Managing Access Control in Large-Scale Multi-party IoT Systems

Pavana Pradeep Kumar, Krishna Kant
CIS Department, Temple University, USA
{pavana.pradeep, kkant}@temple.edu

Amitangshu Pal
Indian Inst. of Technology, Kanpur, India
{amitangshu.pal}@gmail.com

***Abstract*—Large-scale IoT systems are likely to involve multiple subsystems deployed and operated by different "parties", which must collaborate to ensure that their operational rules do not conflict. We codify the smooth functioning of the entire system through a set of "safety properties" that must be enforced collaboratively. However, this requires cross-party access to the sensors/actuators state and the ability to request remote actuations. In this paper, we define an access control architecture for such situations where we distinguish between the static authorization problem that selects parties tasked with safety property enforcement and the dynamic (run-time) control over accesses. This results in a unique enforcer selection problem for which we develop efficient algorithms and quantify their performance through a comprehensive emulation of an extensive smart home. We also show that the additional cost of granting access rights to the parties is quite small in a medium-size emulated multiparty IoT environment.**

***Index Terms*—Access control, multi-party IoT system, safety properties, combinatorial optimization**

I. Introduction

IoT deployments continue to accelerate worldwide for various applications, particularly the multifaceted smart city applications. Most of such deployments invariably consist of multiple subsystems, each controlling a different aspect of the system. For example, a smart building would include subsystems focused on smart lighting, HVAC control, surveillance, fire management, etc. These subsystems are likely to be designed/deployed by different vendors and managed by various administrators, referred to as "parties". Consequently, their operational rules (ORs) are developed independently and may primarily concern only the sensors/actuators of the subsystem (though we do allow for cross-party ORs). Thus these subsystems may conflict when operated in a shared environment.

In [1], we have explored such inter-party conflict detection and resolution. The approach is based on defining a set of "Safety Properties" (SPs), again expressed in the same way as ORs, except that these are now requirements for acceptable collective behavior. The SPs necessarily involve some cross-party aspect since we can assume that any intra-party conflicts have already been resolved. The conflict detection and resolution are then reduced to checking for conflicts between ORs and SPs and *enforcement* or suitable dynamic alteration of ORs to avoid the conflicts. Although the inherent conflicts between ORs and SPs can be detected statically, much of the conflict detection and resolution depends on the context and thus needs to be done at run-time.

This research was supported by NSF grant CNS-1527346.

Our work in [1] assumed that any accesses needed to evaluate ORs and SPs (which would require the current state of sensors and actuators) and any resolution actions (which would require sending appropriate actuation command to a controller) could be done freely since we did not consider any access restrictions. This is undesirable even when assuming that the parties are mutually trusting and non-malicious; as we do, a successful attacker can get free access to the entire system. This motivates us to consider the issue of accessibility driven by the principle of least privilege. In this paper, we define an architecture that separates authorization (i.e., selection of the most suitable party to enforce each SP, which is done rather statically) and the run-time access management. We develop algorithms for each and evaluate them comprehensively by emulating a large smart-home environment. *To the best of our knowledge, access control in the context of dynamic conflict detection in multiparty IoT systems has not been considered in the literature.*

The rest of the paper is organized as follows. Section II discusses the related work, describes our overall architecture and highlights specific contributions. Section III provides the necessary background and describes the two sides of the problem: namely authorization (determining SP enforcers) and run-time access control. The next two sections (IV and V) then discuss the enforcer selection algorithms and section VI focuses on the run-time access management. Section VII evaluates the efficiency of our approach. Finally, section VIII then concludes the discussion.

II. Related Work and Our Contributions

A. Related works and their limitations

Access control is a very well studied topic with many well-known approaches, including Role-Based Access Control (RBAC) [2] which controls access based on the role of the individual, Attribute-Based Access Control (ABAC) [3] which controls access based on the attributes of the users and objects, and Capability-based Access Control (CAC) [4] which provides an unforgeable ticket for access to the authorized entities. However, traditional access control methods are inadequate in large-scale multi-party IoT systems due to both the complexity of ABAC and RBAC models [5] and the dynamic and context-specific nature of the access control problem. On the other hand, CAC models have limitations like lack of context awareness, access rights propagation, revocation, etc

Ouaddah et al. [6] analyze various access control models for IoT systems. The article discusses the advantages and disadvantages of different access control models and protocols in an IoT environment. Pal, et.al. [7] present a policy-based approach in the context of IoT for providing fine-grained access for authorized users to services while protecting valuable resources from unauthorized access. An attribute-based access control mechanism is presented in [8] that can enforce access control based on the attributes of the devices, users, and environment context. However, these mechanisms do not consider dynamic context based access control needed in a multiparty IoT system.

Recently, the Blockchain technology has been examined extensively to manage cooperation among non-trusting parties. In [9] the authors present FairAccess, a decentralized access control framework for IoT based on blockchain. Other similar studies on access control models using blockchain are reported in [10]–[14]. However, these approaches achieve scalability by distributing access permissions via a hub acting as a manager, posing a security risk if the manager is malicious. Additionally, blockchain mechanisms proposed in [9] do not support self-enforced access control policies. The use of Blockchain is not only expensive but also has several downsides. It requires broad consensus, meaning disclosure of a lot of data, and it does nothing about the publication of incorrect data (e.g., incorrect sensor values) unless that data can be verified by independent means (e.g., by giving all parties isolated mechanisms to access each sensor's value). But even then, if the sensor is faulty or otherwise compromised, Blockchain cannot address that.

B. Proposed Access Control Architecture

We split the access control problem into two parts: (a) *Authorization*, which chooses, for each SP, which party is best suited to do the enforcement, and (b) Real-time *Access Control*, which must be applied during SP enforcement at run-time. Note that the authorization (and hence enforcer selection) is essentially static, although it can be redone if needed. Accordingly, authorization relates to the more static aspects of the access, including (a) Communications difficulty (e.g., intermediate hops, provisioning cost, etc.), (b) Sensor/actuator state sharing risk as perceived by the target party, (c) Quality/reliability of the data generated, etc. We capture these aspects via a "cost" factor and regard it as only a party-to-party cost for simplicity. We specifically avoid defining this cost in dynamic terms (e.g., number of bytes of data) since the focus here is not performance but rather data exposure across parties or the difficulty of communication. In particular, if a party needs to enforce two different SPs that require a state of the same remote sensor/actuator, the cost needs to be considered only once from an authorization perspective, even though these SPs will be enforced as and when necessary at run-time.

Since a SP involves *attributes* (defined as sensor/actuator values and actuation operations) from two or more parties, there is no "natural" enforcer for a SP. Overall, we want to distribute the enforcement responsibilities among parties to minimize the "cost" (which includes risk, among other things) of non-local attribute access and avoid too much concentration in one party (which is both risky and may cause performance bottleneck). In general, the SPs may range in importance from critical to merely desirable so that the least the important ones may even be ignored if they require too much work.

Following the selection of enforcers, we have the access control problem, which determines when under what conditions or how long the actual access is provided to the authorized parties.

