IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 15, 2021, accepted July 27, 2021, date of publication July 30, 2021, date of current version August 6, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3101218

Attribute-Based Access Control for AWS Internet of Things and Secure Industries of the Future

SMRITI BHATT[1], THANH KIM PHAM[2], MAANAK GUPTA[2], (Member, IEEE), JAMES BENSON[3], JAEHONG PARK[4], (Member, IEEE), AND RAVI SANDHU[3], (Fellow, IEEE)

[1]Department of Computer and Information Technology, Purdue University, West Lafayette, IN 47907, USA
[2]Department of Computer Science, Tennessee Technological University, Cookeville, TN 38505, USA
[3]Department of Computer Science, Institute for Cyber Security, The University of Texas at San Antonio, San Antonio, TX 78249, USA
[4]Department of Management, Marketing and Information Systems, The University of Alabama in Huntsville, Huntsville, AL 35899, USA

Corresponding authors: Maanak Gupta (mgupta@tntech.edu) and Smriti Bhatt (bhattsmriti1@gmail.com)

This work was supported in part by the National Science Foundation at Tennessee Technological University under Grant 2025682, and in part by the NSF CREST Center at The University of Texas at San Antonio (UTSA) under Grant HRD-1736209.

ABSTRACT Internet of Things (IoT) is revolutionizing and enhancing the quality of human lives in every aspect. With a disruption of IoT devices and applications, attackers are leveraging weak authentication and access control mechanisms on these IoT devices and applications to gain unauthorized access on user devices and data and cause them harm. Access control is a critical security mechanism to secure the IoT ecosystem which comprises cloud computing and edge computing services along with smart devices. Today major cloud and IoT service providers including Amazon Web Services (AWS), Google Cloud Platform (GCP), and Azure utilize some customized forms of Role-Based Access Control (RBAC) model along with specific authorization policies enabled by policy-based access control models. To enable fine-grained access control and overcome limitations of existing access control models, there is an imminent need to develop a flexible and dynamic access control model for securing smart devices, data and resources in the cloud-enabled IoT architecture. In this paper, we develop a formal attribute-based access control (ABAC) model for AWS IoT by building upon and extending previously developed access control model for AWS IoT, known as AWS-IoTAC model. We demonstrate the applicability of our proposed model through an industrial IoT use case and its implementation in the AWS IoT platform. Our proposed fine grained model for AWS IoT incorporates its existing capabilities and introduces new attributes for IoT entities and attribute-based policies for enabling expressive access control in AWS IoT. We also evaluate the performance of our model on the AWS cloud and IoT platform with the future smart industries use-case to depict the feasibility of our model in a real-world platform.

INDEX TERMS Internet of Things, smart industries, future manufacturing, access control, security, privacy, digital twins, attribute-based access control.

I. INTRODUCTION

Internet of Things (IoT) is a rapidly emerging domain with billions of connected devices and data-driven applications that are enabling various smart infrastructures, such as smart homes, E-Health, smart transportation, smart farming [1], [2], and smart manufacturing. Today, we live in a giant interconnected ecosystem where IoT devices collect large amount of data associated with users and their surroundings and leverage cloud computing resources for storing and analyzing data in order to extract useful insights from the data. This evolving paradigm incorporating cloud computing, edge computing, and IoT is known as cloud-enabled IoT (CE-IoT) [3]. A real-world industry realization of CE-IoT is evident with major cloud services providers, such as Amazon Web Services (AWS) [4], Google Cloud Platform (GCP) [5], and Microsoft Azure [6], and their IoT platforms. In such largely interconnected and diverse cyberspace, new security and privacy risks associated with cloud and IoT users, devices, data, and applications are surfacing every day. In this paper, we focus on access control and authorization aspects in a real-world CE-IoT platform, AWS IoT [7].

The associate editor coordinating the review of this manuscript and approving it for publication was Vyasa Sai.

AWS is one of the largest[1] cloud services provider today. It offers various services including Internet of Things (IoT), AWS Greengrass[2] (an edge computing service), AI and Machine Learning and IoT Analytics. AWS IoT is a real-world cloud-enabled IoT platform which is build on AWS's cloud services. It allows IoT devices to securely connect and communicate with each other as well as other cloud services and applications. We envision that AWS cloud services and its IoT service can enable real-world implementations of various IoT domains including smart communities, smart industries, and smart healthcare in the near future. However, in such smart ecosystem with ubiquitous IoT devices and cloud computing services available to users, both legitimate users to benefit and malicious users to exploit, securing access to cloud resources, connected devices, and data collected from these devices is an ongoing research challenge. There is an imminent need for a dynamic and flexible access control model for CE-IoT.

A. MOTIVATION

The current state-of-art access control model utilized in most cloud computing platforms is Role-Based Access Control (RBAC) [8], [9] model, a dominant and mature access control model in the industry. Another access control model that has recently become popular in cloud platforms is Policy-Based Access Control (PBAC), which is also used in AWS cloud. However, there are some inherent limitations of these models, such as role explosion in RBAC. For example, creating various roles for assigning permissions on billions of devices and users would result in role-explosion problem. Previously developed AWS cloud access control (AWSAC) [10] and AWS IoT Access Control (AWS-IoTAC) [11] models are based on policy based approach and also utilize groups and customized roles for assigning permissions to various entities. With a large number of IoT devices, various policies and roles are needed for assigning permissions, which again raises the same issue with policy explosion and role explosion for billion of devices, especially in the futuristic smart connected environment. For instance, in a smart manufacturing plant, there can be various departments, units, and employees, and several IoT devices (e.g., sensors, actuators). A complex access control scenario is that within a specific department, we want to restrict access of employees working in unit 1 on the smart devices that they own within unit 1. Furthermore, we need to check the shift hours for each employee before allowing the access to factory devices and resources. Here, we need to adopt a more flexible access control approach, such as attribute-based access control (ABAC) [12], [13], which allows to define attributes for entities and environmental conditions (e.g., time, location, etc.) and specify access control policies based on those attributes.

Another scenario where we need an ABAC approach is when we want to control access of specific type of devices and their communication with other devices during office hours. For enabling this, we need to define specific device attributes, data communication channel attributes, and contextual attributes, which would then be used in attribute-based access control policies.

In this paper, we develop a formal ABAC model for AWS IoT by extending AWS-IoTAC model, (discussed in detail in Section 2). The ABAC AWS-IoTAC model allows to define attributes of different entities (e.g., IoT devices, virtual objects or digital twins, topics) and use these attributes and their values while specifying authorization policies which determine an access decision, either allow or deny, based on these policies. The new ABAC model for AWS IoT incorporates its existing access control capabilities and further enable fine-grained and flexible access control on IoT devices, things, data and resources, and services. Ultimately, lessons learned from developing an ABAC model based on AWS IoT will be valuable for similar development in other platforms and further benefit studies on a platform-independent model also.

To demonstrate the capabilities of our ABAC model, we present a secure future industries use case, more specifically a smart oil refinery factory, and its proof-of-concept implementation using our proposed ABAC AWS IoT model. Industries of the future including advance manufacturing and smart factories will be shaping the economy of a nation. Major funding agencies as National Science Foundation (NSF) and Department of Energy (DoE) have announced more than $1 billion in awards for establishing research and development centers to advance industries of the future.[3] In our smart factory use case, we define different types of entities such as users, groups, IoT devices, things and their virtual objects [14] (which are digital representations of the physical devices, also known as device shadows in AWS, and/or digital twins in other platforms), thing groups, and topics/channels used for communication in publish/subscribe communication paradigm, and their specific attributes. We utilize the AWS IoT, AWS Greengrass, and AWS Lambda service to demonstrate the proof-of-concept implementation and specify attribute-based access control policies that allows specific operations on protected entities from authorized actors based on attributes and their values. A detailed discussion on this is presented in implementation section later followed by the performance evaluation of our model that depicts its feasibility in a large real-world CE-IoT platform.

A summary of our contributions is as follows:

- We develop a formal ABAC model for AWS IoT to enable flexible and fine-grained access control for billions of smart devices, users, resources, and applications in a real-world IoT platform that can enable secure smart industries of the future.

[1] https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/

[2] https://aws.amazon.com/greengrass/

[3] https://www.arcweb.com/blog/us-nsf-doe-invest-1-billion-5g-ai-quantum-computing-advance-industries-future

