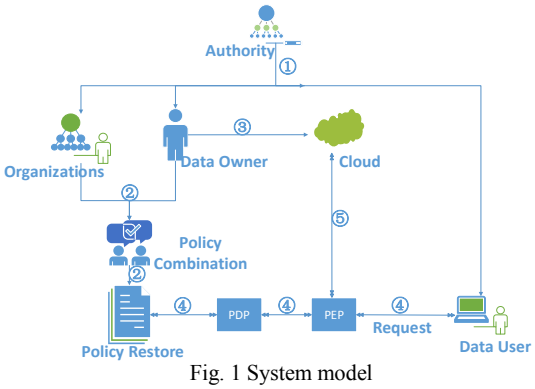

Fig. 1 System model

When different organizations generate XACML policies, system can compress the rules and code for all attributes. When a request comes to PEP, PEP can judge it by the stored combination policy information.

III. POLICY COMBINING ALGORITHM

In this paper, we aim to automatically generate combined policy based on the multi-rules. A compacting algorithm is introduced, based on the above definitions about rule and policy conflict algorithm. P_1, P_2, …. , P_n is defined by various anticipant entities.

A. Algorithm Describing

As the compaction method described, the subject and the action must be the same. So we can integrate policies as the following steps. The first step is to compress the rules, transforming from XACML modeling policies to 3-value formulation. Then encode the rules based on the attributes, after that, the rules are represented as binary which can reduce the storage and quickly seek the intersection. The last step is policy integration, sorting the binary, we can easily get the classes of the same value, including the attributes and actions. After above three steps, we can get the entirely combinable rule set.

Step 1: Rule Compressing

First, indexing every organization from 1 to *N* sequentially, which represents policy 1 to policy *N* in the policy set. In order to classify the rules, we need to compress all rules in each policy.

Then, transforming the XACML modeling to basic triples, i.e. subject, action, and constraint. Owing to the object is the same data, it can be simplified by compression.. The subject as we mentioned before, is multiple attributes representing the user identification; the action is some operations the user can do to the data which designated by the rules, if there are multiple actions in one policy, we divide it into several policies that each one has one action; as for the constraints, we define it as string format to handle at last. The algorithm input is N policies set and output is N-triple set.

Step 2: Rule Coding

Scanning *N* policies sequentially and coding the attributes according to its value range to insure the same attribute or action has the same binary code. When dealing with rule

coding, each attribute in rules is restored in system dictionaries. With the help of coding operation, not only space consuming is saved than the 3-values formulation, but also more convenient to seek the intersection. After the coding step, rules are represented as binary constraint part. The coding entries include the attributes and action part, which identities the user. Next is Policy Belong that represents which policy the rule belongs to. The last is the Policy Type and the conflict algorithm, which plays an important role in the integration process. The input of the algorithm is N policies; output is N binary policies with attributes and action dictionaries.

Algorithm 1. Rule Coding

```
     Input: N policies, each policy has M rules
     Output: N  policies , each rule represent binary
1    Def condingrules(ruleset):
2      att_dictionary ={}
3      act_dictionary ={}
4      for rule in ruleset:
5        if rule.attribute not in dictionary_attributes:
6          att_dictionary[rule.attribute] = binary(count_att )
7          count_att = count_att+1
8        if rule.action not in dictionary_action:
9          act_dictionary[rule.action] = binary(count_act)
10         count_act = count_act +1
11     Rerutn [att_dictionary,act_dictionary]
```

The pseudo-code of the rule coding algorithm is showing upper. The 2, 3 lines define the dictionary of attributes and action. The 4th line to 10th line sequentially scan all rules, matching each attribute and action to the binary count. At the end of algorithm, the coded rule and two dictionaries returned.

There are two attributes in Tab.2 that shows the transform expression in the coding process. The first attribute is gender which has two candidates and the second attribute is position which is currently assumed to be five. The map between the key and value is shown in Tab.2.

