Blockchain Based Auditable Access Control for Distributed Business Processes

Ahmed Akhtar
Computer Science Department
Lahore University of Management Sciences
Lahore, Pakistan
16030059@lums.edu.pk

Basit Shafiq
Computer Science Department
Lahore University of Management Sciences
Lahore, Pakistan
basit@lums.edu.pk

Jaideep Vaidya
MSIS Department
Rutgers University
Newark, USA
jsvaidya@business.rutgers.edu

Ayesha Afzal
Computer Science Department
Air University
Multan, Pakistan
ayesha@aumc.edu.pk

Shafay Shamail
Computer Science Department
Lahore University of Management Sciences
Lahore, Pakistan
sshamail@lums.edu.pk

Omer Rana
School of Computer Science
& Informatics, Cardiff University
Cardiff, UK
RanaOF@cardiff.ac.uk

***Abstract*—The use of blockchain technology has been proposed to provide auditable access control for individual resources. However, when all resources are owned by a single organization, such expensive solutions may not be needed. In this work we focus 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 can provide an attractive decentralized solution to provide auditability. However, the underlying access control policies may be overlapping in terms of the component conditions/rules, and simply using existing solutions would result in repeated evaluation of user's authorization separately for each resource, leading to significant overhead in terms of cost and computation time over the blockchain. To address this challenge, we propose an approach that formulates a constraint optimization problem to generate an optimal composite access control policy. This policy is in compliance with all the local access control policies and minimizes the policy evaluation cost over the blockchain. The developed smart contract(s) can then be deployed to the blockchain, and used for access control enforcement. We also discuss how the access control enforcement can be audited using a game-theoretic approach to minimize cost. We have implemented the initial prototype of our approach using Ethereum as the underlying blockchain and experimentally validated the effectiveness and efficiency of our approach.**

***Index Terms*—Blockchain, XACML, Business Processes, Access Control, Workflows**

I. Introduction

The emerging cloud and edge computing infrastructure has enabled development of next generation internet-centered distributed applications that are autonomous, co-operative, adaptive, evolvable, emergent, and trustworthy. Such Internet-centered distributed applications include business processes (BPs), distributed workflows, and web service mashups [1], [2], [3]. 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 [4], [5].

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 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 [6] 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, the access control enforcement is at the same time outsourced and auditable. However, existing solutions [6], [7] 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. This smart contract then needs to be executed in order to return 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 [8]) to provide auditability at 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. The set of individual access control policies of all services involved in a BP is likely to have some common clauses, as shown in Example 1. If each of the smart contracts evaluating the access control policy of every service is executed separately, these common clauses/conditions will be re-evaluated by their respective smart contracts, as many times as they appear in the individual access control policies as, illustrated in Example 1. This kind of re-evaluation causes cost of evaluation to increase significantly for a blockchain based access control system. This is due to the increased computation and storage requirements, resulting in an increased transaction fee being charged by the miners.

Example 1: Consider a distributed data analytics workflow of a Supply Chain Monitoring application that collects, integrates, and analyzes real-time data from different data sources (e.g., emergency reporting systems, social media) and IoT devices (e.g., sensors – environmental, traffic, chemical, fire detection, surveillance cameras, etc.). This data analytics workflow is depicted in Figure 1 and is instantiated dynamically in response to a highway accident in which a tractor trailer carrying several drums of liquid acetone overturned and exploded generating a smoke plume that spreads a large area.

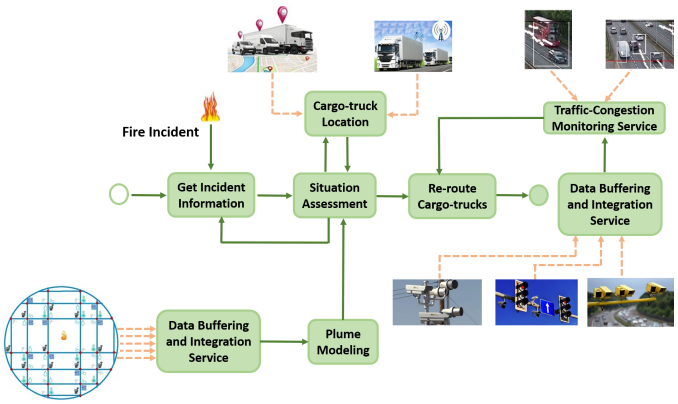

Fig. 1. Emergency Management Workflow

Now, consider the access control policies of some of the web services used in the emergency management workflow.

- *Traffic-Congestion Monitoring Service:* This service takes feeds from surveillance cameras at road intersections and analyzes them to determine traffic congestion. The surveillance cameras are owned by Smart City Project authority, which requires that only officers from the transportation or Police department can access their camera feeds. Therefore, Traffic-Congestion Monitoring Service can only be accessed by officers from the transportation or police department.
- *Plume Modeling:* This service takes the sensory data observations in the vicinity of the incident site and it identifies the at-risk areas that could be affected by the plume. This sensor data comes from the devices installed by the Environmental Protection Department and they allow access to only officers of at least grade 18 from the environmental or transportation department. Therefore, Plume Modeling service can only be accessed by officers of at least grade 18 belonging to the environmental or transportation department.
- *Cargo-truck Location:* This service is used to retrieve the locations of all the cargo trucks of the Supply Chain Monitoring Company. According to the policy of the Supply Chain Monitoring Company, location of its trucks can only be accessed from within the same city by any officer from the environmental, transportation or Police department. Therefore, Cargo-truck Location service can can only be accessed from within the same city by any officer from the environmental, transportation or Police department.

Re-evaluation of Common Conditions: Now consider an officer of grade 18 from the transportation department, who is in the same city and wants to execute the emergency management workflow. Clearly, the condition that the access subject belongs to the transportation department, gets evaluated again and again, for each of the aforementioned web services, before the officer is granted access to execute the workflow. Even if access is denied for a subject, still the conditions of police and environmental department are re-evaluated across two services.

A composite access control policy, is hence needed to reduce this cost of evaluation for a blockchain based access control system, by removing these 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 cost in the form of mining fee incurred due to blockchain transactions. This is precisely the problem that we address in this work. Our key contributions are to:

- Propose an approach that formulates a constraint optimization problem to generate a composite access control policy for a BP that encapsulates the local access control policies of component services and minimizes the policy evaluation cost over the blockchain.
- Present a game-theoretic approach to reduce the cost of auditing the access control enforcement.
- Implement and experimentally validate the proposed approach using Ethereum testnet Rinkeby.

