Towards Policy Engineering for Attribute-based Access Control*

Leanid Krautsevich, Aliaksandr Lazouski, Fabio Martinelli, and Artsiom Yautsiukhin

Istituto di Informatica e Telematica Consiglio Nazionale delle Ricerche
{name.surname}@iit.cnr.it

Abstract. Attribute-based Access Control (ABAC) was recently proposed as a general model which is able to capture the main existing access control models. This paper discusses the problems of configuring ABAC and engineering access policies. We question how to design attributes, how to assign attributes to subjects, objects, actions, and how to formulate access policies which bind subjects to objects and actions via attributes.
Inspired by the role mining problem in Role-based Access Control, in this paper we propose the first attempt to formalise ABAC in a matrix form and define formally a problem of access policy engineering. Our approach is based on the XACML standard to be more practical.

Keywords: ABAC, policy engineering, access control, attributes, attribute mining problem (AMP), role mining.

1 Introduction

Attribute-based Access Control (ABAC) [1] was recently proposed as a general model which is able to capture the main existing access control models, like Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role-based Access Control (RBAC). The core components of ABAC are attributes assigned to all entities, e.g., subjects, objects, actions. Access policies define conditions (predicates over attributes) when access requests are permitted. Although ABAC provides a rich flexibility in defining access policies, there are a lot of challenges regarding a conceptual and formal definition of the model.

This paper discusses problems of configuring ABAC and engineering access policies. We question how to design attributes, how to assign attributes to subjects, objects, actions, and how to formulate access policies which bind subjects to objects and actions via attributes. To the best of our knowledge, currently there is not even a formal definition of these problems for ABAC neither solutions. We see a role mining in RBAC [2] as the most relevant problem which might be extrapolated and used to address similar problems of ABAC.

* This work was partly supported by EU-FP7-ICT NESSoS (256980) and PRIN Security Horizons funded by MIUR with D.D. 23.10.2012 n. 719, and EIT ICT Labs activity 13083.

Role mining, introduced in 2003 [3], gained a lot of attention in last years and a large number of different approaches to the problem were proposed [4–6]. Role mining is generally considered as the automatic creation of roles, assignment of subjects to roles and roles to permissions (object-action pairs). Frank et al. [2] specify three aspects of a role mining problem: (i) a formal definition, (ii) an algorithm, and (iii) quality measures of the algorithm.

It is convenient to use matrices for defining the role-mining problems formally [4, 2, 6]. Matrices help to capture relations between subjects and permissions (**UPA**), subjects and roles (**UA**), and roles and permissions (**PA**). Operations on matrices allow expressing the required relations (e.g., $\mathbf{UPA} = \mathbf{UA} \times \mathbf{PA}$) and perform simplifications, if necessary.

Inspired by the role mining problem, in this paper we propose the first attempt to formalise ABAC in a matrix form and define formally a problem of engineering access policies. Our approach is based on XACML model, an open standard of ABAC proposed by OASIS [7] and widely used in industry and research. We define only the most general problem, and leave the space for further detailed elaboration of the problem for future research, when concrete scenarios are to be considered. We only propose a definition of the problem, and leave possible algorithms and their quality measurements for the future work. We show that the role mining problem is a specific case of policy engineering problem.

The paper is structured as follows. In Section 2 we provide a simple formalisation of ABAC model, based on XACML. Section 3 contains the basics for multidimensional matrix used for our model. We provide a matrix form of ABAC formalisation in Section 4. In Section 5 the policy engineering problem for ABAC is defined and exemplified for RBAC case. Finally, we provide discussion (Section 6), related work (Section 7), and conclusions (Section 8).

2 General ABAC Model

We recall important notions relevant to the ABAC model. Currently, there is not a formal definition of the general ABAC model. However, the first steps towards definition were done by Jin et al. [1] where the authors presented ABAC_α model. ABAC_α is developed to include existing models for access control such as DAC, MAC, and RBAC. Moreover, there is an OASIS standard XACML [7] that defines a language for access control policies for ABAC. Jin et al. [1] focus on the basic, minimal features of ABAC, when the XACML standard is ready for a practical use. In this paper we formalise the core ABAC features close to the XACML standard to make our approach closer to practice. Naturally, we are not able to capture all features of XACML. In the paper we focus on attribute assignment, rules definition, and simple policy decision making. Moreover, we operate with four possible decisions used in XACML. We leave formalisation of policies for the future work.

The essential goal of any access control model is to guarantee that only legitimate subjects have permission to access objects. Following XACML, suppose there is a set S of subjects $s \in S$, a set O of objects $o \in O$, a set Act of possible

actions $act \in Act$ (e.g., "read"). We also may add environment to the model, but skip this part for simplicity. We define a set of all entities in the system as $E = S \cup O \cup Act$. We assume an attribute as a function $ATTR$ which assigns a value to an entity $e \in E$ such that:

$$ATTR : E \mapsto D \tag{1}$$

where D is a finite set of values, i.e., the domain of the attribute. For example, an attribute function mapping $S \mapsto \mathbb{N}$ may specify the age of the subject.

The set of rules maps attributes into one of four possible outcomes:

$$RULE : \bigotimes_{\forall i_a=1}^{n} (D_{i_a}) \mapsto \{\bot, \top, \boxtimes, \varnothing\} \tag{2}$$

where by $\bigotimes_{\forall i_a=1}^{n}$ we mean, that a $RULE$ function requires n attribute values of entities. It is important to note, that to identify the used value precisely we need to specify a triple: the entity (to which the attribute belongs to), the attribute (we consider), and the value. In Equation 2 the information about the entity and the attribute is present only implicitly, but in the following we will need to specify the triple explicitly. Note, that a rule does not bind a value to a specific entity when it speaks about the subject, object, or action of a request. We say that an attribute is *bound* by a rule if the rule states exactly to which entity the attribute belongs to. For example, consider the following rule "*access is allowed only if user John is on vacations*". Here we know, that the value "is on vacation" of "current work status" attribute must belong to John. Consider a rule "*access is allowed only if the subject has an e-mail from iit.cnr.it domain*". Here we know that the attribute "e-mail" belongs to a subject, but we do not know in advance who the subject is. Thus, in the first example we have a bound (to a specific entity) attribute, when in the second case we speak about a *free* attribute.

The result of the $RULE$ function is one of four possible values of domain $\Re = \{\bot, \top, \boxtimes, \varnothing\}$, where $\top$ means positive result (e.g., allow access), $\bot$ means negative result (deny access), $\boxtimes$ means undefined result (e.g., caused by division by zero), and $\varnothing$ means not available or not applicable result. Rules are further aggregated in policies (and further in policy sets) in XACML, but we skip this part in our initial model.

Finally, a Policy Decision Point (PDP) should consider all rules (policies and policy sets in the original XACML standard) and provide the final decision. In short PDP collects all authorisation decisions provided by the rules and returns "permit" if at least one rule returns "permit" decision, and "deny" otherwise.

$$PDP : \bigotimes_{\forall i_r=1}^{n_r} (\Re_{i_r}) \mapsto \Re \tag{3}$$

Ideally, the result of the PDP should be either $\top$ or $\bot$, but $\boxtimes$ and $\varnothing$ are also possible and leave the decision for Policy Enforcement Point.

Since matrix form has proven to be convenient for solving the role mining problem, we aim for a similar matrix form for ABAC model. Indeed, the input information for the attribute mining problem is simply the number of triples (direct access control assignments): subject-object-actions, which were allowed (or denied) in the latest period of time. Such information is a three-dimensional matrix with dimensions denoting subjects, objects, and actions. Therefore, the matrix form should explicitly link all functions defined in the current section and result in direct access control assignments. Such assignments explicitly show whether a user can perform an action on an object. Finally, using such form we will be able to define the attribute mining problem.