TABLE II EXAMPLE OF CODING RULE

Attribute	***Key***	***Value***
Gender	Man	0
	Female	1
Position	Internship	000
	Nurse	001
	Doctor	010
	Manager	011
	dean	100

The gender has two candidates to be chosen, so the length of values is 1, which can represent two states. The later one is 3 bits to hold more than 4 conditions. In the first attribute, We use state 0 to represent the man and state 1 to note female. The length of attributes varies depending on the size of candidates and the length of value is logarithm scale of that.

Step 3: Rule Combination

As described in step 2, all rules are represented as binary formulation where all information the rule carrying is 0 and 1. Based on the idea of compression, the subject and the action must be the same. In this paper, the intersection is the rules that have the same attributes and action. Now we need to find the intersection according to the numerical values. As our designation, there are 5 steps in all to complete that.

3.1 Shift and Weighted the attributes

We weight different attributes and action to make the rules computable. The result of shift is based on the attributes number and length. They are sequentially joint together and form the binary string, which represents the value of rule.

3.2 Sorting the rule

Shifting the attributes and calculating the value of each rule, we are able to sort them to get the combinable class. After the sorting operation, we can easily get the same attributes and action rules. Here is an example about how it executes detailed.

TABLE III EXAMPLE OF SORING RULE

	Attribute				*sorted*	
	gender	code	position	code	index	value
1	Female	1	Internship	000	2	0001
2	Man	0	Nurse	001	3	0001
3	Man	0	Nurse	001	5	0100
4	Female	1	Nurse	001	1	1000
5	Man	0	dean	100	4	1001

The sorting operation is shown in Tab.3, the length of attributes represented as binary is varied. In the shuffle process, the binary shifts and then joint together. For the Rule1, the attributes are 1 and 000 so that the joint binary is 1000. After the soring step, it located before Rule4 which is 1001 and after Rule5 which is 0100. When all rules are sorted, the same attributes are collected in sequential place. So the same rules in one class can come to the process of combination.

3.3 Checking rules.

However, in practical system we need to consider the detailed process of the combination. We need to promise every policy has the same description about the subject and action value, and only in that case the rules set can be integrated. The incomplete class that don't contain all policies should be removed. For example, there are five policies in one policy set, and if one class has 4 different rules totally, the class should be abandoned rather than participating in the combination process.

3.4 Deal with the contradictory situation

In the process of 3.3, we handle incomplete class, and then the rules that belong to one policy must clear the internal conflict. Based on the common policy conflict algorithm, the constraint intersections are divided into corresponding positive and negative sets. For example, if the policy P contains two rules of the same attributes, except the rule type and constraint. Rule1 holds the permit type and Rule2 is negative, and the constraint of Rule1 is A and the latter is B. If the conflict algorithm of P is permit override, the intersection part $A \cap B$ belongs to A for rule1 holds the same type as policy. Rule2 constraints become B-A and the rule1 keep steady.

3.5 Form the result

When all policies inner conflicts have been cleared, the R_y and R_n start to form. We establish R_y as the union of all permit rules constraint, and R_n as the union of all negative rules. Then

we handle the policies conflict as the inner processing that set R_y' is the union of all permit rules and R_n' is the union of all denied. If policy set conflict algorithm is PD(Permit-unless-Deny), the result Ry is the subtraction of R_y' and R_n', and Rn is R_n', and if the conflict algorithm is DP (Deny -unless- Permit), the result R_y stay and R_n is the subtraction of R_n' and R_y'. After we have finished the process, the system will restore the R_y and R_n respectively. When the request suits for R_y or R_n, the PDP can respond to it directly. If not, the request is suitable for NotApplicable, then the system will ignore the request.

There is a pseudo-code about the processes. The algorithm input is N binary rules; output is multiple intersections that contain the same attributes and action.

