DACIoT: Dynamic Access Control Framework for IoT Deployments

Ashraf Alkhresheh, *Member, IEEE*, Khalid Elgazzar, *Senior Member, IEEE*, and Hossam S. Hassanein, *Fellow, IEEE*

***Abstract*—This article presents a dynamic access control framework for the Internet of Things (DACIoT). The main objective of DACIoT is to prevent unauthorized access to IoT devices and tightens the authorized access while an IoT device is in use. The rigidness of existing access control (AC) techniques in terms of manual policy specification, discontinuity of access decision making, and immutability to changing access behaviors makes these solutions fall short in highly dynamic IoT environments. DACIoT supports three functionalities that are lacking in existing AC solutions: 1) automatic policy generation; 2) continuous policy enforcement; and 3) adaptive policy adjustment. The DACIoT extends the standard reference model of the extensible AC markup language (XACML) with the added three functionalities to improve the adaptability of attribute-based AC policies to highly dynamic IoT environments. Results show that DACIoT provides improved security, dynamic adaptability, and can scale efficiently to IoT environments.**

***Index Terms*—Access control, artificial intelligence, context awareness, device authorization, information technology, Internet of Things (IoT), mobile computing, security and privacy.**

I. Introduction

THE TRACES of the IoT concept go back to the early work done by Ashton in 1999, which received worldwide attention from both academia and industry [1]. The general premise of the IoT is to extend everyday objects with computing capabilities to identify, generate, and communicate information regarding their physical environments. Typically, these objects are physical and virtual sensors, radio-frequency identification (RFID) tags, and smartphones [2]. A mixture of enabling software and hardware technologies have contributed to the success of the IoT. These technologies can be grouped into two categories: 1) IoT functional supporting technologies, such as RFID, GPS, cellular technologies, Wi-Fi, microcomputers, and microprocessors, which add intelligence to the connected objects enabling them to acquire, communicate, and process contextual information and 2) IoT nonfunctional supporting technologies, such as cryptography, authorization, and access control (AC) technologies, which improve the security and privacy of IoT environments by securing the data stored, processed, and exchanged between IoT devices and data consumers, hence increasing the adoption of the IoT, thereby using the technology to its full potential. The increasing number and density of connected IoT objects coupled with the developments in the integrated technologies have widened the range of potential IoT applications in several domains, including health care, intelligent transportation, logistics, and smart buildings [3].

Despite the persuasive advantages and benefits that IoT offers, the new technology has not achieved widespread adoption [4]. This is primarily due to the security and privacy concerns that IoT raises. The omnipresence of IoT devices allows for the collection, processing, and dissemination of what could be considered very sensitive information about individuals, which undermines their security and intrudes on their privacy. The overprovisioning of access permission to the data generated by IoT devices and advancements in data analysis techniques have opened the door for third parties (e.g., data brokers) [5] to aggregate, infer, and release sensitive information about the users of these devices. For example, collecting real-time energy data at fine granularity allows an electric utility company to study the power consumption patterns of the consumer for the purpose of improving customer experience. However, off-the-shelf data analytic tools can reveal subtle inferences about the behavior of the occupants, including daily schedules, repeated usage patterns, and anomalies.

Therefore, it has become evident that security and privacy concerns will impede the widespread use of the IoT technology, unless access to IoT devices can be tightly controlled. Otherwise, the security and privacy risks of the new technology will outweigh any of its benefits. AC techniques offer primary solutions to address these concerns, but the current state of the art falls short to adequately fulfill the requirements of highly dynamic environments, where many attributes consistently change and hence, the context on which access decisions are made. A novel dynamic access framework is required to live up to the challenge and meet the stringent requirements of IoT environments.

II. Background and Motivations

AC is a combination of three security concepts: 1) authentication; 2) authorization; and 3) accountability. Authentication is a two-step process, including an identification step in which the system asks the user to provide their valid and recognized

Manuscript received December 10, 2019; revised April 7, 2020 and May 18, 2020; accepted June 3, 2020. Date of publication June 16, 2020; date of current version December 11, 2020. *(Corresponding author: Khalid Elgazzar.)*

Ashraf Alkhresheh and Hossam S. Hassanein are with the School of Computing, Queen's University, Kingston, ON K7L 3N6, Canada (e-mail: khashraf@cs.queensu.ca; hossam@cs.queensu.ca).

Khalid Elgazzar is with the Department of Electrical, Computer, and Software Engineering, Ontario Tech University, Oshawa, ON L1G 0C5, Canada (e-mail: khalid.elgazzar@ontariotechu.ca).

Digital Object Identifier 10.1109/JIOT.2020.3002709

credentials to the system (e.g., username and password, smart card, and retina scan); a verification step by which a system verifies the credentials of the user. The authentication is completed when the credentials of accessing the entity are valid and accepted, otherwise, authentication fails [6]. Authorization determines whether an authenticated accessing entity has sufficient privileges to access system resources and what operations are allowed or prohibited for this specific entity on the resources of interest [7]. The level of authorization that an accessing entity can be assigned is determined by evaluating its associated properties, such as identity, role(s), proximity, access history, privacy preferences, against a set of predefined access rules. Accountability is the process that guarantees that all operations carried out by users, systems, or processes can be identified and that the mapping to the user and the operation are maintained [6]. This work focuses on the authorization concept. Both authentication and accountability are beyond the scope of this research.

There are three primary phases through which AC systems are developed.

- Phase I) It determines the high-level security and privacy rules according to which AC must be regulated. The collection of these rules referred to as access policy (e.g., HIPAA) [8].
- Phase II) It is the formal representation of the access policy and procedures, referred to as an AC model (e.g., role-based AC model (RBAC) [9]).
- Phase III) It is the development of the low-level software and hardware functionalities that implement and enforce the security rules defined in the access policies and formalized by the AC model. Typically, these functionalities are referred to as policy enforcement mechanisms (e.g., antivirus software and firewalls).

The three core elements of an AC system are the subject, object, and operation. The subject is the accessing entity that actively causes information to flow between system components or that changes the system state (e.g., a person, device, application, and process). The object is the passive entity that receives data (e.g., light bulb and door lock). The operation is the action invoked by a subject and applied to an object (e.g., set and get).

AC schemes have been studied extensively over the past few decades and remain an area of intense research interest. This is due to rapidly changing technologies and growing security and privacy concerns. However, controlling who can access what under which conditions is challenging in the IoT context. The extremely large number of heterogeneous and resource-constrained IoT devices communicating over dynamic, distributed, and *ad hoc* networks via low bandwidth connections creates a unique set of authorization and AC challenges, rendering standard AC policies, models, and mechanisms unfit for IoT scenarios.

The three major limitations in traditional AC mechanisms for IoT environments are: 1) manual access policy management; 2) discontinuous access decision enforcement; and 3) static access permission assignment. In the following, we summarize these limitations and their respective challenges.

A. Manual Access Policy Management

Traditional AC systems and authorization techniques assume closed computing environments where all users and resources of the system are known in advance. In addition, these techniques rely on AC policies that are configured by security experts, and are unlikely to change during runtime. However, due to the dynamic nature of IoT these assumptions are not valid for the following reasons: 1) unknown IoT entities (e.g., users, applications, and devices) can join and leave the system anytime and at their discretion; 2) the unbounded number of interactions among these entities can result in frequent changes in security and privacy requirements of the resource administrator and consequently frequent changes in the underlying AC policies; and 3) access to IoT devices will be controlled mostly by owners who may not have sufficient policy management skills to define AC policies with adequate security measures. In such environments, manual policy management, such as adding/removing access rules and identifying and resolving conflicts in access policies, becomes a complex and error-prone task. Therefore, the automation of the access policy generation process is required to overcome inflexibility in traditional policy management techniques, eliminate errors and conflicts in AC policies, and ensure authorized access in highly dynamic IoT environments.

B. Discontinuous Access Decision Enforcement

Traditional AC models are proposed to protect data that is permanently stored with static or infrequently changing AC policies. Typically, these models enforce access decisions only at the time access is requested and do not consider changes in access conditions while the resource is in use. Advancements in IoT enabling technologies along with ubiquitous connectivity have led to a new generation of smart services based on real-time data access. A delay in making access decisions when context changes may result in negative consequences. Therefore, continuity in access policy enforcement becomes a necessity in highly dynamic IoT environments not only at the time of request but also for the entire access session.

C. Overprovisioning of Access Permissions

Typically, AC models assign access permissions based on static considerations, such as identity or roles. However, these models usually result in defining access policies that assign more access permissions than are required by the accessing entity. To alleviate this problem, researchers have focused on improving the flexibility of AC policies such that policy administrators can define customized AC policies that consider, beside the identity and role, the dynamic access condition factors, such as time, location, purpose of access, and interrelationships among users [10]–[12]. The AC literature refers to these factors collectively as access context. However, the highly dynamic nature of IoT makes it difficult, if not impossible, to predict all access contexts and assign the appropriate access permissions to each possible context. This unique characteristic of IoT requires an adaptive permission assignment technique that adjusts AC policies at runtime in order to prevent the exploitation of outdated AC policies

and reduces the chance for users to abuse or misuse excessive access permissions.

The main contributions of this work are given as follows.

1) We introduce an automatic access policy specification schema that, in response to changes in access context and in compliance with business objectives, generates new AC rules at runtime. We use context, attributes, and predication to describe the core AC elements.
2) We design and implement a continuous policy enforcement (CPE) mechanism that captures with high granularity the frequent changes in the access context and potential modifications of the access policies while an IoT resource is in use.
3) We introduce an adaptive policy adjustment (APA) technique that dynamically refines the system access policies in response to changes in the device-to-device access behavior.

III. Related Work

Many efforts have been put to extending basic AC models with several features in order to address IoT-specific AC challenges. Research efforts in this direction can be broadly categorized into: improving AC policy management, enabling lightweight and real-time policy enforcement, and enabling dynamic and self-adjusted AC policies.

Improving Policy Management: Anggorojati *et al.* [13] proposed a capability-based context-aware AC delegation model (CCAAC) for federated IoT networks. The main objective of this work is to enable IoT devices to delegate access rights to another IoT device based on the attributes and context of the delegate and resources of interest. Although CCAAC improves access policy management through dynamic rule specification, it cannot adapt to dynamic context changes. Jindou *et al.* [14] integrated social network services (SNSs) with RBAC to control access in Web of Thing (WoT) environments. The authors extend the standard RBAC to consider the user profile and social links in the user-role assignment, enabling personalization of access policies. Alkhresheh *et al.* [12] proposed a context-aware automatic access policy specification schema to prevent unauthorized data access in highly dynamic IoT environments. In this work, attribute-based AC (ABAC) [15] policies are automatically generated to overcome the inflexibility in traditional access policy specification techniques and improve adaptability to dynamic changes. The experimental evaluation of this proposal shows that it offers great flexibility and improved scalability in policy specification.

Enabling Real-Time Policy Enforcement: One approach to enable real-time policy enforcement is to implement the AC logic at end devices, enabling them to make and enforce access decisions in a peer-to-peer fashion. Hernåndez-Ramos *et al.* [16] proposed a distributed capability-based AC (DCapBAC) to implement the authorization logic in constrained end devices. This solution is a promising attempt toward achieving end-to-end security and trust between IoT devices. However, DCapBAC uses a central entity to issue access tokens to devices to enforce access decisions, which does not scale well in IoT environments. This idea is extended in [16] to control device-to-device communications using an extensible AC markup language (XACML)-based architecture to issue authorization tokens. Both approaches use static context in the decision making.

Another approach to real-time access policy enforcement is to perform the authentication process at the device level and offload the authorization task to less restricted edge devices (e.g., gateway). Alkhresheh *et al.* [17] proposed a continuous access policy enforcement mechanism for IoT deployments called CAPE. CAPE describes AC elements using predicates, and stores them as primitive facts (PFs) in a k-dimensional tree (K-D tree) data structure [18]. The mechanism matches access requests with PFs, generates access policies, and makes context-aware access decisions at runtime.