II. Preliminaries

Distributed applications requiring access to resources from different autonomous organizational domains can be represented as distributed BP workflows. Typically the access control policies of organizations governing these resources are specified in XACML. We now present a brief review of BPs and XACML.

A. Business Processes

A Business Process (BP) is a composition of interrelated activities materialized in the form of web services to achieve a well defined business outcome [9]. While the workflow in Example 1 can be naturally expressed in the form of a distributed BP, it is quite complicated and it would be difficult to clearly explain all of the steps of the proposed approach given the space limitation. Therefore, for the rest of the paper,

we consider a simple alternative example in the university context which serves to comprehensively illustrate the access control requirements in a multi-organizational environment, though note that all of the work applies in general to any distributed workflow.

Example 2: Consider an assignment grading BP (shown in Figure 2) in a virtual university environment where three universities (LUMS, Rutgers, and Cardiff) collaborate. An instructor from one university can teach a course to students of other universities. The first step in this BP is to download the assignments of all the students in the course from the Learning Management Systems (LMS) of their respective universities. The LMS of the universities LUMS, Rutgers, and Cardiff are accessed for downloading the assignments using *downloadLUMSAssignments*, *downloadRutgersAssignments* and *downloadCardiffAssignments* services respectively. After downloading, these assignments need to be transferred to a server machine on which an Auto Grader application runs. There are two servers Codec and Rustam, each of which can be used for running autograding service. Codec is the first choice, which is available 90% of the times via *transferToCodec* service, while the remaining 10% of the times, Rustam is used via *transferToRustam* service. After grading the assignments, the scores need to be uploaded to the grade management systems, Zambeel of LUMS, Sakai of Rutgers, and SIMS of Cardiff using *uploadLUMSMarks*, *uploadRutgersMarks*, and *uploadCardiffMarks* services respectively. Finally, the students are notified of their grades. The email notification service (*notifyViaEmail*) is selected 80% of the time, while the SMS notification service (*notifyViaSMS*) is selected 20% of the time.

The probabilities in Figure 2 represent the likelihood of a service being called based on service availability statistics and/or execution paths. The access control policies of some of the services used in Example 2 are given in Figure 3.

B. XACML

The Extensible Access Control Mark-up Language (XACML) is a published standard that defines: i) a declarative fine-grained, attribute-based access control policy language; ii) an architecture; and iii) a processing model describing how to evaluate access requests according to the rules defined in policies [10]. An XACML policy consists of a Target, a Rule set, and a rule combining algorithm.

In XACML, the authorizations of subjects over resources for a given action are specified as rules. A rule in an access control policy is comprised of conditions, which defines the requirement of specific attribute(s) that must be present in the access request or could be inferred/derived from the access request. We consider three type of conditions: Atomic Condition, Composite Condition and Permit/Deny Condition, each of which is defined below.

Definition 1: **Atomic Condition:** *An Atomic Condition is a boolean expression that compares the value of an attribute or functions over attributes to another attribute or value.*

Definition 2: **Composite Condition** *A Composite Condition is a boolean expression over atomic conditions and/or other composite conditions, which can be joined by conjunction or disjunction.*

Definition 3: **Permit/Deny Condition:** *A Permit/Deny Condition is a composite condition that specifies the Resource as well as the Action on that resource on which the permission is granted or denied.*

Definition 4: **Authorization Rule:** *An authorization rule can be represented as a Directed Acyclic Graph $G = (V, E)$, where V is the set of vertices corresponding to either Permit/-Deny Condition, or Atomic Condition, or Composite Condition. $E \in V \times V$ is a set of ordered pairs representing directed edges between vertices in V*

The XACML representation of the access control policy of *downloadLUMSAssignments* service of Example 2 is shown in Listing 1 and its corresponding graph is shown in Figure 3.

Listing 1. Sample XACML Policy

```
<Policy PolicyId="downloadLUMSAssignments Policy" ...>
 <Target><AnyOf><AllOf>
  <Match MatchId="...function:string−equal">
   <AttributeValue DataType="...string">LUMS_LMS</AttributeValue
        ↪ >
   <AttributeDesignator AttributeId="...resource−id" Category="...
        ↪ resource" .../></Match>
  <Match MatchId="...string−equal">
   <AttributeValue DataType="...string">Download Assignment</
        ↪ AttributeValue>
   <AttributeDesignator AttributeId="...action−id" Category="...action"
        ↪ .../></Match>
 </AllOf></AnyOf></Target>
 <Rule Effect="Permit" RuleId="Rule−1">
  <Target><AnyOf><AllOf>
   <Match MatchId="...string−equal">
    <AttributeValue DataType="...string">inst123</AttributeValue>
    <AttributeDesignator AttributeId="...supervisor" Category="...
        ↪ access−subject" .../></Match>
   <Match MatchId="...string−equal">
    <AttributeValue DataType="...string">CS101</AttributeValue>
    <AttributeDesignator AttributeId="...teaches" Category="...
        ↪ instructor" .../></Match>
   <Match MatchId="...boolean−equal">
    <AttributeValue DataType="...boolean">true</AttributeValue>
    <AttributeDesignator AttributeId="...isPhDStudent" Category="...
        ↪ access−subject" .../></Match>
  </AllOf></AnyOf></Target>
  <Condition>
   <Apply FunctionId="...any−of">
    <Function FunctionId="...double−less−than−or−equal"/>
     <AttributeValue DataType="...double">3.5</AttributeValue>
     <AttributeDesignator AttributeId="...cgpa" Category="...access−
        ↪ subject" .../>
</Apply></Condition></Rule></Policy>
```

In the policy graph of *downloadLUMSAssignments* service, "$Permit\langle LUMS\_LMS, DownloadAssignments\rangle$" is a permit condition that authorizes a subject satisfying the predecessor conditions for the action "$DownloadAssignment$" on the resource "$LUMS\_LMS$". The composite condition "$isSupervisorCourse$" is a conjunction of two atomic conditions: i) "$isSupervisor(rollNumber, instNumber)$" – the instructor referred to by "$instNumber$" must be the supervisor of the requesting student referred to by "$rollNumber$"; ii) "$teaches(intNumber, courseCode)$" – the instructor teaches the course referred to by "$courseCode$". There are two other atomic conditions: i) "$isPhDStudent(rollNumber, universityID)$" – the requesting student is a PhDStudent; ii)

