Blockchain Based Auditable Access Control For Business Processes With Event Driven Policies

Ahmed Akhtar, Masoud Barati, Basit Shafiq, Omer Rana,
Ayesha Afzal, Jaideep Vaidya, and Shafay Shamail

Abstract—The use of blockchain technology has been proposed to provide auditable access control for individual resources. Unlike the case where all resources are owned by a single organization, this work focuses on distributed applications such as business processes and distributed workflows. These applications are often composed of multiple resources/services that are subject to the security and access control policies of different organizational domains. Here, blockchains provide an attractive decentralized solution to provide auditability. However, the underlying access control policies may have event-driven constraints and can be overlapping in terms of the component conditions/rules as well as events. Existing work cannot handle event-driven constraints and also does not sufficiently account for overlaps leading to significant overhead in terms of cost and computation time for evaluating authorizations over the blockchain. In this work, we propose an automata-theoretic approach for generating a cost-efficient composite access control policy. We reduce this composite policy generation problem to the standard weighted set cover problem. We show that the composite policy correctly captures all the local access control policies and reduces the policy evaluation cost over the blockchain. We have implemented the initial prototype of our approach using Ethereum as the underlying blockchain and empirically validated the effectiveness and efficiency of our approach.

Index Terms—Blockchain, Business Processes, Workflows, Access Control, Event-driven Policies, Automata-theoretic Approach

1 Introduction

The emerging cloud and edge computing infrastructure has enabled the development of next-generation internet-centered distributed applications that are autonomous, cooperative, adaptive, evolvable, emergent, and trustworthy. Such Internet-centered distributed applications include business processes (BPs), distributed workflows, and web service mashups [1], [11], [19]. Since these applications are architected and developed using resources and services that may belong to different organizational domains, access to the underlying resources and services is governed by the security and access control policies of the respective resource owner domains [23], [26].

Access control ensures that resources are used only according to the access control policies defined by the resource owners. Typically, resources are protected by access control systems deployed within the organization. However, this does not directly provide auditability of the access control enforcement and also causes a significant organizational burden since the organizations now need to bear the overhead of configuration, deployment, and management of the system along with the hardware, software, and manpower cost. A recently proposed alternative solution is to use blockchain technology [17] for access control. The basic idea here is to transform the access control policy evaluation process into a completely distributed smart contract execution. With this transformation, access control enforcement is at the same time outsourced and auditable. However, existing solutions [17], [21] focus on the access control policy for individual resources and encode the access control policy for each resource within a single smart contract on the blockchain. The execution of this smart contract returns the access decision for that particular resource. This is quite expensive, and if all of the resources are within a single organization, issues of trust do not exist and alternative solutions exist (such as tamper-proof logs [25]) to provide auditability at a lower cost. On the contrary, distributed applications, including BPs and distributed workflows, are composed of multiple resources/services that are subject to the security and access control policies of different autonomous organizational domains. Consequently, each of these services in a BP will have its own access control policy and a corresponding smart contract to evaluate it. Here, utilizing a blockchain is very attractive since it provides auditability in a decentralized environment. However, directly using existing blockchain-based solutions to manage access control for such distributed applications requires evaluating the user's authorization separately for each service that needs to be accessed. This may have significant overhead in terms of cost and computation time for blockchain transactions, especially when the individual domain's access control policies are overlapping, resulting in repeated evaluations of these overlapping parts. Additionally, the access control policies of services may be context dependent and may have event-driven constraints [26], [27]. These event-driven constraints cannot be formulated as rules in simple predicate logic and require state-space models (e.g., automata, Petri nets, etc.)

- *A. Akhtar, B. Shafiq and S. Shamail are with the Department of Computer Science, Lahore University of Management Sciences, Lahore, Pakistan. E-mail: 16030059@lums.edu.pk, basit@lums.edu.pk, sshamail@lums.edu.pk*
- *M. Barati is with the School of Information Technology, Carleton University, Canada. E-mail: masoudbarati@cunet.carleton.ca*
- *O. Rana is with School of Computer Science & Informatics, Cardiff University, Cardiff, UK E-mail: ranaof@cardiff.ac.uk*
- *A. Afzal is with the Department of Computer Science, Air University, Pakistan. E-mail: ayesha@aumc.edu.pk*
- *J. Vaidya is with Rutgers University, Newark, NJ 07102. E-mail: jsvaidya@rbs.rutgers.edu*

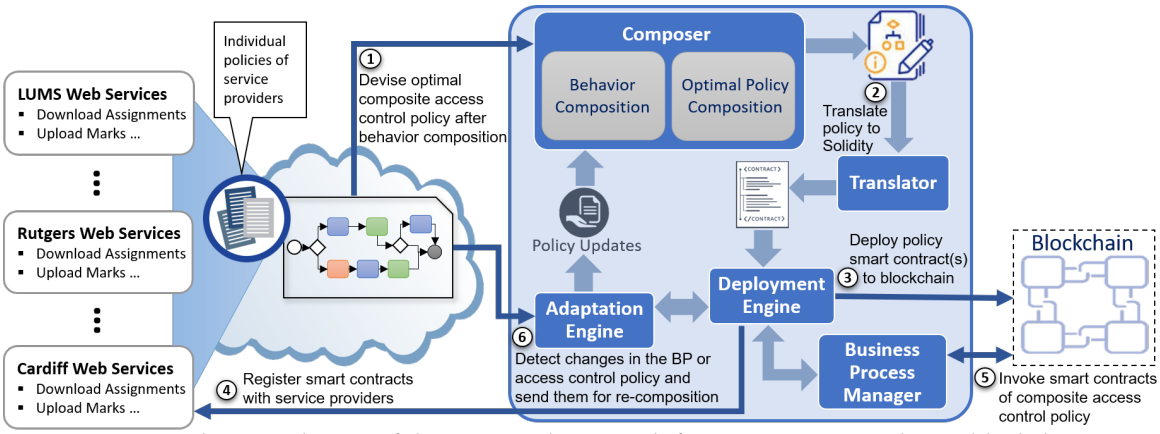

Fig. 1: Architectural view of the proposed approach for BP access control over blockchain

for their representation and analysis. The individual access control policies of all services involved in a BP may be overlapping in both attribute-based and event-driven policies. If each of the corresponding smart contracts evaluating the access control policy of every service is executed separately, the clauses/conditions in the overlapping parts will be re-evaluated, as many times as they appear in the individual access control policies. This re-evaluation of overlapping conditions results in a significant cost of policy evaluation for a blockchain-based access control system. This is due to the increased computation and storage requirements.

A composite access control policy can reduce this cost of evaluation for a blockchain-based access control system, by removing the repetitive evaluations. Such a plan should be designed keeping in view the access control policy needs of the particular BP in question, and it should be optimal with respect to computation and storage requirements.

This is precisely the problem that we address in this work. The goal is to save the cost by trying to combine the evaluation of multiple conditions, hence avoiding repeated evaluation. However, this is not always possible because the execution of the BP often depends on certain user-initiated events which cannot be predetermined, and in such cases, we want to defer the evaluation of the conditions to the point where they are needed to save cost by avoiding unnecessary evaluation. Specifically, we propose an approach to generate an efficiently evaluatable composite access control policy using the local access control policies of component services that include both attribute-based and event-driven policies. Since a BP and event-driven policies can be abstracted as automata, we can make use of a behavior composition framework to automatically compose the BP and event-driven policies for realizing a desired specification – called a target behavior [4], [6]. The framework uses a sound and complete automata-theoretic technique for synthesizing a controller/orchestrator that at each step of execution delegates a requested operation by the target to a proper available service. Fig. 1 provides an architectural overview of the proposed approach. The proposed approach is designed for a cloud services environment wherein the cloud service provider hosts the BP and is also responsible for access control management of the BP based on the local access control policies of component services.

We note that an initial study of this problem was conducted in [2] where we also proposed a blockchain-based solution for auditable evaluation of access control policies of distributed BPs. However, [2] only considered static policies, whereas this work can be used for event-driven policies which are dynamic in nature. The automata-theoretic approach of this work is also completely new and allows for the restructuring and optimization of smart contracts that reduce the overall evaluation cost.

Thus, the key contributions of this work are to:

- Propose an automata-theoretic approach for the generation of a cost-efficient access control policy for a BP that encapsulates the local access control policies of component services. The optimality criterion is based on the cost of policy evaluation over the blockchain.
- Reduce the problem of efficient evaluation of policy over the blockchain to the standard weighted set cover problem for which several approximation techniques exist.
- Implement and empirically validate the proposed approach using Ethereum's testnet Rinkeby.

The rest of the paper is organized as follows. Section 2 presents an illustrative example to explain the problem. Section 3 presents the preliminaries and the problem statement. Section 4 presents the methodology. Section 5 presents the experimental results. Section 6 discusses the related work in the literature. Finally, Section 7 concludes the paper and discusses future work.

2 Illustrative Example

Distributed applications requiring access to resources from different autonomous organizational domains can be represented as distributed BP workflows. The underlying access control policy of each resource may specify authorization rules based on user attributes (e.g., the user should be a graduate student having a CGPA of at least 3.5) or the occurrence of certain user-initiated events (e.g., the IT administrator will approve the request for access). A distributed BP workflow may require access to many such resources and hence it will be subject to multiple access control policies.

We now present an illustrative example in a virtual university context that serves to comprehensively illustrate the access control requirements in a multi-organizational business process (BP).

Example 1. Consider a virtual university environment in which three universities (e.g., LUMS, Rutgers, and Cardiff) collaborate to offer online courses to students. An instructor from any of these universities can teach

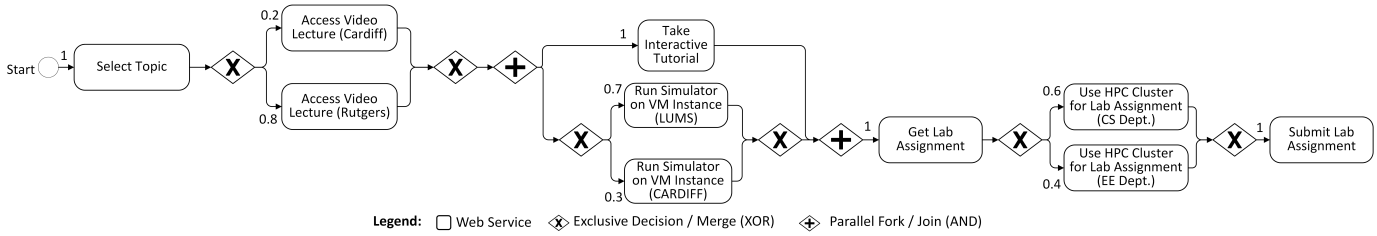

Fig. 2: Course assessment BP from virtual university domain

Select Topic: A **senior** student can select a topic.	**Access Video Lecture (Cardiff)**: A student belonging to a specific department, who has also studied a pre-requisite course, can access the video lecture from the digital library of Cardiff, only after the **approval of the digital library administrator**.	**Use HPC Cluster for LabAssignment(CSDept.)**: A **registered** graduate student working as an RA in an HEC-recognized university can access the HPC cluster of the CS department only after an **instructor designates them and provides approval** to access the cluster followed by the approval of the Computer Science administrator.	**Use HPC Cluster for LabAssignment(EEDept.)**: A **registered senior** PhD student can access the HPC cluster of the EE department only after an **instructor designates them and provides approval** to access the cluster followed by the approval of the Electrical Engineering administrator.
Access Video Lecture (Rutgers): A **senior** student working in the same area as the course can access the video lecture from the digital library of Rutgers only after the **approval of the digital library administrator**.			