Enabling Self-Adjusted AC Policies: Atlam *et al.* [19] proposed an adaptive risk-based AC (AdRBAC) model for IoT environments. The model controls access based on contextual and risk factors, including user historical context (i.e., access behavior), resource sensitivity, action severity, and risk history. It uses smart contracts [20] to monitor the user's behavior to prevent any potential security breaching during access sessions. Recently, the authors extended their work to adopt XACML as a policy language and an architecture for AC in IoT settings [21]. Zhang *et al.* [22] proposed a smart contract-based AC framework for IoT systems. The framework uses blockchain to provide a distributed and trustworthy AC scheme. Also, Outchakoucht *et al.* [23] used the blockchain technology to build a trusted and distributed AC architecture in decentralized IoT environments. The authors adopt reinforcement machine learning algorithms to update the AC policies at runtime. Access policies are stored in smart contracts that execute automatically in response to authorized access requests. Although these approaches can provide some form of dynamic and adaptive access policies, they lack consistency in real-time access due to updating the policies after the fact and cannot continuously enforce authorized access based on dynamic changes.

IV. Proposed Framework

Although many basic principles of standard AC models continue to apply in IoT contexts, a holistic AC solution that caters to the highly dynamic nature of IoT is still missing. This article proposes a dynamic AC framework for the Internet of Things (DACIoT), a dynamic AC framework for IoT environments. DACIoT introduces three novel AC concepts that we deemed lacking in current AC frameworks: 1) automatic policy generation; 2) CPE; and 3) APA, which address, in one-to-one correspondence, the three aforementioned limitations in existing IoT AC solutions. We propose DACIoT as an extension to the successful XACML reference model defined by the organization for the advancement of structured information standards (OASISs) [24].

A. XACML Reference Model

XACML organizes access policies into three levels: 1) policy set; 2) policy; and 3) rule. It uses targets to index policy

sets, policies, and rules. A target is a predicate of one or more variables defined on the subject, the resource and the operation of access requests. The target specifies the type of requests a policy set or policy can be applied to. If a policy applies to a given access request, its rules are evaluated to make the access decision, otherwise, the policy is skipped.

The main components of the XACML AC framework are given as follows.

1) *Policy Enforcement Point (PEP):* PEP intercepts access requests to system resources and forwards them to the policy decision point (PDP) where access decision is made. PEP executes the access decisions it receives from PDP. It implements fulfilling obligations that PDP may include in the access decisions. An XACML obligation is an instruction from the PDP to the PEP on what action(s) must be performed before and/or after an access is approved.
2) *Policy Decision Point:* PDP evaluates access requests and makes access decisions based on available attributes that describe the access requests elements (subject, object, and operation), and applicable AC policies.
3) *Policy Information Point (PIP):* PIP acts as the source of attributes that PDP requires to evaluate access requests against ABAC policies. PIP collects attributes that pertain to the subject, resource of interest, and environment in which the access requests take place, and provides these attributes through the context handler (CH) to the PDP upon access decision making.
4) *Policy Administration Point (PAP):* PAP supports policy management functionalities, including adding/removing and modifying access policies. It also stores the access policies defined by the policy administrator.
5) *CH:* It controls the workflow of the AC system. It represents a hub through which PEP, PDP, and PIP communicate. CH forwards access requests it receives from PEP to PDP and returns the access decisions from PDP to PEP. It also fetches the attributes and resource content required for PDP to make access decisions.

We build our AC framework on the foundations of XACML for the following reasons.

1) XACML provides a standard AC architecture, policy language, and a request/response protocol [25], [26]. It offers system administrators high flexibility to define, modify, and reuse the authorization logic.
2) XACML uses ABAC policies to control access based on a wide range of attributes, including subject, resource, and environmental attributes. This provides high flexibility to describe general AC requirements by combining multiple AC policies and support specific business AC needs.
3) XACML provides an extensible and dynamic authorization approach, where AC policies can be dynamically extended to include new subject, object, and/or attribute in the authorization process.

B. Extending XACML Access Model

Despite the superiority that ABAC policies show over all previous AC policies, it suffers from three limitations in highly dynamic environments: 1) manual access policy management; 2) discontinuous access decision enforcement; and 3) overprovisioning of access permissions. To overcome these limitations, we replace three components in XACML and identify the lacking functionality in each component followed by our proposed solution. Then, we highlight the advantages that our solution provides over the corresponding XACML standard component. Fig. 1 depicts the XACML reference architecture with the proposed replacement components in dashed lines. We replace the three shaded components: PDP, PEP, and PAP with automatic policy specification (APS), CPE, and APA, respectively. In the following, we point out the limitations of the old components and describe the functionalities of the new replacing components.

1) Automatic Policy Specification: One limitation of traditional ABAC AC policy is that it describes core AC elements and their attributes holistically. Holistic in the sense that policy administrators need to consider all possible associations of the AC elements and define the attributes and conditions that govern every single association (i.e., rule) in the policy specification process. The tight coupling describes parallel relationships among AC elements, which limits the flexibility in policy expression. For example, if policy administrators are to add a new type of subject to the access policy, they need to define an access rule that describes the attributes and conditions that associate the subject with every type of object he is authorized to access. The same applies for adding new types of objects or operations.

Another limitation of ABAC policy is that it is static in the sense that access policies are specified at the setup time and do not change. For example, if the policy administrator is to define an access rule that increases or limits the access permissions of a certain subject on a certain object, the policy administrator needs to look up and update all predefined access rules that associate with the subject and object. Otherwise, the new access rule would create conflicts granting the subject inconsistent access permissions on the same object. Such an approach limits the adaptability of AC policies in dynamic access scenarios; it complicates the policy management and introduces significant policy maintenance overhead (e.g., policy conflict resolution).

APS is designed to overcome the inflexibility in policy specification methods. APS breaks down AC rules to their fundamental elements: subject, operation, and object, and creates two types of association: 1) element–element and 2) element–element–context. The former, which we call the element association, simply associates elements into subject–object, object–operation, and operation–subject pairs. The latter, which we call the guard context (GC), attaches the contextual conditions described in the original access rule to all element associations. We store the element associations and their GCs as PFs in the policy database as shown in Fig. 2. When an access request is received, an access rule(s) is automatically generated if there exists in the policy database element associations that match the elements of the access request; the access is granted if the real-world conditions of the request elements satisfy the GCs defined for each element association. Otherwise, access is denied by default.

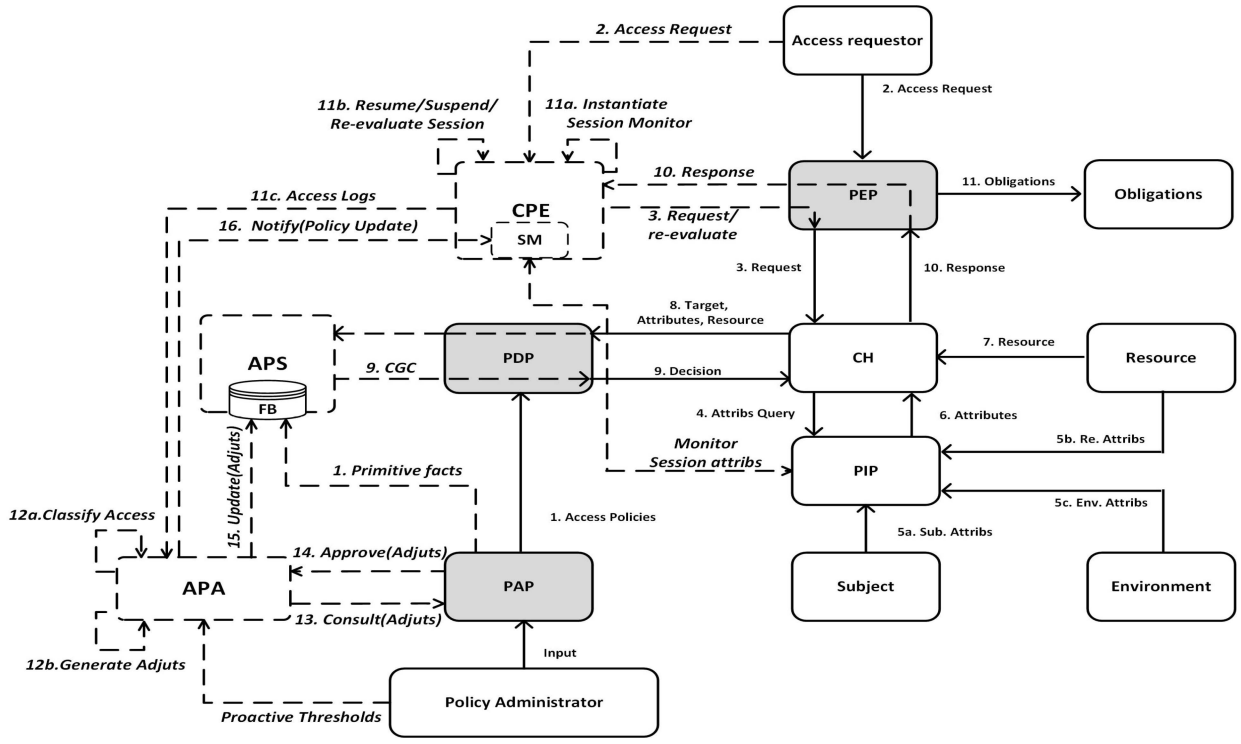

Fig. 1. Extended XACML reference model.

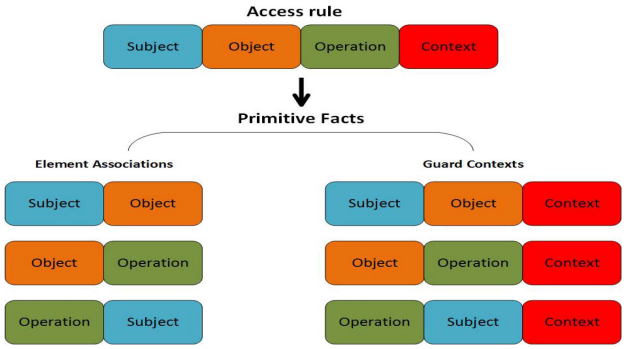

Fig. 2. Primitive facts.

GC: It is a set of predefined application-dependent values or thresholds represented as key–value pairs, for example (location: "lab room"), (time, "9:00 A.M."). These key–value pairs describe the environmental conditions defined by the object administrator under which a subject of certain attributes (e.g., identity, role, and access credits) can perform operations of certain attributes, such as (type: "read"), (granularity: "per minute") on an object of certain attributes, such as (CPU utilization: "$<$ 70%"), (energy level: "$>$ 80%").

Operational Context (OC): It is represented by key–value pairs similar to GC. However, values in OC are real-time measurements that reflect the real-world conditions of the subject, object, and the requested operation at the time of access. If these measurements satisfy the GC conditions that are defined separately for each element, then access is permitted. Otherwise, access is denied by default.

PF: We describe the core AC elements using abstractions, in a key–value pair representation, which contain both the attributes that characterize elements and the GC that determines the qualification context (or constraints) relevant to each element that controls access to objects. We build these basic abstractions and represent them in predicates as follows.

Element Abstraction: Element (X) is a descriptor represented by a tuple of the form `{type:value, key1: value1, key2: value2,...}`. The tuple contains a key *type* whose value is `{subject|operation|object}` to indicate this tuple is pertaining to the which of the access core elements. A descriptor is likely to contain a *time* and *location* keys that identify a certain time frame and location that control access to objects. For example, a subject must be in location x and time t to perform an operation of a certain object. The element descriptor is a unified fact defined by the object administrator on attributes of the access elements and their associated context. For example, the following basic abstraction represents a factual tuple for a subject x:

element x = `{type:"subject",name:"Any", study-level: "undergraduate",advisee: "True",location:"IoTResearchLab",time: "6:00-16:00"}`.

This descriptor contains a type key, three subject attributes that basically identify the subject, and two access constraints that determine the GC required to gain access to a protected object.