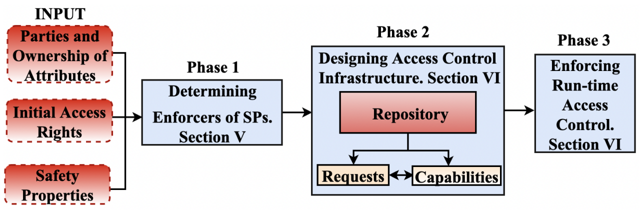

INPUT
Parties and Ownership of Attributes
Initial Access Rights
Safety Properties
Phase 1
Determining Enforcers of SPs. Section V
Phase 2
Designing Access Control Infrastructure. Section VI
Repository
Requests
Capabilities
Phase 3
Enforcing Run-time Access Control. Section VI

Fig. 1: Authorization and Access Control Model

We assume that all subsystems have physical interconnectivity through which any pair of them can communicate. We can assume a standard web-services interface for each controller through which they can discover each other and then communicate [15], [16]. To protect the system from external security threats, we assume standard cryptographic mechanisms such as each controller using PKI to authenticate other controllers and Diffie-Hellman key exchange for establishing session keys. It should be adequate for a controller to verify a standard group key for all its devices [17], but we do not address that part here. We need to provide appropriate authorization to enforce the SPs for inter-controller interaction. We assume that the ORs and SPs are themselves not considered sensitive and thus can be pooled and operated upon at a single node, hereafter called the *root node*. This could be either one of the controllers or yet another trusted node.

Fig. 1 shows the overall structure of our work. In phase1 (authorization), the root node executes the suitable algorithm from section V by using as inputs the details of each subsystem (party) and the SPs to determine the enforcing party for each SP. Next, in phase2 (access control), discussed in section VI, the root node prepares suitable attribute lists along with "capabilities" (or cryptographically signed access specifications) and distributes them to both SP enforcers and attribute owners. Phase3 then uses these for actual access control.

C. Our contributions

As discussed in section II-B, *the authorization cost has a unique property in that the cost of an attribute is counted only once even if multiple SPs enforced by a party need it.* This property sets our problem apart from the very well-studied assignment problems in operations research. *To the best of our knowledge, such a problem has not been considered in the past in the literature.* Since the problem is easily shown to be NP-hard, we devise two different algorithms, one based on a step-by-step assignment of safety properties to parties and the other based on recursive partitioning. We also address the problem of distributing the access rights, verifying them, and ensuring that authorized enforcers get limited ability to request actuations.

As stated above, we evaluated our solutions comprehensively using the Home-IO simulator [18], whose extensive physics modeling capabilities of the environment make it much more extensive and flexible than simple experiments that one could conduct with actual devices. The experimental evaluation indicates that both approaches rapidly converge to the optimal solution, but partitioning is better. We also conducted experiments to request/grant the needed access rights and have shown that the cost is rather modest.

In this paper, we have assumed that the parties in our application are cooperative and non-malicious. We believe that is a proper assumption in a setting where the subsystems are deployed and operated by reputable companies/organizations along with severe legal and market-related costs of deliberate misbehavior. Dealing with cooperation in an arbitrary malicious environment is generally infeasible or expensive. For example, the Byzantine agreement [19] concerns only broadcast consistency, assumes only a limited number of dishonest parties, and still is very expensive.

III. Access Control in IoT Systems

A. Multiparty Operation and Conflicts

As mentioned above, a large-scale IoT system may have multiple subsystems, each with one or more controllers. These controllers can become interdependent for several reasons, as discussed in [20] and thus may conflict. Fig. 2 illustrates some of the access control issues and conflicts with a simple example of two IoT subsystems deployed and operated by two different parties. Controller 1 owns sensor 1,2,3 and actuator 1, Controller 2 owns sensor 4,5,6 and actuator 2. These relationships are shown using solid lines. As shown by dotted lines, some of the operational rules also need nonlocal sensor values. The two subsystems operate in two adjacent and overlapping areas called areas1 and 2. When a single-step greedy algorithm is used to solve this simple example, the assignment cost slightly differs compared to greedy used in combination with some additional heuristics.

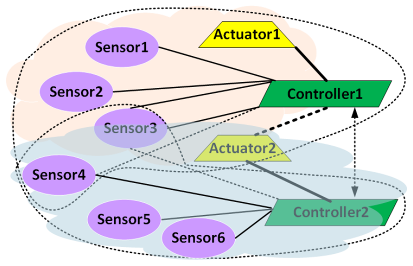


Fig. 2: Illustration of a multi-party IoT system

For example, suppose that in Fig. 2, the two systems are fire management (controller 1) and HVAC (controller 2), Sensor3 is the smoke detector for the entire space, and Actuator2 opens/closes a window. Suppose that controller1 determines smoke and opens the window in area2, letting cold air inside. This would cause HVAC to be triggered and may act oppositely (i.e., run the heater). Note that in this case, the HVAC cannot enhance its actuation condition to suppress heating *since it does not know why the temperature dropped.* Instead, if Controller 1 shares the sensor readings with Controller 2, such an exception is not only possible but can be incorporated as a more direct rule independent of the temperature. Such actions do require a deep analysis of various possibilities and their consequences. For example, it would be inappropriate to use a rule that stops the window from opening due to smoke if the heater is running.

IoT deployments continue to evolve over time due to physical, operational, and environmental changes. For example, devices may fail, be taken out temporarily for maintenance, new devices added, operational policies changed either permanently or temporarily due to some event (e.g., fire), etc. This requires the ability to rerun authorization algorithms whenever needed, grant new authorizations, and revoke unneeded ones.

B. Operational Policies and Rules

An *operational policy* $\mathcal{P}_i$ for controller i essentially moves the subsystem from one state to another. The policy can be expressed as the following triple:

$$\mathcal{P}_i = [C_i^{(pre)}, \mathbb{A}_i, C_i^{(post)}] \tag{1}$$

Where $C_i^{(pre)}$ and $C_i^{(post)}$ are the precondition and post-condition assertions about the state before and after taking the stated action $\mathbb{A}_i$, the index i here emphasizes that this policy deals only with what is visible to controller i. The pre/post conditions are Boolean expressions over various state variables, including the status of an actuator and the sensor's values.

An operational policy can also be considered as an *operational rule* (OR), denoted $\mathcal{R}_i^{(o)}$, expressed in first order logic by simply viewing it as follows:

$$\mathcal{R}_i^{(o)} = [C_i^{(pre)} \& \mathbb{A}_i \Longrightarrow C_i^{(post)}] \tag{2}$$

As a concrete example, consider a policy that turns on cooling when the room temperature goes above 25C. This can be expressed as [(temp>25C & cooling=off & turnon_cooler) $\Longrightarrow$ cooling=on]. As another example, rule R4 in Table II can be expressed at [Motion & luminance$< B_0$ & turnon_lights $\Longrightarrow \forall_i$light$_i = on$].

C. Safety Properties

A *Safety Property* (SP) requires an action to be taken so that "something bad" (which we generically call as *conflict*) should not happen due to independent operation of different subsystems. For example, the SP "if smoke detected then open windows" intended to avoid an undesirable behavior and may be needed since the climate control subsystem controls the window. In contrast, smoke detection is a function of the fire management subsystem. The SPs, like the ORs, may involve time and even frequency (e.g., turning a device on and off in quick succession may be undesirable). One issue with specifying SPs explicitly is that the "bad" things need to be anticipated in advance. In a complex system, this may be difficult, and new problems may emerge over time either via actual occurrence or through some prediction mechanism (e.g., machine learning). However, this is not a problem since the

TABLE I: Table of notations

Notations	Meaning
S_{tot}	Set of SPs (N denotes #SPs)
P_{tot}	Set of Parties (K denotes #parties)
$C_{\text{enf}}(s)$	Cost of non-enforcement for each SP $s \in S_{tot}$
$C_{\text{nenf}}(s)$	Cost of enforcement for each SP $s \in S_{tot}$
$C_{\text{auth}}(p,p')$	Inter-party authorization cost between parties p and p'

conflict detection/resolution in [1] and the access control are dynamic, and the authorizations can be reassigned occasionally as needed.