Fig. 3: Access control policies of five services from course assessment BP

a course to the students of other universities. Also, resources from one university can be used for teaching courses to students of other collaborating universities. However, each university is an autonomous entity with its own policies governing access to its resources.

Fig. 2 shows a course assessment BP in which the student first goes through the lecture materials for a specific topic and then takes an interactive tutorial and completes a lab assignment for that topic. The first step in this BP is the selection of a topic followed by a video lecture on that topic served from the digital library of one of the different universities. After accessing the video lecture, the student starts an interactive tutorial about that topic. To complete this tutorial a simulator needs to be run on a VM instance on one of the available servers of different domains. After the tutorial, the student gets a lab assignment. To complete this assignment, the student needs to access one of the HPC clusters of different domains. Finally, the lab assignment is submitted.

The web services in this BP belong to different organizational domains and each of them has an access control policy associated with it. The access control policies of five of the services used in Example 1 are given in Fig. 3. Some access control policies depend upon the attributes of the user. For instance, the *Use HPC Cluster for Lab Assignment (CS Dept.)* service from the BP of Example 1 can only be accessed if the requesting person is a registered graduate student working as a research assistant in a university which is recognized by the Higher Education Commission (HEC). Such policies do not have any event-driven constraint associated with them and they give the same result upon each evaluation as long as the attributes themselves do not change. There are other access control policies that depend upon certain events and may give a different result upon each evaluation depending on the occurrence of the underlying events. For instance, there is an additional event-driven constraint on the policy for *Use HPC Cluster for Lab Assignment (CS Dept.)* service from the BP of Example 1. *HPC Cluster (CS Dept.)* is a shared resource for running batch processing jobs and it can only be accessed if a faculty member designates a user and provides approval to the user to access the cluster and after that, the computer science administrator based on the instructor's approval, provides approval to run the job.

The probabilities in Fig. 2 represent the likelihood of a service being called based on service availability statistics and/or execution paths.

2.1 Overlap in Access Control Policy Evaluation

Since the services in the BP are from different organizational domains, there can exist overlap between the underlying access control policies of the services. For instance, the condition that the requesting student should be a senior student is required by the access control policies of three services: i) *Select Topic*; ii) *Access Video Lecture (Rutgers)*; and iii) *Use HPC Cluster for Lab Assignment (EE Dept.)*. The condition that the student should be registered is also common across the access control policies of two services: i) *Use HPC Cluster for Lab Assignment (CS Dept.)*; and ii) *Use HPC Cluster for Lab Assignment (EE Dept.)*. Similarly, an overlap between the policy events is also present, like the approval of the administrator of the digital library is required by the policy for both services: i) *Access Video Lecture (Cardiff)*; and ii) *Access Video Lecture (Rutgers)*. The event of approval by the instructor is also common across the policies of the two services: i) *Use HPC Cluster for Lab Assignment (CS Dept.)*; and ii) *Use HPC Cluster for Lab Assignment (EE Dept.)*; however, subsequent events in their respective event-driven policies are different as they require approval from their respective department administrators.

2.2 Avoiding Re-evaluation of Overlapping Policies

The problem of increased blockchain cost due to the re-evaluation of overlapping policies can be solved by combining the conditions from policies of multiple services, to be

evaluated within a single smart contract while removing the duplicate conditions. This combined access control policy includes all the authorization rules of the individual policies of the combined services. However, depending on the BP execution state and event-driven constraints, only a few of these authorization rules may be relevant for a given user request. A naive approach would be to evaluate the user authorization against the combined access control policy and depending on the result of this evaluation, start the execution of the combined services. However, this may result in an unnecessary evaluation of user attributes and authorization rules of the combined access control policy in the case that the BP execution path and event-driven constraints render such attributes/authorization rules to be irrelevant. For instance, in the access control policies of the BP of Example 1 given in Fig. 3, the condition that the student should be working in the same area as the course, is required only if the user selects the service *Access Video Lecture (Rutgers)* and not otherwise. Similarly, the condition that the student belongs to an HEC-recognized university is only required if the event of the approval of the Computer Science administrator occurs in succession to the event of the approval of the instructor and not otherwise. Evaluating such conditions for paths on which they are not required, results in an increased blockchain cost due to their redundant evaluation. Also, since these paths depend upon user-initiated events occurring at runtime, they cannot be predetermined. It would therefore be beneficial to delay the evaluation of conditions to the point where they are really needed to avoid evaluating any unnecessary conditions.

A composite access control policy is needed which takes the above criteria into consideration for efficient access control policy evaluation over the blockchain.

3 Defining the Problem

We model the access control policies of service providers and the given BP as automata which enables us to verify the BP structure and constraints with respect to the service provider's policies. This representation also facilitates the efficient evaluation of policies by identifying and removing redundancies across the policies.

We use the following definition of automaton (similar to "Transition Systems" in action languages [10]) to model access control policies and BP.

Definition 1. An automaton is a tuple of $\langle \mathcal{A}, Q, q_0, F, \Phi, \Theta, \eta \rangle$, where $\mathcal{A}$ is a finite set of actions; Q is a finite set of states; $q_0 \in Q$ is the initial state; $F \subseteq Q$ is the set of final states; Φ is a set of guards over the actions; Θ is a set of state constraints over the actions; $\eta \subseteq Q \times \mathcal{A} \times 2^{\Phi} \times 2^{\theta} \times Q$ is the transition relation. The tuple $\langle q, a, \phi, \theta, q' \rangle \in \eta$ also denoted as $q \xrightarrow{a, \phi, \theta} q'$ is a transition from q to q' on action a with set of guards ϕ and a state constraint θ.

We represent the access control policies as automata given in Definition 1 by considering the set of actions $\mathcal{A}$ to represent the events of the policy and the set of guards Φ to represent the predicates of the attribute-based conditions of the policy. The set of state constraints Θ is empty for a policy automaton and will be useful when we define the BP automaton later. The automaton of the access control policy of *Use HPC Cluster for Lab Assignment (CS Dept.)* service of Example 1 is shown in Fig. 4. In this policy automaton, the set of events $\mathcal{A}$ includes *instructor approval* and *CS admin approval* while the set of guards Φ includes *isAnRA*, *isRegistered*, *isGradStudent* and *isUniHECRecognized*.

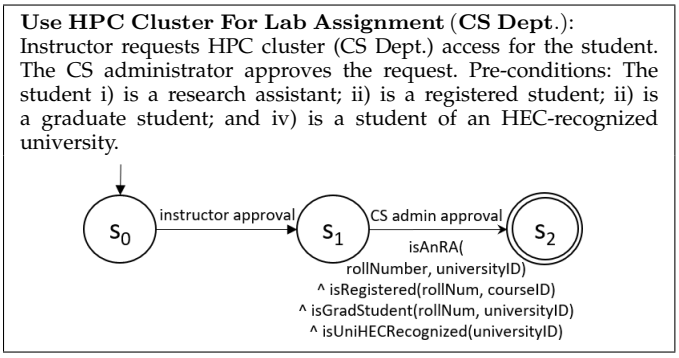

Fig. 4: Access control policy automaton of a service from the course assessment BP

We represent the BP as an automaton given in Definition 1 by considering the set of actions $\mathcal{A}$ to represent service selection or service execution events of the BP and the sets Φ and Θ, of guards and state constraints respectively, to represent the access control policies of service providers as pre-conditions to service execution events of the BP. Each of the guards from Φ specifies a predicate of the attribute-based conditions of the respective service providers policy and each of the constraints from Θ specifies a particular final state of the policy automaton representing an event-based condition of that policy. Table 1 shows the events associated with the BP of Example 1 and Fig. 5 shows its automaton along with the access control policies of underlying services. Note that events labeled with pe and ue are both user-initiated whereas events labeled with se are service-execution events. In this BP automaton the set of events $\mathcal{A}$ includes Execute *Select Topic* service, Select *Access Video Lecture (Cardiff)* service, Execute *Access Video Lecture (Rutgers)* service, etc., the set of guards Φ includes *isSenior*, *isSameArea*, *isGradStudent*, etc., while the set of state constraints Θ includes $FSM_1.state = q_1,\ FSM_2.state = p_2$, etc. Note that Definition 1 requires each transition to only have one state constraint (associated with a particular policy). If multiple state constraints need to be captured, it would be necessary to create a composite policy automaton encompassing all required state constraints, with a single final state, which would then allow expression with a single state constraint in accordance with Definition 1.

To verify the composability of the BP structure and the constraints with respect to the service provider's policies, a target behavior is specified. Let us consider k automata $\mathcal{T}_1, \mathcal{T}_2, \cdots \mathcal{T}_k$ such that $\forall\, i \in [1..k]\ \mathcal{T}_i = \langle \mathcal{A}_i, Q_i, q_{0_i}, F_i, \Phi_i, \Theta_i, \eta_i \rangle$. $\mathcal{T}_1$ is the BP automaton and $\mathcal{T}_2, \cdots \mathcal{T}_k$ are the policy automata. Then the target behavior is defined as follows:

Definition 2. The *target behavior* $\mathcal{T}_T$ is an automaton obtained using the BP automaton $\mathcal{T}_1$ and the policy automata $\mathcal{T}_2, \cdots \mathcal{T}_k$ and is a tuple:

$$\mathcal{T}_T = \langle \mathcal{A}_T, Q_T, q_{0_T}, F_T, \Phi_T, \eta_T \rangle, \text{ where}$$