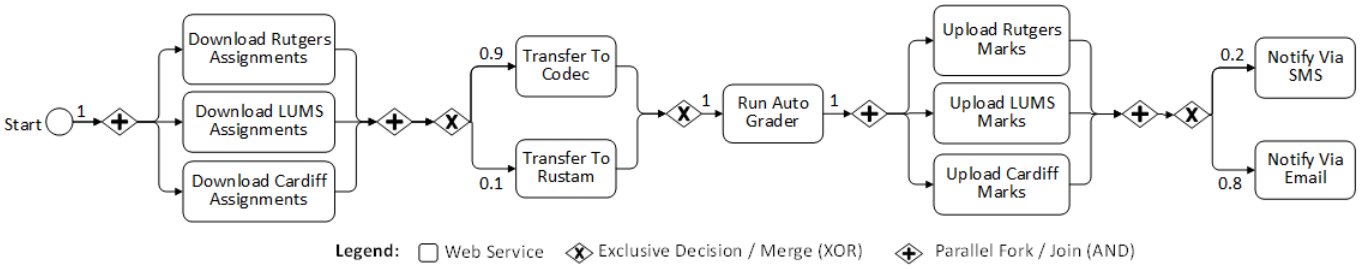

Fig. 2. Assignment grading BP from virtual university domain

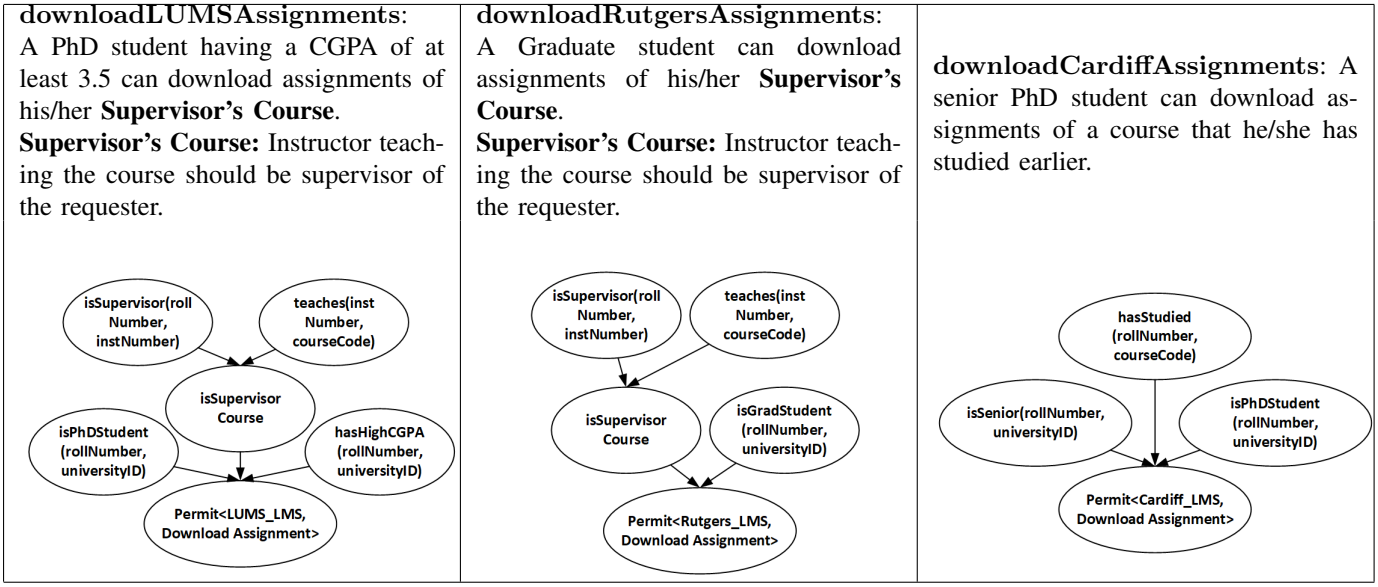

Fig. 3. Access control policy graphs of a few services from assignment grading BP

"$hasHighCGPA(rollNumber, universityID)$" – the requesting student has CGPA of at least 3.5.

III. Problem Statement

Considering the access control policies associated with the underlying web services, we can formally define a distributed BP as follows:

Definition 5: **Business Process:** *A business process is denoted by a Graph, $BP = (S, T, U, Q)$ where:*

- *$S = \{s_1, \ldots s_n\}$ is the set of web services represented as vertices of BP;*
- *$T \subseteq S \times S$ is the set of all edges in BP;*
- *$U : T \rightarrow [0, 1]$ is the function defining probabilities of edges in T*
- *$Q = \{G_1, \ldots G_n\}$, where $G_j = (V_j, E_j)$ is a graph which encodes the rules based on Definition 4, and represents the access control policy of service $s_j \in S$.*

We consider a services cloud environment, where BPs are composed using Web services from multiple organizational domains as depicted in Figure 4. The cloud service provider hosts such BPs and also ensures that the users executing the BPs satisfy the access control policies of the underlying organizational domains. The cloud service provider has knowledge of the access control policies of these organizational domains and uses this knowledge to develop an optimal composite access control policy for a BP and deploys it over the blockchain as one or more smart contracts to ensure auditability. Here we assume that the access control policies of these organizations do not include sensitive information and are publicly available.

To ensure access control, the objective is to deploy one or more smart contract(s) containing all the authorization conditions of underlying web services on the blockchain. However, deploying a single unified smart contract by taking union of all the access control policies may not always be optimal in terms of blockchain cost. Therefore, the unified access control policy, denoted by G, can be partitioned in many ways into a set of fragments, where each fragment contains one or more authorization conditions. We refer to such a partition of G as a composite access control policy $F = \{F_1, F_2 \ldots, \}$. The structure of the BP will determine which partition of the unified policy may result in minimal cost of deployment and evaluation. Formally, the optimal composite access control policy generation problem can be defined as:

Definition 6 (Optimal Composite Policy Generation Problem): Given the BP, find a partition $F^* = \{F_1, \ldots, F_m\}$ of the global unified access control policy G such that:

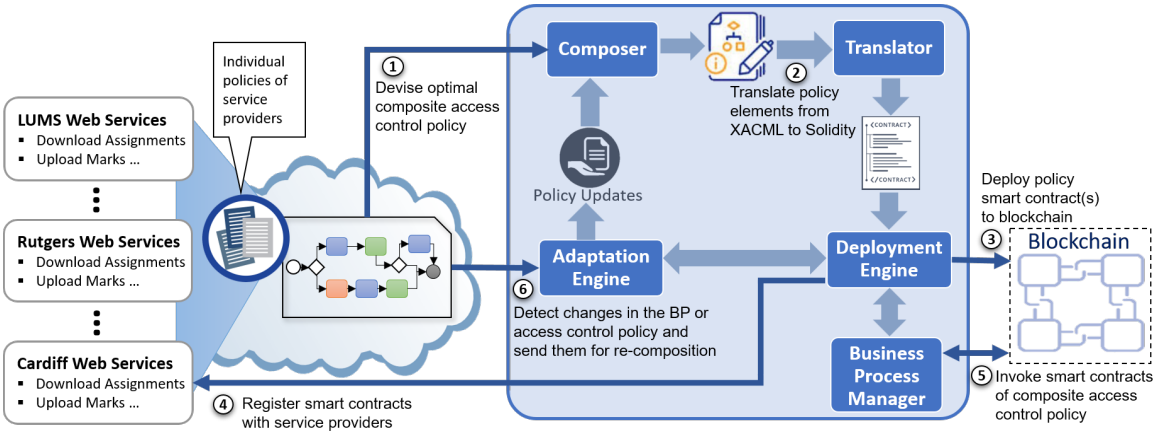

Fig. 4. Architectural view of the proposed approach for BP access control over blockchain

- $\bigcup_{i=1}^{m} F_i = \bigcup_{j=1}^{n} G_j = G$; and
- Sum of the costs of the smart contracts corresponding to all fragments in F^* is minimum over all partitions of G.

IV. Proposed Approach

The proposed approach for finding the optimal composite access control policy consists of five major components as depicted in Figure 4. For a given BP and associated access control policies, the *Composer* generates the composite access control policy, which is an optimal partitioning of the unified 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, which 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 keeps 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 evaluation and enforcement of the composite access control policy through smart contracts. The *Adaptation Engine* monitors changes in the service providers' policies and triggers revision of the composite access control policy with minimum changes. We now discuss the proposed approach in detail:

A. Optimal Policy Composition

The composer combines the multiple service views of the access control policies Q, for all services S in the BP, to give a global unified view of the access control policy. To do this, the first step is to merge all the access control policies of the individual web services in the BP to obtain the global mediated access control policy G. Note that G should not have any duplicate conditions while all the conditions of each individual access control policy in Q are satisfied.

Next, the composer needs to partition G into the composite access control policy such that the cost of deployment and evaluation on the blockchain is minimal. To find the optimal composite access control policy, the composer needs to estimate the combined cost for deployment and evaluation on the blockchain. This cost depends on the size and number of fragments of the partition and the probability that the smart contract corresponding to each fragment gets executed, given the structure of the underlying BP. The overall sequence of steps is given in Algorithm 1. We now explain each step.

1) Global Mediated Access Control Policy: The global mediated access control policy $G = (V, E)$ is simply the union of all of the underlying access control policy graphs G_j, along with two sink nodes, one for Permit and one for Deny. All of the permit conditions are connected to the Permit sink node, while all of the deny conditions are connected to the Deny sink node. The steps to generate the global mediated access control policy are given in Algorithm 1 (lines 1 - 8). Figure 5 depicts the global mediated access control policy graph (shown in the outermost dashed rectangular box) for the BP in Figure 2. This global mediated policy also includes the *assignment download* policies of Figure 3 depicted in bold.

2) Optimal Composite Access Control Policy: A composite access control policy can be generated in multiple ways from the global mediated access control policy G. In order to find the optimal composite access control policy, we need to decide whether to keep each of the conditions in G as an independent smart contract, or make it part of a smart contract of a composite condition. This decision needs to be made for each condition in G, keeping in view of the contribution to the overall cost made by this decision. The selection of conditions which incur the least cost will be considered as the optimal composite access control policy for G. An example of a possible composite access control policy is shown in Figure 5 where conditions selected for deployment, as a separate smart contract, are shown in small rectangular boxes, while those

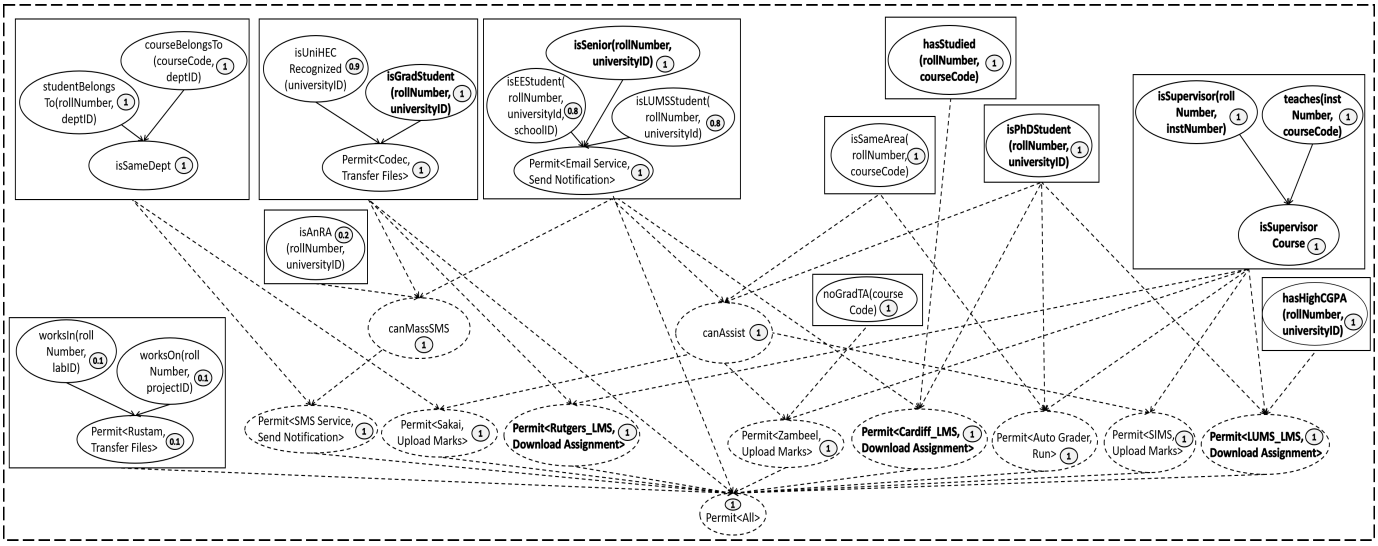

Fig. 5. Global mediated access control policy graph shown in outermost dashed rectangular box, and fragments of composite access control policy shown in small rectangular boxes