Algorithm 2.Rule Combination

```
     Input: N policies , each rule is expressed with binary coding
     Output: multiple intersections
1    Def rule Combination (ruleset):
2    for rule in ruleset:
3       weigh all attributes and action as rule.val
4    sort(rule.val)
5    result=[]
6    while i and j in rule:
7    set i is the first, j is the last flag point same value
8    if rule i to rule j contain all policy:
9       then result.append(rule i)
10   else
11      abandon rule i to rule j
12   Return result
```

The 2nd to 5th lines of algorithm do the preliminary work of seeking intersection. The processor firstly weights all attributes in rules and then sorts them. Rules with the same values are arranged sequentially and the flag i to j can scan them in line 6 to 11. As long as the intersection contains all policies, they should be added to the result, waiting for combined, while the deficiency should be abandoned.

Here is an example about the enforcement of the full algorithms:

As the Tab.4 describing, the first line represents the policy attributes. The column 1 is the index number, which identities the rules. Next 3 columns are represented the basic attributes to be integrated. If the rules units have the same value, then the rules have the possibility to combine. As we mentioned before, the attributes are varied for different applications. The 5th column is the rule belonging, and next is the rule effect. The 7th is conflict algorithm dealing with the situation that the same rule in one policy and there are intersection in their constraints.

TABLE IV EXAMPLE OF COMBINATION

Index	*Att1*	*Att2*	*Action*	*Policy belong*	*Rule effect*	*Conflict Algorithm*	*constraint*
1	000	001	01	**1**	Y	Po	A
2	000	001	01	1	N	Po	B -A
3	000	001	01	**2**	Y	Do	C
4	000	001	01	**3**	N	Po	D
5	000	001	01	**4**	Y	Do	E -F
6	000	001	01	4	N	Do	F
7	000	001	01	**5**	Y	Po	G
8	000	002	02	1	Y	Po	H

The Tab.4 has five policies and totally eight rules, which attributes and other information have been mapped into binary and sorted.(The policy belong is shown in numerical terms for easily understand; the policy type and the conflict algorithm can represent their attributes with two values, which are denoted as 0 and 1 respectively).The constraint is indicated as A-H, which is the restrain that organizations set for the request.

The combiner firstly scans the sequential rules, until find the first rule with different attributes and action value, which represents that the rule is the start of next intersection to be classified. Secondly, system checks whether the intersection contains all policies in the policy set, and if not, then the intersection is not our obligation because there could be such a situation that no organization has appointed this class. Thirdly, the conflict between the single policy should be abandoned. Assume that Rule1 and Rule2 belong to Policy1, which are positive and negative respectively. And policy 1 conflict algorithm is Po (permit override), which means the intersection of Rule1 and Rule2 $A \cap B$ is belong to Rule1, owing to the former is positive rule. So the constraint of Rule2 should be B-A. Similarly, the Rule5 is the E-F. Finally, the fourth step is output the result of Ry and Rn, which contains the positive request and the negative. We define

$$Ry' = \{A \cup C \cup (E\text{-}F) \cup G\}, \text{ and}$$

$$Rn' = \{(B\text{-}A) \cup D \cup F\}.$$

When there is an intersection between Rn' and Ry', it means that the different policies occur contradictory, the result should be modified by the conflict algorithm of the policy set. If the conflict algorithm is PD(Permit-unless-Deny), the result is Ry=Ry'-Rn', and Rn=Rn. However, if the conflict algorithm is DP (Deny -unless- Permit), the result is Ry=Ry', and Rn=Rn'-Ry'.

B. Complexity Analysis

We consider the policy combination algorithm, including step 1 to 3. Assuming that there are N policies totally, and each policy has M rules in average. In the first algorithm, system scans all rules sequentially. So the complexity of time is O (N*M). When dealing with the coding process in step 2, combiner scans all rules and codes the attributes and other information respectively, where the time consuming is O(N*M). In the step of 3, the rules are represented as binary values. Other information can be indicated as 0 or 1 as well . So the intersection seeking process is to sort the values of all the rules. There are N*M rules and the sorting time complexity will be O(Log(N*M)* (N*M)) for any steady sort algorithm. In the merge process, combiner scans all rules and seeks whether they suit the condition or not, which consumes O(N*M) time occupation. In the output stage, the generation outputs the Rn and Ry which costs constant time. Consequently, the totally process time complexity is O(Log(N*M)*(N*M)).