3 Mathematical Basis

Before we will be able to define our model we would like to specify the mathematical basis for our model. In the paper we use multidimensional matrices, i.e., the matrices which may have an arbitrary number of dimensions. Multidimensional matrices are usually considered as tensors. In contrast, we build our theory using Multidimensional Matrix Mathematics proposed by Solo [8]. This mathematics adopts all operations from tensor analysis and keeps it simple and close to the classical (2-dimensional) matrix mathematics.

In the paper we denote matrixes with bold capital capital letters e.g., $\mathbf{A}$ (when functions are denoted with capital letters not in bold, e.g., $RULE$), and minuscule letters to denote elements of this matrix, $\mathbf{a}$. Elements always contain indexes to specify the element, e.g., $\mathbf{a}_{i,j}$. Indexes denote the dimensions of matrices. We use indexes for matrices (e.g., $\mathbf{A}_{i,j}$) only when we would like to specify the dimensions of the matrix explicitly. In this section we also use sets of indexes. For example, if we have matrix $\mathbf{A}_{i_1,i_2,i_3}$ with elements $\mathbf{a}_{i_1,i_2,i_3}$ for brevity we write $\mathbf{A}_I$, where $I = \{i_1, i_2, i_3\}$. Let IJK denote any combination of indexes from sets $I = \{i_1, i_2\}$, $J = \{j_1, j_2\}$, and $K = \{k_1, k_2\}$ preserving the order for every set (e.g., $i_1, j_1, j_2, k_1, i_2, k_2$) and $\overline{IJK}$ denote the ordered set of indexes where all indexes from I are followed by all indexes of J and then are followed by all indexes of K (e.g., $i_1, i_2, j_1, j_2, k_1, k_2$).

In the paper we use summation of matrixes, two types of multiplication [8] and a special *diag* operation, defined as follows.

Definition 1. *Let $\boldsymbol{A}_I$ and $\boldsymbol{B}_I$ be two matrices of $|I|$ dimensions. Then $\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$ is also an $|I|$-dimensional matrix with values $\boldsymbol{c}_I = \boldsymbol{a}_I + \boldsymbol{b}_I$;*

Definition 2. *The multidimensional matrix outer product is multiplication of every element of one matrix by every element of another matrix. The multidimensional matrix outer product $\boldsymbol{A}_I \otimes \boldsymbol{B}_K$ is a multidimensional matrix $\boldsymbol{C}_{\overline{IK}}$ every element of which is computed as: $\boldsymbol{c}_{\overline{IK}} = \boldsymbol{a}_I * \boldsymbol{b}_K$.*

Definition 3. *The multidimensional matrix inner product is defined as a contracted multiplication of elements of both matrixes with different indexes. The multidimensional matrix inner product $\boldsymbol{A}_{IJ} \times \boldsymbol{B}_{JK}$ is a multidimensional matrix $\boldsymbol{C}_{\overline{IK}}$ every element of which is computed as: $\boldsymbol{c}_{\overline{IK}} = \sum_{\forall j \in J} \boldsymbol{a}_{IJ} * \boldsymbol{b}_{JK}$.*

In our work, the elements of all matrices belong to a specific domain of results $\Re = \{\varnothing, \boxtimes, \bot, \top\}$ (we also use $\Re' = \{\varnothing, \top\} \subset \Re$). Thus, we have to define the "$*$" and "$\sum$" (or "$+$") operation on the elements of the domain to be able to apply operations defined in Definitions 1, 2, and 3 on matrices.

Definition 4. *Multiplication ("*") and addition ("+" or $\sum$) operations are defined by two corresponding tables*

"$*$"

	$\top$	$\bot$	$\boxtimes$	$\varnothing$
$\top$	$\top$	$\bot$	$\boxtimes$	$\varnothing$
$\bot$	$\bot$	$\bot$	$\boxtimes$	$\varnothing$
$\boxtimes$	$\boxtimes$	$\boxtimes$	$\boxtimes$	$\varnothing$
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

"$+$"

	$\top$	$\bot$	$\boxtimes$	$\varnothing$
$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	$\top$	$\bot$	$\bot$	$\bot$
$\boxtimes$	$\top$	$\bot$	$\boxtimes$	$\boxtimes$
$\varnothing$	$\top$	$\bot$	$\boxtimes$	$\varnothing$

(4)

In the following we will use multiplication of elements to indicate whether a specific element should be considered (e.g., whether a rule should check the value of an attribute). The addition operation indicates how considered elements should be combined (e.g., whether there is at least one value of an attribute satisfying a rule). Since a rule may result in either $\bot$ and $\top$ if applicable we will never meet the addition of $\bot$ and $\top$ values in our work. For the final combination of rules (see Equation 3) we use a simple algorithm, which allows access if at least one rule allows it (similar to deny-unless-permit rule-combining algorithm in XACML). One observation is useful here: if the access control system only specifies when access is allowed ($\top$) and simply ignores the rest ($\varnothing$) then denoting $\top$ as 1 and $\varnothing$ as 0 we get usual boolean operations for *and* and *or*.

Proposition 1. *The operations defined in Definition 4(proofs are in Appendix):*

1. $*$ *and* $+$ *are commutative,*
2. $*$ *and* $+$ *are associative,*
3. $*$ *is distributive over* $+$*.*

Proposition 2. *Let us have three multidimensional matrices $\boldsymbol{A}_{IJK}$, $\boldsymbol{B}_{IML}$, $\boldsymbol{C}_{JMN}$ and $IJK \bigcap IML \bigcap JMN = \emptyset$.*

1. $(\boldsymbol{C} \times \boldsymbol{B}) \times \boldsymbol{A} = \boldsymbol{C} \times (\boldsymbol{B} \times \boldsymbol{A})$.
2. $(\boldsymbol{C} \times \boldsymbol{B}) \times \boldsymbol{A} = ((\boldsymbol{C} \times \boldsymbol{A}) \times \boldsymbol{B})^{T(K;L)}$*, where* $\boldsymbol{D}^{T(K;L)}$ *means transposition, i.e., interchanging of positions, of indexes from set K and L preserving order.*

Two points can be derived from Proposition 2. First, if $I = \emptyset$ then $\mathbf{B} \times \mathbf{A} = \mathbf{B} \otimes \mathbf{A}$, since there are no indexes for contraction. Second, changing the order of matrices does not change the elements of the resulting matrix, but only the order of dimensions.

Finally, we define an operation *diag* which reduces a set of dimensions J of a matrix to one dimension, using only the diagonal elements of J.

Definition 5. *Let $\boldsymbol{A}$ be a multidimensional matrix with dimensions IJK, where $\forall j_t \in J, t = 1...k$ for some finite k. Then,*

$$\boldsymbol{C} = diag_J(\boldsymbol{A}_{IJK}) \ ; \ \boldsymbol{c}_{I\{j\}J} = \boldsymbol{a}_{I\{j_1=j,j_2=j,\ldots,j_k=j\}K} \tag{5}$$

Proposition 3. *If matrix $\boldsymbol{A}_{IJK}$ has only one dimension $J = \{j_1\}$, then $diag_J(\boldsymbol{A}_{IJK}) = \boldsymbol{A}_{IJK}$*

4 Matrix form for ABAC Model

In this section we define a matrix form of ABAC similar to the one used for the role engineering [2]. This form is required in order to explicitly link such entities as subject, object and allowed actions. In other words, at the end of the modelling process we should get a way to easily say which subject has access to which object and which action it is allowed to do on the object. Note, that although a similar link also exists in XACML policies, it is not explicit. For example, a rule which says that *every user from an IT department which has a permanent position may access a document of a project* requires some analysis before saying that John may access description of work of the project.