Algorithm 1 Generate Optimal Composite Policy

Input: $BP(S,T,U,Q)$ is the graph representing the Business Process

Input: $Q = \{G_j \mid \forall\, j\ \exists\, s_j \in S\}$ is the set of graphs of individual access control policies of services in BP

Input: $d\colon V \to \mathbb{R}$ is the deployment cost function

Input: $e\colon V \to \mathbb{R}$ is the evaluation cost function

Input: N is the number of evaluations

Output: F^* is the optimal partition

{Generate global mediated access control policy, G}

$G = (V, E) \leftarrow \bigcup_j G_j$

$V \leftarrow V \cup \{Permit\langle All\rangle\} \cup \{Deny\langle All\rangle\}$

for each vertex $v \in V$ s.t. v is a permit condition **do**

  $E \leftarrow E \cup \{(v, Permit\langle All\rangle)\}$

end for

for each vertex $v \in V$ s.t. v is a deny condition **do**

  $E \leftarrow E \cup \{(v, Deny\langle All\rangle)\}$

end for

{For each condition, compute the probability that the condition will be evaluated during execution of the BP}

$P \leftarrow Eval.Probabilities(BP, G)$

{For each condition compute the cost of deployment and evaluation over N executions}

$C \leftarrow Eval.Costs(G, P, d, e, N)$

$W_{opt} \leftarrow FindOptimalPartition(G, C)$

return F^*

conditions that are not selected are shown as dotted ovals.

One step which is essential in finding the cost of deciding to keep a condition, as a separate smart contract, is finding the probability that it will be called upon when the underlying BP is executed. This is necessary because there are two types of costs associated with a smart contract i.e. the deployment cost and the evaluation cost, of which the former is a one time cost whereas the latter is a per call cost.

a) Finding Probabilities: The probability of each condition is computed considering the structure of the BP. We first compute the probability of each path in the BP by taking the product of the probabilities of each edge included in the path. Let us denote the probability of the l^{th} path ($path_l$) in the BP as π_l. Next, we compute the probability of each condition in the global mediated access control policy graph G. We need to find for each condition $v \in V$ the probability of it being called when the BP is executed. Let $G_j = (V_j, E_j)$ denotes the access control policy graph of service s_j. The probability of condition v is defined in equation (1) and explained below:

$$P(v) = \begin{cases} \sum\limits_{\forall l, \forall s_j \ \mathtt{s.t.}\ s_j \in path_l \wedge v \in V_j} \pi_l & \text{if } v \text{ is Atomic} \\ \max\limits_{\forall w \in Predecessors(v)} P(w) & \text{otherwise} \end{cases} \tag{1}$$

The probability of an atomic condition is found by summing up the probabilities of all the paths of the BP in which at least one of the services requiring that condition is present. This is due to the fact that the atomic condition is needed for all paths that have a service which requires that condition. The probability of a composite condition or a Permit/Deny condition is found by taking the maximum of the probabilities of all the conditions contained in it. This is due to the fact that a composite condition or a Permit/Deny condition is needed if any of its contained conditions is required. Figure 5 shows these probabilities annotated on each condition.

b) Computing Costs: Selecting a condition $v \in V$ for partitioning of G means that it has to be deployed as a separate smart contract on the blockchain. Hence, it will enatil a deployment cost, given by the deployment cost function $d\colon V \to \mathbb{R}$, and an evaluation cost, given by the evaluation cost function $e\colon V \to \mathbb{R}$ and the number of evaluations. Note that the deployment cost $d(v)$, is a one time cost which is incurred only when the smart contract is initially deployed, whereas the evaluation cost $e(v)$, is a recurring cost which is incurred with every call to evaluate an access control policy. The total cost, denoted by the function $C\colon V \to \mathbb{R}$, is computed for each condition $v \in V_p$ for N evaluations as:

$$C(v) = d(v) + N \cdot e(v) \cdot P(v)$$

c) Determining Optimal Partition: Having the cost function C for every condition $v \in V$ enables us to formulate the problem of finding an optimal partition as a 0-1 integer programming problem, which can be formulated as follows:
Utility Function:

$$\min \sum_{\forall\, v \in V} C(v)\, x_v$$

Subject to:
Path Constraint:

$$\texttt{for each path } \phi \texttt{ in } G, \sum_{v \in \phi} x_v \geq 1 \qquad (2)$$

and

$$x_v \in \{0, 1\}$$

Where, x_v is set to 1 if condition $v \in V$ is selected for deployment as a separate smart contract and 0 otherwise. The path constraints (2) ensures that all atomic conditions of the global mediated access control policy G of BP are covered. The solution to this optimization problem gives us the optimal composite access control policy.

B. Policy Translation and Deployment

The job of the Translator is to transform the composite access control policy, which is expressed in XACML, 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 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 following steps.

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 IV-D.
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.

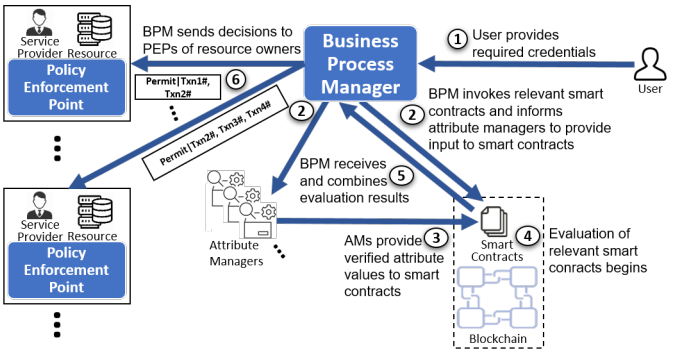

Fig. 6. Policy evaluation and enforcement mechanism for BP over blockchain

C. Enforcement over the Blockchain

The BPM is responsible for evaluation and enforcement of the composite access control policy through smart contracts. Figure 6 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 selects the relevant view of the BP and provides the needed credentials to the BPM. For example, the view of an instructor for assignment grading BP will be different from that of a PhD student, and requires different credentials.
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 to the smart contract.
4) Evaluation of all the relevant smart contracts begins after the 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.

D. Auditing

The primary benefit of using the blockchain for access control is to provide auditability. We now discuss how auditing can actually be done within our framework. Note that, as discussed above, for every resource access, the BPM computes the decision along with the supporting transaction identifiers on the blockchain.