IV. IMPLEMENTATION

In this section we examined time consuming of ours and similar method [9] on the platform of OpenStack Object Storage (Swift), we mainly analyzed the consuming change as the number of attributes and rules grow. All experiments are

conducted on a Linux Server with Inter Xeon(R) E5-2609, running at 64-bit Ubuntu 14.04.

We implemented a XACML policies generator which can generate random attributes based on access control policies. The data has little influence on the generation process time cost but the size is important. Each policy was derived from random rules and conflict algorithm, and the rules contained multiple attributes as we settled. We also analyzed the circumstances that attributes expanding, showing the scale of time consuming tendency when the size of data growing. The attributes scale is controlled from 4 to 7. Considering that typical policy contains different rule numbers may conduct diverse conclusion, the rule index is settled between 50 and 3500 for convincing there are no numerical suspicion.

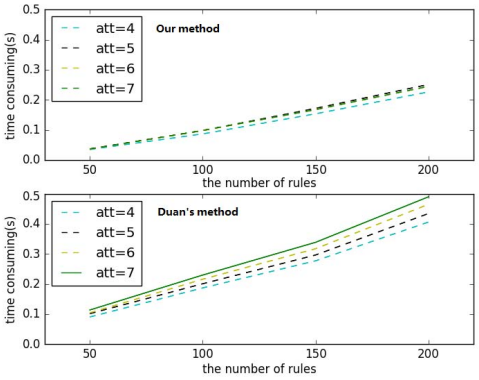

Fig. 2 Methods contrast

In the first set of experiments, we varied the rule numbers and the attribute numbers in one policy to observe the difference of compacting time consuming. From the Fig.2 we can easily find that with the rules number in one policy growing, the contrast method cost more time than ours, and the slope is much sharper than coding-based way. In experiment 2 and 3, we will study rule numbers and attributes influence separately for examining the method on large scale of data situation.

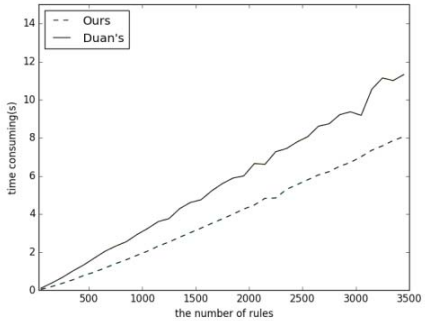

Fig. 3 Rules number expands

In the second set of experiments, the scale of rule numbers influence on the policy was viewed in integration processor. More rules in one policy represents that there are more comparing with each other. So the scale of data in policy certainly is connected with the comparing efficient. The experiment is conducted in the circumstances that there are 50 to 3500 rules for each policy. In the Fig.3, we observed that the difference of time consuming between the two ways increases with rules number growing. The lower line is our method and the upper is the contrast one. This can be an evident that our method is more suitable for large scale data circumstances.

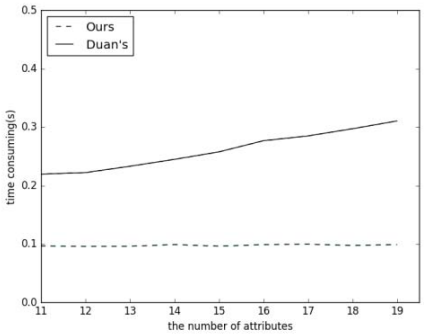

Fig. 4 Attributes number expands

Thirdly, we evaluated the size of attributes influence on the intersection process. The way1 is our method that coding the attributes with binary, the way2 is the contrast that not coding attributes and get intersection directly. Owing to there will be large scale of attributes in user identification process, so the number of attributes should be in consideration. As the Fig.4 shows, with the growing of the attribute numbers in each policy, the performance of coding operation to the attributes is more efficient than the opposite way. Our method almost steady but the contrast is growing steadily. We observed that using our coding-base method substantially decreases the intersection time comparing with the other, as the number of attributes growing, the difference between them is more significant.