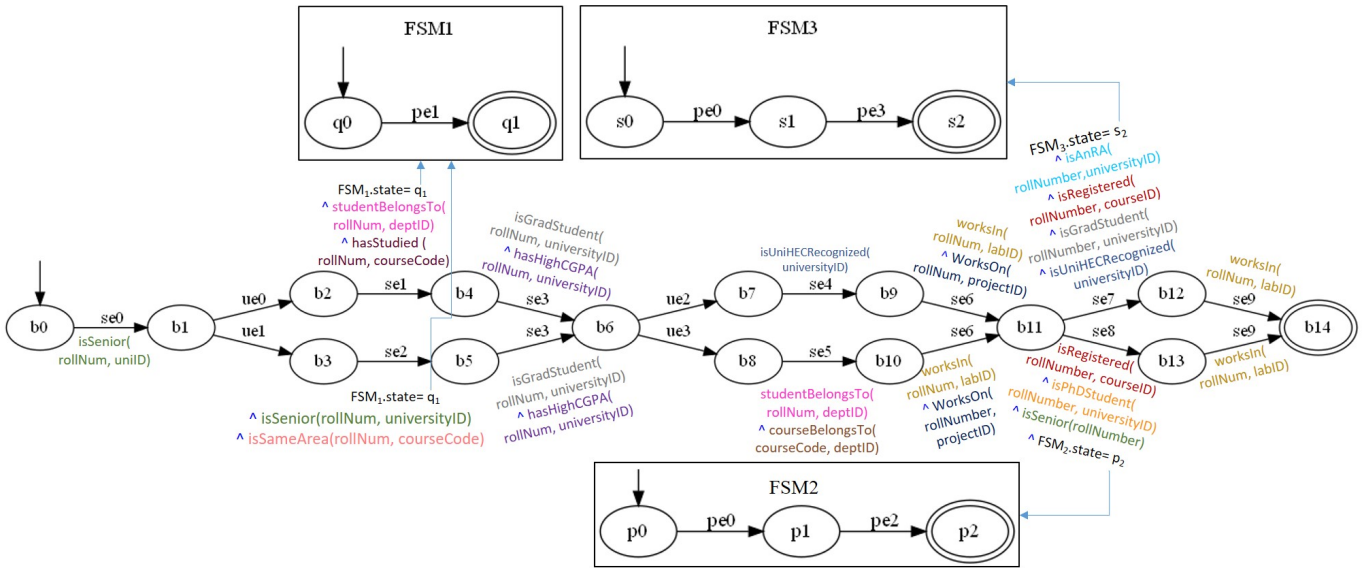

Fig. 5: BP automaton annotated with policies of underlying services

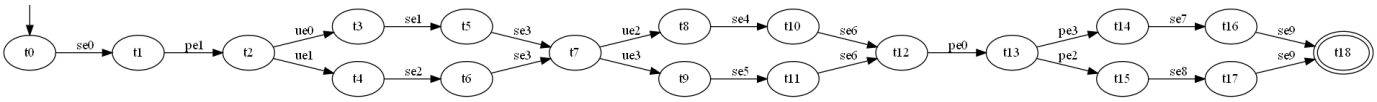

Fig. 6: Target Behavior

TABLE 1: All events associated with the automaton for the course assessment BP

Event	Description
pe0	Instructor Approval
pe1	Digital Library Admin Approval
pe2	Electrical Engineering Admin Approval
pe3	Computer Science Admin Approval
ue0	Select *Access Video Lecture (Cardiff)* service
ue1	Select *Access Video Lecture (Rutgers)* service
ue2	Select *Run Simulator on VM Instance (LUMS)* service
ue3	Select *Run Simulator on VM Instance (Cardiff)* service
se0	Execute *Select Topic* service
se1	Execute *Access Video Lecture (Cardiff)* service
se2	Execute *Access Video Lecture (Rutgers)* service
se3	Execute *Take Interactive Tutorial* service
se4	Execute *Run Simulator on VM Instance (LUMS)* service
se5	Execute *Run Simulator on VM Instance (Cardiff)* service
se6	Execute *Get Lab Assignment* service
se7	Execute *Use HPC Cluster for Lab Assignment (CS Dept.)* service
se8	Execute *Use HPC Cluster for Lab Assignment (EE Dept.)* service
se9	Execute *Submit Lab Assignment* service

- $\mathcal{A}_T = \bigcup_{1 \le i \le k} \mathcal{A}_i$ is a finite set of events
- $Q_T = \mathcal{Q}_1 \bigcup \left[\bigcup_{2 \le i \le k} (\mathcal{Q}_i \setminus q_{0_i}) \right]$ is a finite set of states
- $q_{0_T} = q_{0_1}$ is the initial state
- $F_T = F_1$ is the set of final states
- $\Phi_T = \bigcup_{1 \le i \le k} \Phi_i$ is a set of guards over the events
- $\eta_T \subseteq Q_T \times \mathcal{A}_T \times 2^{\Phi_T} \times Q_T$ is the transition relation where transition $(\sigma,\ a,\ \phi,\ \sigma') \in \eta_T$, also denoted as, $\sigma \xrightarrow{a,\phi} \sigma'$ is in $\mathcal{T}_T$, if one of the following is true:
  - $\exists\, \sigma \xrightarrow{a,\phi,\theta} \sigma'$ in $\mathcal{T}_1$ s.t. $\theta = \varnothing$
  - $\exists\, \sigma \xrightarrow{a,\phi,\theta} \sigma'$ in $\mathcal{T}_i$, where $i \in [2..k]$ s.t. $\sigma \neq q_{0_i}$
  - $\exists\, \sigma \xrightarrow{a',\phi',\theta} \omega$ in $\mathcal{T}_1$ s.t. θ contains a state constraint referring to a final state of policy automaton $\mathcal{T}_j$, where $j \in [2..k]$ and $q_{0_j} \xrightarrow{a,\phi,\theta'} \sigma'$ exists in $\mathcal{T}_j$
  - $\exists\, \omega' \xrightarrow{a,\phi,\theta} \sigma'$ in $\mathcal{T}_1$ s.t. θ contains a state constraint referring to σ, where $\sigma \in F_l$ is a final state of policy automaton $\mathcal{T}_l$, where $l \in [2..k]$

The target behavior $\mathcal{T}_T$ is specified by merging the policy automata into the BP automaton while ensuring that the event dependencies between the policy events and service events are preserved. We do this by directly inserting each policy automaton in the BP automaton before the transition which contains a guard referring to the respective policy automaton. The target behavior for the BP of Example 1 is shown in Fig. 6. Here, note that the policy event *pe1* (Digital Library Admin Approval) comes before the transitions containing the service events *se1* (Execute *Access Video Lecture (Cardiff)* service) and *se2* (Execute *Access Video Lecture (Rutgers)* service) because both of these service events have the guard $FSM_1.state = q_1$ associated with them in the BP automaton. Similar observations can be made about the target behavior for all guards in the transitions of the BP automaton referring to some policy automaton.

Problem Statement: *Given a BP automaton $\mathcal{T}_1$ and automata of policies $\mathcal{T}_2 \cdots \mathcal{T}_k$, compose and restructure the policy conditions to minimize the enforcement cost over the blockchain while ensuring correctness in terms of the policy and BP constraints.*

4 Methodology

The architecture of the proposed approach for finding a cost-efficient access control policy consists of five major components as depicted in Fig. 1. For a given BP and associated access control policies, the *Composer* performs

behavior composition and uses the behavior composition model obtained to generate a cost-efficient access control policy. The *Translator* is responsible for transforming the composite access control policy into one or more smart contract(s). The *Deployment Engine* deploys these smart contracts over the blockchain, on which they are executed to determine the access control decision for any given user request. The composite access control policy may split the individual service provider's policy into multiple smart contracts. Moreover, a single smart contract may be part of multiple service providers' policies. The deployment engine creates a mapping between each web service and the set of smart contracts satisfying its access control policy. The deployment engine also registers the associated smart contracts for each web service with the corresponding service provider. The service provider may verify that the associated smart contracts satisfy its local access control policy. The *BP Manager (BPM)* is responsible for the evaluation and enforcement of the composite access control policy through the smart contracts. The *Adaptation Engine* monitors changes in the service providers' policies and triggers revision of the composite access control policy with minimum changes.

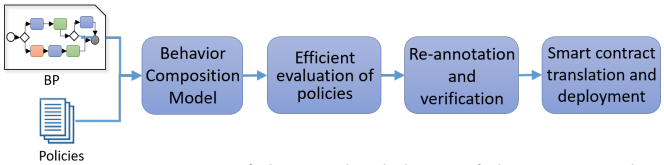

Fig. 7: Overview of the methodology of the proposed approach for BP access control over blockchain

The overall methodology for the generation of a composite access control policy for efficient evaluation of user authorizations for a given BP is shown in Fig. 7. The execution of the BP depends upon user-initiated events in the BP and access control policies. For this reason, we first generate a behavior composition model which represents the BP and all the corresponding access control policies of the component services (Section 4.1). We then reduce the policy evaluation cost over blockchain by combining multiple access control conditions to avoid repeated evaluation of overlapping access control policies to obtain a cost-efficient access control policy. Specifically, we reduce the problem of efficient evaluation of policy over the blockchain to the standard weighted set cover problem for which several approximation techniques exist [5] (Section 4.2). After obtaining a cost-efficient policy, we re-annotate the behavior composition model with it for formal verification to ensure conformance with the access control policies of all component services (Section 4.3). Now, the groups of conditions are encoded into smart contracts and deployed on the blockchain for evaluation, enforcement, and auditing purposes (Sections 4.4, 4.5 and 4.6). Below, we discuss each step in detail.

4.1 Behavior Composition of BP with Policies

Note that the execution path of the BP depends on the role and/or attributes of the user as well as the user-initiated events that occur at runtime and cannot be predetermined. For this reason, we first generate a behavior composition model which represents the BP and all the corresponding access control policies of the component services. To generate the behavior composition model we need the system behavior, which is a combined automaton of the BP automaton and the policy automata and is modeled as a direct product automaton which is defined as follows:

4.1.1 System Behavior

Let us consider a direct product automaton that combines k automata, each of which follows Definition 1. For each $i \in [1 \dots k]$, let $\mathcal{A}_i$ be a finite set of events. Q_i is a finite set of states; $q_{0_i} \in Q_i$ is an initial state; η_i is the transition relation. Note that the global set of events is given by $\mathcal{A}_S = \bigcup_{1 \le i \le k} \mathcal{A}_i$. For each event $a \in \mathcal{A}_S$, the automata in which a can be found is given by the set $loc(a) = \{i \mid a \in \mathcal{A}_i\}$.

Definition 3. The *direct product automaton* of the BP automaton and the policy automata gives us the system behavior $\mathcal{T}_S$ which is a tuple:

$$\mathcal{T}_S = \langle \mathcal{A}_S, Q_S, q_{0_S}, F_S, \Phi_S, \eta_S \rangle, \text{ where}$$

- $\mathcal{A}_S = \bigcup_{1 \le i \le k} \mathcal{A}_i$ is a finite set of shared events
- $Q_S = Q_1 \times Q_2 \times \cdots \times Q_k$ is a finite set of states
- $q_{0_S} = q_{0_1} \times q_{0_2} \times \cdots \times q_{0_k} \in Q_S$ is the initial state
- $F_S \subseteq Q_S$ is the set of final states
- $\Phi_S = \bigcup_{1 \le i \le k} \Phi_i$ is a set of guards over the events
- $\eta_S \subseteq Q_S \times \mathcal{A}_S \times 2^{\Phi_S} \times Q_S$ is the transition relation defined below:
  $\forall \langle q_1, q_2, \dots, q_k \rangle, \langle q'_1, q'_2, \dots, q'_k \rangle \in Q_S, \exists$ a transition $(\langle q_1, q_2, \dots, q_k \rangle, a, \phi, \langle q'_1, q'_2, \dots, q'_k \rangle) \in \eta_S$, also denoted as, $\langle q_1, q_2, \dots, q_k \rangle \xrightarrow{a, \phi} \langle q'_1, q'_2, \dots, q'_k \rangle$ in $\mathcal{T}_S$, if and only if
  - For each $j \in loc(a),\ q_j \xrightarrow{a, \phi_j, \theta_j} q'_j \in \eta_j$
  - For each $j \notin loc(a),\ q_j = q'_j$

  where, $\phi = \bigcup_{j \in loc(a)} \phi_j$

4.1.2 Behavior Composition Model Generation

As defined above, the system behavior $\mathcal{T}_S$ is the direct product automaton and therefore is likely to contain a large number of extraneous states and transitions that are not relevant to the target behavior $\mathcal{T}_T$. Therefore, we perform behavior composition [6] given in Algorithm 1 to obtain a mapping $\mathcal{R}$ which contains only the relevant states and transitions.