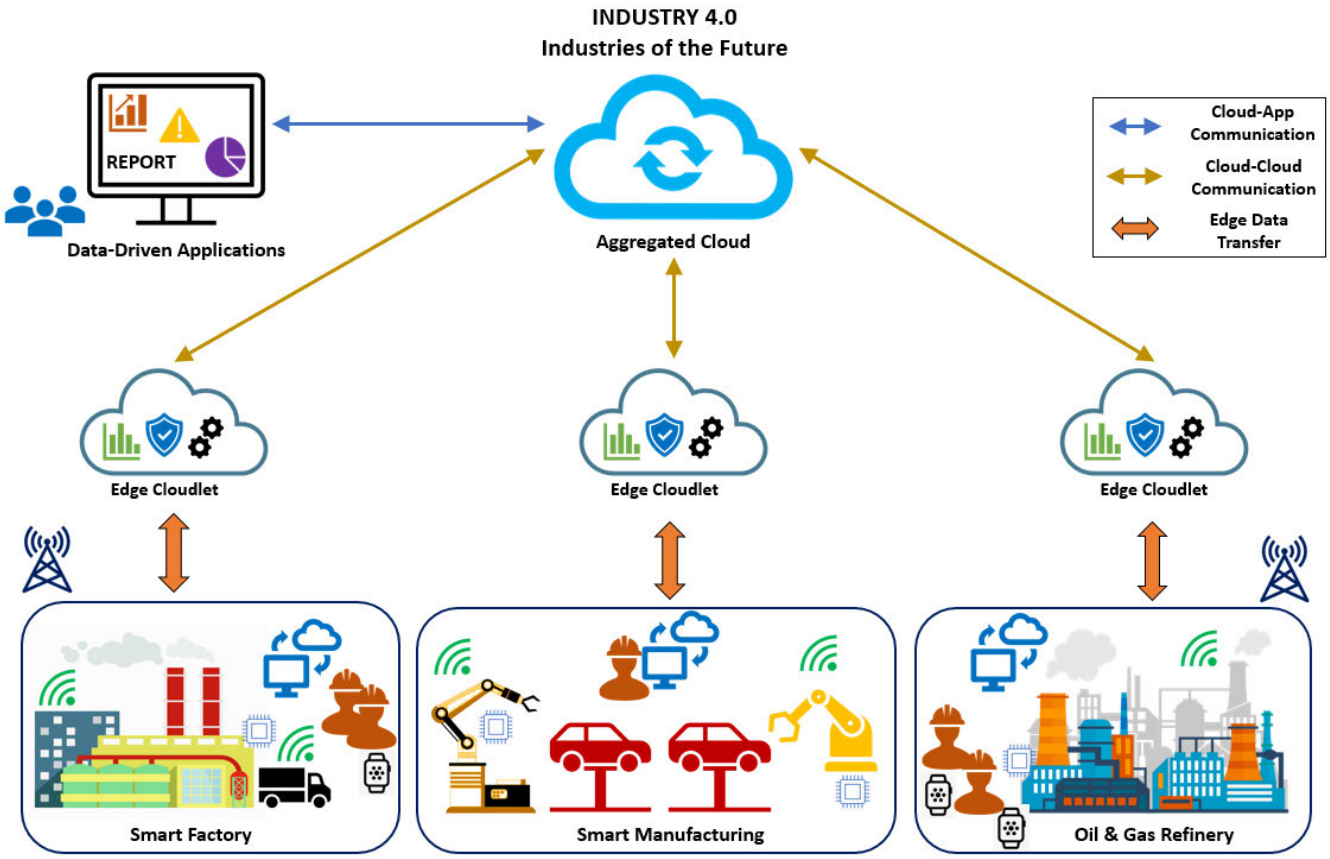


FIGURE 1. Industries of the future with different types of communications.

- We present an Industrial IoT (IIoT) use case scenario which is a smart oil factory with specific entities, groups, and attributes.
- We implement a proof-of-concept of our ABAC model in AWS IoT along with other AWS services. We envision that the proposed ABAC model would act as a foundation to secure industries of the future.
- In order to analyze the performance and feasibility of our ABAC AWS-IoTAC model, we conduct experimental analysis to analyze the performance, usability, and feasibility of our proposed model in a real-world CE-IoT platform - AWS cloud and IoT.

The rest of paper is organized as follows. In Section 2, we first discuss the industries of the future, then provide background on the AWS-IoTAC model followed by a comparison or our model with existing models and AWS-IoTAC model. In Section 3, we present our proposed ABAC model for AWS IoT and define the formal definitions. Section 4 discusses smart industries use case scenario in detail and Section 5 presents the implementation details of the use case utilizing our ABAC model in AWS IoT platform. It also presents the performance analysis and results of our model and then provides a discussion on capabilities and limitations of the model. Finally, Section 5 concludes the paper.

II. BACKGROUND AND RELATED WORK

In this section, we present a brief background on industries of the future, and the AWS IoT Access Control (AWS-IoTAC) model along with its formal definitions. We also provide a comparison of our model with other relevant access control models.

A. INDUSTRIES OF THE FUTURE

Industries of the Future (IotF) is an emerging concept built upon the foundational technology and capabilities of IoT and Cyber Physical Systems (CPS). Recently, IotF is receiving significant attention and investments from major funding organizations, such as National Science Foundation (NSF), and Department of Energy (DoE). A collection of technological domains including artificial intelligence (AI), advanced manufacturing, quantum information science, 5G/advanced wireless technology, and biotechnology together form the industries of the future (IotF). It is also referred as Industry 4.0 (more recently as Industry 5.0) which is enabled by the convergence of various technology domains including IoT, CPS, Cloud and edge computing and intelligent systems utilizing AI. IotF will play a major role in strengthening national infrastructure and driving national economy in coming years.

Figure 1 shows some of specific smart industries – Smart Factory, Smart Manufacturing, and Oil & Gas Refinery.

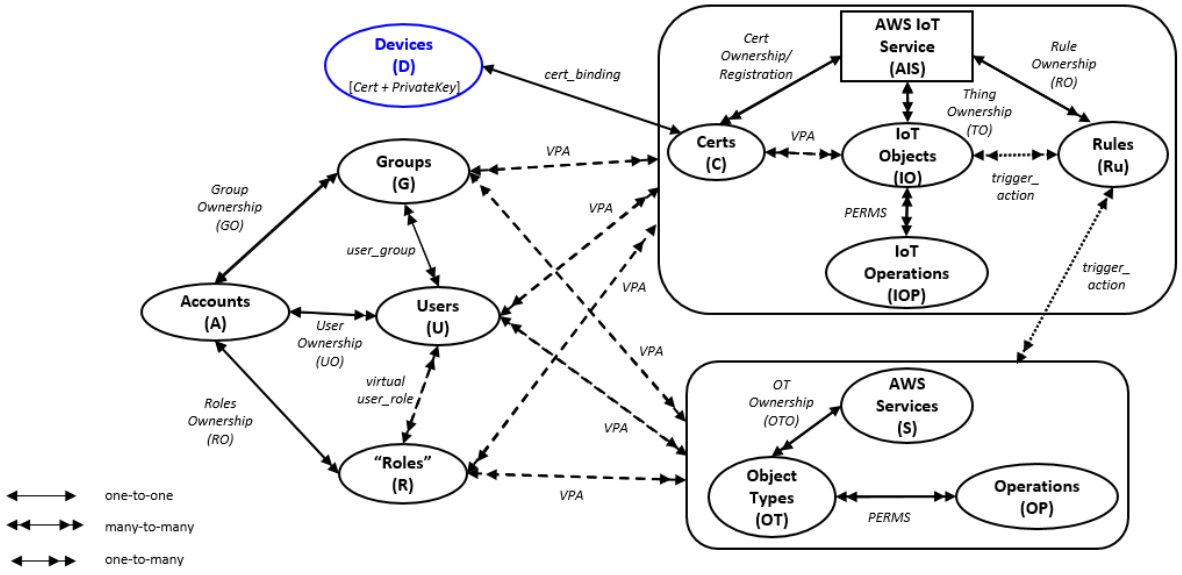


FIGURE 2. AWS-IoT access control (AWS-IoTAC) model within a single project [11].

These smart industries are also referred as Industrial Internet of Things (IIoT). These industries encompass various components including smart sensors, actuators, machinery, industry employees, services at edge and cloud and data-driven applications. With various components involved, there are different types of communications and data transfer that can happen between these components (as shown in Figure 1) which enable to accomplish specific tasks autonomously with minimal human intervention. The core objective of IIoT is to enhance operational efficiency, resource utilization and management, and cost optimization in these industrial domains. However, this new technology advancement also introduces new security and privacy risks in addition to existing threats. A recent ransomware attack[4] on the Colonial Pipeline disrupted oil company's operations and crippled fuel delivery on the Atlantic coast in US. The company paid 75 Bitcoin (i.e., 5 million US dollars) as ransom to the Darkside, the hacker group responsible for the cyber attack, for recovering their data and resources and resume normal operation. This cyber attack is a prime example of the threat we are facing and we can only imagine how it will expand with smart industries in the future. Thus, we need to develop secure access control and data communication models for enhanced security and privacy of future smart industries.

In this paper, we present a smart oil and gas refinery use case to demonstrate the applicability of our attribute-based access control model for AWS IoT. The use case involves various IoT sensors and actuators, wearable devices, and services and applications to increase operational efficiency, to perform real-time monitoring of workers and machinery, to achieve effective and efficient management of resources, and to enable autonomous operations through connected machines within a factory. We implement this use case in a real-world cloud-enabled IoT platform, AWS IoT, and demonstrate how attribute-based access control model and policies are employed to secure access and communications between sensors, actuators, and wearable devices of the employees. A detailed discussion is presented in Section IV and V.

[4]https://www.nytimes.com/2021/05/14/business/darkside-pipeline-hack.html

B. AWS IoT ACCESS CONTROL (AWS-IoTAC) MODEL

In this subsection, we briefly describe the Amazon Web Services (AWS) IoT Access Control (AWS-IoTAC) model [11] and its formal definitions. The AWS-IoTAC model is build upon AWS cloud access control model, known as AWSAC [10] and has incorporated basic cloud access control components and complemented by the novel IoT access control components including entities and access control operations. Figure 2 shows the developed AWS IoT access control (AWS-IoTAC) model along with the formal definitions in Table 1 focusing on the AWS-IoTAC components. The model has the following components: *Accounts (A), Users (U), Groups (G), Roles (R), Services (S), Object Types (OT), Operations (OP), AWS IoT Service (AIS), Certificates (C), Devices (D), IoT Objects (IO), IoT Operations (IOP),* and *Rules* along with specific relationships as described below.

In AWS cloud, **Accounts** are basic resource containers that enables customers to use cloud resources and manage their resource utilization and billing. The **Users** are individuals authenticated and authorized to access resources through their accounts. A user is the owner of an account who can create multiple users within that account and can assign them specific permissions on cloud resources. **Groups** are a set

TABLE 1. AWS-IoTAC formal model definitions.