V. CONCLUSIONS

In this paper we proposed a coding based solution as an important step when dealing with large scale of policies combinations. We express attributes in binary, thereby reducing the time of time in the intersection process. The experiments show that with the attributes expanding violently, the time consumption of our method is not sensitive, while the contrast one rises steadily. Besides, since all attributes are encoded in binary rather than strings form, the memory occupation of our method is much less than the comparison one. In the end, owing to the intersection has been found and integrated policy has been generated before the system starts, PEP can immediately respond to the request as the generation processor handled offline.

We plan to extend this work in several interesting directions. One direction involves with the expression about basic algebraic operation, such as or-gate and not-gate. After we have settled the target classes problem, there is a giant step to finish the left composing problem. A suitable algebraic expression may efficiently decrease the integration consume and memory occupation. We believe that our method can help us to figure out a suitable algebraic expression to a large extent.

ACKNOWLEDGMENT

The authors would like to thank the anonymous referees of TrustCom2018 for their reviews and suggestions to improve this paper. This work is supported by the National Key R&D Program of China(2016YFB0800402), partially supported by the National Natural Science Foundation of China under Grant No.61232004 and the Fundamental Research Funds for the Central Universities(HUST: 2016YXMS020).

REFERENCES

[1] Gai K, Qiu M, Xiong Z, et al. Privacy-preserving multi-channel communication in Edge-of-Things[J]. Future Generation Computer Systems, 2018.

[2] Bonatti P, Sabrina D C D V, Samarati P. An algebra for composing access control policies[J]. Acm Transactions on Information & System Security, 2002, 5(1):1-35.

[3] Backes M, Dürmuth M, Steinwandt R. An Algebra for Composing Enterprise Privacy Policies[J]. Lecture Notes in Computer Science, 2004, 3193:pp. 33-52.

[4] Yau S S, Chen Z. Security Policy Integration and Conflict Reconciliation for Collaborations among Organizations in Ubiquitous Computing Environments[C]// International Conference on Ubiquitous Intelligence and Computing. Springer-Verlag, 2008:3-19.

[5] P. Mazzoleni, E. Bertino, and B. Crispo. XACML policy integration algorithms. In Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT), pages 223–232, 2006.

[6] R. Jagadeesan, W. Marrero, C. Pitcher, and V. Saraswat. Timed constraint programming: a declarative approach to usage control. In Proc. of the 7th ACM SIGPLAN international conference on Principles and practice of declarative programming (PPDP), pages 164–175, 2005.

[7] P. Bonatti, S. D. C. D. Vimercati, and P. Samarati. An algebra for composing access control policies. ACM Transactions on Information and System Security (TISSEC), 5(1):1–35, 2002.

[8] L. Duan, S Chen, et al. Automated policy combination for data sharing across multiple organizations, in Proc. IEEE Int. Conf. Services Compute (SCC), 2015, 51(111): 226-233.

[9] L.Duan, Y Zhang, et al. Automated Policy Combination for Secure Data Sharing in Cross-Organizational Collaborations, IEEE ACCESS, 2017, 4:3454-3468

[10] P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Bertino. Xacml policy integration algorithms. In ACM Transactions on Information and System Security (TISSEC), pages 852-869, February 2008.

[11] Extensible access control markup language (XACML) version 2.0. OASIS Standard, 2005.

[12] P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo, An algebra for fine-grained integration of xacml policies, In Proc. ACM Symposium on Access Control Models and Technologies, pp.63-72, 2009.