4.1 Attribute Assignment

First we consider attribute assignments to different entities specified by functions $ATTR$ in Equation 1. Let A be a set of all attributes considered in the system, i.e., for an $a \in A$ we have one corresponding $ATTR$. All functions $ATTR$ may be considered as a three dimensional boolean matrix:

$$\begin{aligned} &\mathbf{ATTR} = (\mathbf{attr}_{i_e,i_a,i_d});\ \mathbf{attr}_{i_e,i_a,i_d} \in \{\varnothing, \top\} \\ &i_e = 1...|E|;\ i_a = 1...|A|;\ i_d = 1...|D_{i_a}| \end{aligned} \tag{6}$$

which assigns $\top$ to an element if an entity $i_e = index(e)$ $e \in E$ has the value of an attribute $i_a = index(a)$ $a \in A$ equals to $i_d = index(d)$ $d \in D_{i_a}$. By $index()$ we mean a function which returns the index of the input. In the following we simply write $i_a = a$. It is very important to note, that in our notations *indexes also explicitly point out the kind of dimension they refer to*. This means, that we should not care much about the order of indexes, since we always can identify them using the name of indexes.

When we define a matrix we first specify its name and element with all indexes, e.g., $\mathbf{ATTR} = (\mathbf{attr}_{i_e,i_a,i_d})$, then we specify the values of the elements ($\mathbf{attr}_{i_e,i_a,i_d} \in \{\varnothing, \top\}$) and finally, we list the ranges of indexes. Indexes of matrices, also denoting the dimensions, are specified as i with a subscript pointing to the nature of the dimension, e.g., i_e denotes an entity dimension. Because of this explicit binding the order of indexes is not important in our work. If we want to refer to a specific element we assign values to the indexes: $\mathbf{attr}_{i_e=\text{“}John\text{''},i_a=\text{“}age\text{''},i_d=\text{“}22\text{''}}$. We use superscripts in brackets for subjects (s), objects (o), actions (act) to denote the corresponding subsets of entities (e.g., $S = E^{(s)} \subseteq E$), attributes (e.g., $A^{(s)} \subseteq A$), and domains ($D^{(s)} \subseteq D$). We also use a specific notation for indexes used only for a subset of entities (e.g., index (i) for attributes (a) of subjects (s) is i_{a_s}).

Note, that for computational reasons values are considered specific for each attribute (i.e., D_{i_a}). In this case, one dimension of **ATTR** matrix will be different for different attributes, but this deviation from the classical representation of matrices does not affect further discussion (but simply requires careful consideration when operations on matrices are defined). Also note, that although

some attribute domains may be infinite we almost always can make them finite, e.g., a domain of natural numbers may be truncated at some value (e.g., 100) and a special value (≥ 100) added to denote the other possible values.

We would like to separate subjects, objects, actions as it is done in XACML (w.l.o.g., we do not use environmental types):

$$\mathbf{ATTR}^{(s)} = (\mathbf{attr}^{(s)}_{i_s,i_{a_s},i_{d_s}});\ \mathbf{attr}^{(s)}_{i_s,i_{a_s},i_{d_s}} \in \{\varnothing, \top\}; \tag{7}$$

$$i_s = 1...|E^{(s)}|;\ i_{a_s} = 1...|A^{(s)}|;\ i_{d_s} = 1...|D^{(s)}_{i_{a_s}}|$$

$$\mathbf{ATTR}^{(o)} = (\mathbf{attr}^{(o)}_{i_o,i_{a_o},i_{d_o}});\ \mathbf{attr}^{(o)}_{i_o,i_{a_o},i_{d_o}} \in \{\varnothing, \top\} \tag{8}$$

$$i_o = 1...|E^{(o)}|;\ i_{a_o} = 1...|A^{(o)}|;\ i_{d_o} = 1...|D^{(o)}_{i_{a_o}}|$$

$$\mathbf{ATTR}^{(act)} = (\mathbf{attr}^{(act)}_{i_{act},i_{a_{act}},i_{d_{act}}});\ \mathbf{attr}^{(act)}_{i_{act},i_{a_{act}},i_{d_{act}}} \in \{\varnothing, \top\} \tag{9}$$

$$i_{act} = 1...|E^{(act)}|;\ i_{a_{act}} = 1...|A^{(act)}|;\ i_{d_{act}} = 1...|D^{(act)}_{i_{a_{act}}}|$$

4.2 Rules Definition

Now we need to specify the matrix for $RULE$ functions. For this matrix we need a set of rules $r \in R$. Every element of set R relates to one $RULE$ function. We also need to capture the parameters of the $RULE$ function. Here we would like to recall, that for precise description of the parameters of $RULE$ function we need to use triples: entity-attribute-value (or separate triples for subject, object, and action; see Equations 7, 8, and 9). We define **RULES** matrix for $RULE$ functions (using the specified triples) as:

$$\mathbf{RULES} = (\mathbf{rules}_{i_r,i^1_{a_s}...i^{n_s}_{a_s},i^1_{d_s}...i^{n_s}_{d_s},i^1_{a_o}...i^{n_o}_{a_o},i^1_{d_o}...i^{n_o}_{d_o}} \tag{10}$$

$$,i^1_{a_{act}}...i^{n_{act}}_{a_{act}},i^1_{d_{act}}...i^{n_{act}}_{d_{act}},i^1_e...i^{n_a}_e,i^1_a...i^{n_a}_a,i^1_d...i^{n_a}_d);$$

$$\mathbf{rules}_{...} \in \{\bot, \top, \boxtimes, \varnothing\};\ i_r = 1...|R|;$$

$$\forall i_{a_s} = 1...|A^{(s)}|;\ \forall i_{d_s} = 1...|D^{(s)}|;\ \forall i_{a_o} = 1...|A^{(o)}|;\ \forall i_{d_o} = 1...|D^{(o)}|;$$

$$\forall i_{a_{act}} = 1...|A^{(act)}|;\ \forall i_{d_{act}} = 1...|D^{(act)}|;$$

$$\forall i_e = 1...|A|;\ \forall i_a = 1...|A|;\ \forall i_d = 1...|D|;$$

The amount of dimensions in this most generic case is $1 + 2 * n_s + 2 * n_o + 2 * n_{act} + 3 * n_a$, where n_s, n_o, n_{act}, n_a are the maximal number of attributes for subject, object, action and bound attributes used for one rule. Since every rule must be stated for a subject-object-action triple, n_s, n_o, n_{act} cannot be 0, while bound attributes are optional and n_a can be 0. For example, if we want to express a policy, consisting of one rule stating that *"a user may get an object only if the sum of his money at present and possible credit is higher than the cost of the object"*, we have 2 attributes of a subject (money the user has now, possible amount of a credit for the user) and 1 attribute of an object (cost of

this object), one for action (type of action, e.g., "get"). Thus, $n_s = 2$, $n_o = 1$, and $n_{act} = 1$ and the amount of dimensions to consider is $1 + 4 + 2 + 2 = 9$.

It is important to note, that in practice, the table itself should not be defined manually (unless specific modifications are required), but should be either automatically derived from the defined rules or found using attribute mining with different heuristic methods (and then transformed to usual XACML policies).