Basic Sets, Functions and Relations
– A, U, G, R, S, OT and OP are finite sets of accounts, users, groups, roles, services, object types, and operations respectively.
– AWS IoT Service (AIS) is one of the Services(S) in AWS.
– C, D, IO, IOP, and Ru are finite sets of X.509 certificates, physical IoT devices, IoT objects, IoT operations, and rules defined in the rules engine of AIS respectively.
– User Ownership (UO): $U \rightarrow A$, is a function mapping a user to its owning account, equivalently a many-to-one relation $UO \subseteq U \times A$
– Group Ownership (GO): $G \rightarrow A$, is a function mapping a group to its owning account, equivalently a many-to-one relation $GO \subseteq G \times A$
– Role Ownership (RO): $R \rightarrow A$, is a function mapping a role to its owning account, equivalently a many-to-one relation $RO \subseteq R \times A$
– Object Type Ownership (OTO): $OT \rightarrow S$, is a function mapping an object type to its owning service, equivalently a many-to-one relation $OTO \subseteq OT \times S$
– Cert Ownership/Registration (CO): $C \rightarrow AIS$, is a function mapping a certificate to its owning service (AIS), equivalently a many-to-one relation $CO \subseteq C \times AIS$
– Rules Ownership (RO): $Ru \rightarrow AIS$, is a function mapping a rule to its owning service (AIS), equivalently a many-to-one relation $RO \subseteq Ru \times AIS$.
– Thing Ownership (TO): $IO \rightarrow AIS$, is a function mapping the IoT objects to its owning service (AIS), equivalently a many-to-one relation $TO \subseteq IO \times AIS$.
– $PERMS = OT \times OP \cup IOP$, is the set of permissions (including cloud and IoT operations and associated permissions).
– Virtual Permission Assignment (VPA): $VPA \subseteq (U \cup G \cup R \cup C) \times PERMS$, is a many-to-many virtual relation resulting from policies attached to users, groups, roles, certificates, and resources.
– user_group $\subseteq U \times G$ is a many-to-many mapping between users and groups where users and groups are owned by the same account.
– virtual user_role (VUR): $VUR \subseteq U \times R$ is a virtual relation resulting from policies attached to various entities (users, roles, groups), where users use AssumeRole action to acquire/activate a role authorized in VUR.
– cert_binding $\subseteq C \times D$ is a mutable one-to-one relation between X.509 certificate and IoT devices within a single account.
– trigger_action $\subseteq Ru \times (IO \times S)$ represents a many-to-many mapping between rules and IoT objects and AWS services on which a rule triggers action(s).

of user groups and **user_group** relation specifies the user to group assignment. AWS supports policy-based access control model. There is also a concept of roles though it is different from the known notion of roles in Role-Based Access Control (RBAC) [8]. In AWS, **Roles** are used to enable secure access between multiple accounts (cross-accounts). The *AssumeRole* action allows to specify permissions for specific roles which provides access to corresponding cloud resources across multiple accounts. The user-role mapping is specified through **virtual user_role** relation. To distinguish the AWS ''Roles'' from RBAC roles, quotation marks are used in Figure 2. For simplicity, we understand roles to signify ''Roles'' unless otherwise stated. **Services** refer to AWS cloud services. **Object Types** represent a specific type of an object in a cloud service, such as virtual machines in EC2 service. **Operations** represent allowed operations on the object types based on the access control policies that specify a set of permissions on object types belonging to a particular service.

The access control in AWS takes a policy-based access control approach. In AWS, a **policy** is specified in a JSON file which includes permissions defined on services and resources in the cloud. It comprises of three main parts (or tags) *Effect, Action* and *Resources*, and optional *Conditions*. A policy can be attached to a user, group, role or to specific cloud resource. **Virtual Permission Assignment** is the process of virtually assigning permissions to users, roles, and groups through attaching policies to these entities. In cases where a policy is attached to a resource, a specific *Principal* (an account, a user or a role) needs to be specified in the policy. There could be multiple permissions defined in one policy, and multiple policies could be attached to one or more entities to assign appropriate permissions on them.

The **AWS IoT Service (AIS)** is the novel service added to the AWSAC model. It incorporates all the IoT specific entities and their access and authorizations aspects, thus, it is represented as a separate entity in the model. **Certs (C)** is a set of X.509 certificates, issued by a trusted entity known as the certificate authority (CA). These can be generated by AIS or other trusted CAs for the IoT clients. In AIS, the Certs are used by MQTT based clients (IoT devices or user applications) to authenticate to AIS and also assigning permissions for different entities by attaching access control policies to Certs. MQTT,[5] an OASIS standard, is a machine-to-machine (M2M) lightweight publish/subscribe messaging protocol, especially designed for constrained devices. **Devices (D)** represent a set of connected IoT devices, such as sensors or light bulbs. These devices can exist independent of AIS, thus, we show them in a different color (blue circle) in the model. A valid X.509 certificate and its private key need to be copied onto the device, along with a root AWS CA certificate before authentication and establishment of a secure communication channel with the AWS IoT service. The certificates to devices association is done through the **cert_binding** relation. In the AWS IoT platform, one certificate can be attached to many things/devices. Similarly, many certificates can be copied

[5]https://mqtt.org/

onto one IoT device. However, in our model, we assume *cert_binding* is an one-to-one association between devices and certificates for better authorization management, and is mutable in nature so can be changed by an administrator in cases of certificate expiry or revocation. In AWS IoT, the access control policies are attached to certificates, and are enforced on physical IoT devices associated with these certificates.

IoT Objects (IO) represent virtual IoT objects created in the AWS cloud. *Virtual objects* are the digital counterparts of real physical devices, or standalone logical entities (applications) in the virtual space [14]. In AWS IoT, a *Thing* and a *Thing Shadow* represents the IoT objects which are the virtual counterparts of real physical IoT devices associated with the cloud. For each IoT device, we assume that there is at least one *thing* with its *thing shadow* instantiated in the cloud, which provides a set of predefined MQTT topics/channels (associated with this device) to allow interaction with other IoT devices and applications, even when the device is offline. *Thing shadow* maintains the identity and last known state of the associated IoT device. **IoT Operations (IOP)** are a set of operational actions defined for IoT service, which exclude the administrative operations such as create things, certificates, attach certificates or policies etc. The basic set of IoT operations can be categorized based on the communication protocols used by IoT devices and applications to communicate with the AWS IoT service. For MQTT clients, four basic IoT operations are available: *iot:Publish* allows devices to publish a message to a MQTT topic, *iot:Subscribe* allows a device to subscribe to a desired MQTT topic, *iot:Connect* allows a MQTT client to connect to the AWS IoT service, and *iot:Receive* allows devices to receive messages from subscribed topics. Similarly, for HTTP clients, *iot:GetThingShadow* allows to get the current state of a thing shadow, *iot:UpdateThingShadow* allows to send messages to update/change the state of a thing shadow, and *iot:DeleteThingShadow* deletes a thing shadow. Whenever a device or application sends message to a virtual thing in the cloud, a new thing shadow is automatically created, if one does not already exist.

Rules (Ru) are simple SQL statements which trigger predefined actions based on the conditions defined in the rule. A rule receives data from a device/thing and triggers one or more actions. The actions route the data from one IoT device to other IoT devices, or to other AWS services. Each rule must be associated with an IAM (Identity and Access Management) role which grants it permissions to access IoT objects and AWS services on which actions are triggered. The relation **trigger_action** represents a many-to-many mapping between rules and IoT objects and AWS services on which the rule triggers action(s). The access control policies in AWS have been modified to include IoT operations and resources, and are thereby named as IoT policies. AWS IoT utilizes both IoT policies and IAM policies to assign specific permissions to IoT devices, IAM users, and IoT applications. Consequently, **Virtual Permission Assignment (VPA)** has been updated to include the IoT policies, and these policies are attached to X.509 certificates. The policy attached to a certificate is enforced on the device which uses that certificate to connect and authenticate to the AWS IoT service. One policy can be attached to multiple certificates, or multiple policies can be attached to one certificate.

All the components and relations of AWS-IoTAC model is defined within the scope of a single AWS account. The AWS IoT service has been evolved over past years and a new formal definition along with formal ABAC model is needed in AWS IoT to offer fine grained and flexible policy specification. In this paper, we propose extension of AWS-IoT service with ABAC considering the new capabilities of the AWS IoT service and incorporate new components and capabilities in our ABAC model for AWS IoT, as described in Section III.

C. COMPARISON WITH EXISTING ACCESS CONTROL MODELS

Here, we briefly discuss related work on both general IoT access control models and IoT domain specific access control models that have been developed. Numerous access control models have been proposed to address security and privacy issues in IoT and CPS systems. A comprehensive survey of IoT access control models is presented by Ouaddah *et al.* [15]. These models are developed based on state-of-art access control models, such as RBAC, ABAC, and Capability-Based Access Control (CapBAC) [16], [17]. In [18], authors proposed an identity authentication and capability-based access control (IACAC) model where devices use an access point and the CAC model to connect with each other. The CapBAC model is used to control access on services and information. With the help of use cases, authors also present that CapBAC supports rights delegation, least privileges access principle, more fine-grained access control, reduced security issues, and fewer issues related to identities of the entities compared to ACLs, RBAC and ABAC. More recently, a blockchain-enabled decentralized capability-based access control, also known as BlendCAC, has been proposed for enhancing the security of large scale IoT systems [19]. However, there are specific limitations of CapBAC, such as propagation and revocation [20]. Besides, researchers have developed platform independent IoT access control models based on RBAC, such as [21]–[23], and ABAC, such as [24]–[26]. Several access control Lists (ACLs), RBAC and ABAC models for virtual objects communication are proposed in [27]. In [28], authors proposed a hybrid access control model for IoT based on RBAC and ABAC. It utilizes user attributes to assign roles to specific users. However, the scope of this paper is focused on developing an attribute-based access control model for a real-world CE-IoT platform, AWS IoT, which can allow researchers and developers to implement and adopt an ABAC approach in other CE-IoT platforms with similar capabilities.