Request Abstraction: The request abstraction consists of a request template that defines the request elements and OC. The request descriptor contains patterns of the form

p = `{type:value, key1: value1, key2: value2,...}`.

Listing 1: Automatic Access Policy Specification Algorithm

```
   input : Primitive Facts PF, Access Request AR
   output: Access Decision, CGC
1  begin
2      for AR do
3          Extract all attributes values of the subject
4          x = {type : "subject", key_1 : value_1, key_2 : value_2 ...}
5          Extract all attributes values of the operation
6          y = {type : "operation", key_1 : value_1, key_2 : value_2 ...}
7          Extract all attributes values of the object
8          z = {type : "object", key_1 : value_1, key_2 : value_2 ...}
9      end
10     Generate PF-query{x}, PF-query{y}, PF-query{z}.
11     Result := Deny
12     for all element ∈ PF do
13         if ∃ subject(x) and operation(y) and object(z) in PF: then
14             Result:= Permit
15             CGC:= GC_subject ∩ GC_operation ∩ GC_object
16             Break
17         end
18     end
19     Return (Result,CGC)
20 end
```

For example, the following tuple represents a print service request (pertaining only to operation):

```
{type:"operation",name:"print",loca-
tion:"6th floor",time:"8:00-14:00"}.
```

A request descriptor matches an object if there is an operation in the fact base (FB) that satisfies every pattern in the request template.

In Listing 1, the access policy specification algorithm takes the PFs and the access request parameters as inputs and automatically produces the AC decision and the common GC (CGC) as an output.

When APS receives an access request (i.e., target and attributes) from the CH, it extracts the attributes associated with the access request at runtime, lines 2–6, and uses the PFs, stored in the FB, to dynamically compute access rules and make an access decision in response to the access request at hand, lines 7–11. In line 13, the algorithm calculates the CGC, which is the intersection of GCs associated with the subject, operation, and object elements of the generated access rule. CGC is a core parameter upon which *CPE* is ensured during an access session. Breaking the access rules down into PFs has many benefits: 1) it enables the AC system to generate and update access rules automatically and at runtime; 2) it allows the AC system to reuse PFs in the generation of access rules, thus reducing the space complexity as the number of rules increases; and 3) it removes the burden of detection and resolution of access rule conflicts because access rules are stored in form of PFs and a change in one fact would automatically affect all possible access rules that can be generated based on this fact.

2) Continuous Policy Enforcement: The literature of AC models that use context information in making access decisions divide into two implementation approaches. The first (and most simply implemented) is discontinuous enforcement, this approach verifies the context information, assigns the access privileges, and makes the access decision only at the time access is requested. It does not consider changes in context after the access decision is made, such as the work by authors in [13] and [27]–[29]. This approach can result in security and privacy breaches by disclosing system resources to users whose access context has changed since the time the access decision was enforced, and therefore, they become nonauthorized users. The second is continuous enforcement. This approach constantly monitors the OC of an access session and continuously updates the assigned access privileges according to changes in access context and discloses system resources only to those users who are authorized under the current context. However, this approach, if not carefully implemented, may place considerable computation and communication burdens on the AC system. While discussion on the first approach has dominated the research area in recent years, there is little, if any, research investigating the practicality of the second approach in high dynamic environments, such as IoT.

CPE aims to capture changes in access context at fine granularity and continuously enforce the appropriate access permissions in highly dynamic IoT scenarios, while not posing significant computation or communication overhead. It can handle two types of context changes: 1) when the OC changes, which are more frequent and happen as a result of changes in real-world conditions of the access elements (e.g., mobility) and 2) when the GC changes, which are less frequent and happen as a result of updating the access policies (i.e., add/remove access rules), or inferring new facts that could affect the access decision of an ongoing access session. The CPE supports the following functionalities.

1) *Session Registry (SR):* The SR stores a status record for each active session. The status record maintains the session id, the initial GC upon which the request was granted, and a status flag that indicates the current status of an access session (e.g., active, inactive, or suspended).
2) *Session Monitor (SM):* We extend the CH with an SM functionality that continuously reads the environmental attributes from available sensors, and potential policy update notifications from the *adaptive policy adjuster component*. The SM keeps track of changes in the OC and CGC associated with the access elements involved in an active session.

When APS grants a new access session, it passes the target, access decision, and CGC of the granted access request to the CPE component. In Listing 2, CPE uses the information it receives from APS and performs the following steps: 1) it creates an access session that connects the subject to the resource and stores the session parameters in the SR, lines 4–9; 2) it initiates an instance of SM to monitor the OC of the access session, line 6; 3) in lines 12–18, the SM enforces the CGC constraints over the lifetime of an access session; it ensures that the initial CGC of an ongoing access session is always satisfied by the current sensor inputs. Otherwise, the access session is terminated. If the PFs are updated, our algorithm, lines 21–24, does not interrupt the ongoing access sessions, rather it re-evaluates all ongoing access sessions against the changes in the PF. For each ongoing access session a re-evaluation access request is submitted to the CH, and the CH proceeds with the normal request evaluation procedure. If an

Listing 2: Continuous Access Policy Enforcement Algorithm

```
   input  : Target T , CGC, Session registry SR, policy_update
   output : Session History SH
   initialize: SR ← {} , SH ← {}
1  begin
2      if T ∉ SR then
3          Create new session S
4          S.id := T // Attributes
5          S.status := Active
6          S.OPC := new SM ( T ) // monitor
7          SR ← {S}
8      end
       // Update CGC for re-evaluated sessions
9      S.CGC := CGC
10     while True do
11         if policy_update = False then
12             for all S ∈ SR do // Check OPC changes
13                 if S.OPC ⊈ S.CGC then
14                     SR.Remove(S)
15                     S.terminate()
16                     SH ← {Req_{T,CGC} terminated} // Logs
17                 end
18             end
19         end
20         else
21             for all S ∈ SR do // re-evaluate sessions
22                 CH ( S.id )
23                 policy_update = False
24             end
25         end
26     end
27 end
```

access request is denied upon re-evaluation, our algorithm, line 9, updates its CGC with the new value calculated by APS (ϕ in case of denial), therefore, the SM terminates the corresponding session once it checks the session OC against the new CGC. Otherwise, the access session resumes, however, with a new CGC that may restrict, relax or keep the access permissions previously associated with the access session; and 4) CPE provides the access logs it collects in line 16 to the adaptive policy adjuster component for further processing.

3) Adaptive Policy Adjustment: A major limitation in existing AC approaches is that they rely on policy administrators to define AC policies that always assign the appropriate access privileges to requesting entities. These approaches assume that all access conditions under which system resources can be accessed are known beforehand and are unlikely to change. However, the highly dynamic nature of IoT environments creates access contexts in which predefined AC policies cannot meet the security and privacy objectives of the policy administrator. In these access contexts, obsolete AC policies may either assign less access privileges than that actually needed by requesting entities; blocking access requests that become legitimate (e.g., emergencies), or assign more access privileges than that actually needed by requesting entity; exposing system resources to insider attacks [30]. While the former case reduces the availability and utilization of system resources, the latter case can extend further to compromise the confidentiality and integrity of these resources. In our framework, we give priority to addressing the problem of overprovisioning of access privileges and leave the former case for future work.

In IoT context, insider threats come from current or former connected IoT devices which have or had access privileges on the system resources, and the users in control of these devices, intentionally abuse these access privileges in a manner that negatively affects the security and privacy objectives of the resource's administrator. Unlike traditional Internet devices, a compromised IoT device can cause damage that extends to the physical world. The severity of these damages increases in sensitive contexts especially when these devices are controlled by system insiders. Detecting abnormal access behaviors, such as insider attacks in highly dynamic IoT environments is challenging due to highly dynamic access contexts under which the IoT device can be accessed. In addition, AC policies become obsolete quickly in IoT environments due to frequent changes in security and privacy requirements, which further increases the attack surface of the system resources.

Recently, the detection and prevention of insider threats have attracted the interest of researchers in the IoT security field. A number of solutions have been proposed to approach insider threats in IoT environments based on behavioral models and anomaly detection techniques [31], [32]. Although these approaches do not focus on the adaptation of the AC policies as a countermeasure to prevent insider attacks, they give great insights for insider threat detection in IoT environments.

APA is designed to react instantly to changes in access context and adjusts the AC policies at runtime with minimal or no human intervention. With this, we prevent users of IoT devices from abusing their access privileges or exploiting obsolete AC policies to gain unauthorized access. We leverage the experience from the field of anomaly detection to build the APA components. APA classifies the device access behaviors based on proactive measures provided by the policy administrator and uses the knowledge it acquires from detected abnormal accesses to generate access policy adjustments at runtime.

APA implements two subcomponents: 1) access behavior classifier and 2) access policy adjuster.

Access Behavior Classifier: An access behavior represents the manner in which a user utilizes the system resources. We define the access behavior as the set of access requests submitted by the user to the authorization server (AS) within a predefined time window. Formally, we define the access behavior, denoted by AB, as follows:

$$\mathrm{AB} = \{\mathrm{AR}_{t-k}, \mathrm{AR}_{t-k+1}, \ldots, \mathrm{AR}_t\}$$

where AR_{t-k} is the first access request in the access behavior AB in a time window k. Each access request is represented by the set of attributes that characterize the elements of the access request (i.e., user, resource, and operation) and the context information that describes the environment in which the access request takes place (e.g., time and location). Formally, we define the access request, denoted by AR, as follows:

$$\mathrm{AR} = \{\mathrm{AT}_u, \mathrm{AT}_r, \mathrm{AT}_o, \mathrm{AT}_c\}$$

where $\mathrm{AT}_u = \{at_1^u, \ldots, at_x^u\}$ is the set of user attributes, $\mathrm{AT}_r = \{at_1^r, \ldots, at_y^r\}$ is the resource attributes, $\mathrm{AT}_o = \{at_1^o, \ldots, at_z^o\}$ is the operation attributes, and $\mathrm{AT}_c = \{at_1^c, \ldots, at_w^c\}$ is the context or environment attributes. An attribute is expressed as

Listing 3: Access Policy Adjustment Algorithm

```
   input : Abnormal Access Behavior AB_0, Fact Base FB, Proactive
           Measures D, adjustment threshold TH_adj
   output: adjustments ADJ
1  begin
2      ADJ ← {}   // initialize
3      for Req ∈ AB_0
4          for A ∈ Req
5              Calculate Pr(D|A)
6              if Pr(D|A) ≥ TH_adj
7                  Query.target ← {A.key, penalty}
                   ADJ ← ADJ ∪ {Query}
8              end
9          end
10     end
11     ADJ ← {Consult_Admin(ADJ)}
12     for Fact ∈ FB
13         for Query ∈ ADJ
14             if Query.key ⊆ Fact.target
15                 Fact.update(Query.value)
16             end
17         end
18     end
19     Notify(CPE)
20 end
```

at = name op $value$, where op is a relational operator (e.g., $=, \neq, <$) between the attribute name and a value from the range of possible values of this attribute.

We define an abnormal behavior, denoted by AB_0, as the AB that consists of one or more access requests that violates administrator-defined proactive measures (e.g., frequent access denials and resource overuse).

The access behavior classifier performs two tasks: 1) the offline training, during which the classifier models the access behaviors based on the historical access information submitted by the CPE component (i.e., user access logs) and administrator-defined proactive measures and 2) the access behavior classification, during which the classifier component classifies incoming access requests and reports abnormal access behavior(s) to the access policy adjuster component.

Access Policy Adjuster: Once the classifier model is built, we need to apply the knowledge it acquires during the training phase to adjust the AC policies. Listing 3 shows the access policy adjustment algorithm. The adjustment algorithm takes the abnormal access behavior reported by the classifier and the administrator-defined proactive measures as inputs, and returns a set of access policy adjustments in the form of PFs update queries. The algorithm performs four steps given as follows.