Now, $\mathcal{R}$ can be used to obtain a reduced automaton from the system automaton to realize the target behavior. This reduced automaton is called the behavior composition model and is formally defined below.

Definition 4. The *behavior composition model* $\mathcal{T}_B$ is an automaton which is obtained using $\mathcal{T}_T$, $\mathcal{T}_S$ and $\mathcal{R}$ is a tuple:

$$\mathcal{T}_B = \langle \mathcal{A}_B, Q_B, q_{0_B}, F_B, \Phi_B, \eta_B \rangle, \text{ where}$$

- $\mathcal{A}_B = \mathcal{A}_S$ (also $= \mathcal{A}_T$) is the finite set of shared events
- $Q_B = \{(q_T, q_S) \in Q_T \times Q_S \mid (q_T, q_S) \in \mathcal{R}\}$ is the set of states, formed by all pairs of $\mathcal{T}_T$ and $\mathcal{T}_S$ states belonging to the largest ND-simulation relation $\mathcal{R}$; given $\sigma = (q_T, q_S)$ we denote q_T by $part_T(\sigma)$ and q_S by $part_S(\sigma)$
- $q_{0_B} = (q_{0_T}, q_{0_S}) \in Q_B$ is the initial state
- $F_B \subseteq Q_B$ is the set of final states
- Φ_B is a set of guards over the events

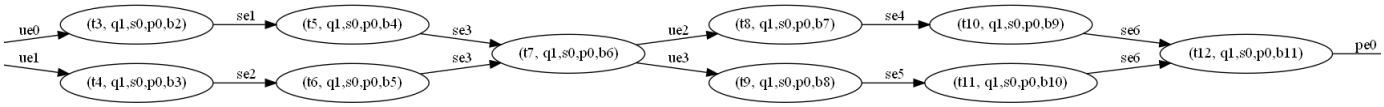

Fig. 8: A fragment of the behavior composition model for the course assessment BP

Algorithm 1 $NDS(\mathcal{T}_T, \mathcal{T}_S)$ - Largest Non-Deterministic Simulation to obtain mapping

Input: $\mathcal{T}_T$, $\mathcal{T}_S$
Output: $\mathcal{R}$

1: $\mathcal{R} := Q_T \times Q_S \setminus (q_T, q_S) \,|\, q_T \in F_T \wedge q_S \notin F_S$
2: **repeat**
3: $\mathcal{R} := (\mathcal{R} \setminus C)$, where C is the set $(q_T, q_S) \in \mathcal{R}$ such that for event $a \in \mathcal{A}_S$ and its associated set of guards ϕ there exists a transition $q_T \xrightarrow{a,\phi} q'_T$ in $\mathcal{T}_T$ such that either:
(a) there is no transition $q_S \xrightarrow{a,\phi} q'_S$ in $\mathcal{T}_S$ or
(b) there exists a transition $q_S \xrightarrow{a,\phi} q'_S$ in $\mathcal{T}_S$ but $(q'_T, q'_S) \notin \mathcal{R}$.
4: **until** $(C = \varnothing)$
5: **return** $\mathcal{R}$

- $\eta_B \subseteq Q_B \times \mathcal{A}_B \times 2^{\Phi_B} \times Q_B$ is the transition relation where $\exists\ (\sigma,\ a,\ \phi,\ \sigma') \in \eta_B$, also denoted as, $\sigma \xrightarrow{a,\phi} \sigma'$ in $\mathcal{T}_B$, if and only if:
  - there exists:
    - a transition $part_T(\sigma) \xrightarrow{a,\phi} part_T(\sigma')$ in $\mathcal{T}_T$
    - a transition $part_S(\sigma) \xrightarrow{a,\phi} part_S(\sigma')$ in $\mathcal{T}_S$
  - $\forall\ \sigma'' \in Q_T \times Q_S$ such that $part_T(\sigma) \xrightarrow{a,\phi} part_T(\sigma'')$ in $\mathcal{T}_T$, $part_S(\sigma) \xrightarrow{a,\phi} part_S(\sigma'')$ in $\mathcal{T}_S$, it is the case that $(part_T(\sigma''),\ part_S(\sigma'')) \in Q_B$

Since the behavior composition model represents the entire set of policies as well as the BP in a single automaton, it can be used to compose and restructure the policy conditions to minimize the enforcement cost over the blockchain while ensuring correctness in terms of the policy and BP constraints, as further discussed below. Fig. 8 depicts a fragment of the behavior composition model for the course assessment BP obtained using Definition 4.

4.2 Efficient Evaluation of Policies

As discussed earlier, the access control policies of web services in a BP may be overlapping. One way to avoid repeated evaluation of overlapping conditions is to group the conditions of the access control policies of multiple services and consequently evaluate the repeated conditions only once. However, such grouping across all services of the BP may not be possible because there are user-initiated events that cannot be predetermined, as their occurrence time is not known a priori. Our objective is to save policy evaluation cost by avoiding unnecessary evaluation by deferring the evaluation of the conditions to the point where they are needed.

For this reason, we first partition the behavior composition model over the user-initiated events (Section 4.2.1). We then reduce the repeated evaluation by grouping the conditions of the access control policies of all the services in a partition and restructuring the policies across partitions. For a given partition, we identify all the duplicate conditions that have been pre-evaluated in prior partitions and remove them from the current partition. Moreover, the conditions that have to be evaluated on all subsequent paths leading from the current partition are pre-evaluated in the current partition. After restructuring the policy, the conditions to be evaluated for a given partition may be encoded as a single smart contract or multiple smart contract(s). While encoding the restructured access control policy using a single smart contract for each partition (Section 4.2.2) reduces the repeated policy evaluation, due to the presence of multiple execution paths in the BP, we cannot completely eliminate the repeated policy evaluation. Using multiple smart contracts can further reduce the repeated policy evaluation. This splitting of the conditions into multiple smart contracts considers the cost of policy evaluation which depends on the probability of the execution path taken in the BP. We develop a heuristic to optimize the cost for each partition to reduce the overall cost. We do this by selecting the least cost configuration out of multiple smart contract configurations for each partition to obtain a cost-efficient access control policy. This heuristic is run iteratively for each partition (Section 4.2.3). We now discuss these steps in detail.

4.2.1 Partitioning of the behavior composition model

User-initiated events in the BP and access control policies can be used to partition the behavior composition model. Accordingly, the user authorization for the BP will be evaluated for only those partitions that are reachable from the current state of the behavior composition model encoding the BP structure and access control policies.

In order to partition, we take the transition set η_B of the behavior composition model $\mathcal{T}_B$ and remove the user-initiated event transitions from it to obtain the reduced transition set η'_B:

$$\eta'_B = \eta_B \setminus U \text{ where,}$$
$$U = \{x \mid x \in \eta_B, x \textit{ is based on a user-initiated event}\}$$

This essentially results in multiple connected components each of which is a fragment of the behavior composition model. We term each of these connected components as partitions. Since the removal of a specific user-initiated event transition results in the creation of a specific partition (which follows the transition), we label that partition with the event transition. The first partition (which does not have a preceding user-initiated event transition) is labeled as the root partition. We use $\mathcal{P}_{\mathcal{T}_B}$ to denote the set of these partitions. Note that we are only concerned with the transitions within each partition. We stress that while there may be an overlap between the states in different partitions, there is no overlap between the transitions in different partitions.

Fig. 9 shows the behavior composition model $\mathcal{T}_B$, of the BP of Example 1, marked along with the partitions and

TABLE 2: Categories of access control policy conditions

Name	Description
$SubsequentDefinitiveConditions(\mathcal{T}_B, p)$	Subsequent definitive conditions include all the (non mutually exclusive) conditions that will be evaluated along all the execution paths originating from the current partition p in $\mathcal{T}_B$.
$PriorDefinitiveConditions(\mathcal{T}_B, p)$	Prior definitive conditions include all the conditions that have already been evaluated along all the execution paths leading to the current partition p in $\mathcal{T}_B$.
$SubsequentPartialConditions(\mathcal{T}_B, p)$	Subsequent partial conditions include all the conditions that will be evaluated along some (but not all) the execution paths originating from the current partition p in $\mathcal{T}_B$.
$PriorPartialConditions(\mathcal{T}_B, p)$	Prior partial conditions include all the conditions that have already been evaluated along some (but not all) the execution paths leading to the current partition p in $\mathcal{T}_B$.
$baseConditions(\mathcal{T}_B, p)$	Base conditions include all the conditions to be evaluated for the partition p in $\mathcal{T}_B$ which do not overlap with either $PriorDefinitiveConditions(\mathcal{T}_B, p)$ or $PriorPartialConditions(\mathcal{T}_B, p)$

annotated with the access control policies of underlying services. Note that the transitions in the partitions therein do not include any user-initiated event transition.

4.2.2 Single smart contract per partition

Once the partitions are created, we first consider the approach of encoding the conditions to be evaluated for each partition as a single smart contract. We try to reduce the repeated evaluation of overlapping conditions of the access control policies across the partitions $\mathcal{P}_{\mathcal{T}_B}$ of the behavior composition model $\mathcal{T}_B$ to obtain the restructured policy. The evaluation of these conditions depends upon the execution path of the behavior composition model $\mathcal{T}_B$, taken during a particular execution of the BP. Therefore, we categorize the access control policy conditions, represented as guards in the policy automata given in Definition 1, into five types, which are given in Table 2 and our approach to deal with each of them is discussed below.

Given a current partition p of the behavior composition model $\mathcal{T}_B$:

- $SubsequentDefinitiveConditions(\mathcal{T}_B, p)$ need to be evaluated for all paths originating from p and they can be pre-evaluated in p.
- $PriorDefinitiveConditions(\mathcal{T}_B, p)$ have already been evaluated before reaching p and do not need to be re-evaluated in p.
- $SubsequentPartialConditions(\mathcal{T}_B, p)$ should not be pre-evaluated in p and their evaluation is deferred until the partition containing them is reached.
- $PriorPartialConditions(\mathcal{T}_B, p)$ may or may not have been pre-evaluated upon reaching p depending on the execution path leading to p. Therefore, we need to evaluate these conditions along with the $baseConditions(\mathcal{T}_B, p)$ to check the user authorizations for the services in p. These conditions can be grouped together with the base conditions in a single or multiple smart contract(s) for evaluation, depending on their pre-evaluation probability as well as the cost of the resulting smart contracts. We formulate the problem of grouping these conditions with the base conditions of p to reduce their evaluation cost over the blockchain in Section 4.2.3

Algorithm 2 uses the first two types of conditions, given in Table 2, to generate the restructured policy. We visit each partition p of the behavior composition model $\mathcal{T}_B$ starting from the *root* partition (line 1). We take the union of the set of guards of all transitions in p to collect the relevant conditions of the access control policy for p (line 2). Then we look at the transitions subsequent to p to find all definitive conditions (see Table 2) (line 3). Then we merge these subsequent definitive conditions to the conditions to be evaluated for p and remove them from the partitions subsequent to p (lines 4 and 5). We also remove the prior definitive conditions (see Table 2) from the conditions to be evaluated for p (line 6). Lastly, we add the conditions to be evaluated for p as guards of the first transition of p (line 7). We return the behavior composition model re-annotated with the restructured policy (line 9). Fig. 10 shows the restructured behavior composition model $\mathcal{T}'_B$ marked along with the identified partitions.