Moreover, there are specific access control models developed for real-world cloud-enabled IoT platforms, i.e, AWS,

Google, Azure, and IoT domains, such as smart homes, smart farming, smart health and remote patient monitoring, and smart transportation. In [11], Bhatt *et al.* developed a formal access control model for AWS IoT, known as AWS-IoTAC. This model was developed based on AWS Cloud Access Control (AWSAC) model [10]. Gupta et al developed access control models for Google Cloud Platform (GCP) and GCP IoT, known as GCPAC and GCP-IoTAC model respective [29]. Another access control model is developed for Azure IoT cloud which is known as Priority-Attribute-Based RBAC Model for Azure IoT Cloud (PARBAC) [30]. While the above mentioned access control models have been developed for cloud-enabled IoT platforms, these models are based on RBAC and PBAC approaches, and even PARBAC also utilizes a special type of attribute. Thus, these models are not based on ABAC model. In this paper, we mainly focus on ABAC approach for developing a fine-grained and flexible access control model for AWS IoT.

Similarly, several access control models are developed for specific IoT application domains. In [31], authors presented the current state of access control for smart home devices, mainly focusing on the access control aspects in three devices of smart lighting system, bathroom scale, and door lock. They found that these heterogeneous smart home devices do not have a unified access control approach but rather have their own mechanisms to enable access. This depicts the complexity of developing an access control model for specific IoT application domains. In [32], authors proposed a privacy-aware smart health access control system (PASH) which is based on ciphertext-policy attribute-based encryption (CP-ABE). Their approach is stated to incorporate a large universe for CP-ABE and also hides sensitive attribute values in the access control policies, which could reveal sensitive health related information.

Recently, Ameer *et al.* [33] developed an extended generalized role based access control (EGRBAC) model for smart home IoT. However, to enable more fine-grained access control while capturing complex relationships between users and devices, they developed smart home IoT attribute-based access control (HABAC) [34] particularly for smart homes. Besides, a role-based administrative model is proposed for EGRBAC in [35] by Shakarami and Sandhu. In the context of smart transportation, several access control models have been proposed by Gupta et al for securing access between different entities, such as sensors, actuators, virtual objects, edge cloudlets and cloud services and applications [36]–[39]. An attribute-based access control model is proposed for smart farming in [40]. Another example in healthcare domain is [41] which proposes a lightweight and fine-grained access control system for smart healthcare utilizing attribute-based encryption and leveraging cloud and edge enabled architecture. More recently, in [42], authors proposed a convergent access control framework to enable synergistic convergence between access control models, such as RBAC, ABAC, relationship-based access control (ReBAC) [43], [44], and other access control models, such as Usage Control (UCON) [45], as per the access control requirements of specific domains. Authorizations in several cloud-enabled IoT domains along with current trends and several use cases are presented in [46]. Moreover, an attribute-based communication control (ABCC) is developed to secure data communication and data flow between various components in cloud-enabled IoT architectures [47].

We discussed several access control models from different aspects here including general IoT access control models, cloud and edge enabled IoT access control models, and convergent models. However, in this paper, our goal is to develop an attribute-based access control (ABAC) model for AWS IoT platform which provides a cloud and edge based architecture as well as enable real-world implementations of various IoT application domains. Therefore, it is critical to understand the formal access control aspects in such real-world cloud and IoT platforms and propose a comprehensive ABAC model for it. The ABAC model for AWS IoT is significantly different than the prior AWS IoT access control (AWS-IoTAC) model. AWS-IoTAC is a policy-based model, whereas the new model is an ABAC model. In addition, the AWS IoT platform has evolved over time and has introduced new IoT entities and capabilities. We have captures these new IoT entities in our proposed model and also proposed different types of attributes for these entities. To implement and incorporate these attributes and ABAC authorization policies, we utilized several AWS cloud services, primarily AWS Lambda and AWS Greengrass, with detailed discussion presented in the Implementation and Results section later.

III. EXTENDING AWS-IoTAC WITH ABAC

In 2017, Bhatt et al proposed a formal access control model for AWS IoT known as AWS-IoTAC [11]. The AWS-IoTAC model was developed based on its policy-based access control capabilities and its documentation, as available in 2017. The authors also proposed a set on ABAC enhancements for supporting fine-grained access control in contrast to its policy-based approach. However, with the proliferation of IoT and new use cases, the access control capabilities of AWS-IoT have changed significantly over last few years. Hence, a revised formal model is needed. Here, we develop and enforce an Attribute-Based Access Control (ABAC) model for AWS IoT by extending the AWS-IoTAC model with new AWS IoT capabilities. This novel ABAC model support the specification of fine grained security policies using attributes of various entities, such as IoT devices, virtual objects, and topics. In addition to the new access control components, our proposed ABAC model uses components of AWS-IoTAC and the current cloud Identity and Access Management (IAM) entities.

Figure 3 shows our ABAC enhanced AWS-IoT access control model along with different components which are formally defined in Table 2. In this model, we primarily focus on the IoT service and its operational access control and authorization aspects. The operational access control

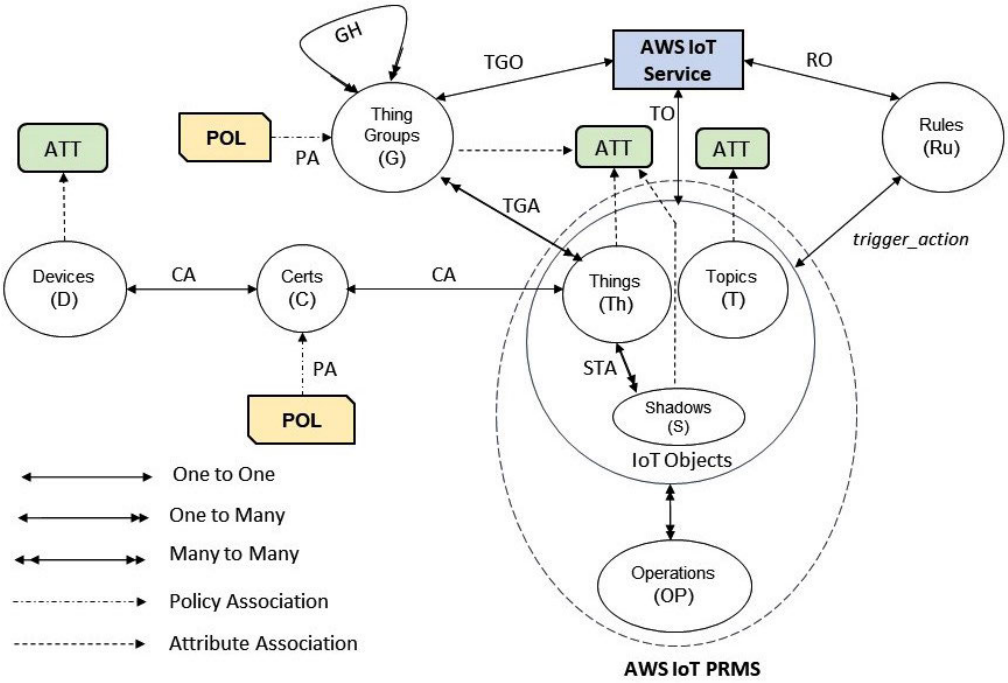


FIGURE 3. Conceptual attribute based access control for AWS-IoT service.

in AWS IoT encompasses IoT operations, such as *connect, publish, subscribe*, etc. The administrative access control of AWS IoT is outside the scope of this paper. For instance, 'create or delete operations on particular entities' (e.g., IoT things, users, groups) are examples of admin operations. We define the IoT operations in detail later while describing the formal definitions. While some of the components overlap with the AWS-IoTAC model, we still describe them briefly here in the context of the proposed ABAC AWS-IoTAC model for reader coherence and clarity. In addition, some of the components have been changed as AWS IoT service is continuously evolving to provide new capabilities and applications for users. As shown in the Table 2, there are eight core entities of the model - *Devices (D), Certificates (C), Things (Th), Shadows (Sh), Topics (T), Rules (Ru), Groups (G), and Operations (OP)*.

Devices (D) represent physical smart devices connected to the internet which collect/generate data and send or receive data to or from other devices, edge gateways, applications, and cloud services. These devices exists in the physical environment. However, digital counterparts of these physical devices are created in AWS cloud IoT service, known as **Things (Th)**. These *things* are also referred as virtual objects and are defined as the digital representation of physical devices (e.g., sensors, actuators, etc.) or logical entities, such as instance of an application or a logical entity that is associated with a physical entity. For example, there could be multiple sensors and control units within a car (referred to as clustered objects [36], [48]), thus, a thing can represent a sensor within a connected car which may itself not be connected to the AWS IoT service. Hence, there could be multiple things associated with a connected car or vehicle. The **parentD** function maps a *thing* to a physical device and multiple *things* can be associated with one physical device. IoT *Things* enable devices to send and receive message from cloud, other smart devices, and applications. These *things* are created and managed within a registry provided by the IoT service. Each *thing* can be associated with a specific *Thing Type* which must be defined in IoT and is immutable, (i.e., it cannot be changed once created). *Thing type* is a newly added component and are optional, however, it provides a mechanism to represent some common information or characteristics of things through a set of three possible attributes. When a *thing* is associated with a specific *thing type*, it gets attributes which can then be assigned values for each attribute independently. An advantage of *thing type* is that it allows the associated *things* to have up to fifty attributes, otherwise, a *thing* can only have three attributes in AWS IoT.

Groups (G), another new component, allows to group things together for ease of management of hundreds or probably thousands of objects in the ecosystem. There are two types of groups – **static groups** and **dynamic groups.** Static group is a set of things grouped together on some feature whose members remain fixed. Dynamic groups on the other hand are a set of things that match a certain query as specified by user or administrator. The **directG** relation maps a thing to a group. There also exists hierarchical [49] relationships between thing groups, known as **Group Hierarchy (GH)**. GH is a partial order relationship on G, which implies that thing groups can have parent and child groups and allows to

TABLE 2. Formal ABAC model definitions for AWS-IoT service.

Basic Sets and Functions

– D, C, Th, Sh, T, Ru, G, and OP are a set of devices, certificates, IoT things, shadows, topics, rules, thing groups, and operations respectively.