1) It extracts the set of attributes that describe the subject, resource, operation, and environment associated to each request in the abnormal access behavior, and for all combinations of these attributes, it calculates the probability of a resource misuse given that the attribute (or combination of attributes) A is present in the abnormal behavior AB_0, lines 3–5.
2) It evaluates the calculated probabilities against an administrator-defined adjustment threshold, and generates new policy, lines 6–8, in the form of update queries. The update query is defined on the attribute(s) that contribute most to the abnormal access behavior. The query target consists of a key–value pair. The query key is set to the value of the attribute key or combination of keys; specifying to which access element(s) of a PF, the update query applies (e.g., subject, operation, object, environment, or a combination of these). The query value determines the penalty of resource misuse.
3) It consults the policy administrator and gets the final approved policy adjustments, and updates the FB by updating all PFs that contains a key–value pair that matches the query target, lines 12–18.
4) It notifies the CPE of the policy update line 19.

We calculate the conditional probability, line 5, as follows:

$$\Pr(D|A) = \frac{\Pr(A \cap D)}{\Pr(A)} \tag{1}$$

where $\Pr(D|A)$ represents the contribution of the attribute A to the abnormal behavior AB_0.

$\Pr(D)$ is the probability of resource misuse in AB_0, defined as follows:

$$\Pr(D) = \frac{\#\text{ of misuse incidents}}{k}. \tag{2}$$

$\Pr(A)$ is the probability of an attribute A to present in AB_0, defined as follows:

$$\Pr(A) = \frac{\#\text{ of } A \text{ presences in } AB_0}{k}. \tag{3}$$

$\Pr(A \cap D)$ is the probability of an attribute A to present in a resource misuse incident in AB_0, defined as follows:

$$\Pr(A \cap D) = \frac{\#\text{ of } A \text{ presences in misuse incidents}}{k} \tag{4}$$

k is the number of access requests in AB_0.

The adjustment threshold TH_{adj} is a predefined application-dependant value that is set by the policy administrator. The evaluation of the calculated probability can result in the generation of one of two types of policy adjustment: 1) revocation adjustment, which applies only to the element association type of PFs or 2) limitation adjustment, which applies to the GC type of PFs. Both types of policy adjustments can be defined to adjust PFs based on single or combined access elements. The former is a recommendation for the policy administrator to adjust the PFs based on access element (e.g., subject), while the latter is a recommendation for the policy administrator to adjust the PFs based on two or more access elements (e.g., subject and object).

Setting TH_{adj} to a small value increases the probability of the adjuster component to pose more restrictions on the access policies. For example, if we set TH_{adj} to a value that is less than $\Pr(A \cap D)$, the adjuster component will always recommend to adjust all PFs that contain the attribute A.

For attributes that have categorical values, such as the names of core access elements and locations, we set the value of misuse penalty to `Null`. For numerical attributes values, such as the time, we define the penalty, denoted as P_0, as a function of the attribute contribution to the abnormal behavior as follows:

$$P_0 = A.\text{value} - \big(\Pr(D|A) - \text{TH}_{\text{adj}}\big)(A.\text{value}). \tag{5}$$

For example, if the probability of the attribute (`subject-id: "Adam"`) to present in a misuse incident, in abnormal behavior, is greater than or equals to

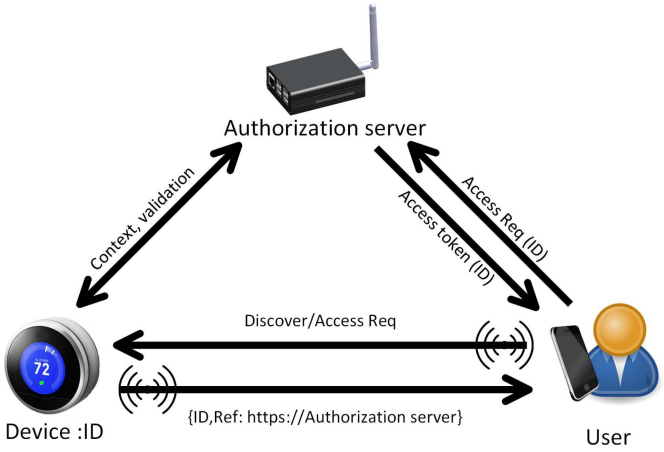

Fig. 3. Proposed AC architecture for DACIoT.

(TH_{adj}), the adjuster component generates an update query and sets the query target as follows:

$$\text{Query.target} \leftarrow \{\text{subject} - \text{id} : \text{“Adam”}, \text{Null}\}.$$

If the update query is approved by the policy administrator, the attribute (`subject-id:"Adam"`) is updated with (`subject-id: "Null"`) in all element associations whose subject is Adam, thus all access permissions that subject Adam has on the system resources will be revoked.

Assume that the adjustment threshold (TH_{adj}) equals 0.5 and the probability of the combined attributes (`subject-id:"Adam", time:"10:00-11:00"`) to present in a misuse incident is 0.75, the adjuster component generates an update query and sets the query target as follows:

$$\text{Query.target} \leftarrow \{\text{subject} - \text{id} : \text{“Adam”} \quad \text{time} : \text{“10:00} - \text{11:00”}, \text{time} : \text{“10:00} - \text{10:45”}\}.$$

If the update query is approved by the policy administrator, the combined attributes (`subject-id:"Adam", time:"10:00-11:00"`) is updated with (`subject-id: "Adam",time:"10:00-10:45"`), thus the time frame during which Adam can access the system resources is reduced by 15 min.

C. DACIoT Access Control Architecture

Fig. 3 depicts our proposed AC architecture for DACIoT. The architecture consists of three interacting entities: 1) the AS; 2) the accessing user; and 3) the target IoT device. AS is the entity that decides which IoT resources a user can access, by implementing the functionalities of the APS, CEP, and PAP. The implementation and placement of the AS may vary depending on the scale, latency constraints, and the number of users/devices in a specific area. An area of intense users/IoT devices deployment may require multiple servers, each serving a cluster of resources/users.

Deciding whether to locate the APS centrally to support multiple CEPs, to dedicate one APS to each CEP, or to adopt a hybrid of the two approaches can be associated with various latency and performance features. In particular, the placement of CEP (e.g., at the application side, at the resource manager side, or at the proxy) can contribute to how efficiently the AS handles a large number of access requests.

For these reasons, we chose to design a unique decentralized proxy-based AC architecture and we placed all AS components on the network edge. The proxy devices can be WI-FI access points, smartphones, or other edge node devices with sufficient computational resources and battery capacity. Our designed architecture brings many advantages that are given as follows.

1) *Technical:* The proxy device provides the computation and communication capabilities that IoT devices cannot support for resource-demanding computations, such as the automatic access policy specification and context monitoring. In addition, it lowers end-to-end latency required for continuous access policy enforcement, by having the access decision made close to where access requests are originating and real-time context information is provided. Furthermore, placing the APS and CEP within the same trust domain, removes the need for securing the communication channel between the two components.
2) *Nontechnical Advantages:* Decentralization preserves data privacy; it provides device owners or administrators with the ability to exert direct control over their data, rather than allowing it to be manipulated or shared by third parties. The architecture allows the data owner to determine who can access which device and for what reason before the data leaves their trust domain.

In the following section, we present a use-case scenario that illustrates the interactions among the three entities of the DACIoT architecture: 1) user; 2) IoT device; and 3) AS. The use-case provides several examples of access requests and demonstrate how the proposed AC functionalities (i.e., APS, CPE, and APA) react to these access requests. More information about the DACIoT architecture and control flow is also provided.

V. Use-Case Scenario

Suppose Adam, a graduate student at University X, is meeting his supervisor Eve in the School's conference room. According to Eve's calendar, the meeting is scheduled for 1 h (e.g., 10:00–11:00) every week. When Adam comes on campus, based on his current context (i.e., student's profile, location, and time), DACIoT grants Adam access to some resources campus wide, such as smart parking, Wi-Fi network, and school's main entrance. When Adam enters the building, DACIoT considers the change in Adam's context and dynamically assigns additional access privileges to Adam, based on the new fact (i.e., Adam's new location), such as entrance to the conference room. While waiting for Eve, Adam may not have privileges to access the conference room facilities. However, when Eve arrives, Adam's context changes again so that during the meeting time, DACIoT considers the coexistence of both Eve and Adam in the conference room and allows Adam to access conference room-specific resources [e.g., air conditioner (HVAC)]. Suppose that Eve has another meeting at 12:00. When Eve leaves the conference room, DACIoT considers the change in Adam's context and

Listing 4: XACML AC Policy

```
1 PolicyCombiningAlgId="Permit-Overrides">
2 <Target/>
3 <Policy PolicyId="SoC resources"
4 RuleCombinationAlgId="First-Applicable">
5 <Target/>
6 <Rule RuleId="1" Effect="Permit">
7 <Target>
8 <Subjects>
9 <Subject> faculty-member </Subject>
10 <Subject> staff </Subject>
11 <Subject> grad-Student </Subject>
12 </Subjects>
13 <Resources>
14 <Resource> smart-parking </Resource>
15 <Resources><Resource> main-entrance
   </Resource>
16 <Resources><Resource> wi-fi </Resource>
17 </Resources>
18 <Actions>
19 <Action> implied-action </Action>
20 </Actions>
21 </Target>
22 </Rule>
23 <Rule RuleId="2" Effect="Permit">
24 <Target>
25 <Subjects>
26 <Subject> supervisor </Subject></Subjects>
27 <Subjects>
28 <Subject> grad-student </Subject></Subjects>
29 <Resources><Resource> HVAC </Resource>
30 </Resources>
31 <Actions><Action> implied-action </Action>
32 <Environments>
33 <Environment>Location:conf-room <Environment>
34 <Environment> Time:10:00-11:00 <Environment>
35 </Environments>
36 </Target>
37 </Rule>
38 </Policy>
39</PolicySet>
```

TABLE I
PFs FOR ACCESS RULE 1

PF_1	{ type: "subject", subject-id :"Adam", role: "grad-stu", object-id:"smart-parking"}
PF_2	{ type: "operation", operation-id:"reserve", subject-id :"Adam", role: "grad-stu"}
PF_3	{ type: "object", object-id:"smart-parking", operation-id:"reserve" }
PF_4	{ type: "subject", subject-id :"Adam", role: "grad-stu", object-id:"Wi-Fi"}
PF_5	{ type: "operation", operation-id:"connect", subject-id :"Adam", role: "grad-stu"}
PF_6	{ type: "object", object-id:"Wi-Fi", operation-id:"connect" }
PF_7	{ type: "subject", subject-id :"Adam", role: "grad-stu", object-id:"main-entrance"}
PF_8	{ type: "operation", operation-id:"unlock", subject-id :"Adam", role: "grad-stu"}
PF_9	{ type: "object", object-id:"main-entrance", operation-id:"unlock" }

TABLE II
PFs FOR ACCESS RULE 2

PF_{10}	{ type: "subject", subject-id:"Adam", role:"grad-stu", object-id:"HVAC", time:"10:00-11:00", location:"conf-room", coexistence:"true" }
PF_{11}	{ type: "operation", operation-id: "control", subject-id :"Adam", role: "grad-stu", time:"10:00-11:00", location:"conf-room", coexistence:"True"}
PF_{12}	{ type: "object", object-id: "HVAC", operation-id:"control", time:"10:00-11:00", location: "conf-room" }

immediately revokes all access permissions Adam has to the conference room facilities. This scenario illustrates how DACIoT dynamically and continuously controls access to university resources.