Now, we are able to see which subject-object-action triples satisfy defined rules. For this purpose we need to provide the required parameters for $RULE$ functions. In the matrix form, this means that we need to multiply **RULES** matrix by a corresponding $\mathbf{ATTR}^{(s)}$, $\mathbf{ATTR}^{(o)}$, $\mathbf{ATTR}^{(act)}$ or/and **ATTR** matrix one time for a required attribute. Thus, in the case of the previous example, we need to multiply **RULES** by $\mathbf{ATTR}^{(s)}$ twice, by $\mathbf{ATTR}^{(o)}$ once, and once by $\mathbf{ATTR}^{(act)}$. First we consider bound attributes (we hide all dimensions which do not take part in the multiplication for brevity). Let $\mathbf{RULES\_RES}'$, $\mathbf{RULES\_RES}''$, $\mathbf{RULES\_RES}'''$ be three auxiliary matrices.

$$\mathbf{RULES\_RES}' = (...((\mathbf{RULES} \times \mathbf{ATTR}) \times \mathbf{ATTR}) \times ... \times \mathbf{ATTR}) = \quad (11)$$
$$\mathbf{RULES} \times (\mathbf{ATTR})^{n_a}$$

$$\mathbf{rules\_res}'_{i_r,...} = (\sum_{\forall i_e^1,...,i_e^{n_a}} \sum_{\forall i_a^1,...,i_a^{n_a}} \sum_{\forall i_d^1,...,i_d^{n_a}} \mathbf{rules}_{i_r,...,i_e^1...i_e^{n_a},i_a^1...i_a^{n_a},i_d^1...i_d^{n_a}} *$$

$$* \mathbf{attr}_{i_e^1...i_e^{n_a},i_a^1...i_a^{n_a},i_d^1...i_d^{n_a}}$$

By $(\mathbf{ATTR})^{n_a}$ we denote the outer matrix product applied several times to the same matrix (Proposition 2 for the proof). Although i_e^1 and i_e^2 denote the same dimension (i.e., entity) they refer to different entity-attribute-value triples. Thus, we cannot apply contraction to them computing $(\mathbf{ATTR})^{n_a}$.

When we multiply the resulting matrix on $\mathbf{ATTR}^{(s)}$, $\mathbf{ATTR}^{(o)}$, $\mathbf{ATTR}^{(act)}$ we do not simply do contraction of all indexes, as it was in case of **ATTR** matrix. In these cases the dimensions denoting the entities to which the attributes belong to (i.e., free dimensions) are added to the resulting matrix. Since we would like to consider one subject, one object and one type of actions we should take the elements with the same indexes (i.e., apply $diag$ function to $I_s = \{i_s^t | t = 1...n_s\}$).

$$\mathbf{RULES\_RES}''_{i_r,i_s,...} = diag_{I_s}(\mathbf{RULES\_RES}' \times (\mathbf{ATTR}^{(s)})^{n_s}) = \quad (12)$$
$$diag_{I_s}(\mathbf{RULES\_RES}'''_{i_r,i_s^1...i_s^{n_s},...})$$

Let $I_o = \{i_o^l | l = 1...n_o\}$ and $I_{act} = \{i_{act}^k | k = 1...n_{act}\}$. The matrix of result of rules for subjects performing actions on objects is:

$$\mathbf{RULES\_RES}_{i_r,i_s,i_o,i_{act}} = diag_{I_{act}}(diag_{I_o}(diag_{I_s}(\mathbf{RULES} \times (\mathbf{ATTR})^{n_a})$$
$$\times (\mathbf{ATTR}^{(s)})^{n_s}) \times (\mathbf{ATTR}^{(o)})^{n_o}) \times (\mathbf{ATTR}^{(act)})^{n_{act}} \quad (13)$$

Note, that according to Proposition 2 the order of multiplication changes only the order of dimensions in the resulting matrix, but not the elements of the matrix. Thus, we may apply multiplications in any order, respecting the converged dimensions and $diag$ operations.

4.3 Access Control Matrix

We know the decisions for every rule with respect to a subject-object-action triple. A Policy Decision Point (PDP) should consider all rules and provide the final decision. Let $\mathbf{PDP}_{i_r}$ be the final rules-combining matrix, used by PDP to combine all rules and make an authorisation decision.

$$\mathbf{PDP} = (\mathbf{pdp}_{i_r});\ \mathbf{pdp}_{i_r} = \top; i_r = 1...|R| \tag{14}$$

In short PDP simply collects all authorisation decisions provided by the rules. Thus, the access control matrix (**ACM**), which defines which action a subject may perform on which objects, can be found as follows:

$$\mathbf{ACM}_{i_s,i_a,i_{act}} = \mathbf{RULES\_RES}_{i_r,i_s,i_a,i_{act}} \times \mathbf{PDP}_{i_r} \tag{15}$$

4.4 Example

Assume we consider access control policies for a hospital. We consider four subjects $S = \{John, Peter, Paul, Eve\}$ which may access three records $O = \{rec1, rec2, rec3\}$ of three different patients $\{Ada, Felix, Rebecca\}$. In this small hospital there are only two departments (surgery and infection departments ($D_1^{(s)} = \{sur, inf\}$) in which two doctors $\{John, Peter\}$ and two nurses $\{Paul, Eve\}$ work ($D_2^{(s)} = \{doctor, nurse\}$). Three rules are defined for the access control:

1. *rule1: Doctors are allowed to write all patient records*;
2. *rule2: Nurses from surgery are not allowed to write the record of Rebecca*;
3. *rule3: Anyone from infection department can read all records*;

We see, that there are 2 attributes of subjects we should consider: the role in the hospital and the department the subject belongs to. For object we have only one parameter: name of the patient. Finally, we would like to consider two types of access: read and write. Thus, the three matrixes of attributes are:

$$\mathbf{ATTR}^{(s)} = \begin{matrix} \\ s_1(John) \\ s_2(Peter) \\ s_3(Paul) \\ s_4(Eve) \end{matrix} \begin{bmatrix} \begin{bmatrix} doctor & nurse \\ \top & \varnothing \\ \top & \varnothing \\ \varnothing & \top \\ \varnothing & \top \end{bmatrix} & \begin{bmatrix} sur & inf \\ \top & \varnothing \\ \varnothing & \top \\ \top & \varnothing \\ \varnothing & \top \end{bmatrix} \end{bmatrix} \tag{16}$$

$$\mathbf{ATTR}^{(o)} = \begin{matrix} \\ rec1 \\ rec2 \\ rec3 \end{matrix} \begin{bmatrix} Ada & Felix & Rebecca \\ \top & \varnothing & \varnothing \\ \varnothing & \top & \varnothing \\ \varnothing & \varnothing & \top \end{bmatrix} \quad \mathbf{ATTR}^{(act)} = \begin{matrix} \\ act_w \\ act_r \end{matrix} \begin{bmatrix} write & read \\ \top & \varnothing \\ \varnothing & \top \end{bmatrix} \tag{17}$$

Now we need to model **RULES** table. We see that we use at most 2 attributes of a subject in rule 2. As for attributes of object and action we use only one of

them maximum. Thus, we need $1 + 2 * 2 + 1 * 2 + 1 * 2 = 9$ dimensions. On the other hand, since we have only one attribute for object and action we can skip dimensions for them, using only the dimensions for the values of these attributes. We also see that rules 1 and 2 are defined for the *write* action only, when rule 3 is defined for the *read* action. Thus, all elements related to rule 3 for action *read* and 1 and 2 for action *write* are $\varnothing$. Therefore, for brevity, we show only the meaningful parts of the **RULES** matrix (see Equation 18).