Algorithm 2 Policy Restructuring

Input: $\mathcal{T}_B$ - behavior composition model annotated with service provider policies
Input: $\mathcal{P}_{\mathcal{T}_B}$ - partitions of $\mathcal{T}_B$
Output: $\mathcal{T}'_B$ - restructured behavior composition model

1: **for** each partition $p \in \mathcal{P}_{\mathcal{T}_B}$ **do**
2: $AC_p \leftarrow ConditionsInPartition(\mathcal{T}_B, p)$
3: $D \leftarrow SubsequentDefinitiveConditions(\mathcal{T}_B, p)$
4: $AC_p \leftarrow AC_p \cup D$
5: $removeConditions(\mathcal{T}_B,\ D)$
6: $AC_p \leftarrow AC_p \backslash PriorDefinitiveConditions(\mathcal{T}_B, p)$
7: $firstTransition(p).\phi \leftarrow AC_p$
8: **end for**
9: **return** $\mathcal{T}'_B$

4.2.3 Multiple smart contracts per partition

Once we have the restructured policy, we can optimize the evaluation of the set of conditions for each partition, by considering the approach of encoding multiple smart contracts, to obtain a cost-efficient policy. Note that within the restructured policy, there can still be cases where some conditions may have been pre-evaluated when a partition is reached. This is specifically the case with prior partial conditions given in Table 2 (depicted as underlined conditions in Fig. 10). However, we cannot completely omit

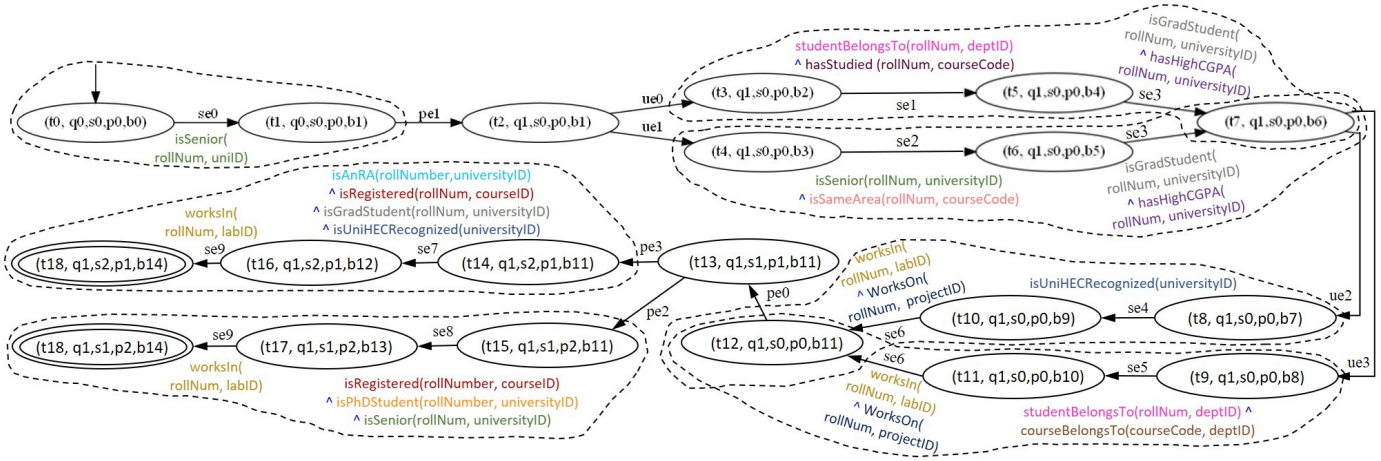

Fig. 9: Behavior composition model with identified partitions and annotated with service providers' access control policies

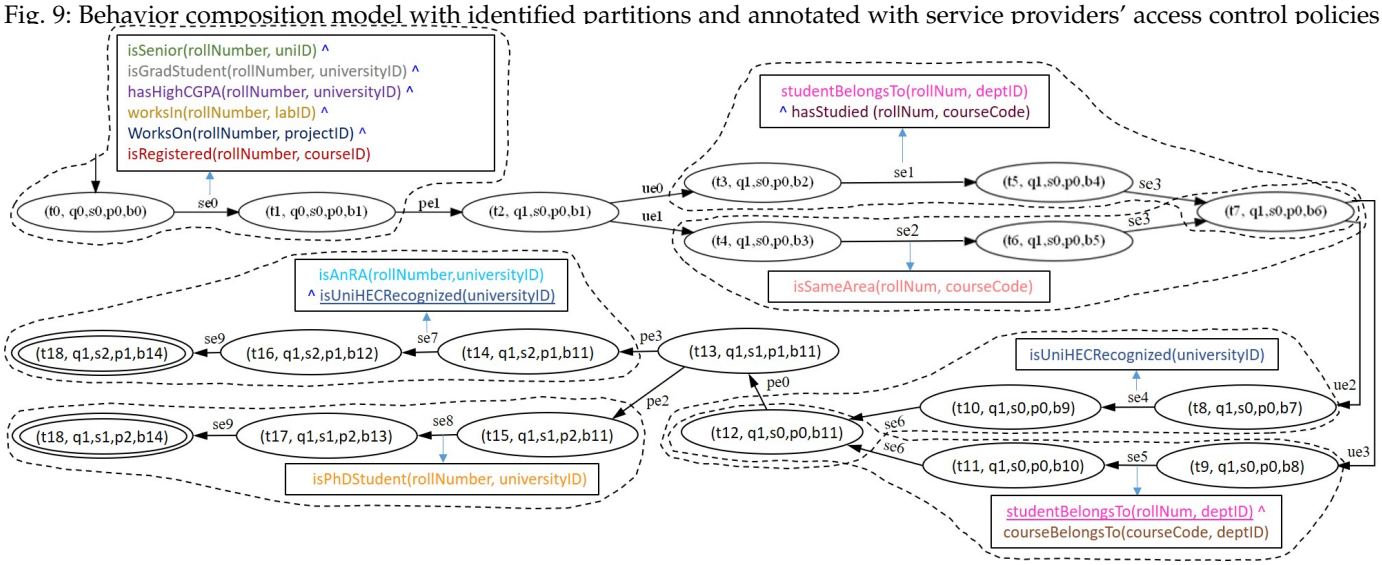

Fig. 10: Behavior composition model with identified partitions and annotated with restructured access control policy

these conditions from the smart contract of the current partition because it is uncertain if they have already been evaluated. Instead, we can make a decision of whether to keep such conditions within the smart contract of the current partition to be evaluated regardless of the previous path taken to reach the current partition. For example in Fig. 10, the prior partial condition $isUniHECRecognized$ in partition $pe3$ can be evaluated in the smart contract of $pe3$ regardless of the previous path taken to reach $pe3$. Alternatively, they can be removed from the smart contract of the current partition, while ensuring that they are only evaluated if the path having a partition containing them was not taken to reach the current partition. That is, the condition $isUniHecRecognized$ can be removed from the smart contract of partition $pe3$ and it can be ensured that it is only evaluated if the path containing $ue2$ is not taken to reach $pe3$. In this case, the missed conditions can either be evaluated as a separate smart contract or as part of the smart contract of the previous partition lying on the path that was not taken to reach the current partition. That is, the condition $isUniHECRecognized$ can be evaluated as part of a separate smart contract or as part of the smart contract of partition $ue2$, the path of which was not taken to reach $pe3$.

Ideally, the decision of how to evaluate the aforementioned conditions needs to be made after exploring the cost of all possible combinations across all the partitions of the restructured behavior composition model $\mathcal{T}'_B$. However, this may not be a viable solution because the execution path of the behavior composition model depends on user-initiated events in the BP and access control policies. These user-initiated events occur at runtime and cannot be predetermined. Therefore, we devise a heuristic that iteratively solves an optimization problem for each partition of the restructured behavior composition model to reduce the overall cost of policy evaluation on the blockchain, based on the probabilities of the BP execution paths. The resulting policy gives us an improvement, in terms of cost, over the restructured policy. We use the following notations before discussing this heuristic:

Given a partition p of the restructured behavior composition model $\mathcal{T}'_B$ let,

- AC_p be all the access control policy conditions required to check the authorization for all the services in p
- $BC_p \subseteq AC_p$ be the $baseConditions(\mathcal{T}'_B, p)$ (see Table 2)
- $OC_p \subseteq AC_p$ be the overlapping conditions between AC_p and the prior partial conditions (see Table 2) i.e.

$$OC_p = AC_p \cap PriorPartialConditions(\mathcal{T}'_B, p) \quad (1)$$

We consider two types of smart contracts for each partition:

- Integral smart contract
- Additional smart contract

There is one integral smart contract for each partition which is always executed when that partition is reached. The integral smart contract contains all the base conditions BC_p along with one or more of the overlapping conditions OC_p. There can be multiple additional smart contract(s) for each partition which may not be always executed. The additional smart contract(s) are composed of different combinations of the overlapping conditions OC_p. An additional smart contract for a partition is executed only if the underlying conditions have not been pre-evaluated when that partition is reached. For a given partition p, the leftover conditions (which are not part of the integral smart contract or any of the additional smart contract(s) for p) are evaluated as part of the smart contract of some previous partition, only if they have not been pre-evaluated when p is reached.

We formulate the problem of selecting the overlapping conditions OC_p for the integral smart contract or any of the additional smart contract(s) as a partition-specific policy evaluation (PSPE) problem. We call the policy obtained as a solution to the PSPE problem, the PSPE policy. The PSPE problem can be reduced to the standard weighted set cover problem. The weighted set cover problem is an NP-complete problem. However, several approximation techniques exist for finding a near-optimal solution to this problem [5]. For this formulation, we define three sets of groupings of the overlapping conditions OC_p using the following notations: Let,

- $\overline{U}$ denote the set of all possible combinations of the integral smart contract for a partition p i.e.

$$\overline{U} = \bigcup_{\forall\, O \in 2^{OC_p}} \{BC_p \cup O\} \tag{2}$$

- $\overline{V}$ denote the set of all possible combinations of the additional smart contract(s) for a partition p i.e.

$$\overline{V} = 2^{OC_p} \tag{3}$$

- $\overline{W}$ denote the set of previously deployed smart contracts covering any combination of the overlapping conditions OC_p and no other condition i.e.

$$\overline{W} = \{W \mid W \text{ is a previously deployed smart such that } W \subseteq OC_p\} \tag{4}$$

Each of the smart contracts in $\overline{U}, \overline{V}$ and $\overline{W}$ has a cost associated with it. This is the cost of blockchain transactions of the smart contract encoding these conditions. There are two types of costs associated with a smart contract i.e. the deployment cost and the execution cost, of which the former is a one-time cost whereas the latter is a per execution cost that is incurred every time the underlying policy is evaluated. For a given number of executions of the BP, the execution cost of a smart contract for a given partition depends upon the probability that the smart contract needs to be executed when that partition is reached. We want to select the smart contracts which give the least cost. The cost of each of the smart contracts in $\overline{U}$ and $\overline{V}$ is given by the cost function C (given in equation (5)). While the cost of each of the smart contracts in $\overline{W}$ is given by the cost without deployment function CWD (given in equation (7)).