A. DACIoT Policy Specification

Listing 4 shows the AC policy defined in XACML language to control access to the university resources considered in this access scenario. The policy includes two access rules: rule 1, lines 6–21, states that a faculty member, staff or a graduate student can reserve a parking spot in the school smart-parking, unlock the main entrance and connect to the school's Wi-Fi network; rule 2, lines 22–36, states that a graduate student can control the air conditioner (HVAC) in the conference room only in the presence of their supervisor and within the time frame 10:00–11:00. Table I shows the PFs that DACIoT generates to give the same effect of access rule 1 defined for graduate students in this use-case scenario. Note that no environmental (or contextual) conditions are defined in access rule 1. Therefore, the corresponding PFs simply define the subject–object, operation–subject, and object–operation associations for each object. For example, DACIoT generates PF_1 to PF_3 to control the access of the graduate students to the smart-parking object. Access rule 2 is an example of a context-aware access rule, it defines three contextual conditions a subject needs to satisfy to gain access to the protected object. The contextual conditions define restrictions on the subject location, request time, and subjects' coexistence referring to the collocation of subjects. We assume that the policy administrator uses the DACIoT *administrator interface* to define the university AC rules during system setup time. Table II shows the PFs that DACIoT generates to give the same effect of the access rule 2 defined for graduate students.

In PF_{11}, the key `coexistence` refers to collocation of both Eve and Adam in the conference room.

B. DACIoT Workflow

The AC goes through three operational phases: 1) discovery and authentication; 2) authorization; and 3) secure access. The mechanisms of device discovery and user authentication are beyond the scope of this research. Therefore, we assume that their functionalities are in place. The access starts by the requesting entity (i.e., Adam's smartphone) discovering resources in its proximity. We assume that Adam has a smartphone that supports Bluetooth low-energy technology (BLE) [33] to scan and discover available IoT devices and

services. If a resource of interest is publicly discoverable, the resource responds with its profile and the reference to the designated AS. Adam then submits an access request to AS. AS replies with a challenge asking the accessing entity for access credentials. If the requesting entity is authenticated, the AS evaluates the access request based on the attributes of the requesting entity, requested resource, and environment and sends the decision along with the set of access permissions back to the requesting application in form of self-contained access token. The user application attaches this token to every request to the protected IoT device. The IoT device verifies the access token every time a new access request is submitted. If the token is valid and the user has permissions, the IoT device allows access to its resources based on the set of permissions and access expiration time embedded in the access token. Otherwise, access is denied.

Fig. 4 shows the time sequence diagram that illustrates how the different components of DACIoT interact with each other to control access to IoT resources and maintain up-to-date AC policies during the lifetime of an access session.

The innovation in DACIoT is that, after an access request is permitted, the AS continuously checks the OC of the requesting and requested entities. If the OC does not satisfy the GC specified in the access token, the access token is deemed invalid and the access session is terminated immediately. For example, if Eve leaves the conference room at 10:30, all access tokens that Adam has obtained during meeting time continue to be valid time-wise until 11:00, however, if Adam submits an access request, for instance, to control the HVAC, the HVAC verifies the access token associated to Adam's request through the SM module. The SM module in turns extracts the current OC of Adam as well as the HVAC through the CH, and finds that the OC of Adam has changed (i.e., coexistence is false) and does not comply with the GC specified in the access token. As a result, the SM terminates the access session, returns an access token invalidation to HVAC. The HVAC in turn denies all subsequent access requests that use this access token.

To elaborate on how DACIoT handles APS and adjustment, suppose that the access logs generated by the SM module show that Adam, either accidentally or intentionally, attempted to access school resources (i.e., office or lab rooms he is not authorized to) without plausible reasons. Although the university access policy would deny Adam's access requests, this scenario may indicate that Adam is either abusing his access privileges for personal benefits or Adam's access key is being used by an attacker.

When Adam's access denials exceed, for instance, the threshold defined by the policy administrator for graduate students, the classifier module classifies Adam's access behavior as abnormal and triggers the adjuster module to provide the appropriate countermeasures. The adjuster module analyzes the abnormal access behavior of Adam and finds, for instance, that Adam's identity attribute (i.e., subject-id: "Adam") contributes most to the abnormal behavior with a probability of 95%, followed by Adam's role attribute (e.g., role: "grad student") with probability of y%, Adam's environmental attributes (e.g., location) with probability of z%, and so on. Based on the administrator-defined adjustment threshold (e.g., 0.75), the adjuster module recommends to revoke all access permissions assigned to Adam until he gets reviewed. In such a case, all policy adjustments are defined based on the subject identity (i.e., Adam). If the policy administrator approved the recommended policy adjustments, the adjuster module sets the subject-id to `Null` in all PFs in which the subject-id value is "Adam" (i.e., PF_1, PF_2, PF_4, PF_5, PF_7, PF_8, PF_{10}, and PF_{11}), and notifies the SM module of the PF update.

The SM does not interrupt Adam's other ongoing access sessions (e.g., Wi-Fi connection), rather it re-evaluates them against the up-to-date PFs. For each ongoing access session that Adam has, the SM submits a re-evaluation access request to the APS module. The APS considers the re-evaluation access requests as new access requests and queries the FB searching for PFs that match the access rules. The APS updates the CGC for each re-evaluated access session, which is null in this case (i.e., Adam has no permissions), and sends it back to the SM module. The SM module immediately terminates the corresponding access session (all access sessions that belong to Adam in this case).

Another interesting scenario is when the adjuster module finds that most of Adam's access denials took place in specific contexts. For example, 85% of Adam's access denials were in the conference room waiting for his supervisor. In such cases, the adjuster module recommends to limit Adam's access permissions by updating all PFs in which the subject is Adam and the location is the conference room (i.e., PF_{10} and PF_{11}). Such scenario results in partial revocation of access permissions assigned to Adam such that Adam is denied access to the resources in the conference room only, but still can access other university resources considered in this use-case scenario.

VI. Experimental Validation

To validate the functionality of DACIoT, we developed a dynamic AC prototype on an IoT environment. We carried out a number of experiments to validate DACIoT operation in order to ensure that it functions as expected. The prototype consists of two parts: 1) the client and 2) the AS. The client part is a mobile-based interface that supports two operational modes: 1) user and 2) administrator. The user interface enables the IoT users to search for and connect to IoT devices available in their proximity, including the designated ASs of the device. The administrator interface enables the policy administrator (or device owner) to define and manage AC policies for their IoT devices. This part of the prototype is deployed on a Samsung Galaxy S7 smartphone (quad-core 2.2-GHz Snapdragon 820, 32-GB storage, 4-GB RAM, Super AMOLED 2560x1440 pixels display, 5.1 inches) with a rooted Android 6.0 Marshmallow platform [34], connected to a Wi-Fi network. In addition, we assume that users in this case scenario use a smart key to access the external and internal doors of a university building.

The AS part is deployed on a Raspberry Pi 3 (quad-core 1.2-GHz ARM Cortex A53, 1-GB SDRAM, 10/100 MBPS Ethernet, 802.11n Wireless LAN, Bluetooth 4.0) [35] and

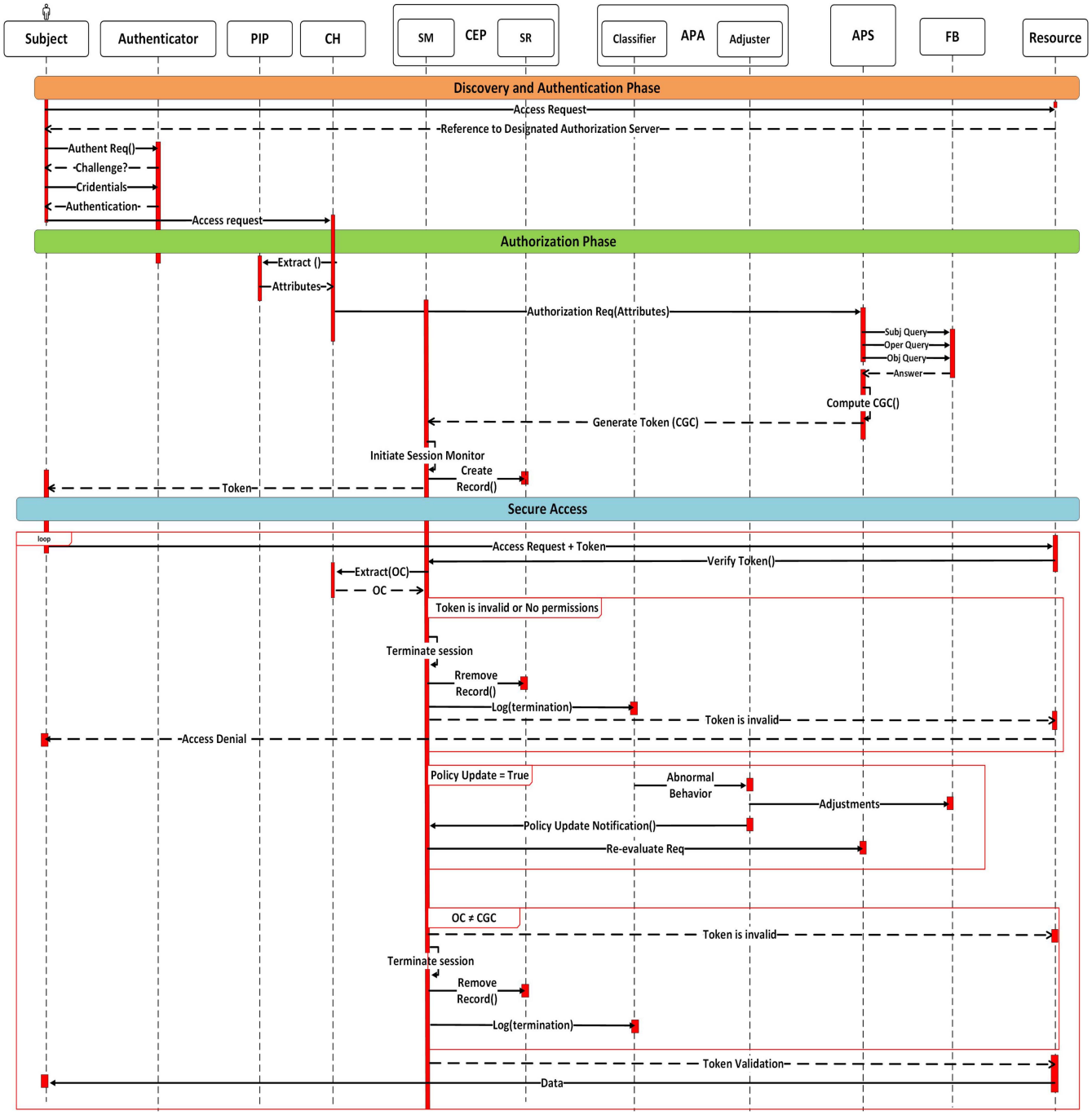

Fig. 4. DACIoT AC flow.

implements the authorization functionalities introduced in this article: APS, CPE, and APR. In addition, it handles device registration (i.e., authentication and attribute management) as well as context monitoring. For authentication, we authenticate users based on their cell phone numbers. For attributes management (i.e., PIP), we use a simple form of a database to store the subject, object, and operation attributes in a CSV formatted excel file and retrieve these upon access request evaluation. For environmental attributes, we consider two attributes: 1) subject location and 2) request time. The subject location is monitored based on Bluetooth proximity to the AS, and the time of requests is determined based on server local time. For IoT resources, we use two commercial IoT devices in this prototype: 1) a SensorTag [36] device representing an IoT device that continuously generates data, such as room temperature and 2) a WeMo smart switch [37] representing an IoT device that can be actuated. We use a smart switch and sensor tag devices to replace the mechanical door lock and Wi-Fi connectivity resources, respectively.

In our experiments, we look into three policy validation metrics, including access policy correctness, consistency, and completeness as follows.

Access Policy Correctness: This metric checks if an AC policy leaks access permission to unauthorized or unintended

users. In order to ensure the correctness of the AC policies that DACIoT generates, we used the DACIoT administrator interface to create the AC policy described in Listing 4. Next, we used the DACIoT user interface to submit access requests to the DACIoT server and observed the results. To ensure test coverage, we generated a set of test requests that encompasses all permutations of attribute values considered in our use-case. To simplify the generation of test requests, we assumed that the OC of an access request either satisfies or does not satisfy the contextual conditions (e.g., Adam is either in or out of the conference room). The total number of test requests is 512 test requests (2 subject-identities $\times$ 2 subject-roles $\times$ 4 objects-identities $\times$ 4 operations-identities $\times$ 2 location values $\times$ 2 time values $\times$ 2 coexistence values). A sample request is of the following format: `{subject-id:"Adam",role:"grad-stu",object-id:"HVAC",operation-id:"control",location:"conf-room",time:"9:00",coexistence:"true"}` and the generated decision is `{permit}`, similar to XACML response. We verified that the access decision generated by DACIoT for all the 512 test requests is the same as the decision generated by XACML. This provides confidence that the DACIoT logic is correct.