IV. Deciding Safety Property Enforcers

A. Rule Enforcer Problem

We have a universal set of attributes denoted A_{tot}, i.e., the set of state variables (or attributes) corresponding to all the sensors and actuators in the entire system. We also have a set of parties denoted as P_{tot}. Each attribute $a \in A_{tot}$ has an associated owner party $P_{attr}(a) \in [1..K]$. Similarly, let us define the opposite function $A_{party}(p)$ that provides the attributes owned by party p. We also have a set of safety properties, denoted S_{tot}. Each of these SPs involves some subset of the attributes which we denote as $A_{sp}(s)$. Obviously, $\cup_{s \in S} A_{sp}(s) \subseteq A_{tot}$. We denote by $P_{\text{enf}}(s) \in P_{tot}$ the "enforcer party" that we want to select for each SP s.

For party $p \in P_{tot}$, $R(a,p) \in [0,1]$ denotes the access rights to attribute a (0 means no access, 1 means access). The baseline accessibility is given, i.e., $\forall a,p$, $R_0(a,p)$ has the following accesses enabled: (a) By default, each party has access to its attributes, i.e., $R_0(a,O(a)) = 1$, and (b) $R_0(a,p) = 1$ for $p \neq O(a)$ if an operational rule of party p requires such an access. For all others $R_0(a,p) = 0$.

For each authorization granted to attribute a to party p, there is a positive cost denoted as $C_{\text{auth}}(a,p)$. We assume that all SPs that already hold in the baseline scenario and do not require any additional access rights are excluded. Thus, each SP $s \in S_{tot}$ SPs does involve some cost for non-enforcement, denoted as $C_{\text{nenf}}(s)$ and enforcement, denoted, $C_{\text{enf}}(s)$. The key notations are summarized in Table I.

The problem now is to determine the weighted partial cover $S_0 \subset S_{tot}$ with a given threshold for the total cost of the uncovered SPs. That is, we want $\sum_{s \in S_{tot} - S_0} C_{ne}(s) \leq w_0$ while minimizing the total authorization cost of the attributes in S_0, i.e., the SPs that are enforced. For any $s \in S_0$, we need new access rights for the enforcer party $P_{\text{enf}}(s)$ over attributes for which the access does not already exist, i.e., for all $a \in A_{sp}(s)$ such that $R_0(a, P_{\text{enf}}(s)) = 0$. That is, if we define $\eta(s)$ as the total cost of all attributes in SP s that the enforcer does not initially have rights to, i.e., $\eta(s) = \sum_{a \in A_{sp}(s), R_0(a, P_{\text{enf}}(s)) = 0} C_{\text{auth}}(a, P_{\text{enf}}(s))$, we want to minimize $\sum_{s \in S_0} \eta(s)$.

B. Formulating Minimum Cost Enforcement Problem

The goal of minimum cost enforcement problem is to partition the set of SPs S_{tot} among the P_{tot} parties such that the overall cost of nonlocal accesses is minimized. Let $x_{s,p}$ denote the assignment function for SP s to party p, i.e., $x_{s,p} = 1$ party p is the enforcer of SP s, i.e, $p = P_{\text{enf}}(s)$. Let

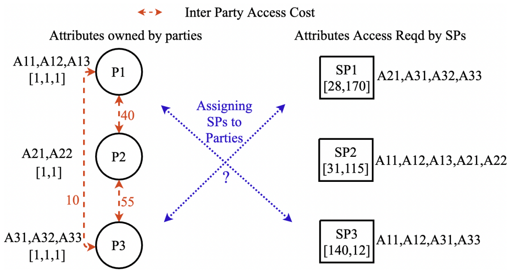


Fig. 3: An Illustrative Example

X denote the enforcement assignment matrix $x(s,p)$s. Thus our assignment problem can be stated as:

$$\begin{aligned} &\text{Minimize } C_{tot}(X) \\ &\text{Subject to } \textstyle\sum_{p=1}^{N} X(s,p) = 1, \; \forall \; s = 1,..,N \\ &\qquad\qquad x_{s,p} \in (0,1) \forall s = 1,..,N, \forall p = 1,..,K \end{aligned} \tag{3}$$

The constraint above states that each SP will be assigned only to a party. Now if SP s is assigned to party p, the attribute set $X(s,p)$ will be local and thus have zero (or small) cost, whereas everything else will have a nonzero cost. Furthermore, if another SP s' is also assigned to party p, the cost of common non-local attributes accessed by s and s' should be counted only once. That is, if $X(s,p) = X(s',p) = 1$ the costs of attributes $A_{sp}(s) \cup A_{sp}(s')$ should be counted only once.

Now the set $A_{com}(p,p') = \cup_{s=1}^{N} X(s,p)[A_{party}(p') \cap A_{sp}(s)]$ represents all the attributes that are *common* parties p and p', i.e., in party p' but also accessed by SP's assigned to party p. The total cost then is the sum of costs of the elements of $A_{com}(p,p')$ times the inter party authorization cost denoted by $C_{\text{auth}}(p,p')$ as shown in the equation below.

$$C_{tot}(p,p') = \sum_{\forall a \in A_{com}(p,p')} c(a) * C_{\text{auth}}(p,p') \tag{4}$$

Then the objective function corresponding to a given assignment matrix X is given by $C_{tot}(X) = \sum_{p=1}^{K} \sum_{p'=1, p' \neq p}^{K} C_{tot}(p,p')$. This function is being minimized above. The resulting optimal $X(s,p)$ is precisely the enforcer list, i.e., $X(s,p) = 1$ iff $p = P_{\text{enf}}(s)$.

C. Complexity of the Enforcement Algorithm

The classical assignment problems are NP-hard, and so is ours. However, as explained in section II-C, ours is not a classical assignment problem because of the unique cost structure that we need to assume to deal with accessibility. This requires developing new efficient heuristic algorithms that we develop in the next section.

Because of this unique cost structure, a simple greedy approach may not provide good results. A simple example illustrates this in Fig. 3, which is later used as a running example to illustrate our algorithms in the next section as well. In this example, with three parties <P1,P2,P3> and three SPs <SP1,SP2,SP3>. The figure depicts the attributes owned by parties P1, P2, and P3, and the attributes SPs SP1, SP2, AND SP3 require access. When a single-step greedy algorithm is used to solve this simple example, the assignment cost slightly differs compared to greedy used in combination with some additional heuristics. The detailed steps in the calculation of cost are shown in section VII-C.

V. Algorithms for Enforcement Problem

A. Greedy+Movement+Deletion (GMD) Approach

Since the cost of the assignment is unknown in our problem until all the SPs have been assigned to the parties, we consider a greedy assignment followed by a few rounds of movement and/or deletion.