– ATT is the finite set of attributes associated with D, Th, Sh, T, and G.

– For each attribute att in ATT, $\mathrm{Range(att)}$is a finite set of atomic values.

– attType: ATT = {set, atomic}, defines attributes to be set or atomic valued.

– Each attribute att in ATT maps entities in D, Th, Sh, T, and G to attribute values. Formally,

$$\mathrm{att} : \mathrm{D} \cup \mathrm{Th} \cup \mathrm{Sh} \cup \mathrm{T} \cup \mathrm{G} \rightarrow \begin{cases} \mathrm{Range(att)} \cup \{\perp\} & \text{if attType(att) = atomic} \\ 2^{\mathrm{Range(att)}} & \text{if attType(att) = set} \end{cases}$$

– IoT-PRMS = (Th ∪ Sh ∪ T) × OP , is a set of permissions on IoT things, shadows, and topics

– POL is a finite set of authorization policies that can be attached to certificates (C) and thing groups (G), such that $\mathrm{POL} \rightarrow 2^{\mathrm{IoT-PRMS}}$

– directPOL: $\mathrm{C} \cup \mathrm{G} \rightarrow 2^{\mathrm{POL}}$, assigns policies to certificates and thing group. Equivalently, $\mathrm{PA} \subseteq \mathrm{POL} \times (\mathrm{G} \cup \mathrm{C})$

– certAssign: $\mathrm{D} \rightarrow \mathrm{C}$, assigns physical device (and implicitly to its corresponding thing) to certificate. Equivalently, $\mathrm{CA} \subseteq \mathrm{C} \times \mathrm{D}$.

– parentD: $\mathrm{Th} \rightarrow \mathrm{D}$, maps each thing to a physical device where multiple things can be associated with a device.

– directG : $\mathrm{Th} \rightarrow \mathrm{G}$, maps each IoT thing to a group, equivalently $\mathrm{TGA} \subseteq \mathrm{Th} \times \mathrm{G}$

– parentTh: $\mathrm{Sh} \rightarrow \mathrm{Th}$, maps each Shadow to a Thing where multiple shadows can be associated with a Thing, equivalently $\mathrm{STA} \subseteq \mathrm{Sh} \times \mathrm{Th}$

– Group Hierarchy is a partial order relation $\succeq_g$ on G, defined by $\mathrm{GH} \subseteq \mathrm{G} \times \mathrm{G}$
This is equivalent to a group mapped to set of parent groups, stated as parentG : $\mathrm{G} \rightarrow 2^{\mathrm{G}}$, mapping group to a set of parent groups in hierarchy.

Effective Attributes of Thing Groups, Things, and Shadows (Derived Functions)

– For each attribute att in ATT such that attType(att) = set:

- $\mathrm{effG_{att}} : \mathrm{G} \rightarrow 2^{\mathrm{Range(att)}}$, defined as $\mathrm{effG_{att}(g_i)} = \mathrm{att(g_i)} \cup \left(\bigcup_{\mathrm{g} \in \{\mathrm{g_j} \mid \mathrm{g_i} \succeq_g \mathrm{g_j}\}} \mathrm{effG_{att}(g)} \right)$
- $\mathrm{effTh_{att}} : \mathrm{Th} \rightarrow 2^{\mathrm{Range(att)}}$, defined as $\mathrm{effTh_{att}(th)} = \mathrm{att(th)} \cup \mathrm{att(parentD(th))} \cup \mathrm{effG_{att}(directG(th))}$
- $\mathrm{effSh_{att}} : \mathrm{Sh} \rightarrow 2^{\mathrm{Range(att)}}$, defined as $\mathrm{effSh_{att}(sh)} = \mathrm{att(sh)} \cup \mathrm{effTh_{att}(parentTh(sh))}$

– For each attribute att in ATT such that attType(att) = atomic:

- $\mathrm{effG_{att}} : \mathrm{G} \rightarrow \mathrm{Range(att)} \cup \{\perp\}$, defined as $\mathrm{effG_{att}(g_i)} = \begin{cases} \mathrm{att(g_i)} & \text{if } \forall \mathrm{g'} \in \mathrm{parentG(g_i)}.\ \mathrm{effG_{att}(g')} = \perp \\ \mathrm{effG_{att}(g')} & \text{if } \exists\ \mathrm{parentG(g_i)}.\ \mathrm{effG_{att}(parentG(g_i))} \neq \perp \text{ then select} \\ & \text{parent g' with } \mathrm{effG_{att}(g')} \neq \perp \text{ updated most recently.} \end{cases}$
- $\mathrm{effTh_{att}} : \mathrm{Th} \rightarrow \mathrm{Range(att)} \cup \{\perp\}$, defined as $\mathrm{effTh_{att}(th)} = \begin{cases} \mathrm{att(parentD(th))} & \text{if } \mathrm{effG_{att}(directG(th))} = \perp \\ \mathrm{effG_{att}(directG(th))} & \text{otherwise} \end{cases}$
- $\mathrm{effSh_{att}} : \mathrm{Sh} \rightarrow \mathrm{Range(att)} \cup \{\perp\}$, defined as $\mathrm{effSh_{att}(sh)} = \begin{cases} \mathrm{att(sh)} & \text{if } \mathrm{effTh_{att}(parentTh(sh))} = \perp \\ \mathrm{effTh_{att}(parentTh(sh))} & \text{otherwise} \end{cases}$

Effective Policies of Thing Groups and Things (Derived Functions)

– $\mathrm{effG_{pol}(g : G)} \rightarrow 2^{\mathrm{POL}}$, mapping a group g onto a set of policies in the presence of a thing group hierarchy.
Formally, $\mathrm{effG_{pol}(g : G)}$ = {p ∈ POL | (∃ g′ $\succeq_g$ g) [(p , g′) ∈ PA]}

– $\mathrm{effTh_{pol}(th : Th)} \rightarrow 2^{\mathrm{POL}}$, mapping a thing th onto a set of policies in the presence of a thing group hierarchy.
Formally, $\mathrm{effTh_{pol}(th : Th)}$ = directPOL(certAssign(parentD(th))) ∪ $\mathrm{effG_{pol}(directG(th))}$

Authorization Functions (Policies)

– Authorization Function: For each op ∈ OP, $\mathrm{Auth_{op}(s : S, tr : TR)}$ where S = D ∪ Th ∪ Sh and TR = D ∪ Th ∪ Sh ∪ T, is a propositional logic formula returning true or false, which is defined using the following policy language:

- $\alpha := \alpha \wedge \alpha \mid \alpha \vee \alpha \mid (\alpha) \mid \neg\alpha \mid \exists\ \mathrm{x} \in \mathrm{set}.\alpha \mid \forall\ \mathrm{x} \in \mathrm{set}.\alpha \mid \mathrm{set} \triangle \mathrm{set} \mid \ \mathrm{atomic} \in \mathrm{set} \mid \mathrm{atomic} \notin \mathrm{set}$
- $\triangle := \subset \ \mid \ \subseteq \ \mid \ \not\subseteq \ \mid \ \cap \mid \cup$
- $\mathrm{set} := \mathrm{effG_{att}(g : G)} \mid \mathrm{effTh_{att}(th : Th)} \mid \mathrm{effSh_{att}(sh : Sh)} \mid \mathrm{att(i)}$ for att ∈ ATT, i ∈ D ∪ Th ∪ Sh ∪ T, attType(att) = set
- $\mathrm{atomic} := \mathrm{effG_{att}(g : G)} \mid \mathrm{effTh_{att}(th : Th)} \mid \mathrm{effSh_{att}(sh : Sh)} \mid \mathrm{att(i)} \mid \mathrm{value}$ for att ∈ ATT, i ∈ D ∪ Th ∪ Sh ∪ T, attType(att) = atomic

Authorization Communication Property

– A source s ∈ S is allowed to perform an operation op ∈ OP on a target tr ∈ TR, where S = D ∪ Th ∪ Sh ∪ T and TR = D ∪ Th ∪ Sh ∪ T, if all required policies stated in $\mathrm{Auth_{op}(s : S, tr : TR)}$ are satisfied. Formally, $\mathrm{Auth_{op}(s : S, tr : TR)}$ = True.

- For a set of primitive operations supported in AWS IoT service, the authorization decisions are defined as follows.
  – For connect ∈ OP, the authorization is given by $\mathrm{Auth_{connect}(s : D, tr : Th)}$, where s ∈ D and tr ∈ Th.
  – For shadow delete operation op ∈ OP, the authorization is given by $\mathrm{Auth_{op}(s : D, tr : TR)}$, where s ∈ D and TR = Th ∪ Sh.
  – For shadow update and get operations op ∈ OP, the authorization is given by $\mathrm{Auth_{op}(s : D, tr : TR)}$, where s ∈ D and TR = Th ∪ Sh.
  – For publish, subscribe, receive operations op ∈ OP, the authorization is given by $\mathrm{Auth_{op}(s : D, tr : TR)}$, where s ∈ D and TR = Sh ∪ T.
- To communicate messages among two devices, all the required set of authorization policies must be satisfied.
  Formally, for a device ($\mathrm{D_1}$) to communicate with device ($\mathrm{D_2}$):
  $\mathrm{Auth_{publish}(D_1 : D, tr : Sh \cup T)} \wedge \mathrm{Auth_{subscribe}(D_2 : D, tr : Sh \cup T)} \wedge \mathrm{Auth_{receive}(s : Sh \cup T, D_2 : D)}$ = True.

develop a thing group hierarchy. An access control policy can be assigned to a static group, as defined by the **PA** relation. In a static group hierarchy, a policy attached to the parent group is inherited by subsequent child groups and member things within those groups. Thing groups are useful when permissions need to assigned to a large number of things and

rather can be assigned to a group containing those things. A thing inherits permissions from its direct parent group and from all other groups in the group hierarchy of its parent group through policy inheritance, specified by $\text{effTh}_{\text{att}}$. However, there are limitations on groups in AWS IoT service, and some are presented here: i) a thing can belong to a maximum of 10 thing groups, ii) within a group hierarchy, a thing can belong to only one direct parent thing group at any level and the maximum depth of a thing group hierarchy is seven, and iii) maximum number of dynamic groups is 100. Dynamic groups are different than static groups. In dynamic groups, there are no hierarchical relationship and group membership cannot be defined explicitly and instead depends on a search query defined by a user or an administrator. In addition, we cannot assign access control policies to dynamic groups which are important in access control. Hence, for our formal model purpose, the thing groups G are static groups and GH is a group hierarchy between static groups.