Access Policy Consistency: This metric ensures that the AC policy is conflict free. A policy or rule conflict occurs when access policies (or rules) grant in-consistent access permissions to an accessing entity [38]. Detecting policy (or rule) conflicts is challenging, it involves detecting the conflicting policies (or rules) and identifying the type of conflict among policies (or rules) at runtime. In addition, it requires dynamic verification to ensure conformance with security requirements specified by access policies. For example, XACML defines four policy or (rule) combination algorithms, including *First-Applicable*, *Only One-Applicable*, *Deny-Overrides*, and *Permit-Overrides*. These algorithms resolve conflicts when an access request applies to more than one policy or (rule). For a first-applicable policy or (rule), the decision of the first applicable policy or (rule) is returned. For an only one-applicable policy or (rule), the decision of the only applicable policy or (rule) is returned; `Indeterminate` (i.e., conflict or error) is returned if there are more than one applicable policy or (rule), or if one or more attribute(s) is missing. For a deny-overrides policy or (rule), `Deny` is returned if any policy or (rule) evaluation returns deny; `Permit` is returned if all policy or (rule) evaluations return permit. For a Permit-Overrides policy or (rule), `Permit` is returned if the evaluation of any policy or (rule) returns permit; `Deny` is returned if the evaluations of all policy or (rule) return deny. If an access request applies to none of these algorithms `Not-Applicable` is returned.

However, DACIoT requires no policy combination algorithm; DACIoT follows the white list authorization approach where all access requests are denied except those defined in the white list in the form of PFs. DACIoT removes the burden of detection and resolution of access policies (or rules) conflicts and always generates consistent access policies because DACIoT does not store access policies or rules permanently, rather it generates them on the fly based on the always up-to-date PFs. Therefore, a change in one fact would automatically affect all possible access rules that can be generated based on that one fact.

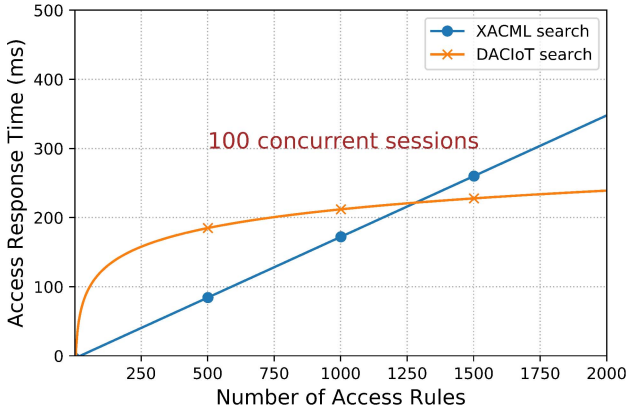

Fig. 5. DACIoT access response time.

Access Policy Completeness: This metric assures that each access request will be either accepted or denied by the AC policy. For example, if an attribute is missing from an access request, XACML policy does not make a final access decision, rather it shifts into an indeterminate state, which requires further processing (i.e., policy or rule combining algorithm). Unlike XACML policies, if DACIoT receives a request with a missing attribute, a subject attribute for instance, the APS will search the FB for a PFs that exactly match the three access elements of this request and can get into one of two states that both leads to an access denial: 1) no PF is defined for the subject with missing attributes, as such, access is denied and 2) one or more PFs are defined for the subject element with a missing attribute. Yet, due to missing attributes, the subject can only match the requested operation or the requested object but not both, hence access is also denied.

VII. Performance Evaluation

In order to evaluate the performance of the DACIoT, we conducted several experiments that investigate various aspects of DACIoT, including access response time, average re-evaluation time, access behavior classification accuracy, and policy adjustment accuracy (PAA). Experiments show how DACIoT can adapt to application security and time-sensitivity requirements offering three re-evaluation strategies. Experiments also demonstrate how DACIoT can maintain up-to-date AC policy recommending precise policy adjustments based on highly accurate access behavior classification.

A. Access Response Time

In our experiments, we define the access response time as the time that DACIoT requires to evaluate an access request and enforce the access decision. This includes access request analysis, attribute extraction, and PF matching.

Fig. 5 depicts the access response time of DACIoT and XACML AC engines for 100 concurrent access sessions versus a varying number of access rules.

To evaluate an access request, the XACML AC engine checks the attributes of the access request against the access

policy or policy set. If a request satisfies the target of a policy, then the request is further checked against the rule set of the policy; otherwise, the policy is skipped without further examining its rules. The same applies to the target of the policy set. The high complexity of XACML policies in terms of policy hierarchy and rule conflicts makes linear searching (i.e., brute force) the natural way of processing access requests.

In DACIoT, we structured the PFs in a K-D tree to achieve faster PFs matching, incur less overhead to access policy enforcement and facilitate runtime policy adjustment. We chose the K-D tree data structure because of its usefulness in applications that involve multidimensional search keys over large scale data sets. We built three separate K-D trees for the subjects, objects, and the operations offline. The time complexity of DACIoT is simply bounded by the time required to search each of the subject, object, and operation K-D trees. Assuming the three K-D trees are of the same size n, and the fact that K-D tree is a special case of binary trees, DACIoT has logarithmic time complexity T which is given as follows:

$$T(n) = 3 * N * O(\log(n)) \tag{6}$$

where N is the number of access rules. When $n >> 2^k >> N$, where k is the number of attributes and context constraints that describe each type of the core AC elements, the time complexity can be further simplified to

$$T(n) = O(\log(n)). \tag{7}$$

For each number of access rules, we calculate the access response time of DACIoT and XACML over 10 runs and take the average. Results show that at a low number of access rules the access response time of DACIoT tends to increase rapidly when the number of access rules increases. This is because the K-D tree search needs to visit relatively more tree branches for a lower number of points (i.e., access rules), however, DACIoT provides more flexible access rules than XACML. When the number of access rules increases significantly, DACIoT search outperforms XACML search for the following reasons: 1) XACML uses brute-force search whose time complexity is linear, while DACIoT uses decision trees in which the time complexity is logarithmic; 2) for a given access request, XACML needs to check all applicable rules in order to return an access decision, whereas DACIoT needs only to check the existence of the request's AC elements in the subject, object and operation K-D trees, starting with the shortest tree, to make an access decision. In the worst case, DACIoT searches all three trees and still outperforms XACML; and 3) XACML policies may have duplicate rules which likely to increase when the total number of access rules increases, and consequently the search time. However, DACIoT relies on PFs to generate distinct and conflict-free access rules, thus requiring less search time. In addition, when the number of access rules increases, the probability of fact-reuse increases which also significantly reduces the search time. For example, DACIoT is 60 ms faster than XACML at 1750 access rules. This result shows that our approach improves the flexibility of access rules and can scale efficiently to large scale environments, such as IoT.

B. Re-Evaluation Time

Re-evaluation time is the time DACIoT takes to re-evaluate all active access sessions in response to changes in the PFs, more specifically, it is the time elapses between the detection of changes to the PFs and the completion of re-evaluation of all ongoing access sessions against the changes to the facts. DACIoT supports three Re-evaluation strategies that cater to various security and time sensitivity requirements of different business applications. The re-evaluation strategies include: *Re-evaluate and Decide*, *Stop and Re-evaluate*, and *Hybrid*.

1) *Re-evaluate and Decide* supports application domains in which resource utilization is a priority rather than limited unauthorized access. Following this strategy, DACIoT does not interrupt ongoing access sessions, rather it first re-evaluates them against the AC policy and based on re-evaluation results, DACIoT only stops the denied access sessions. The strategy maximizes resource utilization giving a chance for users whose access sessions might not be affected by the policy changes to continue seamless access to protected resources. In addition, it incurs less re-evaluation time than other strategies because DACIoT only needs to terminate the affected access sessions and no further processing is required. However, this strategy compromises system security because it may expose system resources to unauthorized access during the re-evaluation time.
2) *Stop and Re-evaluate* strategy supports application domains in which system security is a priority. Following this strategy, DACIoT stops all ongoing access sessions when any changes in access context is detected, reserves their parameters, and re-evaluates the access sessions against the AC policy. Based on the revaluation results, DACIoT restores permitted access sessions. This strategy provides more secured access to system resources than other strategies at the expense of limited interruption. With this strategy, however, both system and users are subject to opportunity costs because legitimate users are blocked from utilizing system resources during re-evaluation time. In addition, the strategy incurs more re-evaluation time re-establishing nonaffected access sessions.
3) *Hybrid* strategy takes advantage of previous strategies and overcomes their limitations. Following this strategy, DACIoT switches between re-evaluation strategies based on application-dependent switching threshold. Several dynamic factors can be considered to calculate the switching threshold, including the user access history, resource sensitivity, and system threat level, among others. This strategy makes a fair compromise between resource utilization and security; it enables users to fully utilize system resources in normal access contexts and maintains sufficient system security in critical or unexpected access contexts.

Fig. 6 depicts the actual re-evaluation time of DACIoT versus a variable number of concurrent access sessions. The figure shows that the re-evaluation time for the three strategies is proportional to the number of concurrent access sessions and tends to increase slowly when the number of concurrent

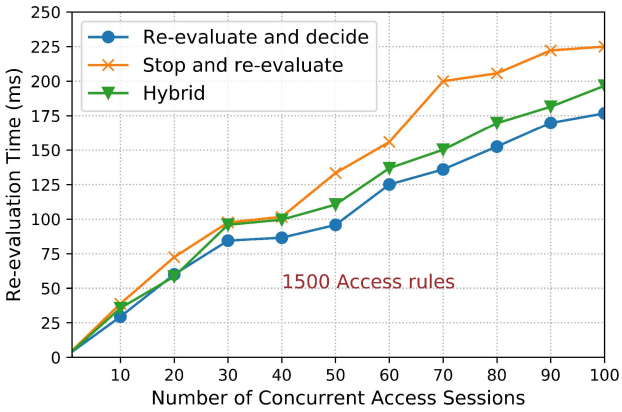

Fig. 6. DACIoT re-evaluation time.

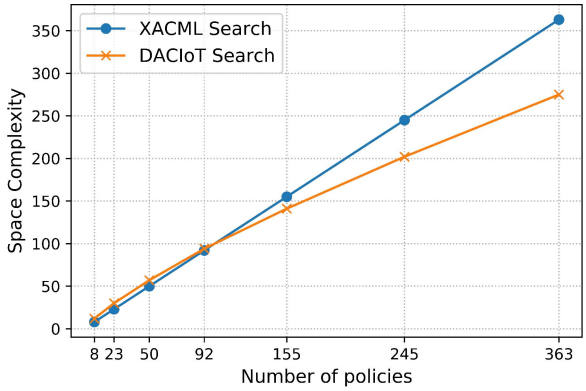

Fig. 7. DACIoT space complexity.

TABLE III
ESTIMATED RE-EVALUATION TIME OF DACIOT STRATEGIES

Strategy	Slope	Intercept	Estimated Re-evaluation Time (ms)
Re-evaluate and Decide	1.68	17.11	186
Stop and Re-evaluate	2.28	18.00	246.36
Hybrid	1.86	19.59	205.86

access sessions increases. In this experiment we use a random variable with normal distribution to generate the switching threshold values for the hybrid strategy.