The initial greedy assignment assigns each SP to the party in the order of the maximum local attributes covered by the party. If multiple parties have the same number of local attributes for SP, the party ID is used to break the tie. Following this greedy assignment of SPs to parties, the next step is to identify a SP that can be reassigned to a different party to reduce the overall cost, which is calculated using equation (4). We perform a lookup of all SPs associated with each party that requires access to non-local attributes. We consider all $K-1$ parties for reassignment for each such SP, excluding its original enforcer, to reduce the overall cost. Given that the parties incur an inter-party access cost reassigning an SP to another party should account for this cost and prefer the party with the lowest cost compared to the original enforcer party. Given that it is not always possible to satisfy SPs while loosening access controls, or vice versa, a balance between SPs and access restrictions may be required. As noted earlier, certain non-critical/optional SPs may have higher cost of enforcement than risk of ignoring them; these are removed in the last step of our algorithm.

With this, we develop a heuristic algorithm named Greedy Movement Deletion (GMD) as shown in Algorithm 1, which is motivated by our earlier work on access control in collaborative multiparty databases in [21]. The first function employs greedy heuristics that map each s to a party p with maximum local attributes and outputs the enforcers(parties) for each s and the total cost of the assignment. The primary approach is to start with the local attributes contained in all parties and attributes in all SPs. This step is done by checking the overlapping of attributes in each SP with the local attribute set of each party. Then for each SP, we determine the number of times it appears in a local attribute set of every party and assigns it to the party with the highest count (line 6). This process is repeated for all the SPs, and the cost of the entire assignment is calculated using Equation. 4 (Line 7 to 8).

The second function takes the set of enforcers obtained via greedy assignment and counts the number of non-local attributes each s should access before evaluating reassignment. The count of non-local attributes is calculated for each s, and its enforcer party p, and s for movement is determined for those that need access to at least one non-local attribute (Line 15,16). Once the SPs to be moved are decided, we consider all the $K-1$ possible parties (excluding the original enforcer) for reassignment and obtain the new set of enforcers by invoking the greedy assignment function for each SP in the new list X (line 19). The function returns a new group of enforcers (E') and the improved cost ($COST'$).

Algorithm 1 GMD: Greedy Movement Deletion Algorithm

```
1: void GMD_Algorithm() //
2: $S_{tot}$ is set of SPs, $P_{tot}$ is set of Parties
3: procedure Greedy_Assignment($S_{tot}$,$P_{tot}$):
4:     $E$= [] // Set of enforcers
5:     $COST$ = 0 // Cost of the entire assignment
6:     $p_{max}$,$s$ $\leftarrow$ getMaxAttributes($S_{tot}$,$P_{tot}$) /* assigns each SP $s$ to a
       party with maximum local attributes ($p_{max}$) */
7:     $E \leftarrow$ Map($p_{max}$,$s$)
8:     Calculate the $COST$ of the assignment Equation. 4
9:     return $E$,$COST$
10: procedure Movement($E$,$COST$):
11:     $E'$= [] // New Set of enforcers
12:     $COST'$ = 0 // Cost of the new assignment
13:     $M$= [] // Set of SPs to be reassigned/moved
14:     for each entry [$p$,$s$] in $E$ do
15:         $s' \leftarrow$ getNonLocalAttributes($p$,$s$) /* determine SP $s$ that require at
            least one non-local attribute access from its enforcer party $p$ */
16:         $M$ = $M \cup \{s'\}$;
17:         if $q$_is_enforcer($s'$) then
18:             $\mathbb{K}'$ = $P_{tot} \setminus q$
19:             $E'$,$COST'$ = Greedy_Assignment($X$,$\mathbb{K}'$)
20:     return $E'$,$COST'$
21: procedure Deletion():
22:     delete($s$ where $C_{\mathrm{enf}}(s) > C_{\mathrm{nenf}}(s)$); // sp is not a critical
23:     Calculate the $COST$ using the Equation. 4
24:     return $COST$
```

The third function considers removing any safety properties to reduce the total cost further. Each safety property s has a cost associated with enforcing and not enforcing it, which is determined by the significance of the s. After finalizing the set of enforcers by the Movement function; if the cost of enforcing a s is greater than the cost of not enforcing it (line 21), the corresponding s is deleted, and the cost is recalculated without this s resulting in the total cost (line 22).

The key quantity in the running time of the algorithm is K, the number of parties, N_{Att}, the total number of attributes across the entire subsystem, S_{tot} is set of SPs. Therefore, the overall worst-case complexity of the greedy algorithm we have developed is $O(K*N*N_{Att})$.

B. Partitioning Approach

In this section, we present a recursive partitioning approach for the problem of assigning all the SPs to the parties. For a large-scale IoT system composed of numerous subsystems, each with its own controller/party operating cooperatively, we propose a modified scheme of the Kernighan-Lin algorithm [22]. The proposed algorithm bisects the IoT system of multiple parties based on the authorization cost between them. The IoT system can be modeled as an undirected weighted graph $G_I = (V,E)$, with vertices v_k, $k = 1..K$ representing parties and edges representing the authorization cost between pairs of controllers/parties. The adjacency matrix W for the graph G_I contains the cost of the edges, for each $w_{ij} \in W$, w_{ij} represents the authorization cost between a party v_i and a party v_j.

The original Kernighan-Lin (KL) algorithm is an iterative algorithm that starts with an initial bi-partition of a graph $G_I = (V,E)$ with $|V| = K$, partitions V into Y and Z such that $V_Y = V_Z$, $V_Y \cap V_Z = \emptyset$ and $V_Y \cup V_Z = V$. If no good initial partition is known, the algorithm is repeated with a variety of randomly chosen initial partitions, and the one with the smallest edge-cut is chosen. The algorithm searches for a subset of vertices in every iteration such that swapping them results in the partition with the smallest edge-cut. If such a subset is found, it is used as the partition for the following iteration. The algorithm repeats the procedure and terminates if it cannot discover two subsets. The KL algorithm finds locally optimal partitions with a good initial partition. Since we are partitioning a smaller graph with fewer parties, the algorithm requires fewer runs to find a good partition in a shorter amount of time.

Based on the KL algorithm, we propose a recursive partitioning algorithm as presented in Algorithm 2. The algorithm has two phases: the partitioning phase and the assigning phase. In the partitioning phase the undirected weighted graph $G_I = (V,E)$, the set $P_0 = V$ is partitioned into two partitions P_y and P_z with minimum average edge-cut, so that two parties in P_y and P_z respectively has least inter-party authorization cost ($C_{\text{auth}}(p,p')$) (Line 4). Each vertex in the resulting partition has an associated external and internal cost which is defined as follows: For each $y \in Y$, the external cost is given by, $E_y = \sum_{\forall v \in Z} c_{yv}$ which is the sum of the costs associated with all edges connecting each vertex in Z. Similarly, the internal cost is defined as, $I_y = \sum_{\forall v \in Y} c_{yv}$ which is the sum of the costs associated with all edges connecting each vertex in the same partition Y. Each vertex in the partition has the D-value, defined as the cost reductions associated with moving a vertex which is calculated using,

$$D_y = E_y - I_y \qquad \text{for any vertex } y \text{ in } Y \tag{5}$$

The primary parameter that determines whether two vertices $y \in Y$ and $z \in Z$ is swapped or not is given by Gain, g_{yz}:

$$g_{yz} = D_y + D_z - 2 * c_{yz} \tag{6}$$

where g_{yz} is the sum of the D-values of vertex Y and Z and difference from the value multiplied by twice the cost of the edge connecting vertex Y to vertex Z.

At Line 8, the function pick_max() calculates the D-values and chooses two vertices y and z that maximize the $Gain$, g_{yz} from the partitions P_y and P_z, respectively. The vertices with the highest gain are swapped between partitions and marked as locked, indicating that they are not subject to further exchanging (Line 9 and 10).