Computing Costs:
We now discuss how the cost functions C and CWD are computed. Suppose $T \subseteq AC_p$ denotes the set of conditions encoded as a smart contract. This smart contract will entail a deployment cost, given by the deployment cost function $d\colon 2^{AC_p} \rightarrow \mathbb{R}$, and an execution cost, given by the execution cost function $e\colon 2^{AC_p} \rightarrow \mathbb{R}$. The total estimated cost, for N number of evaluations (where N could be in 100s or 1000s), denoted by the function $C\colon 2^{AC_p} \rightarrow \mathbb{R}$, is computed as:

$$C(T) = d(T) + N \cdot e(T) \cdot Pr(T) \tag{5}$$

Where $Pr(T)$ is the execution probability of the smart contract encoding the set of conditions T. There may be multiple smart contracts for a given partition. However, not all of these smart contracts are executed for each run of the BP. The probability of a smart contract being executed depends on the probability that the underlying conditions are evaluated for a given execution of the BP. Consequently, the execution cost of a smart contract for a given partition depends upon the probability that the smart contract needs to be executed when that partition is reached.

Computing Probabilities:
We now discuss how to compute the execution probability $Pr(T)$ of the smart contract encoding the set of conditions T. Given a partition p of the restructured behavior composition model $\mathcal{T}'_B$ let,

- Π denote the set of paths of $\mathcal{T}'_B$ reaching partition p in which all the conditions contained in T appear
- $prob(\pi)$ denote the path probability of any path $\pi \in \Pi$

Then the probability that the smart contract, encoding the set of conditions $T \subseteq AC_p$, will be executed upon reaching the partition p, is defined in equation (6) and explained below:

$$Pr(T) = 1 - \sum_{\forall\, \pi \in \Pi} prob(\pi) \tag{6}$$

If all the conditions in T have been evaluated prior to reaching partition p, then we do not need to execute the smart contract encoding the set of conditions T for partition p, otherwise, we need to execute it. Note that, since Π is a subset of all the paths reaching partition p, the sum of the probabilities of the paths in Π is always less than or equal to 1.

We also need to compute the cost of evaluating one or more of the leftover overlapping conditions OC_p by executing the pre-deployed smart contract of a previous partition. Such an evaluation cost estimate does not include any deployment cost because an already deployed smart contract of a previous partition is being used to evaluate some conditions for this partition. So, in addition to the cost function (C), there is another cost without deployment (CWD) function. This function gives the total estimated cost, for N number of evaluations of a set of overlapping conditions $W \subseteq OC_p$ using a pre-deployed smart contract of a previous partition and it is computed as:

$$CWD(W) = N \cdot e(W) \cdot Pr(W) \tag{7}$$

Where, W is the set of conditions of a previously deployed smart contract *s.t.* $Pr(W)$ follows equation (6).

Partition Specific Policy Evaluation:
Having defined the cost we now discuss the formulation of the PSPE problem.

Let Z be a collection of all the smart contracts in $\overline{U}, \overline{V}$ and $\overline{W}$ (given in equation (2), (3) and (4)) i.e.

$$Z = \overline{U} \cup \overline{V} \cup \overline{W}$$

Then the PSPE problem is defined as:

Definition 5. (Partition Specific Policy Evaluation Problem): Given a set of conditions AC_p to be evaluated for partition p in the restructured behavior composition model $\mathcal{T}'_B$ and a set $Z \subseteq 2^{AC_p}$ of smart contracts, each of which encodes a subset of the conditions in AC_p, with costs of the smart contracts in R defined by the cost functions C and CWD. The goal is to find a set $H \subseteq Z$ such that:

- all conditions in AC_p are covered by the smart contracts in H, and
- the sum of the costs of the smart contracts in H is minimized

The set of smart contracts in H will give the minimum cost while covering all the conditions required for partition p in the restructured behavior composition model $\mathcal{T}'_B$.

Reduction To Weighted Set Cover Problem:
The PSPE problem can be reduced to the standard weighted set cover problem for which several approximation techniques exist [5]. Here we give the steps of the reduction.

The weighted set cover problem is defined as follows:

Definition 6. (Weighted Set Cover Problem): Given a set of elements $E = \{e_1, \ldots, e_n\}$, and a set $S \subseteq 2^E$ of m subsets of $E, S = \{S_1, \ldots, S_m\}$ with weights $Q = \{q_1, \ldots, q_m\}$. The goal is to find a set $I \subseteq S$ such that:

- all elements in E are covered by I, and
- the sum of the weights of the subsets in I is minimized

The PSPE problem is reduced to the weighted set cover problem as follows:

- The set of conditions AC_p in the PSPE problem corresponds to the set of elements E in the weighted set cover problem i.e. $E = AC_p$ and $n = |AC_p|$
- The set $Z \subseteq 2^{AC_p}$ in the PSPE problem corresponds to the set $S \subseteq 2^E$ in the weighted set cover problem i.e. $S = Z$ and $m = |Z|$
- The costs of the smart contracts in Z, defined by the cost functions C and CWD, in the PSPE problem correspond to the weights Q in the weighted set cover problem i.e.

$$q_i = \begin{cases} C(S_i) & \text{if } S_i \in \overline{U} \text{ or } S_i \in \overline{V} \\ CWD(S_i) & \text{if } S_i \in \overline{W} \end{cases} \quad (8)$$

- The resulting set of smart contracts H which gives the minimum cost while covering all the required conditions in the PSPE problem corresponds to I in the weighted set cover problem i.e. $I = H$

The solution to the weighted set cover problem gives us the minimum set cover I, which is equivalent to the set of minimum cost smart contracts H from the PSPE problem.

The steps to generate the PSPE policy are given in Algorithm 3. We visit each partition p of the restructured behavior composition model $\mathcal{T}'_B$, starting from the *root* partition (line 1). For each partition p, we fetch the guards of the first transition of p to get the conditions AC_p to be evaluated for p (line 2). Then we find the overlapping conditions OC_p which are common between the conditions to be evaluated for this partition AC_p and the prior partial conditions (see Table 2) (line 3). After that, we update the guards of the first transition of p to the conditions to be evaluated for p after optimization (line 4). We return the behavior composition model re-annotated with the PSPE policy $\mathcal{T}''_B$ (line 6).

Algorithm 3 Partition Optimization

Input: $\mathcal{T}'_B$ - restructured behavior composition model
Input: $\mathcal{P}_{\mathcal{T}'_B}$ - partitions of $\mathcal{T}'_B$
Output: $\mathcal{T}''_B$ - PSPE behavior composition model

1: **for** each partition $p \in \mathcal{P}_{\mathcal{T}'_B}$ **do**
2: $\quad AC_p \leftarrow firstTransition(p).\,\phi$
3: $\quad O_p \leftarrow AC_p \cap PriorPartialConditions(\mathcal{T}'_B, p)$
4: $\quad firstTransition(p).\,\phi \leftarrow$
$\qquad OptimizePartition(\mathcal{T}'_B, AC_p,\ OC_p)$
5: **end for**
6: **return** $\mathcal{T}''_B$

4.3 Policy Re-annotation

The behavior composition model $\mathcal{T}_B$ which is annotated with the service providers' policies is re-annotated with the restructured policy and the PSPE policy to obtain the restructured behavior composition model $\mathcal{T}'_B$ and PSPE behavior composition model $\mathcal{T}''_B$ respectively. Once all partitions in $\mathcal{T}'_B$ are re-annotated with the optimal selection of smart contracts according to the PSPE problem, we obtain the PSPE behavior composition model $\mathcal{T}''_B$. We can then formally verify $\mathcal{T}''_B$ against $\mathcal{T}_B$ for conformance with the access control policies of all component services. This verification involves checking whether all the accesses that are allowed in $\mathcal{T}_B$ are also allowed in $\mathcal{T}''_B$ and those that are not allowed in $\mathcal{T}_B$ are also prohibited in $\mathcal{T}''_B$. In fact, we show that there is no need for such verification since the restructured policy is equivalent to the service providers' policies. For this purpose, we give two theorems and their proof sketches below:

Theorem 1. The restructured policy, which is annotated on $\mathcal{T}'_B$ (see Algorithm 2) and the PSPE policy, which is annotated on $\mathcal{T}''_B$ (see Algorithm 3), are equivalent to the service providers' policies, which are annotated on $\mathcal{T}_B$ (see Fig. 9)

Proof Sketch: To prove Theorem 1, we see that $\mathcal{T}_B$ transitions into $\mathcal{T}'_B$ and then into $\mathcal{T}''_B$ by passing through Algorithm 2 and Algorithm 3 in succession. If the changes made to the policy by Algorithm 2 and Algorithm 3 are such that they do not violate the service providers' policies, then the resulting policy will be equivalent. So, the proof of Theorem 1 is reduced to the proof, that both Algorithm 2 and Algorithm 3 either make changes to the policy without violating the service providers' policies or keep the policy unchanged.

First, we consider Algorithm 2. In lines 4 and 5 of this algorithm, the subsequent definitive conditions are merged into the current partition and they are removed from the subsequent partitions. This means, that we pre-evaluate these conditions because we are sure that they will eventually have to be evaluated anyway. So, this pre-evaluation will not violate the policy of the underlying

service providers. In line 6 of this algorithm, we remove the prior definitive conditions from the current partition. This means, that we avoid re-evaluation of identical conditions that have already been evaluated. So, avoiding this re-evaluation will not violate the policy of the underlying service providers. The rest of the conditions, which are not moved by this algorithm, keep the policy unchanged.

Now, we consider Algorithm 3. In line 4 of this algorithm, the conditions to be evaluated for the current partition are updated after optimization. This optimization, is based on the PSPE problem, given in Definition 5. This problem selects the least cost choice for evaluating overlapping conditions (see equation (1)) of a partition out of three cases: 1) evaluating an overlapping condition in the integral smart contract of the current partition; or 2) evaluating it in an additional smart contract for the current partition; or 3) using an already deployed smart contract of a previous partition to evaluate it. So, all three cases of this problem will evaluate the overlapping conditions in one way or the other, while the base conditions (see Table 2) will always be evaluated as part of the integral smart contract (see Section 4.2.3). Hence, the solution to this problem will not violate the policy of the underlying service providers.

This shows, that both Algorithm 2 and Algorithm 3 always either make changes to the policy without violating the service providers' policies or keep the policy unchanged. This concludes the proof.

Theorem 2. The PSPE policy always gives lesser or equal cost than the restructured policy which gives lesser or equal cost than the service providers' policies.