Shadow (Sh) is a JSON object that stores the current (reported) state and new (desired) state. In the AWS-IoTAC model, there was only one shadow associated with an IoT thing. In the proposed ABAC extended model, there can be multiple shadows associated with one thing. For instance, an autonomous vehicle could have multiple sensors and things, which can be represented as multiple shadows associated with a vehicle thing. The **parentTh** function maps a shadow to a thing where multiple shadows can be associated with a thing. Shadows in AWS can be updated using updateShadow operation, retrieved using getShadow operation, and deleted using deleteShadow operation, supported by the IoT service. Besides, there is a *shadow document* that stores all the information about state changes on a particular device as per the reported and desired state. AWS cloud have introduced an edge computing service known as AWS Greengrass,[6] which enables edge communication and local computation. The shadows can also reside at the edge gateway or cloudlet [38] as a virtual representation of the device at the edge of the network. **Certificates (C)** are X.509 certificates created by a trusted certificate authority and are assigned to Things in cloud and also copied onto the corresponding physical devices. It enables mutual authentication between IoT devices and AWS IoT service. To the authorization front, access control policies are attached to these certificates which in-turn assigns available permissions for the device and its equivalent thing residing in cloud or at edge in a gateway or cloudlet. The **CA** relation in Figure 3 assigns a certificate to a physical device and its corresponding thing.

AWS IoT supports two communication protocols – MQTT protocol based on publish-subscribe paradigm, and HTTPS for web interfaces. In a publish-subscribe paradigm, topics/channels are used for sending/publishing and receiving/subscribing messages among devices and applications. In our ABAC model, we have proposed **Topics (T)** as an entity which is a set of topics and act as a conduit for sending and receiving IoT messages within different objects. These topics are channels and follow a tree like structure separated by slashes, for example, *home/living_room/bulb* is a topic where a device can send and receive messages related to a bulb in living room of a smart home. These topics are user defined and are created instantaneously which provides great flexibility and creating multiple topics for a large number of IoT devices. In the MQTT, there is a broker to manage these topics and a list of subscribers for each topic. It forwards any new message publish to a specific topic to all the subscribers of that topic. For instance, if Device1 publishes to *home/living_room/bulb*, then any other device, say Device2, that has subscribed to that topic will receive that message. **Rules (Ru)** are the same as discussed in the AWS-IoTAC model, which are SQL statements and can be triggered based on IoT message content (e.g., temperature > 100) or specific condition being met as per the rule. A rule can perform a list of actions through the **trigger_action** relation.

Operations (OP) are IoT operations that can be performed in AWS IoT. Previously, in AWS-IoTAC model, IoT Operations (IOP) represented the operational IoT operations in the AWS IoT service. A set of operations were defined for two communication protocols: MQTT and HTTP, and clients using these protocols, as discussed in Section 2. The MQTT operations are – *iot:Connect, iot:Publish, iot:Subscribe* and *iot:Receive*. To enable publish and subscribe operations, connect operations must be allowed for clients (device, application, etc.), and receive operation is checked every time a message is being delivered to a client. Therefore, if a client has subscribed to a topic then it must also have permissions for receive operation. Besides MQTT operations, there are shadow operations – *iot:DeleteThingShadow, iot:GetThingShadow,* and *iot:UpdateThingShadow*, which we consider administrative operations. In AWS IoT, there are also job execution actions, though these types of actions are outside the scope of our model.

ATT is a set of attributes representing the properties of specific entities including devices, things, shadows, topics, and groups. An Attribute is function that takes the entity as input and assigns value(s) which can be atomic or set valued, within the range (Range(att)) of that attribute. For example, *age* is a user attribute function whose domain range is defined as real numbers or integer values starting from 0 to 150. So, for a specific user Alice, $age(\text{Alice}) = 35$, which means a user Alice is 35 years old. Similarly, specific attributes and their values can be defined and identified for other entities. The set valued attributes can have a set of values, and atomic attributes have one value from the range of the attribute. These attributes with a given range can be defined by the administrator for specific entities as required by the application scenarios.

In the AWS IoT ABAC model, devices, things, shadows, topics, and thing groups can have attributes which represent specific properties of entities and are defined accordingly. For example, device attributes can be *owner* and *location* which

[6]https://aws.amazon.com/greengrass/

represents the owner of the device and location of the device respectively. Similarly, things and thing groups also have a set of attributes associated with them. If there is thing th which belongs to a thing group, then the thing will have its own attributes as well as attributes inherited from its parent group. Moreover, there could be group hierarchy of thing groups. This means, effective attributes of a thing, defined by $\text{effTh}_{\text{att}}(\text{th})$ are the union of its directly assigned attributes and all the groups' attributes inherited through the group hierarchy. Since a physical device have a virtual thing associated with it, where physical and virtual thing essentially have same properties, the device and its thing attributes may be the same. However, the device attributes are a subset of thing attributes since things have their own attributes, attributes inherited from their direct parent group and attributes inherited from the group hierarchy, if any. Thus, a virtual thing may have more attributes than its corresponding physical device through thing groups and group hierarchy.

Within the context of thing and shadow attributes, we can define multiple shadows for a single thing and these shadows can have a subset of thing attributes assigned to them based on the nature of a shadow of the thing. For instance, a thing can have five attributes defined for it and has two associated shadows. Now, each shadow can have all the five attributes of the thing, or both shadows may have a subset (some or all of the thing attributes) of thing which may or may not be common attributes among the shadows. Hence, the shadow attributes att are a subset of thing attributes and effective attributes of a shadow sh would be its own attributes union all effective attribute values of associated thing (thing attributes and its inherited attributes from parent group as well as its GH), defined by $\text{effSh}_{\text{att}}(\text{sh})$ in Table 2.

Topics also have a set of attributes that represent their characteristics. The topic attributes are essential in enabling fine-grained access control that allows to control publish-subscribe communications based on topic attributes. For instance, there could be some sensitive topics to which only authorized devices, things, and/or clients (e.g., applications) can send and receive messages. In this case, we can use topic attributes along with entities attributes in authorization policies. A simple ABAC policy example in a smart factory would be *A worker in section* 1 *can send messages to devices that belongs to the same section using those topics that are active in section 1.* In this case, the worker, devices, and topic attribute values need to match for sending and receiving messages between a worker and devices using a specific topic. If an attacker (who is not in section 1) tries to publish a message on a topic of section 1, then the operation will be denied and an alert notification can be sent to an administrator. How do we define and assign these attributes to specific entities? It actually depends on use case or application scenario, since smart home would have different attributes than a smart factory or industry.

Recently, AWS IoT introduced attributes that can be defined for things or thing groups, however, there are limited number of attributes that can be defined. Moreover, there is limited capability of how these attributes can be used while defining authorization policies. Due to these limitations, AWS IoT does not support *complete* ABAC capabilities (at the time of writing of this manuscript) as proposed by this paper. As per the AWS IoT documentation,[7] the attributes are available as a policy variable (i.e., it can be used in the policy) only while connecting over MQTT or MQTT over the WebSocket protocol. AWS IoT also introduced tags for managing resources. These tags are similar to attributes with key-value pair, however, tags are not used for defining authorization. Hence, tags are not considered in our model. IoT permissions (IoT-PRMS) are a set of permissions on IoT things, shadows, and topics that specifies what operations (MQTT and Shadow operations) are allowed for IoT objects. IoT objects comprise things, shadows, and topics as shown in Figure 3. These permissions are specified in a **Policy (POL)** which can be attached to certificates and thing groups. Through certs and thing groups, authorization policies are applied on things and physical devices. There are two types of policies assigned to certificates and thing groups. First, directly assigned policies, captured by **directPOL** relation that explicitly assigns policies directly to certificates and thing groups. Second, there are implicit effective policies (specified by $\text{effG}_{\text{pol}}(\text{g} : \text{G})$ and $\text{effTh}_{\text{pol}}(\text{th} : \text{Th})$) derived based on hierarchical groups.

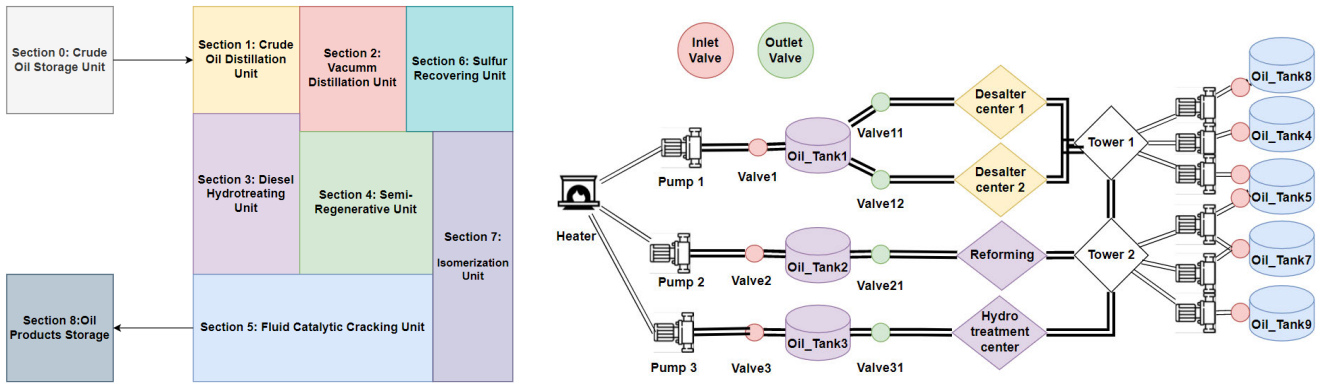
The **Authorization Function** for a specific operation op is represented as $\text{Auth}_{\text{op}}(\text{s} : \text{S}, \text{tr} : \text{TR})$ for a given source s and a target tr. The source and target can be a device, thing, shadow or a topic. The propositional logic based policy language for specifying ABAC policies is also defined in Table 2. Based on different types of IoT entities (source and target entities) in the IoT space, different authorization communication properties for various operation as supported by AWS IoT are defined in Table 2. The authorization function and policies depend on specific application scenario, such as smart home, smart industry, smart manufacturing, etc. and are defined accordingly.