If any pair of vertices $y \in Y$ and $z \in Z$ is swapped, the algorithm updates the D-values (Line 10), denoted by the symbol D' and calculated using

$$\begin{aligned} D_i^{'} &= D_y + 2c_{iy} - 2c_{iz}, \quad && \forall i \in Y - \{y\} \\ D_j^{'} &= D_z + 2c_{jz} - 2c_{jy}, \quad && \forall j \in Z - \{z\} \end{aligned} \tag{7}$$

The swapping process is repeated until the sum of the gain values between each pair of vertices equals zero. After optimizing the partition, in the assigning phase by calling function at Line 16, we divide the N SPs into resulting the partitions by calculating the number of overlapping attributes between SPs and the parties in the partitions, and the cost of this assignment is calculated using Equation. 4. The process is then repeated recursively until we reach a partition size of one (Line 14). Then, we delete the non-critical SP and calculate the resulting cost. (Line 15).

Algorithm 2 Partitioning Algorithm

```
 1: The initial partition is $\prod_{initial} = \{P_0\}$, store $\prod_{initial}$, $P_0 = V$, $P = P_0$
 2: procedure compute $partition(P)$
 3:     while $|P| > 1$ do
 4:         Initially partition $P$ into $P_y$ and $P_z$ arbitrarily
 5:         repeat
 6:             /* Now improve the partitions */
 7:             compute $D$-values, $\forall y \in P_y$ and $\forall z \in P_z$ using (Eqn.5)
 8:             $(y,z)$=pick_max$(P_y,P_z,W)$ /* Pick pair of unlocked vertices
                $y \in P_y$ and $z \in P_z$, such that $Gain$, $g_{yz}$ in Eqn.(6) is maximized
                */
 9:             swap_vertices$(y,)$ between $P_y$ to $P_z$
10:             Lock vertices $y$ and $z$, store $g_{yz}$, Update new $D$-values, $\forall y \in P_y$
                and $\forall z \in P_z$ (Eqn.7)
11:         until sum($Gain$ = 0);
12:         Store the partition $P \longrightarrow \{P_y, P_z\}$
13:         assign($P$)
14:         Compute $partition(P_y)$ and $partition(P_z)$,
15: delete($s$ where $C_{\text{enf}}(s) > C_{\text{nenf}}(s)$); // sp is not a critical
16: procedure assign($P$)
17:     for each partition p in $P$ do
18:         Assign SPs based on count of overlapping attributes and calculate
            cost of assignment using (Eqn.4)
```

VI. Designing Access Control Infrastructure

The algorithms in the previous section would partition the SPs into sets, say S_i, $i = 1..K$, where S_i denotes the set of SPs enforced by party i. (Note that some of these sets could be null.) Following this, each enforcer and the regular party must be provided the appropriate access lists (phase2). Next, these lists should be used to request and enforce access control at run-time. We discuss these in the following.

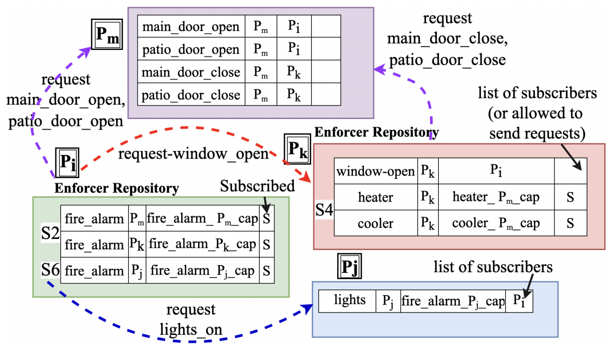
A. Constructing Enforcement Repositories

Let $S_i = \{S_{ij}, j = 1,2,..\}$ denote the individual SPs that party i will enforce. Let RA_{ij} and WA_{ij} denote the set of *nonlocal* read and write (actuation) attributes in SP S_{ij}. Then for enforcement, party i needs to obtain values of RA_{ij} from their owner parties, evaluate S_{ij}, and if necessary send a request to the owner parties in WA_{ij}. This brings in three key questions: (a) when and how often does party i evaluate S_{ij}, (b) when and how party i gets access to RA_{ij} and W_{ij}, and (c) how is the access claim verified by the target party? We address these questions in the following.

Since the "state" represented by RA_{ij} can change anytime, and it is nonlocal to party i, a straight pull or push is required to read them. We will assume a web-services-based publish-subscribe interface for reads. Thus an attribute owner publishes a new value whenever it changes, and the subscribers can decide how to act on it. In our case, the subscription is not open; instead, only the enforcers should be allowed to subscribe to the attributes they need. The notion of capabilities [4] provides a convenient mechanism for this.

The root distributes the capabilities to both the publisher and subscriber nodes to do the appropriate checking as part of the subscription protocol. The changed values can be delivered to a subscriber either via a push mechanism (usually based on a change threshold) or pulled in periodically by the subscriber.

The pull mechanism is more straightforward but may incur high communication costs (frequent pulls) or miss out on events that substantially change the pulled value (infrequent pulls). Besides, it implies permanent access granted to the subscriber. The push mechanism amounts to a remote interrupt delivery to each subscriber and may have scalability issues. However, it can control (a) the change threshold for the pushes and hence the overhead vs. accuracy tradeoff, and (b) the accessibility itself – effectively, the subscriber has no access until a new value is pushed.



Pm
main_door_open Pm Pi
patio_door_open Pm Pi
main_door_close Pm Pk
patio_door_close Pm Pk
request main_door_close, patio_door_close
request main_door_open, patio_door_open
list of subscribers (or allowed to send requests)
Pi
request-window_open
Pk
Enforcer Repository
window-open Pk Pi
S4
heater Pk heater_ Pm_cap S
cooler Pk cooler_ Pm_cap S
Enforcer Repository
Subscribed
S2
fire_alarm Pm fire_alarm_ Pm_cap S
fire_alarm Pk fire_alarm_Pk_cap S
S6
fire_alarm Pj fire_alarm_Pj_cap S
Pj
list of subscribers
lights Pj fire_alarm_Pj_cap Pi
request lights_on

Fig. 4: Enforcing Run-time Access Control

As for WA_{ij}, when an enforcer determines that actuation is required, it needs to send an explicit request to the owners, who perform the requested actuation. We again propose to use capabilities for this so that every actuation request can be checked. The root node again does the capability distribution. A key advantage of the capability mechanism is that the access rights can be revoked if enforcers are changed or are found to be faulty/compromised. The access information is maintained in a repository at each node as illustrated in Fig. 4 where we arbitrarily chose 3 SPs (S2, S4, S6) and four parties as indicated. S2/S6 are enforced by party P_i, and S4 by P_k. Parties P_m and P_j host some of the sensors/actuators needed for the enforcement.

B. Enforcing Run-time Access Control

As stated earlier, the root node prepares the necessary access lists (along with capabilities) and distributes them to all the parties for use in access control, arranging to publish the required state data, handle subscriptions, and start the run-time operation. As an illustration, for party P_i to enforce S2, it must be subscriber to P_m and P_k for "main_door, "patio_door" and "windows" respectively. Also, P_m and P_k must honor request from P_i to open the "main_door", "patio_door" and "window".