Proof Sketch: The proof of Theorem 2 is much more intuitive. The PSPE policy tries to bring down the cost of policy evaluation for each partition of the restructured policy. In the worst case, it simply does not reduce the cost of the policy evaluation of any partition giving the same cost as that of the restructured policy. Whereas, if it reduces the cost of the policy evaluation for even one of the partitions, then the overall cost of the PSPE policy will be less than that of the restructured policy. We can deduce in a similar way that the restructured policy tries to remove the repeated evaluation of conditions for the service providers' policies. In the worst case, if it does not remove any repeated evaluation, it still gives the same cost as that of the service providers' policies. Whereas, if it removes the repeated evaluation of even one of the conditions, then the overall cost of the restructured policy will be less than that of the service providers' policies. This concludes the proof of Theorem 2.

4.4 Smart Contract Translation and Deployment

The job of the Translator is to transform the composite access control policy into one or more smart contract(s) specified in the appropriate language for the blockchain. In our implementation, we use Solidity, the programming language of smart contracts for the Ethereum blockchain. However, our approach is agnostic to the underlying blockchain.

The deployment engine is responsible for the deployment of smart contracts corresponding to the composite access control policy to the blockchain. In addition, the deployment engine registers the smart contracts with relevant service providers. This registration involves the following:

1) Deployment engine sends the details of smart contracts to relevant service providers for verification. Each service provider checks whether the deployed smart contract(s) satisfies its local access control policy.
2) The deployment engine and service providers agree on a penalty, which needs to be paid by the BPM if the service provider's policy is not correctly enforced. The service provider may selectively audit the smart contract executions to find violations of its access control policy as discussed in Section 4.6.
3) For each smart contract, the deployment engine generates a public/private key pair and shares the private key with all the service providers whose access control policy is evaluated using this smart contract. The attribute managers use the public key to encrypt the attributes needed for smart contract evaluation. The service providers can use their private key to decrypt the attribute values stored in blockchain transactions, of the corresponding smart contract, for auditing.

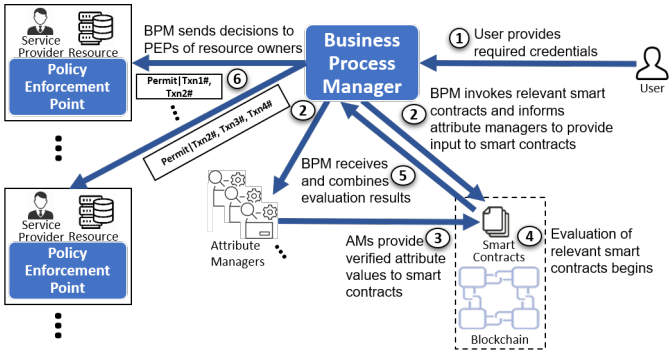

Fig. 11: Policy evaluation and enforcement mechanism for BP over blockchain

4.5 Enforcement over the Blockchain

The BPM is responsible for the evaluation and enforcement of the composite access control policy through smart contracts. Fig. 11 depicts the architecture of the blockchain-based access control system for BPs. The specific steps for policy evaluation and enforcement are discussed below.

1) The user provides the needed credentials to the BPM.
2) The BPM invokes the relevant smart contracts of the composite access control policy to determine the user's authorization. The BPM also notifies the relevant attribute managers to provide the verified attribute values for the given user to the corresponding smart contracts.
3) For a given condition over an attribute (e.g., $CGPA \geq 3.5$), the attribute manager encrypts the attributes, their values, and the user id with the public key for that smart contract. It also includes the condition evaluation result as true or false. This is together encoded in a single message which is digitally signed by the attribute manager and inputted into the smart contract.
4) Evaluation of all the relevant smart contracts begins after the required inputs from attribute managers are received.
5) The BPM receives the evaluation results of all the smart contracts of the composite access control policy. The BPM combines the evaluation results to determine the authorization of the user for the given BP. In case of conflicting

authorizations, the BPM applies the appropriate rule combining algorithm to resolve these conflicts.

6) If the authorization decision is permit, the BPM forwards the user request to the policy enforcement points (PEPs) of all participating service providers, along with the decision and transaction identifiers of smart contracts used for reaching this decision. The service provider(s) can use these transaction identifiers for the sake of auditability.

4.6 Auditing

The primary benefit of using the blockchain for access control is to provide auditability. Typically, auditing might be requested by one of the service providers for a specific service. In this case, one possibility is to simply validate all of the corresponding transactions, but this has a significant cost. Instead, we can leverage the fact that for effective validation, it is not necessary to recheck all the accesses, but only a random fraction of them. We have proposed a game-theoretic mechanism for auditing that reduces the auditing cost while incentivizing honest behavior for the BPM in [2].

5 Experimental Evaluation

We analyzed the effectiveness and efficiency of the proposed approach using the Ethereum testnet Rinkeby as the underlying blockchain. Specifically, we compared the cost of evaluating the smart contracts corresponding to the PSPE access control policy with: i) separate policies (the access control policy of each participating service is encoded as a separate smart contract); and ii) the restructured access control policy. The deployment cost and evaluation cost are computed using Rinkeby, which provides an accurate estimate of the costs of running the actual smart contracts on the Ethereum mainnet. Both deployment cost and evaluation cost are characterized in terms of the size and storage requirements of the underlying smart contracts.

We evaluated the costs of the approach considering the workflow in Example 1 (the corresponding BP is in Fig. 2). Note that each web service in this BP has its own access control policy. Fig. 4 shows the access control policy automaton of *Use HPC Cluster For Lab Assignment (CS Dept.)* web service. We do not show the automata of access control policies of the remaining web services due to space limitations.

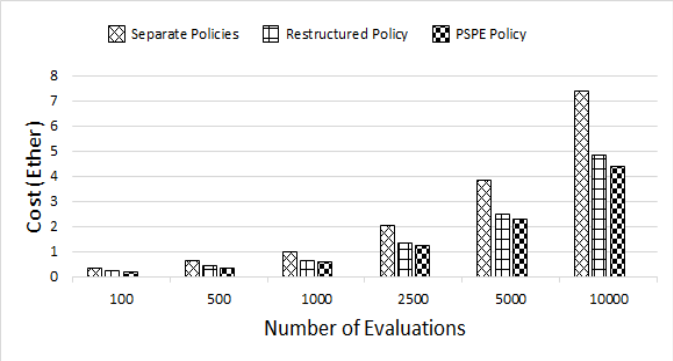

Fig. 12: Cost comparison of course assessment BP for a varying number of evaluations

Fig. 12 shows the cost (in Ethers) of evaluating the separate policies, restructured policy, and, PSPE policy for a varying number of evaluations. As can be seen from this figure, the cost of evaluating separate policies is significantly higher as compared to the restructured policy and the PSPE policy. This is due to the overlap between the access control policies of the component web services of the BP, resulting in the revaluation of overlapping conditions multiple times. The degree of overlap between the web services in a BP can be measured using the Jaccard Index (JI):

$$JI(s_i, s_j) = \frac{|\text{Intersection of conditions in } s_i \text{ and } s_j|}{|\text{Union of conditions in } s_i \text{ and } s_j|}$$

$$\text{Degree of overlap} = \forall s_i, s_j \in BP, \frac{\sum_i \sum_j JI(s_i, s_j)}{\text{No. of service pairs}}$$

The degree of overlap between the access control policies of the course assessment BP example given in Fig. 5 is 0.0741. The evaluation cost of the restructured policy and the PSPE policy is close. The cost difference between the two policies depends on the number of branches in the BP and branching probabilities. Indeed, in the case of a sequential BP, the restructured policy is optimal.

To analyze the effect of the degree of overlap on the overall cost, we next measure the cost for different degrees of overlap. For this, we consider the course assessment BP example, given in Fig. 5, with varying degrees of overlapping conditions in the access control policies of the underlying services. To reduce the degree of overlap between the access control policies of the course assessment BP, we replaced some of the overlapping conditions with unique non-overlapping ones. Similarly, to increase the degree of overlap we replaced some of the non-overlapping conditions with common overlapping ones. This gives us 3 different sets of access control policies for the same BP having a similar size in terms of the number of access control policies but different degrees of overlap. We then compute the restructured policy and consequently the PSPE policy for each case. We compute the evaluation costs for each of the 3 cases for 2500 evaluations, repeating the same computation 100 times and report the average.

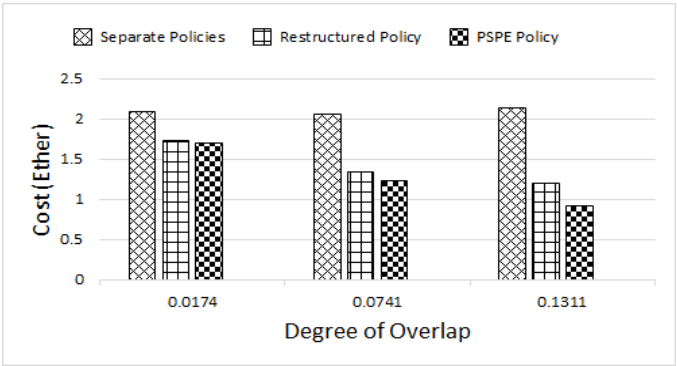

Fig. 13: Cost comparison of different degrees of overlap for 2500 Evaluations

Fig. 13 shows the cost (in Ethers) of evaluating the different policies obtained for varying degrees of overlap. As can be seen from this figure, the cost improvement from separate policies to restructured policy and the PSPE policy increases proportionally to the degree of overlap. This is as expected since we can avoid revaluation of the overlapping conditions. The cost of the PSPE policy is always lower than that of restructured policy because it leverages the BP structure information to minimize unnecessary evaluation.

6 Related Work

The transparency and auditability properties of blockchains and distributed ledgers have attracted huge interest for it to be utilized as the underlying technology for access control enforcement, especially in distributed environments [8], [16], [20], [21], [24], [30], [31], [33]. The initial works on access control using blockchain [16], [21] employ the exchange of access tokens between users via transactions on the blockchain to enable the transfer of access rights. These tokens use a scripting language to express an unlockable public/private key authentication mechanism which can only be unlocked by users holding the corresponding rights. Fine-grained access control policies cannot be expressed using the scripting language used in these works.

Rajput et al. in [22] propose a system based on a permissioned blockchain for managing access to patients' data in emergency situations. They store user role assignments in smart contracts which are evaluated based on pre-defined rules when an emergency situation occurs. Yazdinejad et al. in [32] presented an approach that uses private and public blockchains tailored for an IoT network with the goal of reducing energy consumption and increasing the security of communication between IoT devices. They use a public/private key pair based authentication method evaluated on the tailored private and public blockchains.