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 use a game-theoretic mechanism following the approach taken in [11], [12] to reduce the cost of auditing, which we now discuss. This is based on the fact that for effective validation, it is not necessary to recheck all the accesses, but only a random fraction of them.

We can model the interaction between the BPM and the service providers as a game. Since each service provider interacts independently with the BPM, its interaction with the BPM can be modeled as an independent two party game. For a given service s, denote the cost of correctly enforcing the access control policies on the blockchain as C_s. This is proportional to the number of smart contracts N_s that need to be evaluated for that service. Similarly, denote the revenue obtained by the BPM for the use of a service s as R_s (note: this is the total revenue from the user as well as the service provider, it is unnecessary to consider what the split might be). In that case, the payoff of the honest BPM will be $R_s - C_s$.

To consider the payoff of the BPM when cheating, we first need to consider how the BPM might cheat. Essentially, a BPM cheats by not evaluating the policy correctly on the blockchain – this is done by not invoking the appropriate smart contracts to save cost (and just reporting either incorrect or different transaction identifiers to the service provider in Figure 6. There are two independent possibilities: 1) *all-or-none:* the BPM may decide to cheat on a fraction of the service requests, but when it cheats, it does not evaluate any of the smart contracts associated with the service request; 2) *partial:* the BPM may cheat on every service request by only evaluating some of the smart contracts associated with the service request. Note that in either case, the reduction in cost is proportional to the number of smart contracts that are not actually evaluated. This can be parameterized by $0 \leq \rho \leq 1$, the fraction of smart contracts not evaluated by the BPM, regardless of how they are chosen (note: $\rho = 0$ implies that the BPM is fully honest, where as $\rho = 1$ implies that the BPM fully cheats). If this cheating is not detected, the payoff for the BPM is $R_s - (1-\rho)C_s$. However, if the cheating is detected the service provider will enforce a penalty P on the BPM. Thus, the payoff for the BPM in this case will be $-P$. Table I gives the payoffs, assuming that verification leads to detection of cheating in the case where the BPM cheats.

TABLE I
PAYOFFS FOR SERVICE S

	Service Provider Verifying	Service Provider Not Verifying
Honest BPM	$R_s - C_s$	$R_s - C_s$
Cheating BPM	$-P$	$R_s - (1-\rho)C_s$

Since the BPM will try to choose the smart contracts to cheat on in such a way that can maximize the chance of detection failure, let us consider how verification can be done by the service provider. The service provider can also choose to verify only a fraction of service requests (but all of the smart contracts in those requests) or verify some of the smart contracts associated with every service request. These two behaviors are denoted as *all-or-none* and *partial* as before.

Thus, there are four possible cases of Cheating/Verification: 1) *(all-or-none, all-or-none)*; 2) *(all-or-none, partial)*; 3) *(partial, all-or-none)*; and *(partial, partial)*. Let us now analyze all four cases. If the cheating is *all-or-none*, then *partial* verification has a higher detection probability for the same amount of effort than *all-or-none* verification. This is due to the fact that even the verification of a single smart contract is sufficient to detect cheating for the entire service request, and thus it is better for the service provider to spread its verification among the service requests to maximize detection. If the cheating is *partial* then both *all-or-none* and *partial* have the same degree of detection. Thus, it is clear that the service provider will always prefer *partial* verification. Now, consider the BPM. If the verification is *all-or-none* it makes sense to concentrate the cheating on as few transactions as possible, to minimize the chance of detection (again the reasoning is the same as earlier, it doesnt matter whether you are detected as cheating on one or more than one smart contracts – both count equally as cheating and incur the same penalty). Thus, in this case, the BPM would prefer *all-or-none* cheating. Interestingly, if the verification is *partial* the BPM would prefer *partial* cheating, since this reduces the possibility of detection (with *all-or-none* cheating, the likelihood is larger that the cheating would get detected). Together, this implies that *partial* cheating and *partial* verification (i.e., (*partial* , *partial*) together is a Nash equilibrium [13] since neither party would prefer to change its behavior. Therefore, in the rest of the discussion below, we assume that this is the case.

Given that the verification is also *partial* it is easiest to choose a fraction of the smart contracts uniformly at random for verification. Now, let us compute the probability of detec-

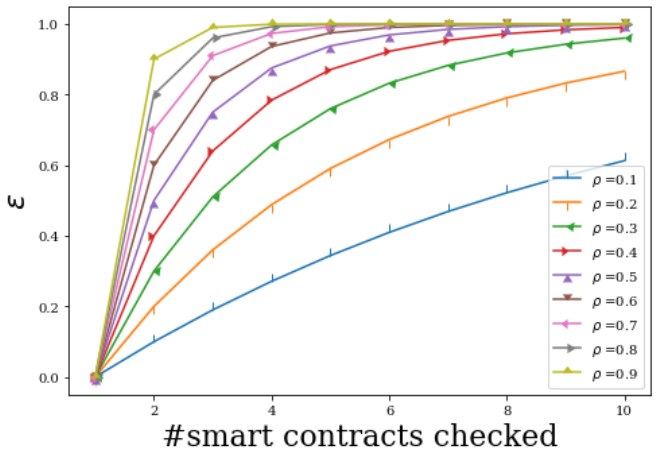

Fig. 7. Cheating Detection Probability for different levels of cheating and different number of smart contracts checked

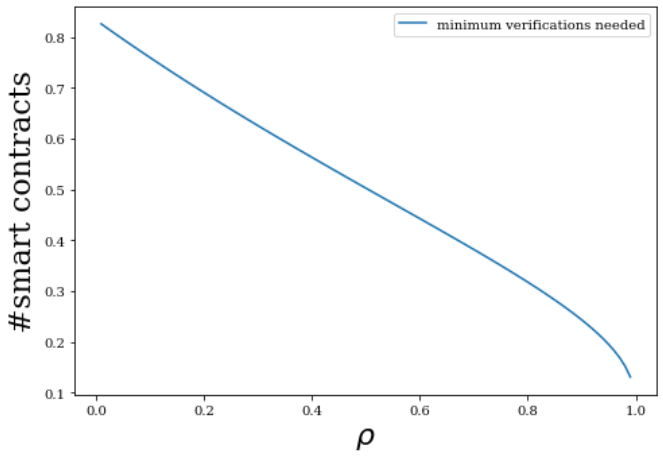

Fig. 8. Minimum number of smart contracts that need to be verified to ensure sufficient detection probability

tion (denoted as ϵ). Since the service has N_s smart contracts, and ρ is the probability of cheating, if x smart contracts are checked, $\epsilon = 1 - (1-\rho)^x$. Figure 7 plots this cheating detection probability for different ρ values (i.e., different levels of cheating), and different number of smart contracts. It is instructive to note that when the cheating level increases to 0.3, just checking 10 smart contracts is already sufficient to give a cheating detection probability of over 95%.