One issue in the design so far is that a legitimate enforcer will be able to make the actuation request any time, which is risky if the enforcer gets compromised. Ideally, we want to allow the actuation only under the precise scenario of the enforced SP, but this is nontrivial. For example, if P_i gets only conditional access, both P_k or P_m must have an independent way of verifying that "fire_alarm=on for >1 minute", but this is not possible in a model where each party "owns" its devices. Furthermore, multiple independent access paths are very expensive, hard to manage, and actually, increase the attack surface of the system.

TABLE II: Operational rules for different controllers

ID	Rule Description
	Operational rules for Light controller
R4	The lights for any area A must be turned on when a motion is detected and luminance value is less than a threshold (B_1)
R5	The lights for any area A must be turned off after 2 minutes, when there is no motion is detected, and luminance value is greater than or equal to the threshold (B_1)
R9	If the motion_detected for any area A is more than 3 times then, user is present in the area
R11	When the user is present in area A for more than 5 minutes then user_present mode is activated
	Operational rules of Fire controller
R13	After two minutes the fire is detected, fire-alarm is turned on
R16	The doors(main door, patio) and windows must be open and unlocked until the smoke_level is less than threshold t
R18	If the room temperature is greater than or equal to 155F then sprinkler_head is activated and water_pump is open
R23	Once the sprinkler is on, turn off the sprinkler after 4 to 6 minutes
	Operational rules for Climate controller
R25	When the room_temperature $\geq$ 75F then turn on cooler
R26	When the room_temperature $\leq$ 68F then turn on heater
R32	If the room_temperature is in between 66F and 71F then turn off the cooler after 3 to 5 minutes
R35	The flood_sensor is deactivated after two minutes and evaporation starts with a constant rate of V until the water_level reached zero
	Operational rules for Security controller
R37	The doors(main door, patio) must be closed and locked when the door sensor is inactive
R39	The garage door must open when motion is detected and when valid_keypad_code is entered
R45	The intruder alarm should go off after 3 minutes
	Operational rules for Surveillance controller
R47	For any area A, Until the user_present mode is activated the security_cameras (indoor_camera,outdoor_camera) should record video for 't' time slots
R50	The security_alarm is turned off after two minutes

Instead, we exploit other physical system constraints to limit remote actuation. At least two types of constraints can be easily recognized in this regard. The first relates to the time/frequency of state change requests from other parties, such as not allowing repeated on/off or open/close requests. The second one concerns the operational constraints of the (local) subsystem, such as a request to open one window and close one nearby in the context of climate control. These can be regarded as *Local Safety Properties* (LSPs), i.e., safety properties that only involve attributes of one subsystem. Recall that we had earlier assumed that such local constraints are always satisfied. While this is reasonable for the correct remote actuation requests, actuation requests from a faulty or compromised remote controller could violate the LSPs. An efficient "what-if" analysis of LSPs before granting the actuation request can avoid this problem. Algorithms for this can be designed along the lines of our work in [1].

VII. Experimental Evaluation

A. Smart-home emulation

To experiment with a wide variety of processes and events in a smart home, we use a comprehensive smart home emulator called Home-IO, built by CReSTIC laboratory [18]. The simulator is a virtual house's real-time simulation software that can modify the environment and automation level.

Home-IO simulates real-time heat transfer via radiation, convection, and conduction. Inter-air mass exchanges are simulated. Due to temperature differences and airflow, the temperature changes when doors and windows are opened and

TABLE IV: Safety Properties

ID	Rule Description
S1	When the sprinkler is on for three minutes, then the water valve is off
S2	When the fire alarm is on for more than 1 minute, main door, patio doors and window are opened for 8 to 10 minutes
S3	The windows must not be unlocked and open when the heater or cooler is on
S4	When the heater or cooler is on, the main door and patio door should not be open for more than 5 minutes
S5	If the garage_door is opened for more than 2 hours then close the garage_door
S6	During night if the fire alarm is on then turn on the lights for 3 to 4 minutes
S7	The AC and the heater must not be on at the same time
S8	All the doors and windows must be closed within 2 minutes when user_present mode must is activated

closed as in a real house. The wind helps heat transfer between the house and the outside air. Upwind house walls react more to heat transfer. Cloud cover reduces solar radiation. The humidity changes the dew point, affecting how the outside air affects the house temperature. The home has multiple rooms with smoke and CO detectors, smart lights, window blinds, and doors. So the house is a data source and a testing ground.

Because of all these features, we believe that the use of Home-IO is far more powerful than a real smart home controller (e.g., Samsung SmartThings [23]), where the readings of various sensors/actuators would be either extremely limited or artificial (e.g., pretending that there is a fire). Our control and analysis logic was implemented in Python (32 bit) using visual studio (version 1.46) installed on an Intel(R) Core(TM) i7-7700 CPU@3.6 GHz, 32 GB RAM and 1 TB SSD.

Fig. 5 shows our simulated scenario using Home-IO. To increase the complexity of the model, we consider eight different rooms, including a kitchen, pantry, garage, and the home's exterior. This yields 130 IoT devices, which helps us emulate rather complex scenarios. Operational rules for automation of these devices are implemented as appropriate Python routines. Table. 6 shows the number and type of devices in the Home-IO model.

B. Emulated IoT system in Home-IO

TABLE III: Home areas & controllers (one party per area)

Subsystem type	total #cntrls	#areas (parties)	#cntrls per party
Lighting	10	2	5
Fire&safety	6	2	3
Security	10	2	5
Surveillance	12	3	4
Climate	14	(2,1)	(5,4)

The emulated IoT system comprises of five types of controllers as shown in Table III, along with their further division into subsystems, each of which is deployed in an area of the home and controlled by a different party. (The precise description of home areas is omitted due to lack of space). Overall, we have 12 various parties. Fig. 6 shows the devices owned by each type of controller. We designed interaction between the smart devices as "routines" that may be either user-invoked or event/time-triggered. The simulated model can catch and respond to any concurrency conflicts and safety property violations in real-time.

For experiment purposes, we created 50 ORs and 7 SPs, and these rules are verified against five types of safety properties defined in our previous work [1]. We have shown some examples of ORs and all the SPs in the Tables II-IV. We only list the textual version of the rules for readability and space reasons rather than the actual LTL version used for the implementation. As an illustration, rule R35 can be translated to LTL using the until (U) operator as follows:

$$\mathsf{G}[X^2(\mathsf{flood\_sensor} = \mathsf{activated} \wedge \mathsf{deactivate\_flood\_sensor} \implies \mathsf{flood\_sensor} = \mathsf{deactivated} \wedge \mathsf{evaporation}(v)\ \mathsf{U}\ (\mathsf{water\_level} = 0)]$$

C. Results and Discussion

The first task in calibrating the model is to generate ORs and SPs. We do this by a systematic substitution of devices from a baseline configuration. For example, the Home-IO model includes five sprinklers and five water valves, using which we generate 25 unique combinations of SPs for SP1. Overall, our model has 130 devices (sensors/actuators) and a total of 1323 SPs to be assigned to 12 parties. Similarly, we generate a total of 1674 ORs.