If one wants to read Equation 18 we propose to start unwrapping it from the middle. Consider any smallest two-by-two matrix with elements easily singled out in any place of the **RULES** matrix. Every row in such matrix means either the role of the subject (marked as "role" for the corresponding rows) or the department a subject belongs to (we use a mark "dep" for the corresponding rows). The columns contain values of role ("doctor" for the first ($d_1^{(s)}$) row and "nurse" for the second ($d_2^{(s)}$)) or department ($d_1^{(s)} = sur$ or $d_1^{(s)} = inf$). We should not be scared by different meanings (and even different size) of domains for different rows, since we will apply operations only with rows of similar kind. Thus, the element of the matrix says whether a subject with a specific value of an attribute is allowed to do something.

Next, we see these smallest two-by-two matrices are combined in other two-by-two matrices (for which the smallest matrices are just elements). We see that the rows again denote roles and departments, and columns denote possible values. We should not be surprised because we considered the attributes of subjects twice. Finally, we see that there are three such higher rank matrices (for Ada, Felix, and Rebecca), which are obviously related to the only object attribute we consider and its three possible values.

Naturally, we should not forget about action value dimension (*read* and *write*) and three rules. Thus, Equation 19 shows the final matrix. This matrix explicitly indicates how a rule is applied to a subject-object-action triple.

Finally, the **ACM** matrix is (using Equation 15):

$$\mathbf{ACM}_{i_s,i_o,i_{act}} = \tag{20}$$
$$\begin{array}{c|c|c|} & act_w & act_r \\ & \begin{array}{cccc} s_1 & s_2 & s_3 & s_4 \end{array} & \begin{array}{cccc} s_1 & s_2 & s_3 & s_4 \end{array} \\ \hline rec1 & \begin{array}{cccc} \top & \top & \varnothing & \varnothing \end{array} & \begin{array}{cccc} \varnothing & \top & \varnothing & \top \end{array} \\ \hline rec2 & \begin{array}{cccc} \top & \top & \varnothing & \varnothing \end{array} & \begin{array}{cccc} \varnothing & \top & \varnothing & \top \end{array} \\ \hline rec3 & \begin{array}{cccc} \top & \top & \bot & \varnothing \end{array} & \begin{array}{cccc} \varnothing & \top & \varnothing & \top \end{array} \\ \hline \end{array}$$

5 Engineering Access Control Policies for ABAC

Role engineering is a set of activities which aim at finding a suitable set of roles, user-role and role-permission assignments. Role engineering is considered either like a top-down approach (when external information about possible roles exists) or as a bottom-up approach, usually referred to as role mining [4, 2]. Similar to role engineering for RBAC we specify a policy engineering problem for ABAC. Here we focus on an attribute mining problem.

$$\mathbf{RULES} = \tag{18}$$

$$\begin{bmatrix}
 & d_1^{(act)} = write; rule1 & & \\
 & \begin{bmatrix} d_1^{(o)} = Ada \\ role \\ \begin{bmatrix} doctor \\ d_1^{(s)} \\ \top \\ \varnothing \end{bmatrix} \begin{bmatrix} nurse \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \\ department \\ \begin{bmatrix} sur \\ d_1^{(s)} \\ \top \\ \varnothing \end{bmatrix} \begin{bmatrix} inf \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \end{bmatrix} & \begin{bmatrix} d_2^{(o)} = Felix \\ role \\ \begin{bmatrix} doctor \\ d_1^{(s)} \\ \top \\ \varnothing \end{bmatrix} \begin{bmatrix} nurse \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \\ department \\ \begin{bmatrix} sur \\ d_1^{(s)} \\ \top \\ \varnothing \end{bmatrix} \begin{bmatrix} inf \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \end{bmatrix} & \begin{bmatrix} d_3^{(o)} = Rebecca \\ role \\ \begin{bmatrix} doctor \\ d_1^{(s)} \\ \top \\ \varnothing \end{bmatrix} \begin{bmatrix} nurse \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \\ department \\ \begin{bmatrix} sur \\ d_1^{(s)} \\ \top \\ \varnothing \end{bmatrix} \begin{bmatrix} inf \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \end{bmatrix} \\
role & & & \\
dep & & & \\
role & & & \\
dep & & & \\
 & d_1^{(act)} = write; rule2 & & \\
 & \begin{bmatrix} d_1^{(o)} = Ada \\ role \\ \begin{bmatrix} doctor \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \begin{bmatrix} nurse \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \\ department \\ \begin{bmatrix} sur \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \begin{bmatrix} inf \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \end{bmatrix} & \begin{bmatrix} d_2^{(o)} = Felix \\ role \\ \begin{bmatrix} doctor \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \begin{bmatrix} nurse \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \\ department \\ \begin{bmatrix} sur \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \begin{bmatrix} inf \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \end{bmatrix} & \begin{bmatrix} d_3^{(o)} = Rebecca \\ role \\ \begin{bmatrix} doctor \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \begin{bmatrix} nurse \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \\ department \\ \begin{bmatrix} sur \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \begin{bmatrix} inf \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \end{bmatrix} \\
role & & & \\
dep & & & \\
role & & & \\
dep & & & \\
 & d_2^{(act)} = read; rule3 & & \\
 & \begin{bmatrix} d_1^{(o)} = Ada \\ role \\ \begin{bmatrix} doctor \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \begin{bmatrix} nurse \\ d_1^{(s)} \\ \varnothing \\ \top \end{bmatrix} \\ department \\ \begin{bmatrix} sur \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \begin{bmatrix} inf \\ d_1^{(s)} \\ \varnothing \\ \top \end{bmatrix} \end{bmatrix} & \begin{bmatrix} d_2^{(o)} = Felix \\ role \\ \begin{bmatrix} doctor \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \begin{bmatrix} nurse \\ d_1^{(s)} \\ \varnothing \\ \top \end{bmatrix} \\ department \\ \begin{bmatrix} sur \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \begin{bmatrix} inf \\ d_1^{(s)} \\ \varnothing \\ \top \end{bmatrix} \end{bmatrix} & \begin{bmatrix} d_3^{(o)} = Rebecca \\ role \\ \begin{bmatrix} doctor \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \begin{bmatrix} nurse \\ d_1^{(s)} \\ \varnothing \\ \top \end{bmatrix} \\ department \\ \begin{bmatrix} sur \\ d_1^{(s)} \\ \varnothing \\ \varnothing \end{bmatrix} \begin{bmatrix} inf \\ d_1^{(s)} \\ \top \\ \top \end{bmatrix} \end{bmatrix} \\
role & & & \\
dep & & & \\
role & & & \\
dep & & & \\
 & & &
\end{bmatrix}$$

$$\mathbf{RULES\_RES}_{i_r,i_s,i_o,i_{act}} = diag_{i_{act}}(diag_{i_o}(diag_{i_s}(\mathbf{RULES} \times (\mathbf{ATTR}^{(s)})^2) \tag{19}$$
$$\times \mathbf{ATTR}^{(o)}) \times \mathbf{ATTR}^{(act)}) =$$

$$\begin{bmatrix}
 & act_w & act_r \\
 & \begin{bmatrix} s_1 & s_2 & s_3 & s_4 \\ \top & \top & \varnothing & \varnothing \\ \top & \top & \varnothing & \varnothing \\ \top & \top & \varnothing & \varnothing \end{bmatrix} & \begin{bmatrix} s_1 & s_2 & s_3 & s_4 \\ \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing \end{bmatrix} \\
rule1\ rec1 & & \\
rec2 & & \\
rec3 & & \\
 & act_w & act_r \\
 & \begin{bmatrix} s_1 & s_2 & s_3 & s_4 \\ \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \bot & \varnothing \end{bmatrix} & \begin{bmatrix} s_1 & s_2 & s_3 & s_4 \\ \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing \end{bmatrix} \\
rule2\ rec1 & & \\
rec2 & & \\
rec3 & & \\
 & act_w & act_r \\
 & \begin{bmatrix} s_1 & s_2 & s_3 & s_4 \\ \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing \end{bmatrix} & \begin{bmatrix} s_1 & s_2 & s_3 & s_4 \\ \varnothing & \top & \varnothing & \top \\ \varnothing & \top & \varnothing & \top \\ \varnothing & \top & \varnothing & \top \end{bmatrix} \\
rule3\ rec1 & & \\
rec2 & & \\
rec3 & &
\end{bmatrix}$$