We apply linear regression to obtain the estimated time for DACIoT re-evaluation. Table III shows the slope and intercept values for the estimated re-evaluation time of the three strategies. The slope values show that for every additional access session, the re-evaluation time increases by an average of 1.69, 2.28, and 1.86 ms for the Re-evaluate and Decide, Stop and Re-evaluate, and Hybrid strategies, respectively. The estimated re-evaluation time is calculated using the following equation:

$$\text{Time}_{\text{EST}} = b_0 X + b_1 \tag{8}$$

where b_0 is the slop, X is the number of concurrent access sessions, and b_1 is the intercept. For example, DACIoT can re-evaluate 100 concurrent access sessions against an entirely new access policy (i.e., 1500 access rules) in 205.86 ms using the Hybrid strategy. These numbers show that DACIoT can efficiently adapt to frequent changes in AC policies and maintain authorized access and usage of protected resources in highly dynamic and large-scale IoT environments.

C. Space Complexity

Unlike traditional AC approaches, such as XACML, DACIoT does not store AC rules in their final form, rather it decomposes AC rules into PFs. To evaluate the space complexity of DACIoT, we assume that each AC element requires one memory unit of storage. Therefore, a single XACML rule (i.e., subject–object–operation) consumes three memory units, whereas the corresponding DACIoT PFs (i.e., subject–object, object–operation, and operation–subject) consume six memory units. This seemingly gives advantage to XACML over our approach. However, assume a system with two subjects S_1, S_2, one object O and one operation OP. For this system, XACML needs to define two access rules as follows: $R_1 : S_1 - O - OP$ and R_2:$S_2 - O - OP$. The two rules require six memory units. Note that the information of the object and operation in R_1 is duplicated in R_2. DACIoT defines the corresponding PFs as follows: $S_1 - O$, $O - OP$, $OP - S_1$ for R_1 and it only defines $S_2 - O$, $OP - S_2$ for R_2. Therefore, DACIoT requires ten memory units to store the two rules. Fig. 7 shows the space complexity of DACIoT and XACML versus the number of access policies. DACIoT begins slightly lagging at a low number of access policies, but significantly outperforms XACML when the number of policies increase.

D. Access Behavior Classification Accuracy

To evaluate the classification accuracy of DACIoT, we used a real-life data set that consists of 180 000 access logs to the doors of the School of Computing at Queen's University. The data collected over the time period from December 1, 2016 to April 30, 2019. Each access log contains the following parameters: user ID (i.e., key ID), door ID, user attributes, door attributes, access time, and access value (i.e., decision). We defined the following cases as abnormal access behaviors: 1) if the user is denied access three times with normal access context (i.e., weekdays, and/or during morning, afternoon and evening) and 2) if the user is denied access two times in a row in sensitive context settings (i.e., weekends and/or during night).

Our classification problem belongs to the many-to-one category of sequence classification problems. We want our classifier to take a sequence of multiple access requests as input and map it to one behavioral class as output. We implement the DACIoT classifier component using a recurrent neural networks (RNNs) [39]. We chose RNNs because it is designed to work with sequence prediction problems. However, we want the classifier to classify the access behavior every time a new access denial is recorded. Therefore, the length of the input sequence is variable in our case. This requires careful engineering of the RNN input layer.

To show the effect of data size on the classification accuracy of different classifiers. We also implemented the

classifier component using the standard random forest classifier (RF) [40]. We used the Python libraries Scikit-learn [41] and Keras [42] to implement the RF and RNN classifiers. For the RNN approach, we used the long short-term memory (LSTM) [43] network. LSTM overcomes the training problem (i.e., vanishing gradients) associated with the RNNs and in turn has been used in a wide range of applications that involve large data sets.

1) Data Preparation and Labeling: To prepare the classification data we performed the following steps. First, we selected the top five users based on the total number of accesses. This cuts down the data size to 13 296 access logs. Since all parameters of the access logs are categorical and do not provide quantitative information, we applied the one-hot encoder (i.e., binarization) to these parameters converting into numerical input features to train the classifiers. Second, we defined 27 input variables as follows: five categories for the users, three categories for user attributes (i.e., faculty, staff, graduate students, and undergraduate students), two categories for the day of access (i.e., weekday and weekend), four categories to represent the access hour (i.e., morning, afternoon, evening, and night), 13 categories to represent the door ID which also represents the door location, and one category for the access value. We added an output label "Abnormal/Normal"İ behavior to the data set. The default value to this label is 0 (normal) to all access logs. We set the value to 1 (abnormal) for access logs that contain abnormal access behaviors. Otherwise, the output label is always set to 0 to denote normal access. Third, we split the data set into training and testing sets with the percentages 80 and 20, respectively. For the RF classifier, we fed the training data without modifications. For the LSTM classifier, however, we group the access logs between each subsequent abnormal behaviors into one access sequence. The resultant access sequences have different lengths, therefore, we calculate the length of the longest sequence and use zero padding to complete the short sequences. Fourth, we optimized the hyperparameters for both models on the testing data.

Finally, we generated 150 synthetic access logs with different probability distributions (e.g., uniform) from the original data, and evaluated the two classifiers using the mock data. To show the effect of data size on the classification accuracy of different classifiers, we conducted two experiments using two different data sizes (6000 and 10 000 access logs).

2) Classification Results and Discussion: One common approach to classification accuracy is to calculate the area under the receiver operating characteristic (ROC) curve, abbreviated to AUC. AUC is a measure of how well a classifier can differentiate between two examination samples (e.g., normal/abnormal access behaviors). We chose the AUC as our evaluation metric for the DACIoT classifier because AUC does not depend on the class distribution, which makes it useful for evaluating classifiers predicting rare events, such as abnormal access behavior as in our case. In contrast, evaluating the performance using the simple accuracy metric (the number of correct predictions made divided by the total number of predictions made) would favor classifiers that always predict a negative outcome (i.e., normal access behaviors) over a rare positive outcome (i.e., abnormal access behaviors). Fig. 8 shows the performance of the two classifiers when trained and tested on a relatively small number of access logs (4800 logs). The two classifiers score approximately the same AUC over variable values of decision thresholds. The results show that both classifiers poorly distinguish normal from abnormal behavior due to the imbalance in the data set (i.e., permit logs versus denial logs).

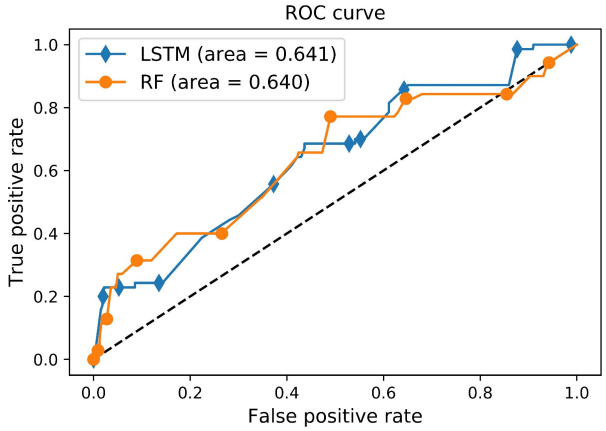

Fig. 8. Performance of LSTM versus RF/training data size = 4800 access logs.

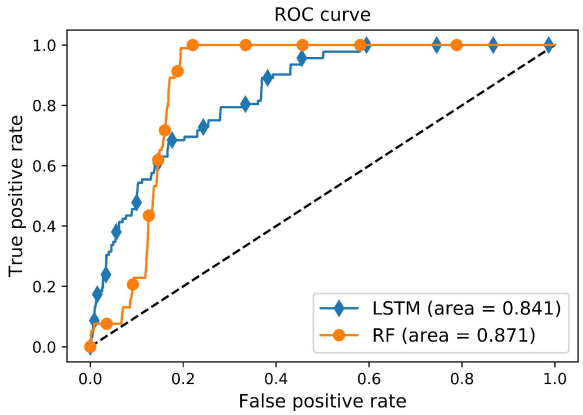

Fig. 9. Performance of LSTM versus RF, validation data = 150 logs.

Fig. 9 shows the performance of the two classifiers when validated using the mock data. RF outperforms LSTM because RF gives priority to the majority class (i.e., permit logs), while LSTM has less chance to learn from a small training data that contains short access sequences.

Fig. 10 shows the performance of the two classifiers when trained and tested on a relatively large number of access logs. Again, the two classifiers score comparable AUCs. However, the AUC results show that both classifiers can distinguish the two access behaviors with a high level of accuracy. LSTM outperforms RF because of the increased chances for the LSTM classifier to learn from longer access sequences in larger training data sets. With longer access sequences, LSTM can learn more about the access behavior of an individual user and build a deep insight on the correlations between the user access context and previous access decisions.

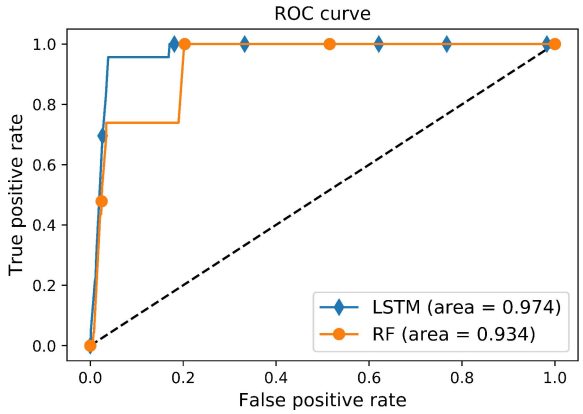

Fig. 10. Performance of LSTM versus RF, training data = 8000 logs.

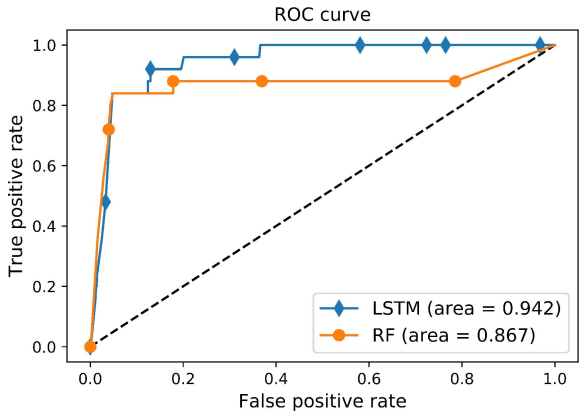

Fig. 11. Performance of LSTM versus RF, validation data = 150 logs.

Fig. 11 shows the performance of the two classifiers when validated using the mock data. The AUC scores indicate that LSTM accuracy improves faster when trained on larger data sets due to the ability to learn the best set of features that can generalize well on unseen data.

E. Policy Adjustment Accuracy

To measure the added security value that DACIoT brings adjusting an AC policy, we calculate the PAA. PAA is the percentage of improvement in the protection of system resources that DACIoT policy adjustment achieves over the original AC policy defined by a system administrator. Assume R is the number of policy adjustments DACIoT recommends and F is the number of policy adjustments the policy administrator refuses. Then, the PAA is calculated as follows:

$$\text{PAA} = \frac{R - F}{R}. \tag{9}$$

It follows from the definition of PAA that the added security value of DACIoT is directly proportional to the number of policy adjustments approved by the policy administrator. In this experiment, we run the LSTM classifier on the entire data set with a total number of 458 users. The classifier reports 466 abnormal access sequences out of a total of 495. We set the adjustment threshold to the minimum value (0.024), which is the maximum allowable number of access denials for a user (three in our case) divided by the maximum sequence length (124 access logs). With this threshold we allow the adjuster component of DACIoT to generate all possible policy adjustments.

TABLE IV
POLICY ADJUSTMENTS

	userID	DoorID	UserID&DoorID	UserID &Time	Total
R	279	301	21	115	716
F	249	291	11	97	648
PAA	0.042 %	0.014 %	0.014 %	0.025 %	0.094%

DACIoT recommended a total of 716 adjustments: 279 adjustments based on the user ID, 301 adjustments based on door ID, 21 adjustments for users on specific doors, and 115 adjustments for users in specific time of the day distributed as follows: 16, 27, 40, and 32 adjustments for night, morning, afternoon, and evening times, respectively. We present the list of adjustments to the policy administrator for analysis. The administrator approved 68 adjustments and provided the following feedback points.