TABLE V: Sequence of Steps in GMD algorithm

Step 1- Initial Greedy Assignment	
Assignment	Cost of the Assignment
SP1 ->P3	55 * 1
SP2 ->P1	40 * 2
SP3 ->P1	10 * 2
Total cost	**155**
Step 2 - Movement/Reassignment	
Movement	Total Cost
SP2 moved to P2	(55 * 1) + (40* 3) + (10 *2) = 195
SP2 moved to P3	(55 * 1) + (40* 3 + 55*2) + (10 *2) = 305
SP3 moved to P2	(55 * 1) + (40* 2) + (40 *2 + 55*2) = 285
SP3 moved to P2	(55 * 1) + (40* 2) + (10 *2) = 155
SP1 moved to P1	(40 * 1 + 10 *3) + (40* 2) + (10 *2) = 175
SP1 moved to P2	**(10 * 3) + (40* 2) + (10 *2) = 130**
Step 3 - Deletion	
Delete SP3? No	20<115? (deletion vs. nonenforcement cost)
Delete SP1? No	30<170? (deletion vs. nonenforcement cost)
Delete SP2? Yes	**20<12? (deletion vs. nonenforcement cost)**
After deletion, Total cost is 110	

D. An Illustrative Example

In illustrating how our algorithms work, we consider a toy example with three parties <P1,P2,P3> and three SPs <SP1,SP2,SP3> as shown in Fig. 3. The figure depicts the attributes owned by parties P1, P2, and P3, as well as the attributes for which SPs SP1, SP2, AND SP3 require access. The cost of enforcing and not enforcing each SP is also shown; for SP1, the value is <28,170>, where the first value of 28 represents the cost of enforcing the SP and the second value of 170 represents the cost of not enforcing the SP. The cost of inter-party access is indicated by red dashed lines.

1) GMD Algorithm: : The key steps in applying the GMD algorithm to our example are shown in Table V. We assign SP1 to P1 and SP2 to P3 due to the most local attributes in each case. SP3 can be assigned to P3 or P1 (equal number of attributes), and we arbitrarily choose P1. The total cost of the assignment is calculated using eqn* 4. In Step 2, we reassign SPs to reduce the cost further by considering all SPs for each party and selecting one that requires access to non-local attributes from the assigned party. For example, SP3 requires access to non-local attributes <A31,A33> of party

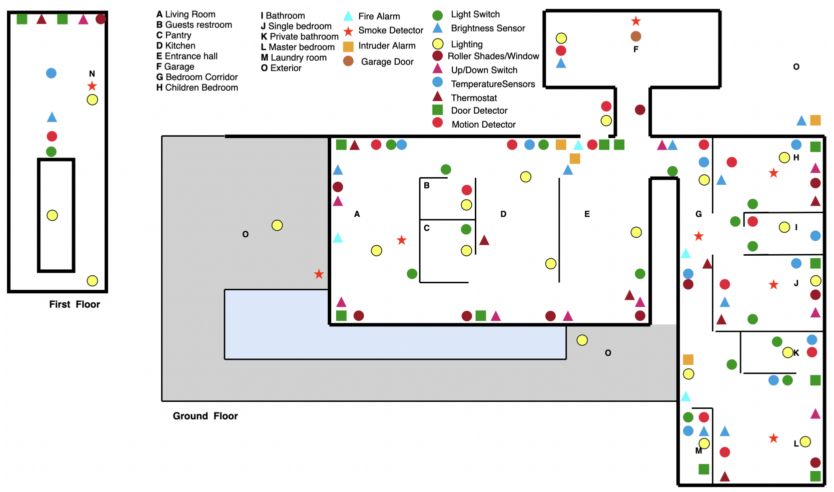


Fig. 5: Virtual floor plan created in Home-IO

Contro-llers	Actuator type & count		Sensor type & count	
C_{Fire}	Fire Alarm	3	Flood	5
	Sprinkler	5		
	Smoke Detector	5	Smoke	5
	Water -Pump	5		
C_{HVAC}	Heater	10	Temp	10
	Cooler	10		
	Windows	10		
	Roller-Shades	10		
C_{Surv}	Indoor-cameras	1	Motion	12
	Outdoor-cameras	1		
	Security Alarm	3		
	Intruder Alarm	3		
C_{Secu}	Main-door	1	Door	10
	Patio-door	1		
	Garage-door	1		
C_{Light}	Light-switches	19	Brightness	9
	User-mode	1		

Fig. 6: Ownership and Count of Devices with Controllers.

P1. Hence SP3 is considered for movement. Similarly, both SP2 and SP1 are considered for reassignment. For movement, we consider all other $K-1$ possible locations for the SP to minimize the total cost. As shown, while no move is cost-effective for SP2 and SP3, moving SP1 to party P2 does reduce the total cost. In the final step, we consider deleting a SP if the cost of enforcing a SP is greater than the cost of not enforcing it by a party (a non-critical SP). In the end, we only enforce SP1 and SP2 at a total cost of 110.

2) Partitioning Algorithm: : Table. VI sketches the overall steps. For initial partition, parties with lower authorization costs assigned to the same group, which results in the partition 1 containing (P1, P3) and partition 2 containing P2. The algorithm then determines the D-values of the vertices(parties) in each partition, which is the difference between the external and internal costs of each vertex as specified by Eq.7. Since the gain in D value is the largest between parties P2 and P3, they are swapped. In the next step, the SPs are assigned based on the maximum number of local attributes covered by the partition, and SP1 is assigned to partition2, SP2 and SP3 are assigned to partition1. The final step is deletion, identical to GMD, and SP2 is deleted, reducing the total cost. Even this example shows that the partitioning approach is significantly faster than GMD and has a lower optimal cost than the GMD algorithm.

3) Results of finding enforcers: We first evaluate our algorithms on a small problem containing eight original SPs and five parties. The exact (Brute Force) solution is feasible in this case and takes only 6-7 minutes and yields an optimal cost of 130.

Fig. 7 compares the GMD, Partitioning, and Brute Force algorithms where the x-axis shows the time taken in milliseconds which is in log-scale; and the y-axis indicates the cost of the solution. Along with GMD/Partitioning, we considered the Greedy approach, which performs only the initial greedy assignment plus the deletion of non-critical SPs. It is seen that in both cases, the cost of the solution converges to the optimal solution as in brute force, but the Partitioning takes less time than the GMD. Initially, the greedy approach has the highest cost value but then converges to GMD. Notably, Partitioning has a lower overall cost than GMD, which has a higher initial cost but still converges to the optimal solution.

TABLE VI: Sequence of Steps in Partitioning algorithm

Step 1- Initial Partition	
Partition1	Partition2
P1 and P3	P2
Step 2 - Calculate D-Value	
Party	Internal, External Cost and Y-Value
P1	$I_{p1}=10,\ E_{p1}=40 \Rightarrow D_{p1}=30$
P2	$I_{p2}=10,\ E_{p2}=55 \Rightarrow D_{p2}=45$
P3	$I_{p3}=0,\ E_{p2}=90 \Rightarrow D_{p3}=95$
Step 3 - Swapping Between Partitions based on Gain	
Gain between P1 and P2	g_{p1p2} = $D_{p1}+D_{p2}-2*C_{p1p2}=-5$
Gain between P3 and P2	g_{p3p2} = $D_{p3}+D_{p2}-2*C_{p1p2}=30$
Swap P3 and P2	**Part#1 - {P1 & P2}, Part#2 - {P3}**
Step 3 - Assigning SPs to the Partitions	
Assignment	Total Cost
SP1 assigned to Part#2	(50 * 1) = 50
SP2,3 assigned to Part#1	(50 * 1) = 50
After Assignment, Total cost is 100	
Step 3 - Deletion	
Delete SP3? No	20<115? (deletion vs. nonenforcement cost)
Delete SP1? No	30<170? (deletion vs. nonenforcement cost)
Delete SP2? Yes	**20<12? (deletion vs. nonenforcement cost)**
After deletion, Total cost is 80	