Definition 6. *(Basic Attribute Mining Problem (AMP)). Given a set of users S, a set of objects O, a set of possible actions Act, a set of considered attributes A and a subject-object-action assignments* ***ACM****, find:*

- ***ATTR*** *matrix (or* $\boldsymbol{ATTR}^{(s)}$, $\boldsymbol{ATTR}^{(o)}$, $\boldsymbol{ATTR}^{(act)}$ *matrixes), i.e., the values of the attributes entities have.*
- ***RULES*** *matrix, i.e., the amount of attributes used at most for one rule* (n_s, n_o, n_{act}, n_a)*; attributes required for every rule; the bound entities the attributes belong to; values of all attributes required for satisfaction of rules;*

The basic AMP is a general and complex problem. One may find a large number of variations of this problem assuming, that some information is available. In some cases, **ATTR** matrix may be known (or at least partially known) a priori [3]. For example, such information as age of a subject, its position in the organisation, time of access, a level of criticality of an object may be known in advance. Naturally, in these cases also the domains of the attributes are known. Sometimes also n_s, and n_o values may be known (or assumed, or bound). Further elaborations on the problem, similar to [4] are possible.

The problem for engineering access control policies in more general, can be defined similar to AMP, where instead of (or in addition to) **ACM** any relevant information is available (e.g., business process, a structure of an enterprise, possible losses of incorrect access granting/denying, etc).

5.1 Role Engineering in ABAC

Here we show how your model can be adapted for RBAC case. RBAC assigns a role to a subject and then maps the role with permissions. Here we consider RBAC without hierarchy, i.e., so called flat model [9]. Although ABAC model also is able to use the role as an attribute, it also may work with other attributes (also attributes of an object, an action, etc.) without the need to create (often meaningless) auxiliary roles.

First, RBAC model specifies only when a subject is allowed (i.e., "Permit" decision) to access an object, and uses "deny" decision otherwise. It does not use neither explicit "deny" decision, i.e., $\perp$, nor "undefined", i.e., $\boxtimes$. Thus, all operations for matrices are boolean "and" for multiplication and "or" for addition.

There is one attribute we should consider in this case: a role of a subject. Thus, $\mathbf{ATTR}^{(s)}$ contains 3 dimensions, one of which, i.e., attribute dimension, has only one element, e.g., "role". Therefore, w.l.o.g., we may consider this matrix as a two-dimensional boolean matrix which assigns subjects to their roles. $\mathbf{ATTR}^{(o)}$ and $\mathbf{ATTR}^{(act)}$ are simple unit matrices, which simply state, that an object is this object and an action is this action. We need only one attribute for specifying this. Thus, these two matrixes are also two dimensional unit matrices (similar to Section 4.4).

RULES matrix needs $1+2*1+2*1+2*1+3*0 = 7$ dimensions. Note, that attribute dimensions for subject, object, and action have only one element and we

can remove them for simplicity. Thus, we have 4 domains: role, attribute values of subject (i.e., role values), attribute values of object (i.e., objects themselves), attribute values of action (i.e., actions themselves).

Let M be a set of permissions $m \in M$, which may be defined as a pair of an object and an action allowed on the object [2, 4]. We may define two matrices **RULES_PERM** and **PERM_OBJ** to break **RULES** in two:

$$\mathbf{RULES\_PERM} = (\mathbf{rule\_perm}_{i_r,i_{d_s},i_m});\ \mathbf{rule\_perm}_{i_r,i_{d_s},i_m} \in \{\varnothing, \top\}$$

$$\mathbf{PERM\_OBJ} = (\mathbf{perm\_obj}_{i_m,i_{d_o},i_{d_{act}}});\ \mathbf{perm\_obj}_{i_m,i_{d_o},i_{d_{act}}} \in \{\varnothing, \top\}$$

$$i_r = 1...|R|;\ i_{d_s} = 1...|D_1^{(s)}|;\ i_m = 1...|M|; \tag{21}$$

$$i_{d_o} = 1...|D_1^{(o)}|;\ i_{d_{act}} = 1...|D_1^{(act)}|;$$

$$\mathbf{RULES} = \mathbf{RULES\_PERM} \times \mathbf{PERM\_OBJ} \tag{22}$$

$$\mathbf{rules}_{i_r,i_{d_s},i_{d_o},i_{d_{act}}} = \sum_{\forall i_m} \mathbf{rule\_perm}_{i_r,i_{d_s},i_m} * \mathbf{perm\_obj}_{i_m,i_{d_o},i_{d_{act}}}$$

We define **RULES_PERM** matrix as a three-dimensional binary matrix assigning $\top$ to the element $\mathbf{rule\_perm}_{i_r=r,i_{d_s}=d^{(s)},i_m=m}$ if a rule r assigns a permission m to every subject with an attribute value (i.e., a role) $d^{(s)} \in D_1^{(s)}$, and $\varnothing$ otherwise. Let also define **PERM_OBJ** which assigns $\top$ to an element $\mathbf{perm\_obj}_{i_m=m,i_{d_o}=d^{(o)},i_{d_{act}}=d^{(act)}}$ if a permission m is defined for the attribute value of object (i.e., object itself) $d^{(o)} \in D_1^{(o)}$ and for the attribute value of action (i.e., action itself) $d^{(act)} \in D_1^{(act)}$.

In this section we do not strictly keep the required order of indexes to simplify the discussion. This relaxation does not violate the computation, but only changes the order of indexes (which can be always changed by transposition).

First, consider Equation 19:

$$\mathbf{RULES\_RES}_{i_r,i_s,i_o,i_{act}} = ((\mathbf{RULES} \times \mathbf{ATTR}^{(s)}) \times \mathbf{ATTR}^{(o)}) \times \mathbf{ATTR}^{(act)}$$
$$= (\mathbf{RULES\_PERM} \times \overline{\mathbf{PERM\_OBJ}}_{i_m,i_o,i_{act}}) \times \mathbf{ATTR}^{(s)} \tag{23}$$

We removed *diag* operation, since there is only one dimension which has to be considered in all three cases (see Proposition 3). For representation reasons, we use the same matrix **PERM_OBJ** for the result of operation $(\mathbf{PERM\_OBJ} \times \mathbf{ATTR}^{(o)}) \times \mathbf{ATTR}^{(act)}$ denoted as $\overline{\mathbf{PERM\_OBJ}}_{i_m,i_o,i_{act}}$, since in fact, this operation does not change the matrix (because $\mathbf{ATTR}^{(o)}$ and $\mathbf{ATTR}^{(act)}$ are unit matrices), but simply renames the dimensions i_{d_o} to i_o and $i_{d_{act}}$ to i_{act}.

Now, we add the result of Equation 23 to Equation 15:

$$\mathbf{ACM} = (\mathbf{RULES\_PERM} \times \overline{\mathbf{PERM\_OBJ}}) \times \mathbf{ATTR}^{(s)}) \times \mathbf{PDP} \tag{24}$$
$$= (\mathbf{RULES\_PERM} \times \mathbf{PDP}) \times \mathbf{ATTR}^{(s)}) \times \overline{\mathbf{PERM\_OBJ}}$$

Let $\mathbf{ROLE\_PERM} = \mathbf{RULES\_PERM} \times \mathbf{PDP}$. We see that this two dimensional matrix assigns value $\top$ if there is a role-permission assignment, and $\varnothing$ otherwise.