1) Out of 279 (User ID) adjustments, 30 were approved. Out of the 30 adjustments, there are two cases where users should have been revoked their access keys. In the first case, the user committed 79 access denials on the same door, and in the second case, the user committed 10 access denials on seven different doors. The other 28 (User ID) adjustments are also recommended as combined attributes adjustments: 10 (User ID and Door ID) and 18 (User ID and Time) adjustments.
2) Out of the 301 (Door ID) adjustments, only 10 were approved.
3) Out of 21 (User ID and Door ID) adjustments, 10 were approved. In the ten cases, users were blocked access to doors they had been authorized to access, but due administrative changes in the access policy, access was denied.
4) Out of 115 (User ID and Time), only 18 adjustments were approved. Out of the accepted adjustments, there were 15 that recommend restricting access on afternoon times, three on night times. One possible justification is that there is an increased possibility that employees left their offices for lunch around noon time, which increases the chances for accidental or intentional use of wrong doors by authorized users.

Table IV summarizes DACIoT policy adjustment results.

For each category of policy adjustment, we present the number of generated and refused adjustments and the PAA. Setting the adjustment threshold to the low value in this experiment, resulted in less approved adjustments relative to the overall recommended adjustments, thus the low value of overall PAA (0.094%).

Fig. 12 shows the DACIoT PAA versus variable values of the adjustment threshold. The results show that DACIoT starts with low PAA at small values of TH_{adj}. This is because low values of TH_{adj} increase the sensitivity of the access policy adjuster to access denials, and also increase the probability of recommending policy adjustments that restrict access to

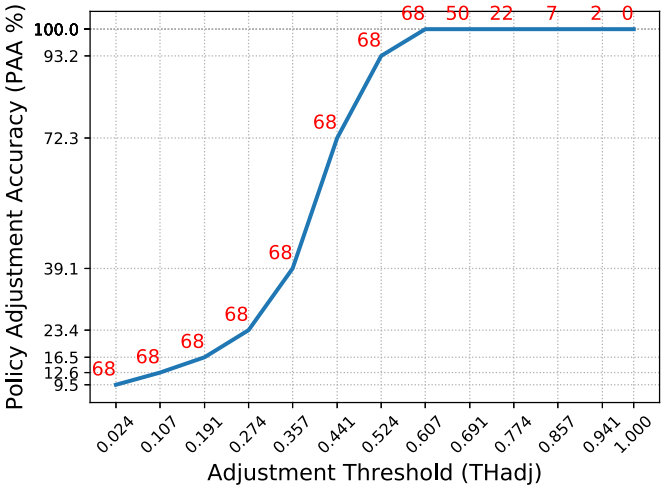

Fig. 12. PAA versus adjustment threshold.

users with infrequent access denials (i.e., high false-positive rate). This increases the number of refused policy adjustments relative to the total number of the recommended policy adjustments and therefore, reduces the PAA.

As TH_{adj} increases the PAA improves significantly because the policy adjuster becomes less sensitive to access denials (i.e., low false-positive rate) and the total number of approved policy adjustments (plotted in the figure) remains constant. When TH_{adj} equals 0.607 the PAA reaches 100%. At this value of TH_{adj}, all policy adjustments recommended by DACIoT are approved by the policy administrator. When TH_{adj} exceeds 0.607, PAA remains constant, however, the total number of recommended policy adjustments starts to decrease until it reaches zero recommendations for TH_{adj} values 0.941 to 100. Therefore, 0.607 is the optimum adjustment threshold value in this use case, where DACIoT is capable of striking a balance between the PAA and the total number of approved policy adjustments.

VIII. Conclusion

AC is a key enabler for the widespread adoption of the IoT technology. The highly dynamic nature of IoT environments poses unique AC challenges, rendering standard AC policies, models, and mechanisms unsuitable for IoT scenarios. In this work, we identified the limitations of current AC approaches and the requirements of a robust and reliable AC mechanism that can efficiently control access to IoT devices and dynamically adapt to frequent changes in access contexts.

This article presents DACIoT, a dynamic AC framework for IoT environments that improves access policy management, ensures continuous authorized access to IoT devices during the lifetime of an access session, and uses deep learning techniques to propose dynamic changes in the AC rules. The experimental validation of the DACIoT framework demonstrates its ability to dynamically generate correct, complete, conflict-free access rules, and continuously enforce context-aware and up-to-date AC policies.

In the future, we plan to extend our work with a policy caching mechanism that temporarily stores the dynamically generated access rules to improve the access response time. We also plan to extend our framework with an anomaly detection component to detect changes in normal access behaviors (i.e., unseen behaviors) at runtime to maintain accurate and up-to-date AC policies.

References

[1] K. Ashton, "That 'Internet of Things' thing," *RFID J.*, vol. 22, no. 7, pp. 97–114, 2009.
[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.
[3] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, 2010.
[4] M. Chui, M. Löffler, and R. Roberts, "The Internet of Things," *McKinsey Quart.*, vol. 2, no. 2010, pp. 1–9, 2010.
[5] "Data brokers: A call for transparency and accountability," Federal Trade Commission, Washington, DC, USA, Rep., Jan. 2014.
[6] P. Samarati and S. C. de Vimercati, "Access control: Policies, models, and mechanisms," in *Foundations of Security Analysis and Design* (Lecture Notes in Computer Science), R. Focardi and R. Gorrieri, Eds. Heidelberg, Germany: Springer, 2001, pp. 137–196.
[7] A. Jøsang, "A consistent definition of authorization," in *Security and Trust Management* (Lecture Notes in Computer Science), G. Livraga and C. Mitchell, Eds. Cham, Switzerland: Springer, 2017, pp. 134–144.
[8] H. C. Assistance. (2003). *Summary of the HIPAA Privacy Rule*. [Online]. Available: https://www.hhs.gov/hipaa/for-professionals/privacy/laws-regulations/index.html
[9] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, Feb. 1996.
[10] G. Zhang and J. Tian, "An extended role based access control model for the Internet of Things," in *Proc. Int. Conf. Inf. Netw. Autom. (ICINA)*, vol. 1, Oct. 2010, pp. 319–323.
[11] A. S. M. Kayes, J. Han, and A. Colman, "A semantic policy framework for context-aware access control applications," in *Proc. 12th IEEE Int. Conf. Trust Security Privacy Comput. Commun.*, Jul. 2013, pp. 753–762.
[12] A. Alkhresheh, K. Elgazzar, and H. S. Hassanein, "Context-aware automatic access policy specification for IoT environments," in *Proc. 14th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2018, pp. 793–799.
[13] B. Anggorojati, P. N. Mahalle, N. R. Prasad, and R. Prasad, "Capability-based access control delegation model on the federated IoT network," in *Proc. 15th Int. Symp. Wireless Pers. Multimedia Commun. (WPMC)*, 2012, pp. 604–608.
[14] J. Jindou, Q. Xiaofeng, and C. Cheng, "Access control method for Web of Things based on role and SNS," in *Proc. IEEE 12th Int. Conf. Comput. Inf. Technol.*, 2012, pp. 316-321.
[15] V. C. Hu *et al.*, *Guide to Attribute Based Access Control (ABAC) Definition and Considerations (Draft)*, DMCA, San Antonio, CA, USA, 2013.
[16] J. L. Hernández-Ramos, A. J. Jara, L. Marín, and A. F. S. Gómez, "DCapBAC: Embedding authorization logic into smart things through ECC optimizations," *Int. J. Comput. Math.*, vol. 93, no. 2, pp. 345–366, 2016.
[17] A. Alkhresheh, K. Elgazzar, and H. Hassanein, "CAPE: Continuous access policy enforcement for IoT deployments," in *Proc. 15th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, 2019, pp. 1576–1581.
[18] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.
[19] H. F. Atlam, A. Alenezi, R. J. Walters, G. B. Wills, and J. Daniel, "Developing an adaptive risk-based access control model for the Internet of Things," in *Proc. IEEE Int. Conf. Internet Things*, 2017, pp. 655–661.
[20] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. IEEE Int. Congr. Big Data (BigData Congr.)*, 2017, pp. 557–564.
[21] H. F. Atlam, M. O. Alassafi, A. Alenezi, R. J. Walters, and G. B. Wills, "XACML for building access control policies in Internet of Things," in *Proc. IoTBDS*, 2018, pp. 253–260.
[22] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1594–1605, Jun. 2019.

[23] A. Outchakoucht, E. Hamza, and J. P. Leroy, "Dynamic access control policy based on blockchain and machine learning for the Internet of Things," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 7, pp. 417–424, 2017.

[24] S. Godik and T. Moses, *OASIS Extensible Access Control Markup Language (XACML)*, OASIS, Burlington, MA, USA, 2002.

[25] A. X. Liu, F. Chen, J. Hwang, and T. Xie, "Designing fast and scalable XACML policy evaluation engines," *IEEE Trans. Comput.*, vol. 60, no. 12, pp. 1802–1817, Dec. 2011.

[26] D. Lin, P. Rao, R. Ferrini, E. Bertino, and J. Lobo, "A similarity measure for comparing XACML policies," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 9, pp. 1946–1959, Sep. 2012.

[27] M. J. Moyer and M. Abamad, "Generalized role-based access control," in *Proc. 21st Int. Conf. Distrib. Comput. Syst.*, Apr. 2001, pp. 391–398.

[28] P. N. Mahalle, B. Anggorojati, N. R. Prasad, and R. Prasad, "Identity authentication and capability based access control (IACAC) for the Internet of Things," *J. Cyber Security Mobility*, vol. 1, no. 4, pp. 309–348, 2013.

[29] L. Gong, "A secure identity-based capability system," in *Proc. IEEE Symp. Security Privacy*, May 1989, pp. 56–63.

[30] E. Shaw, K. Ruby, and J. Post, "The insider threat to information systems (security awareness bulletin no. 2–98)," Washington, DC, USA: Dept. Defense Security Inst., 1998.

[31] Y. Meidan *et al.*, "N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervasive Comput.*, vol. 17, no. 3, pp. 12–22, May 2018.

[32] I. Hafeez, A. Y. Ding, M. Antikainen, and S. Tarkoma, "Toward secure edge networks taming device to device (D2D) communication in IoT," Dec. 2017. [Online]. Available: arXiv:1712.05958

[33] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of Bluetooth low energy: An emerging low-power wireless technology," *Sensors*, vol. 12, no. 9, pp. 11734–11753, 2012.

[34] E. Alepis and C. Patsakis, "Hey doc, is this normal? Exploring Android permissions in the post marshmallow era," in *Proc. Int. Conf. Security Privacy Appl. Cryptography Eng.*, 2017, pp. 53–73.

[35] *Raspberry Pi 3*. Accessed: May 18, 2020.[Online]. Available: https://www.raspberrypi.org/products/

[36] *SensoTag*. Accessed: May 18, 2020. [Online]. Available: http://www.ti.com/tool/CC2650STK

[37] *Belkin's WeMo Switch*. Accessed: May 18, 2020. [Online]. Available: https://www.wemo.com/

[38] S. Jajodia, P. Samarati, M. L. Sapino, and V. Subrahmanian, "Flexible support for multiple access control policies," *ACM Trans. Database Syst.*, vol. 26, no. 2, pp. 214–260, 2001.

[39] D. E. Rumelhart *et al.*, "Learning representations by back-propagating errors," *Cogn. Model.*, vol. 5, no. 3, p. 1, 1988.

[40] T. K. Ho, "Random decision forests," in *Proc. 3rd Int. Conf. Doc. Anal. Recognit.*, vol. 1, Aug. 1995, pp. 278–282.

[41] *Scikit-Learn: Machine Learning in Python—Scikit-Learn 0.21.2 Documentation*. Accessed: May 18, 2020. [Online]. Available: https://scikit-learn.org/stable/

[42] *Keras Documentation*. Accessed: May 18, 2020. [Online]. Available: https://keras.io/

[43] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Dec. 1997.