To assess the scalability of our proposed algorithms, we consider the full system described earlier, which has 1674 ORs, 12 parties/subsystems, and 1323 SPs. Since the exact (Brute Force) algorithm has exponential complexity, it can only go up to 16 SPs and five parties, or 2^{21} combinations (about one day of run-time), but others can all complete very quickly. In addition to GMD/Partitioning, we also considered the *Greedy approach*, which only does the initial greedy assignment plus deletion of non-critical SPs. Fig. 8 shows the comparison (note: y-axis is log scale). The Greedy approach differs slightly in cost compared to both approaches but later converges to optimal. This is because as the number of SPs and parties increases, the marginal benefit of assigning a new SP to any party decreases. This is because of the concept of the fictitious locals, and the majority of SPs will be assigned to the

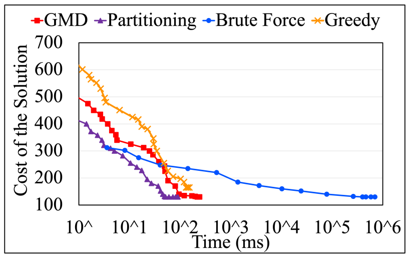


Fig. 7: Enforcement Cost vs. Time (5 parties)

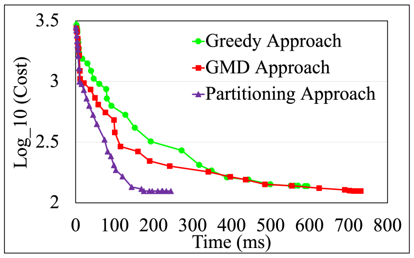


Fig. 8: Enforcement Cost vs. Time (12 parties)

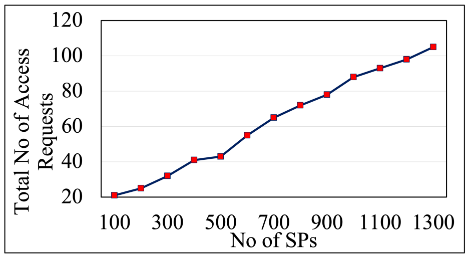


Fig. 9: No of Access Requests vs No of SPs

same party. It is seen that partitioning takes significantly less time than the other two approaches, but their eventual costs are almost identical. The eventual costs of GMD and partitioning are 125, while the optimal cost of the Greedy approach is 135.

4) Results of Run-time Access Control: We evaluate the performance of phase2 and phase3 of our proposed architecture. After assigning enforcers for the SPs, we built a repository for each node as described in section VI. We then conducted experiments to actually request and grant access rights as needed by the enforcers.

In our experimental setup consisting of 1323 SPs and 12 parties, there are a total number of 105 access requests made between the enforcer parties and the parties owning the attributes. The results are shown in Table. VII, it can be seen that the average time taken per SP is around 21.5 milliseconds. This does not account for the network delay, which would vary by the type of network and the message encryption used. However, we do show the number of communications required in Fig. 9 as a function of system size (in terms of the number of SPs). As expected, the total no of communications increases almost linearly with the size, with roughly five comm/SP enforcement. We believe that the access control costs are pretty modest for use in real multiparty IoT systems.

TABLE VII: Run-time Access control results

#Access Requests	105
Min Time (ms)	15.5
Max Time (ms)	35.1
Avg Time (ms)	21.5
Std Dev	2.40

VIII. Discussion and Conclusions

In this paper, we consider the problem of access control in large-scale IoT systems consisting of multiple subsystems (or "parties"), which must collaborate to avoid or resolve operational conflicts. Even in a trusted environment, careful access control is essential to lower the attack surface. We address this problem in three phases. First, we explore approaches to efficiently assign enforcing parties to each safety property (SP) in the system based on an authorization cost metric. All of these approaches quickly converge to the exact solution for small problems (where the exact solution is feasible). For larger problems, they all converge to the same value, thereby strengthening the belief that they provide a good, if not optimal, solution. In phase2, we build the access-control infrastructure, which phase3 then uses for run-time access control. In the future, the work can be extended along many vectors, including placing restrictions on the sharing of operational rules and safety properties, and tighter control over the provided run-time accesses.

References

[1] P. Pradeep *et al.*, "Automating conflict detection and mitigation inlarge-scale iot systems," *Proc. of CCGrid Conference*, May 2021.

[2] D. F. Ferraiolo *et al.*, "Proposed nist standard for role-based access control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

[3] P. A. Bonatti *et al.*, "A uniform framework for regulating service access and information release on the web," *Journal of Computer Security*, vol. 10, no. 3, pp. 241–271, 2002.

[4] P. N. Mahalle *et al.*, "Identity authentication and capability based access control (iacac) for the internet of things," *Journal of Cyber Security and Mobility*, vol. 1, no. 4, pp. 309–348, 2013.

[5] H. Aldowah *et al.*, "Security in internet of things: issues, challenges and solutions," in *IRICT*, 2018, pp. 396–405.

[6] A. Ouaddah *et al.*, "Access control in the internet of things: Big challenges and new opportunities," *Computer Networks*, vol. 112, pp. 237–262, 2017.

[7] S. Pal *et al.*, "Policy-based access control for constrained healthcare resources in the context of the internet of things," *Journal of Network and Computer Applications*, vol. 139, pp. 57–74, 2019.

[8] B. Bezawada *et al.*, "Securing home iot environments with attribute-based access control," in *ACM ABAC*, 2018, pp. 43—53.

[9] A. Ouaddah *et al.*, "Towards a novel privacy-preserving access control model based on blockchain technology in iot," in *Europe and MENA cooperation advances in information and communication technologies*, 2017, pp. 523–533.

[10] T. Hardjono *et al.*, "Verifiable anonymous identities and access control in permissioned blockchains," *arXiv preprint arXiv:1903.04584*, 2019.

[11] 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.

[12] Y. Zhang *et al.*, "Smart contract-based access control for the internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1594–1605, 2019.

[13] J. Qiu *et al.*, "A survey on access control in the age of internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4682–4696, 2020.

[14] S. Ravidas *et al.*, "Access control in internet-of-things: A survey," *Journal of Network and Computer Applications*, vol. 144, pp. 79–101, 2019.

[15] F. Curbera *et al.*, "Unraveling the web services web: an introduction to soap, wsdl, and uddi," *IEEE Internet computing*, vol. 6, no. 2, pp. 86–93, 2002.

[16] T. G. Stavropoulos *et al.*, "awesome: A web service middleware for ambient intelligence," *Expert Systems with Applications*, vol. 40, no. 11, pp. 4380 – 4392, 2013.

[17] J. Zhang *et al.*, "Wireless sensor network key management survey and taxonomy," *Journal of network and computer applications*, vol. 33, no. 2, pp. 63–75, 2010.

[18] B. Riera *et al.*, "Home i/o: a virtual house for control and stem education from middle schools to universities," in *IFAC ACE*, 2016.

[19] L. Lamport *et al.*, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[20] A. Al-Farooq *et al.*, "A formal method for detecting rule conflicts in large scale iot systems," *IFIP/IEEE IM*, 2019.

[21] M. Athamnah *et al.*, "Multiparty database sharing with generalized access rules," in *CloudCom*, 2016, pp. 198–205.

[22] S. Dutt, "New faster kernighan-lin-type graph-partitioning algorithms," in *ICCAD*, 1993, pp. 370–377.

[23] "Samsung smartthings," https://www.samsung.com .