Moreover, perfect detection is unnecessary to ensure honesty. It is only necessary to have a high enough detection probability to deter cheating. Specifically, note that to ensure that the BPM does not cheat, the only pure strategy Nash equilibrium is when, the payoff to the honest BPM is more than the payoff obtained with cheating when (imperfect) verification is done. Specifically, all we need to ensure is that:

$$R_s - C_s > (1-\epsilon)\cdot(R_s - (1-\rho)C_s) - \epsilon P_s \tag{3}$$

Rewriting and simplifying:

$$R_s - C_s > R_s - \epsilon R_s - (1-\rho)C_s + \epsilon(1-\rho)C_s - \epsilon P_s$$
$$\implies C_s < (1-\rho)C_s + \epsilon(R_s - (1-\rho)C_s + P_s)$$
$$\implies \epsilon > \rho C_s/(R_s - (1-\rho)C_s + P_S)$$

This can be further simplified by expressing the penalty in terms of the revenue (for example $P_s = m \cdot R_s$) and in turn, the revenue is some multiple of the cost (i.e., $R_s = m' \cdot C_s$) (note $m, m' \geq 0$). Substituting for R_s and P_s we get:

$$\implies \epsilon > \rho C_s/(m'C_s - (1-\rho)C_s + m \cdot m' \cdot C_S)$$
$$\implies \epsilon > \frac{\rho}{((1+m)m' - 1 + \rho)} \tag{4}$$

Note that the service provider does not know ρ (the degree of cheating) of the server. However, it can simply substitute different values of ρ and find the corresponding ϵ necessary. Since ϵ can be derived for each ρ for the number of smart contracts checked (see Figure 7), it is easy to find the smallest number of smart contracts that must be checked to ensure that ϵ is actually greater than what is necessary to deter cheating as per equation 4.

For example, let us pessimistically assume that $m = 1$ and $m' = 1.1$ (i.e., the penalty is the same as the revenue and the revenue is only 10% more than the cost – typically the penalty is at least 10 times the revenue, and the profit margin is at least 20%). In this case $\epsilon > \frac{\rho}{(1+1)1.1-1+\rho} = \frac{\rho}{1.2+\rho}$. Now, for each value of ρ we can derive the corresponding ϵ and correspondingly determine the minimum number of smart contracts that need to be checked to achieve this ϵ. Figure 8 depicts this for all possible values of ρ from 0 to 1 in increments of 0.01. It is clear, that even in the worst case, when cheating is 0.01, no more than 1 smart contracts need to be validated in order to ensure sufficient detection.

One question that can be legitimately posed with all of the above analysis is what stops the BPM from choosing an even lower value of ρ. To begin with, this does not help the BPM, since the needed ϵ also drops as ρ decreases. In fact, we can simply obtain the derivative for ϵ with respect to ρ and evaluate it at the limit. Indeed, $\lim_{\rho\to 0} \frac{\partial \epsilon}{\partial \rho} = 0.83333$. Furthermore, note that in order to actually cheat, the BPM must not execute at least one contract – which also provides a lower bound on the cheating, if it occurs. For example, in the case where a service request requires 10 smart contracts, the lowest level of cheating corresponds to $\rho = 0.1$. Again, as shown in Figure 8, no more than 1 smart contract needs to be validated to ensure that the payoff of cheating will always be lower than that of honesty.

V. Experimental Evaluation

We analyzed the effectiveness and efficiency of the proposed approach using the Ethereum testnet Rinkeby as underlying blockchain. Specifically, we compared the cost of evaluating the smart contracts corresponding to the optimal composite access control policy with: i) separate policies (access control policy of each participating service is encoded as a separate

smart contract); and ii) the global-mediated access control policy encoded as a single smart contract. Note that the deployment cost and evaluation cost are computed using the Etherium testnet Rinkeby instead of running the actual smart contracts on the Etherium mainnet. Since the Etherium testnet Rinkeby emulates the Etherium mainnet in terms of gas usage, therefore the costs estimated this way are equivalent to the costs in the Etherium mainnet. Both deployment cost and evaluation cost are characterized in terms of the size and storage requirements of the underlying smart contracts.

We first evaluated the costs of the approach considering the workflow in Example 2 (the corresponding BP is in Figure 2). Note that each web service in this BP has its own access control policy. Figure 3 shows the access control policies of *downloadAssignments* web service from the 3 different universities. We do not show the access control policies of the remaining web services due to space limitation.

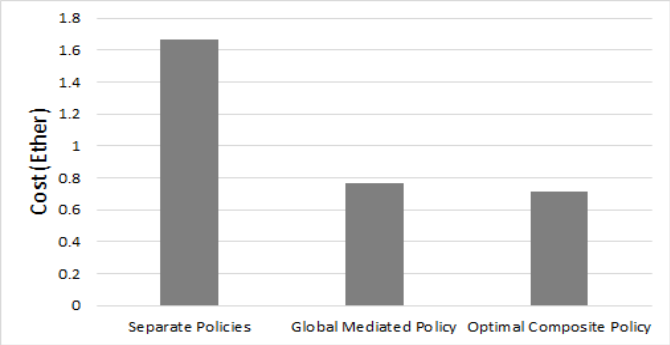

Fig. 9. Cost comparison of Assignment Grading BP for 2500 Evaluations

Figure 9 shows the cost (in Ethers) of evaluating the separate policies, global mediated policy and optimal composite policy for $2,500$ evaluations. As can be seen from this figure, the cost of running separate policies is significantly higher as compared to the global mediated policy and the optimal composite 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):

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

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 for the assignment grading BP of Example 2 was 0.1765. The evaluation cost of global mediated policy and the optimal composite policy is close. The cost difference between the two policies depends on the number of branches in the BP and branching probabilities. Indeed, in case of a sequential BP, the global mediated 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 5 randomly generated BPs with 11 services and varying structure in terms of the number of branches and branching probabilities. We also consider a set of 20 randomly generated access control policy graphs with varying degree of overlapping conditions. We first randomly select 11 policy graphs out of the 20 policy graphs. We then randomly assign the same 11 policy graphs to the services in each of the 5 BP graphs. This gives us 5 different BP structures having the same degree of overlap. We then compute the global mediated policy which again is the same for all the 5 BPs. Next we compute optimal composite policy which however will be different for the 5 BPs due to the structural difference between the BPs. We compute the evaluation cost for the optimal composite policy for each of the 5 BPs and take the average. We repeat the above steps to generate policies and BPs with different degrees of overlap.