In the following section, to demonstrate our proposed model and ABAC authorization policies, we present a Smart Industries use case scenario and its implementation in AWS IoT platform utilizing the AWS IoT edge computing service, AWS Greengrass to provide local connection and communication at the edge of the network.

IV. A FUTURE SMART INDUSTRIES USE CASE

In this section, we present a smart industrial IoT use case, specifically a connected oil refinery including various scenarios within this use case that align with the industries of the future. We first describe the use case and associated scenarios and then demonstrate a proof-of-concept implementation that shows how our proposed ABAC AWS IoT access control model is used to enable secure fine-grained access control in the context of a real-world smart future industry.

[7]https://docs.aws.amazon.com/iot/latest/developerguide/thing-policy-variables.html



(a) Oil Refinery Sections

(b) Oil Transferring Flow Process

FIGURE 4. An overview of a smart oil refinery.

A. OVERVIEW

In the oil refinery industry, the facilities are operated 24 hours a day, 365 days a year. The size of an oil refinery can be equivalent to a hundred football fields combined, employing hundreds of employees in the refinery [50]. While there are thousands of pieces of equipment in an oil refinery, in our use-case we focus on a subset of these equipment such as tanks, valves, pipes, and pumps. We present a smart oil refinery which comprises various users–*employee, manager, worker*, and smart devices–*smart watch, smart valves, pumps and oil tank*. There are several scenarios within this use case which we will discuss later in the implementation section.

Since a significant amount of work is performed by employees or workers in such a complex ecosystem, human errors are inevitable and may result in catastrophic results. It is estimated that 70% of the accidents in the oil and gas industry occur due to human error. Some of the major factors that cause such accidents are miscommunication, physical exhaustion, inadequate supervision, and misinformation resulting in false alarms or notifications [51]. Therefore, it is crucial to integrate smart devices, such as sensors on valves and oil tanks, and smart pumps which can be turned on or off remotely. This enables users and machines to perform actions automatically and efficiently [52]. Currently, IoT devices and applications are critical to future manufacturing and oil-gas industries. Using smart devices, industry employees can remotely control and communicate with these devices in an effective way as well as monitor employees' health who work in hazardous conditions. Under normal operations, smart machines can automatically perform desired tasks and reduce employee's repetitive works while maintaining employee safety and continuous industry operations. For example, messages from various devices are routed and filtered to specific users (managers, workers, etc.), so that they are not overwhelmed with a huge amount of information and a large number of false alarms generated in the system [53].

One of the critical aspects of the oil refinery industry is the safety of employees. It is essential to monitor employees' health in real-time and provide assistance when needed due to use of dangerous chemicals and other hazardous situations. Employees can wear smart wearable IoT devices that can continuously track their individual health parameters and automatically send notifications to respective parties and request for help in case of emergency. For example, if an employee's heart rate is very high and blood oxygen is declining then a notification will be sent to connected ambulance through employee's smart watch or a connected edge cloudlet (which is deployed at the edge network closer to the employee and wearable devices (e.g., smart watch) are connected to these cloudlets.) However, integrating smart IoT devices can expose security vulnerabilities which can be exploited by attackers [54]. For instance, if an unauthorized user discovers a vulnerability to access a device and manipulate the way it behaves in an oil refinery, serious consequences like loss of life, or even explosions may occur. Therefore, we utilize our proposed ABAC model to secure smart oil refinery use case scenarios and depict its applicability in a real-world smart industry through a proof-of-concept implementation in the AWS IoT platform.

Figure 4a (or left side) represents simple working process of an oil refinery. At first, crude oil is imported to the factory as shown in `Section 0`, then it is distilled into many different kinds of oil, which later flows into different treatment processes (represented as various sections `Sections 1 to 7` in the Figure 4a). Finally, the processed oil is transported to the storage unit `Section 8` for sales. Figure 4b (or right side) shows the process of refining the oil including various devices, pumps, valves, and oil tanks. Within this use case, we consider scenarios from two main perspectives: first smart machines and devices in the refinery, and second wearable IoT devices associated with the refinery employees, which can be further categorized as manager, maintenance, and production worker. As per the first perspective, the use case scenarios discuss how smart

devices and associated sensors control the oil flow based on valve pressure and vibration rates and detect if there is any leakage in oil tank based on oil level in the tank and oil flow measured in ''Gallon Per minute'' (GPM). Based on the data collected from smart devices and specified conditions, pump can be turned on or off as needed in the scenario. For example, in Figure 4b, first the oil is heated to a desired temperature through the furnace, and once it has reached the temperature, the *Pump* on each pipeline will be turned on to increase pressure which will transport the oil to its destination tanks through specific valves. The corresponding *Inlet Valve* or *Outlet Valve* must be opened to let the oil flow in or out of the respective oil tanks. However, we need to define attributes for different types of entities (refinery devices and user devices) and specify attribute-based access control policies for authorizing operations on inlet and outlet valves to securely enable the flow of oil in and out of the oil tanks as well as operations on other devices by users (through their individual devices, e.g., smart watch) and other refinery devices.

Another perspective is based on user-wearable devices, where a smart watch can receive notifications and send messages to perform specific tasks and/or get desired device information in the refinery. The conditions to perform specific actions on a device along with particular authorizations required to perform such actions are defined in the attribute-based policy (shown later in Figure 8 in next section).

B. USE CASE AND ABAC AWS IoT MODEL MAPPING

Here, we map the smart oil refinery use case entities and scenarios as per the ABAC AWS-IoTAC model entities, attributes, and operations for our proof-of-concept implementation. In the smart oil refinery, there are employees who perform operations, manage and maintain devices/machines for accomplishing various tasks as shown in Figure 4. Employees are categorized into three main types of users:

- **Manager**: These are assigned to specific sections in the refinery and manage each section that they are assigned to and supervise staff, operate required plant processes and equipment.
- **Maintenance**: The maintenance employees diagnose, identify, and repair devices when they behave abnormally or if there is an issue.
- **Production Worker**: The production workers ensure that the oil quality meets the standards and processes operate properly. They communicate with the Manager and Maintenance to provide updates and resolve any issues.

Employees (Manager, Maintenance, and Production Worker) have smart watches, which are user devices (D) (IoT Devices in ABAC AWS IoT model). An attribute (ATT) for smart watch could be *UserType* whose values can be *Manger, Maintenance, or Production Worker*. Besides, there would be other attributes which can be defined as needed based on the use case scenario.

In this use case, we consider that users can send and receive messages for performing operations on devices and get information or notification through their smart watches. Moreover, they can send messages and trigger actions on other smart devices. Therefore, an employee can perform an action or operation on a specific device or receiving a notification from other devices through his/her smart watch. An employee can also manually interact with the connected machines or devices, if needed. Here, we mainly focus on device to device (user devices and refinery devices) interactions/communications scenarios.

Next, we discuss different types of IoT Devices (D) in the use case. For each physical device, there is a virtual counterpart in the cyber space. In the ABAC AWS-IoTAC model, these virtual counterparts are specific things (Th) and shadows (Sh) associated with the physical IoT devices. These things can further be grouped into thing groups (G). The following IoT devices are used in this smart oil refinery.

- **Watch**: A wearable smart watch is provided to every employee by the smart oil refinery. It monitors the health parameters of the employee (i.e., employee's heart rate, movement, and body temperature) and their surroundings to ensure his/her safety in the refinery. When there is any abnormal health data or conditions affecting the employee, the smart watch will send notifications to the nearest medical center and their managers to inform about the situation. These smart watches are also capable of sending messages to refinery machines for performing specific actions (OPs), such as activate the machine/device (e.g., turn on the valve), or get device data (e.g., get the oil level in tank, etc.). Similarly, the smart watch can receive messages and notifications from connected machines whenever there is a scenario that needs employee attention. For example, if the device in a specific section of the refinery malfunctions or enters into an unsafe state, then a worker in that section will receive a notification about it and can send a message to turn it off.
- **Oil Tank**: In an oil refinery, the amount of oil in a **Oil Tank** (connected device (D)) always needs to be maintained at a specific level. The oil tank should never be drained out entirely since it could cause heating and equipment issues. On the other hand, excessive oil storage could result in an overflow of oil which may trigger critical conditions, such as a fire in the refinery. Therefore, it is necessary to maintain the oil level and safe working conditions in the refinery using the **Inlet Valve** and **Outlet Valve** for the Oil Tank. Whenever the Oil Level is low, the Inlet Valve needs to be open to let the oil flow into the tank from another source, and when the Oil Level is high, the Outlet Valve needs to be open so that the oil flows into a different tank from another destination. Based on different Oil Levels, different types of employees are notified with different messages. For example, when the Oil Tank has a low or high oil level, it only needs to inform the Production

Worker working in that section to check and fix the situation.