In role engineering matrix $\overline{\mathbf{PERM\_OBJ}}$ is considered given. Now, let $\mathbf{ACM} = \mathbf{ACM}' \times \overline{\mathbf{PERM\_OBJ}}$, where $\mathbf{ACM}'$ is a two-dimensional boolean matrix which means that a subject has a permission. Then, we have, that:

$$\begin{aligned}\mathbf{ACM} &= \mathbf{ACM}' \times \overline{\mathbf{PERM\_OBJ}} \\ &= (\mathbf{RULES\_PERM} \times \mathbf{PDP}) \times \mathbf{ATTR}^{(s)}) \times \overline{\mathbf{PERM\_OBJ}} \qquad (25) \\ \mathbf{ACM}' &= \mathbf{ROLE\_PERM} \times \mathbf{ATTR}^{(s)} \qquad (26)\end{aligned}$$

Equation 26 is equivalent to RBAC model in a matrix form $\mathbf{UPA} = \mathbf{UA} \times \mathbf{PA}$ [2] which has a number of solutions presented in the literature [4–6].

6 Discussion

The first and the main point we would like to discuss is the complexity of the proposed approach. Indeed, usage of multidimensional matrices makes the computation and representation hard. The following observation is useful here. Looking to the matrixes required for role engineering to take into account one attribute (role) we see that there is a need to specify all subject-role and role-permission relations, which also highly increase complexity of the task, but this approach proved to be useful in practice. In this paper we proposed a technique, which should take into account any number of attributes. Thus, we should not be surprised of increased complexity. Moreover, the process of creation of such matrices should be automatic, rather than manual. Furthermore, computations can be significantly simplified by marking the dimensions, which contain only $\varnothing$ symbols. Thus, there is no need to compute every operation, but only meaningful ones.

One thing we did not consider in the paper is different environmental conditions. Thus, access may be allowed during the working hours, and forbidden in other time of the day. XACML uses environmental attributes together with attributes of subject, object and action for analysis of access requests. Such dimension can be easily added to the model using the same strategy we apply for subjects, objects, and actions.

XACML also assumes, that sometimes several subjects may be considered for one access request. In this case we cannot simply apply *diag* function in Equations 13, but must consider all subjects separately. In this article we do not consider this sophisticated case, but simply note, that our framework requires little changes for taking this possibility into account.

All in all, the proposed model is the first attempt, to our knowledge, to define the policy engineering problem for ABAC using a matrix form. Thus, we acknowledge that the proposed model may be simplified (e.g., in defining **RULES** matrix) especially, when specific cases of policy engineering problem for ABAC are used (e.g., when we model RBAC case). We also acknowledge that the model can be tuned to address the features of XACML more accurately (e.g., addition of policies and policy sets), but the current version had the main goal to take into account the core concept of ABAC.

7 Related Work

ABAC is a generalisation of traditional access control models [1]. It is capable to express complex security policies and it is resistant against scalability problems which occur when a number of subjects accessing a resource is enormous. The recent Usage Control (UCON) model [10, 11] is also an example of ABAC. The specific features of the UCON model are mutable attributes and continuous policy enforcement. The XACML framework [7], an open standard proposed by OASIS, is an example of application-independent ABAC for access control. In fact, XACML provides a language to express security policies and an enforcement architecture. Recently, XACML was extended in order to encode usage control policies and to support the continuous policy enforcement, i.e., it was extended to capture features of the UCON model [12].

There were several attempts to formalise ABAC. A logic-based formalisation of ABAC for access control is given in [13]. Crampton and Morisset proposed a formal language for ABAC that addresses the same problem space as XACML [14]. UCON formalisation based on temporal logic can be found in [15]. Martinelli et al. [16] proposed a formalisation based on a process algebra. Although these approaches are fruitful for automatic evaluation and enforcement of security policies, they are not suitable for management of attributes and for engineering of security policies. Management of mutable attributes in UCON was addressed in [17, 18]. Authors described how the decision making is affected by uncertain attribute values and how often mutable attributes should be refreshed.

Benefits, shortcomings, and open problems of ABAC models were surveyed in [19]. Attribute design and engineering of security policies were named as problems there. Our paper gives a first step towards defining and solving them.

We consider role mining in RBAC as a starting point. Indeed, the role can be just considered as an attribute, and roles to permissions assignments as a policy. Role mining, introduced in 2003 [3], gained a lot of attention in last years and a large number of different approaches to the problem were proposed [4–6, 20]. The authors of these approaches showed that it is convenient to use matrices for defining formally and solving automatically the role-mining problems.

8 Conclusions and Future Work

In this work we made the first steps towards defining access control policy engineering problem for ABAC. We proposed a matrix-based formalisation of the ABAC model. Our formalisation is based on the XACML standard, which should help adapting our findings in practice. We provided the basic attribute mining problem definition using our formalisation and showed how this bottom-up approach can be generalised for the policy engineering problem for ABAC.

In the paper we specified a large number of directions for future work: closer adaptation of the approach to XACML, considering policies and rules-combining algorithms; elaboration of policy engineering problem for ABAC; reducing the complexity of the model, etc. Naturally, solutions for the specified problem is the main future work we need to consider.

References

1. Jin, X., Krishnan, R., Sandhu, R.: A unified attribute-based access control model covering dac, mac and rbac. In: Proceedings of DBSec '12, Springer (2012) 41–55
2. Frank, M., Buhmann, J.M., Basin, D.: On the definition of role mining. In: Proceedings of SACMAT '10, ACM (2010) 35–44
3. Kuhlmann, M., Shohat, D., Schimpf, G.: Role mining - revealing business roles for security administration using data mining technology. In: Proceedings of SACMAT '03, ACM (2003) 179–186
4. Vaidya, J., Atluri, V., Guo, Q.: The role mining problem: Finding a minimal descriptive set of roles. In: Proceedings of SACMAT '07, ACM (2007) 175–184
5. Vaidya, J., Atluri, V., Guo, Q.: The role mining problem: A formal perspective. ACM TISSEC **13**(3) (2010) 27:1–27:31
6. Lu, H., Vaidya, J., Atluri, V., Hong, Y.: Constraint-aware role mining via extended boolean matrix decomposition. IEEE TDSC **9**(5) (2012) 655–669
7. OASIS: extensible access control markup language (xacml) version 3.0. Available via `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf` (January 2013)
8. Solo, A.M.G.: Multidimensional matrix mathematics. In: Proceedings of The World Congress on Engineering. Volume I., International Association of Engineers, Newswood Limited (2010) 1824–1850
9. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed nist standard for role-based access control. ACM TISSEC **4**(3) (2001) 224–274
10. Sandhu, R.S., Park, J.: Usage control: A vision for next generation access control. In: Proceedings of MMM-ACNS, Springer (2003) 17–31
11. Lazouski, A., Martinelli, F., Mori, P.: Usage control in computer security: A survey. Elsevier Computer Science Review **4**(2) (2010) 81–99
12. Lazouski, A., Mancini, G., Martinelli, F., Mori, P.: Usage control in cloud systems. In: Proceedings of ICITST '12, IEEE (2012) 202–207
13. Wang, L., Wijesekera, D., Jajodia, S.: A logic-based framework for attribute based access control. In: Proceedings of FMSE '04, ACM (2004) 45–55
14. Crampton, J., Morisset, C.: PTaCL: a language for attribute-based access control in open systems. In: Proceedings of POST'12, Springer (2012) 390–409
15. Zhang, X., Parisi-Presicce, F., Sandhu, R., Park, J.: Formal model and policy specification of usage control. ACM TISSEC **8**(4) (2005) 351–387
16. Martinelli, F., Mori, P., Vaccarelli, A.: Towards continuous usage control on grid computational services. In: Proceedings of ICAS-ICNS '05, IEEE (2005)
17. Krautsevich, L., Lazouski, A., Martinelli, F., Mori, P., Yautsiukhin, A.: Integration of quantitative methods for risk evaluation within usage control policies. In: Proceedings of ICCCN '13, *to appear*, IEEE (2013)
18. Krautsevich, L., Lazouski, A., Martinelli, F., Yautsiukhin, A.: Cost-effective enforcement of access and usage control policies under uncertainties. IEEE Systems Journal **7**(2) (2013) 223–235
19. Sandhu, R.S.: The authorization leap from rights to attributes: maturation or chaos? In: Proceedings of SACMAT '12, ACM (2012) 69–70
20. Colantonio, A., Pietro, R.D., Ocello, A., Verde, N.: Mining stable roles in RBAC. In Gritzalis, D., Lopez, J., eds.: Emerging Challenges for Security, Privacy and Trust. Volume 297 of IFIP Advances in Information and Communication Technology. Springer