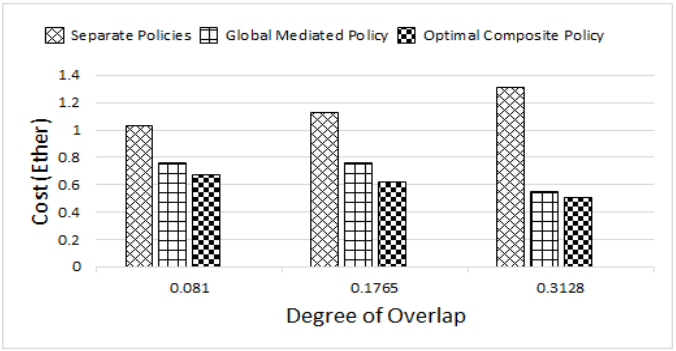

Fig. 10. Cost comparison of different degrees of overlap for 2500 Evaluations

Figure 10 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 global mediated policy and the optimal composite policy increases proportionally to the degree of overlap. This is as expected since we can avoid revaluation of the overlapping conditions. The cost of optimal composite policy is always lower than that of global mediated policy, because it leverages the information of the structure of the BP to minimize unnecessary evaluations.

VI. Related Work

Blockchain technology has gained significant interest for access control enforcement in distributed environment because of the transparency and auditability properties [7], [14], [15], [16], [17], [18], [19], [20]. The earlier work on this topic [7], [14] used access tokens that are exchanged among users through blockchain transactions for transfer of access rights. These tokens are based on basic public/private key authentication scripts which can only be unlocked by the users who hold the corresponding rights. A limitation of this body of work is the lack of support for fine grained policies due to the limited expressive capabilities of the scripting language.

Dorri et al. [21] considered access control enforcement for smart home environment. This work uses blockchain as immutable storage of access control policies, while the access control policies are evaluated off-chain. Zhang et al. [16] propose a framework with multiple access control contracts (ACCs), one judge contract (JC) and one register contract

(RC). Each ACC implements static as well as dynamic access right validation. The JC receives reports of misbehavior from ACC and returns a penalty judgment. The RC registers access control and judgment methods. Access control is manged by IoT gateways serving as agents for the IoT devices. The gateways need to be trusted by the IoT devices because they work on their behalf for access control management, which may not cater for transparency requirement.

The above approaches either consider simple access control policies when evaluated on the blockchain or employ off-chain mechanisms for policy evaluation. Maesa et al. [6] is the first to propose an auditable blockchain based access control solution that considers fine-grained access control policies represented in XACML. In their solution, the access control policy is transformed into a smart contract which is evaluated over the blockchain for determining user authorizations. They have also provided a reference implementation that uses XACML policies, transformed into Solidity smart contracts, and deployed on the Ethereum blockchain for access control evaluation. However, their solution as well as all of the above approaches focus on the access control policy of individual resources, whereas 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 organizational domains. Therefore, employing such solutions to manage access control for distributed applications requires evaluating the user's authorization separately for each resource, which may have significant overhead.

Weber et al. in [22] address the problem of trust in collaborative BP execution using blockchain. Specifically, they use blockchain as a peer-to-peer infrastructure to coordinate BP execution as well as to store BP logs. Therefore, the focus of their work is on auditing the execution behavior of the BP rather than access control enforcement.

VII. Conclusion

In this work we have examined the problem of enabling auditable access control for distributed BPs. We have proposed a solution based on blockchain technology that minimizes the cost of deployment and enforcement. We also propose a game-theoretic mechanism for auditing that reduces the auditing cost while incentivizing honest behavior for the BPM. The cost savings may benefit the service providers, users, or 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.

Acknowledgment

Research reported in this publication was supported by the Higher Education Commission and Planning Commission of Pakistan and LUMS FIF grant as well as the National Science Foundation under awards CNS-1564034, CNS-1624503, CNS-1747728 and the National Institutes of Health under awards R01GM118574 and 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] H. Mei, G. Huang, and T. Xie, "Internetware: A software paradigm for internet computing," *Computer*, vol. 45, no. 6, pp. 26–31, 2012.

[2] 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.

[3] 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*, pp. 462–469, IEEE, 2013.

[4] 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.

[5] 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, pp. 478–493, Sep. 2017.

[6] D. D. F. Maesa, P. Mori, and L. Ricci, "A blockchain based approach for the definition of auditable access control systems," *Computers & Security*, vol. 84, pp. 93–119, 2019.

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

[8] 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.

[9] M. Papazoglou, "Web services and soa: principles and technology 2nd," *Harlow, Essex: Pearson Education Limited*, 2012.

[10] S. Godik and T. Moses, "Oasis extensible access control markup language," *OASIS Specification cs-xacml-specification-1.0*, 2002.

[11] Y. Hong, J. Vaidya, and H. Lu, "Secure and efficient distributed linear programming," *Journal of Computer Security*, vol. 20, no. 5, pp. 583–634, 2012.

[12] J. Vaidya, I. Yakut, and A. Basu, "Efficient integrity verification for outsourced collaborative filtering," in *2014 IEEE International Conference on Data Mining*, pp. 560–569, IEEE, 2014.

[13] H. Gintis, *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Interaction*. Princeton University Press, 2009.

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

[15] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in *2016 2nd International Conference on Open and Big Data (OBD)*, pp. 25–30, IEEE, 2016.

[16] 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.

[17] C. Dukkipati, Y. Zhang, and L. C. Cheng, "Decentralized, blockchain based access control framework for the heterogeneous internet of things," in *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*, pp. 61–69, ACM, 2018.

[18] 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.

[19] 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. 38431–38441, 2019.

[20] 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.

[21] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for iot security and privacy: The case study of a smart home," in *2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops)*, pp. 618–623, IEEE, 2017.

[22] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling, "Untrusted business process monitoring and execution using blockchain," in *International Conference on Business Process Management*, pp. 329–347, Springer, 2016.