The approaches given above either do not evaluate the policy on the blockchain or evaluate simple access control policies on the blockchain. The first work to propose an auditable blockchain-based access control solution considering fine-grained access control policies was that of Maesa et al. [17]. They transform the access control policy into a smart contract which is executed on the blockchain to evaluate the access control policy. Another similar work is that of Islam and Madria [12] which proposes an attribute-based access control system for IoT networks using permissioned blockchain. Resource owners issue attributes that provide access to their resource. The resource owners register access attributes by calling smart contracts which will later be responsible for the evaluation of the access requests. However, these and all of the above works deal with the access control policy of individual resources, while distributed applications contain many resources and are subject to the access control policies of the respective organizational domains of the underlying resources. Marangone et al. [18] propose an approach based on blockchain technology to control data access in the context of a multi-party business process using attribute-based encryption. They encrypt data on the basis of user permissions such that only those who have access to the data are actually able to decrypt it. This way they are able to use a public blockchain while ensuring that access to all resources is restricted. However, they do not consider overlap in the policy for different resources. Therefore, using these solutions for access control enforcement of distributed applications would result in the repeated evaluation of overlapping policies, leading to significant overhead. Another limitation of these works is that they use only attribute-based policies and do not consider event-driven policies.

Di Ciccio, Claudio, et al. in [7] give the conceptual foundation of the use of blockchain for the executions of inter-organizational business processes involving collaboration between untrusted parties. They compare two systems named Caterpillar [15] and Lorikeet [28] which provide support for process execution on the blockchain. They propose that the immutability property of the blockchain can be utilized for auditability and dispute resolution between the untrusted parties. Therefore, these works focus on ensuring compliance of the BP execution using auditibality as opposed to access control enforcement.

There has been significant work on modeling context-based and fine-grained access control policies [3], [14]. Security properties for authorization and access control have also been verified using automata-theoretic and other formal approaches [9], [13], [29]. Shafiq et al. [27] propose a verification framework for the detection and resolution of inconsistencies and conflicts in policies modeled through event-driven RBAC which is a subset of GTRBAC. Shafiq et al. [26] propose a framework for verifying the secure composability of distributed workflows in an autonomous multi-domain environment. The objective of workflow composability verification is to ensure that all the users or processes executing the designated workflow tasks conform to the time-dependent security policy specifications of all collaborating domains.

The above approaches consider temporal and event-based policies, their composability in the context of multi-domain organizational workflows as well as their optimal reconfiguration. In addition to doing all of the above, our approach easily enables auditability of the access control enforcement using blockchain as an immutable log. This introduces a novel aspect of optimization related to the cost of blockchain transactions used to store the audit logs.

7 Conclusion

In this work, we have examined the problem of enabling auditable access control for distributed BPs with event-driven policies. We have proposed a solution based on blockchain technology that minimizes the cost of deployment and enforcement. We also reduce the problem of generating the cost-efficient composite access control policy to the standard weighted set cover problem. The cost savings benefit the service providers, users, and/or the BPM depending on who bears this cost. In the future, we plan to analyze in detail the effect of different cost-sharing models where the cost is split between these parties in different ways. We also plan to work on the adaptation problem which occurs when the BP or policies may change, and propose incremental solutions that are still efficient.

Acknowledgments

The research reported in this publication was supported by the HEC NRPU Grant 14601, HEC and Planning Commission of Pakistan, LUMS FIF grant as well as the National Institutes of Health award R35GM134927. The content is solely the responsibility of the authors and does not necessarily represent the official views of the agencies funding the research.

REFERENCES

[1] A. Afzal, B. Shafiq, S. Shamail, A. Elahraf, J. Vaidya, and N. R. Adam, "Assemble: Attribute, structure and semantics based service mapping approach for collaborative business process development," *IEEE Transactions on Services Computing*, 2018.

[2] A. Akhtar, B. Shafiq, J. Vaidya, A. Afzal, S. Shamail, and O. Rana, "Blockchain based auditable access control for distributed business processes," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 12–22.

[3] N. Baracaldo, B. Palanisamy, and J. Joshi, "G-sir: an insider attack resilient geo-social access control framework," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 1, pp. 84–98, 2017.

[4] M. Barati, "A formal technique for composing cloud services," *Information Technology and Control*, vol. 49, no. 1, pp. 5–27, 2020.

[5] M. Cygan, Ł. Kowalik, and M. Wykurz, "Exponential-time approximation of weighted set cover," *Information Processing Letters*, vol. 109, no. 16, pp. 957–961, 2009.

[6] G. De Giacomo, F. Patrizi, and S. Sardina, "Automatic behavior composition synthesis," *Artificial Intelligence*, vol. 196, pp. 106–142, 2013.

[7] C. Di Ciccio, A. Cecconi, M. Dumas, L. García-Bañuelos, O. López-Pintado, Q. Lu, J. Mendling, A. Ponomarev, A. Binh Tran, and I. Weber, "Blockchain support for collaborative business processes," *Informatik Spektrum*, vol. 42, no. 3, pp. 182–190, 2019.

[8] S. Ding, J. Cao, C. Li, K. Fan, and H. Li, "A novel attribute-based access control scheme using blockchain for iot," *IEEE Access*, vol. 7, pp. 38 431–38 441, 2019.

[9] Y. Du, Y. Wang, B. Yang, and H. Hu, "Analyzing security requirements in timed workflow processes," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 190–207, 2022.

[10] M. GELFOND, "Action languages," *Erectric Trans. on Artificial Intelligence*, vol. 2, pp. 193–210, 1998.

[11] J. Im, S. Kim, and D. Kim, "Iot mashup as a service: cloud-based mashup service for the internet of things," in *2013 IEEE international conference on services computing*. IEEE, 2013, pp. 462–469.

[12] M. A. Islam and S. Madria, "A permissioned blockchain based access control system for iot," in *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2019, pp. 469–476.

[13] S. Jha, S. Sural, V. Atluri, and J. Vaidya, "Security analysis of abac under an administrative model," *IET information security*, vol. 13, no. 2, pp. 96–103, 2019.

[14] J. B. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A generalized temporal role-based access control model," *IEEE transactions on knowledge and data engineering*, vol. 17, no. 1, pp. 4–23, 2005.

[15] O. Lòpez-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, and A. Ponomarev, "Caterpillar: a business process execution engine on the ethereum blockchain," *Software: Practice and Experience*, vol. 49, no. 7, pp. 1162–1193, 2019.

[16] D. D. F. Maesa, P. Mori, and L. Ricci, "Blockchain based access control," in *IFIP international conference on distributed applications and interoperable systems*. Springer, 2017, pp. 206–220.

[17] ——, "A blockchain based approach for the definition of auditable access control systems," *Computers & Security*, vol. 84, pp. 93–119, 2019.

[18] E. Marangone, C. Di Ciccio, and I. Weber, "Fine-grained data access control for collaborative process execution on blockchain," in *International Conference on Business Process Management*. Springer, 2022, pp. 51–67.

[19] H. Mei, G. Huang, and T. Xie, "Internetware: A software paradigm for internet computing," *Computer*, vol. 45, no. 6, pp. 26–31, 2012.

[20] O. Novo, "Blockchain meets iot: An architecture for scalable access management in iot," *IEEE internet of things journal*, vol. 5, no. 2, pp. 1184–1195, 2018.

[21] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman, "Fairaccess: a new blockchain-based access control framework for the internet of things," *Security and Communication Networks*, vol. 9, no. 18, pp. 5943–5964, 2016.

[22] A. R. Rajput, Q. Li, M. T. Ahvanooey, and I. Masood, "Eacms: Emergency access control management system for personal health record based on blockchain," *IEEE Access*, vol. 7, pp. 84 304–84 317, 2019.

[23] R. Ranchal, B. Bhargava, P. Angin, and L. B. Othmane, "Epics: A framework for enforcing security policies in composite web services," *IEEE Transactions on Services Computing*, 2018.

[24] S. Rouhani, R. Belchior, R. S. Cruz, and R. Deters, "Distributed attribute-based access control system using permissioned blockchain," *World Wide Web*, vol. 24, no. 5, pp. 1617–1644, 2021.

[25] B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," *ACM Transactions on Information and System Security (TISSEC)*, vol. 2, no. 2, pp. 159–176, 1999.

[26] B. Shafiq, S. Ghayyur, A. Masood, Z. Pervaiz, A. Almutairi, F. Khan, and A. Ghafoor, "Composability verification of multi-service workflows in a policy-driven cloud computing environment," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 5, pp. 478–493, 2017.

[27] B. Shafiq, J. S. Vaidya, A. Ghafoor, and E. Bertino, "A framework for verification and optimal reconfiguration of event-driven role based access control policies," in *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, 2012, pp. 197–208.

[28] A. B. Tran, Q. Lu, and I. Weber, "Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management." in *BPM (Dissertation/Demos/Industry)*, 2018, pp. 56–60.

[29] E. Uzun, V. Atluri, J. Vaidya, S. Sural, A. L. Ferrara, G. Parlato, and P. Madhusudan, "Security analysis for temporal role based access control," *Journal of Computer Security*, vol. 22, no. 6, pp. 961–996, 2014.

[30] R. Xu, Y. Chen, E. Blasch, and G. Chen, "Blendcac: A smart contract enabled decentralized capability-based access control mechanism for the iot," *Computers*, vol. 7, no. 3, p. 39, 2018.

[31] S. Xu, J. Ning, Y. Li, Y. Zhang, G. Xu, X. Huang, and R. Deng, "A secure emr sharing system with tamper resistance and expressive access control," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[32] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, Q. Zhang, and K.-K. R. Choo, "An energy-efficient sdn controller architecture for iot networks with blockchain-based security," *IEEE Transactions on Services Computing*, vol. 13, no. 4, pp. 625–638, 2020.

[33] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1594–1605, 2018.

Ahmed Akhtar is a PhD candidate in the School of Science and Engineering, Lahore University of Management Sciences, Pakistan. His research interests are in the areas of distributed systems and service-oriented computing.

Masoud Barati received the PhD degree in Computer Science from Sherbrooke University, Canada, in 2018. He is currently an Assistant Professor with the School of Information Technology, Carleton University, Canada. His research interests include distributed systems, Blockchain, formal methods, and cybersecurity.

PLACE PHOTO HERE

Basit Shafiq is an Associate Professor in the School of Science and Engineering, Lahore University of Management Sciences, Pakistan. He has published over 60 research papers in international conferences and journals. His research interests are in the areas of distributed systems, security and privacy, semantic Web, and Web services.

PLACE PHOTO HERE

Omer Rana received the B.Eng. degree in information systems engineering from Imperial College of Science, Technology and Medicine, London, U.K., an M.Sc. in microelectronics systems design from the University of Southampton, U.K., and a Ph.D. in neural computing and parallel architectures from the Imperial College of Science, Technology and Medicine. He is a Professor of performance engineering with Cardiff University, Cardiff, U.K. His research interests include high performance distributed computing, data analytics/mining and scalable systems.

PLACE PHOTO HERE

Ayesha Afzal is an Assistant Professor at Air University, Multan, Pakistan. Her research interests are in the areas of distributed systems, business process management, and big data analytics.

PLACE PHOTO HERE

Jaideep Vaidya is a Distinguished Professor of computer information systems at Rutgers University. He has published over 200 papers in international conferences and journals. His research interests are in privacy, security, and data management. He is an IEEE and AAAS Fellow as well as an ACM Distinguished Scientist.

PLACE PHOTO HERE

Shafay Shamail is an Associate Professor in the School of Science and Engineering, Lahore University of Management Sciences, Pakistan. He has worked both in the software industry and academia. His research interests include software quality, cloud computing, and e-government architectures. His publications include over 60 research papers.