Appendix

Proposition 4. *The operations defined in Definition 4:*

1. $*$ *and* $+$ *are commutative,*
2. $*$ *and* $+$ *are associative,*
3. $*$ *is distributive over* $+$*.*

Proof The commutative property of $+$ and $*$ follows from the symmetry of tables defined in Definition 4.

Now consider associative property of "+": a+(b+c)=(a+b)+c

- Let $a = \top$, then the result of the left part is $\top$ is annihilating element of "+". The right part is also $\top$, since now we apply annihilating element twice ($a + b$ and $((a + b) + c)$). Note, that this means that if any of the elements is $\top$ then the property holds.
- Let $a = \bot$ and $b \neq \top$ and $c \neq \top$. Then $\bot$ is an annihilating elements for the three values which are left and we have the same reasoning that we had for $a = 1$.
- Let $a = \boxtimes$ $b \neq \top$ $b \neq \bot$ and $c \neq \top$ and $c \neq \bot$. Now $\boxtimes$ is an annihilating element
- Let $a = b = c = \varnothing$. Trivial.

The proof for associative property of "*" is the same but we should do it other way round (start with $\varnothing$, which is annihilating element for "*").

Now we prove that $a * (b + c) = a * b + a * c$.

- Let $a = \top$. From the definition of multiplication operator we see that $1*d = d$ for any d. Thus, $a * (b + c) = b + c = a * b + a * c$).
- Let $a = \bot$ and $a * c$ and $a * b$ can be anything, but $\top$.
  - Let b be either $\top$ or $\bot$ then $a * b = \bot$. $\bot$ plus anything, but $\top$ is equals to $\bot$. Then, since $b + c = \bot$ or $b + c = \top$ then $a * (b + c) = \bot$. The same holds for $c = \top$ or $c = \bot$
  - Let $b = \boxtimes$. Then $a*b = \boxtimes$ and $b+c = \boxtimes$. Thus, $a*(b+c) = \boxtimes = a*b+a*c$. The same holds for $c = \boxtimes$
  - Let $b = c = \varnothing$. $a * (b + c) = \varnothing = a * b + a * c$.
- Let $a = \boxtimes$.
  - Let b be either $\top$ or $\bot$ or $\boxtimes$ then $a * b = \boxtimes$. Moreover, $b + c =$ is $\top$ or $\bot$ or $\boxtimes$ and $a * (b + c) = \boxtimes$. Since $a * c$ is either $\boxtimes$ or $\varnothing$, then $a * b + a * c = \boxtimes = a * (b + c)$. The same holds if c is either $\top$ or $\bot$ or $\boxtimes$;
  - Let $b = c = \varnothing$. $a * (b + c) = \varnothing = a * b + a * c$.
- Let $a = \varnothing$. From the definition of multiplication operator we see that $\varnothing * d = \varnothing$ for any d. Thus, taking into account that $\varnothing + \varnothing = \varnothing$ $a * (b + c) = \varnothing = a * b + a * c$).

□

Proposition 5. *Let us have three multidimensional matrices* $\boldsymbol{A}_{IJK}$, $\boldsymbol{B}_{IML}$, $\boldsymbol{C}_{JMN}$ *and* $IJK \bigcap IML \bigcap JMN = \emptyset$.

1. $(\boldsymbol{C} \times \boldsymbol{B}) \times \boldsymbol{A} = \boldsymbol{C} \times (\boldsymbol{B} \times \boldsymbol{A})$.
2. $(\boldsymbol{C} \times \boldsymbol{B}) \times \boldsymbol{A} = ((\boldsymbol{C} \times \boldsymbol{A}) \times \boldsymbol{B})^{T(K;L)}$, *where* $\boldsymbol{D}^{T(K;L)}$ *means transposition, i.e., interchanging of positions, of indexes from set K and L preserving order.*

Proof Let the result of $(\mathbf{C} \times \mathbf{B}) \times \mathbf{A}$ be $\mathbf{D}$.

$\mathbf{d}_{\overline{NLK}} = \sum_{IJ}(\sum_{M}(\mathbf{c}_{JMN} * b_{IML}) * \mathbf{a}_{IJK}) = \sum_{IJM}((\mathbf{c}_{JMN} * \mathbf{b}_{IML}) * \mathbf{a}_{IJK})$ by distributive property of $*$ over $+$. Now, by the associative property of $*$ we have, that $\sum_{IJM}((\mathbf{c}_{JMN} * \mathbf{b}_{IML}) * \mathbf{a}_{IJK}) = \sum_{JM}(\mathbf{c}_{JMN} * \sum_{I}(\mathbf{b}_{IML} * \mathbf{a}_{IJK})) = \mathbf{d}_{\overline{NLK}}$ by distributive property of $*$ in reverse direction and commutative property of $+$. Thus, $\mathbf{D} = \mathbf{C} \times (\mathbf{B} \times \mathbf{A})$

We also see, that $\mathbf{d}_{\overline{NLK}} = \sum_{IJM}((\mathbf{c}_{JMN} * \mathbf{b}_{IML}) * \mathbf{a}_{IJK}) = \sum_{IJM}((\mathbf{c}_{JMN} * \mathbf{a}_{IJK}) * \mathbf{b}_{IML}) = \sum_{MN}(\sum_{J}(\mathbf{c}_{JMN} * \mathbf{a}_{IJK}) * \mathbf{b}_{IML}) = \mathbf{d'}_{\overline{NKL}}$. Thus, $\mathbf{D'}^{T(K;L)}_{\overline{NKL}} = ((\mathbf{C} \times \mathbf{A}) \times \mathbf{B})^{T(K;L)} = \mathbf{D}_{\overline{NLK}}$ □

Proposition 6. *If matrix* $\boldsymbol{A}_{IJK}$ *has only one dimension* $J = \{j_1\}$, *then* $diag_J(\boldsymbol{A}_{IJK}) = \boldsymbol{A}_{IJK}$

Proof Let $\mathbf{C} = diag_J(\mathbf{A}_{IJK})$, then $\mathbf{c}_{I\{j\}J} = \mathbf{a}_{I\{j_1=j\}K} = \mathbf{a}_{I\{j\}K}$. Thus, $\mathbf{C} = \mathbf{A}_{IJK}$ □