In addition, to check any leakage, the difference between outflow and inflow in the tank needs to be measured and is given as ''Gallon Per Minute'', or, ''GPM'' in the use care. If there is any discrepancy between the inflow and the outflow while comparing the oil level, then it means that there is leakage in the oil tank in that specific section. Depending on the severity of the leakage, notifications can be sent to maintenance workers to repair the leak, and also inform other workers for their safety. The leakage can also trigger automated response from the Inlet and Outlet valves (e.g., open or close the valve) to minimize loss and possible damage.

- **Valves and Pump**: To ensure an efficient and safe process of transporting oil, the vibration and pressure levels along the pipeline need to be maintained at specific levels, as shown in Figure 4b. To measure the vibration and pressure levels, various sensors and actuators are placed on the valves along the pipeline in each section. The two main causes of damages along the pipeline are low pressure and/or a high vibration rate. For instance, if the sensors at the Inlet Valve or Outlet Valve detect low pressure, the **Pump** needs to be turned on. If there is a high vibration rate detected at the Valve, the Pump will need to be turned off. The process to turn the Pump On or Off could be autonomous based on the sensor values.

It is critical to secure devices and ensure only authorized entities can send and receive messages to and from devices while also securing the data associated with the smart oil refinery. These devices are simulated and connected to the AWS IoT and has Things and Shadows associated with them. The above entities have specific attributes (ATT) defined for them. Using these attributes and their values, we define the ABAC policies (POL) for specifying the permissions for each entity. Furthermore, these devices can be grouped into different Thing Groups (G), which may have sub-groups within the Thing Groups, thus creating a group hierarchy. The Thing Groups can also have attributes which are inherited by sub-groups and things associated with that group. These devices communicate using the MQTT protocol as supported by the AWS IoT service. The MQTT protocol utilizes a publish-subscribe paradigm and have a set of Topics (T) created for communication purposes. Therefore, in our use case, we create generic topics where the devices can send (or publish) messages and receive messages (by subscribing to those specific topics). We also utilize the reserved AWS topics in AWS IoT platform. We discuss these details in our proof-of-concept implementation in the next section.

V. IMPLEMENTATION AND RESULTS

This section presents the implementation details of our industrial IoT use case utilizing our proposed AWS-IoT ABAC model on the AWS-IoT Platform. To implement this smart refinery use case, we utilize current capabilities of the AWS IoT service, AWS Greengrass service for enabling edge computing capabilities, and also build a novel ABAC policy decision point (PDP) and policy enforcement point (PEP) using the AWS Lambda service for realizing the capabilities of our ABAC AWS-IoTAC model in AWS IoT.

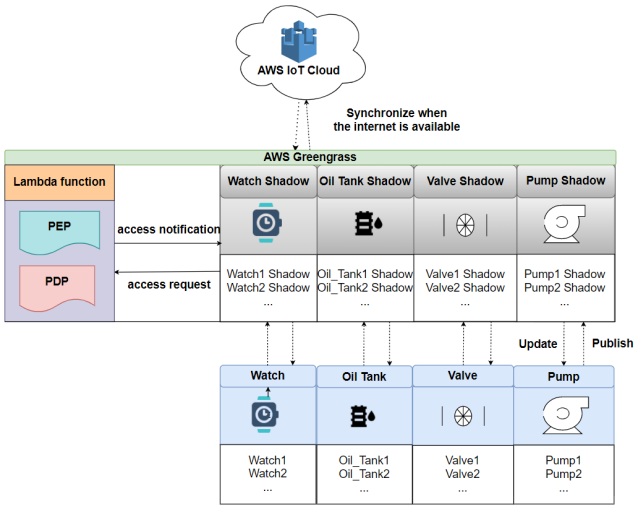


FIGURE 5. System architecture.

A. SYSTEM ARCHITECTURE

Figure 5 represents system architecture along with the components of our proof-of-concept implementation. We simulate the oil refinery physical devices: *Oil Tanks, Valves, Watches, and Pumps*, using the AWS Device SDKs[8] which were deployed on virtual machines (VMs). For each physical device, we create virtual Things in AWS IoT and these things have shadows associated with them. There can be more than one shadow for each thing in AWS IoT. However, for this use case, we consider one shadow for each thing. For each physical device, their corresponding virtual IoT thing, and their shadows, we define attributes based on the characteristics of these entities. These attributes and their values are utilized in the ABAC authorization policies. Further, we group things into groups and sub-groups with specific attributes for each group. This forms a group hierarchy and allows attribute inheritance. These are administrative tasks and will be discussed later in detail in the Administrative Phase.

We utilize the AWS Greengrass to enable edge computing and communication between physical devices. The Greengrass service was used to implement our model, as detailed in the performance evaluation subsection later. First, physical devices need to be registered with the AWS cloud. Once the registration is complete, these devices connect to the AWS Greengrass which holds a copy of physical devices' shadows. These are edge virtual objects. All the data from the devices is stored at the edge on the AWS Greengrass service which allows local computation. The data is synced with the cloud periodically when the Internet connection is

[8]https://github.com/aws/aws-iot-device-sdk-python

available and/or when user/administrator syncs data manually. The AWS Greengrass also hosts the Lambda function and authorization policy which acts as a PDP and PEP. When there is any access request from a device or entity, or on the device or entity, the ABAC authorization policy needs to be evaluated as per the attributes of those entities (entity requesting access and target entity) and the decision (allow access or deny access) is made and enforced. The lambda function on the Greengrass inspects the payload messages sent from devices using the MQTT protocol, and queries the attributes of specific entities to allow or deny actions/operations on those entities. If there is no match found in the authorization policy, then the messages cannot be published and is rejected, and therefore, no actions will be performed.

The attribute-based authorization policies restrict access to services and operations in the oil refinery, such as sending and receiving messages, sending notifications to specific entities. There are single or multi-level policies needed for securing these entities and operations. For enabling fine-grained access in the smart oil refinery, we define policy rules for each type of refinery devices that allow or deny specified actions on specific devices and sending notifications to the relevant group of employees. We simulate different Oil and GPM Levels for Oil Tank, Pressure and Vibration rates for Valves, and Heart Rates and Requests for employees. Besides, we also defined the basic policy in the AWS IoT service for various IoT operations, such as connect, publish, subscribe, and receive messages for IoT devices. There are also specific subscriptions that need to be defined in the AWS Greengrass to enable communication between devices, Greengrass, and cloud. However, in this paper, we mainly focus on the attribute-based authorization policies specified based on our ABAC AWS-IoTAC model and deployed as an external service deployed in Greengrass.

B. IMPLEMENTATION PHASES

The implementation of our ABAC AWS-IoTAC model involves two phases: *administrative phase* and *operational phase*. In the administrative phase, we discuss the group hierarchy, assign things to the group, attributes/policy inheritance from parent group to child groups and to the thing in the group, storing attribute locally, and creating external policy. The operational phase covers how thing groups, thing attributes, policy, and lambda functions are used to authorize IoT operations from and on smart oil refinery devices, i.e., Oil Tank, Valve, Pump, and smart Watch.

1) ADMINISTRATIVE PHASE

Here, we discuss the administrative aspects of our implementation which simulates the smart oil-refinery. There are several admin tasks from creating things, thing groups, attributes, and policies to be able to query the attributes and store them locally for fast policy evaluation. We first create the devices (i.e., Oil Tank, Valves, Watch and Pump) using the AWS IoT SDKs and simulate the data generated from these devices.

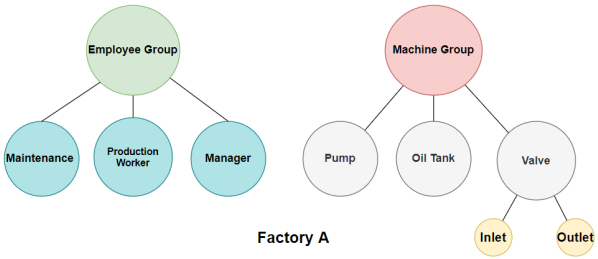


FIGURE 6. Group hierarchy in smart oil refinery.

As an administrator, we also need to create Things, Thing Groups and their attributes in AWS IoT.

Figure 6 shows different things, thing groups, and sub-groups as a group hierarchy along with their direct and inherited attributes. In this group hierarchy, the oil refinery named Factory A is divided into two main Thing Groups: **Machine** and **Employee** based on the types of IoT devices. The Machine Group is further divided into sub-groups – **Oil Tank Group, Valve Group, Pump Group**. Additionally, the sub-group can have other sub-sub groups. For instance, the Valve group has the sub-group of **Inlet** or **Outlet** Valve. The Employee Group has sub-groups **Manager, Maintenance**, and **Production Worker**. A group inherits all the attributes of its parent groups above it through the hierarchy. When a device (represented as a thing) is added to a group, the device inherits all the assigned group attributes through group hierarchy along with its own attributes. For example, a smart sensor named *Sensor1* has the following attributes.

```
'Manufacturer': 'Acme Cooperation',
'Model':'2'
```

The above JSON format implies that *Sensor1* has two attributes, `Manufacturer` and `Model`, assigned to it with values *Acme cooperation* and *2* respectively. However, in Figure 7, *Sensor1* has other attributes as well. These attributes are inherited from its parent groups as per the group hierarchy. When *Sensor1* is assigned to the Inlet Valve group as shown in Figure 7, it will inherit all the attributes of the Inlet Valve group, which has the attribute `SpecificationType`. In addition, Inlet Valve group has parent group Valve, thus it will inherit its parent group's attributes, i.e., `DeviceType`. The Valve group is assigned to the Machine Group, which has attribute `ParentType`, so Valve group will inherit the `ParentType`. The final set of attributes of *Sensor1* based on its group hierarchy includes:

```
'Manufacturer': 'Acme Cooperation',
'Model':'2', 'ParentType':'Machine',
'DeviceType':'Valve','SpecificationType':
'Inlet'
```

The same applies to the Employee group and *Watch_1*. Initially, *Watch_1* has the following attributes:

```
'Manufacturer':'Cooperation B',
'ID': '19456 ', 'DeviceType': 'Watch_1'
```

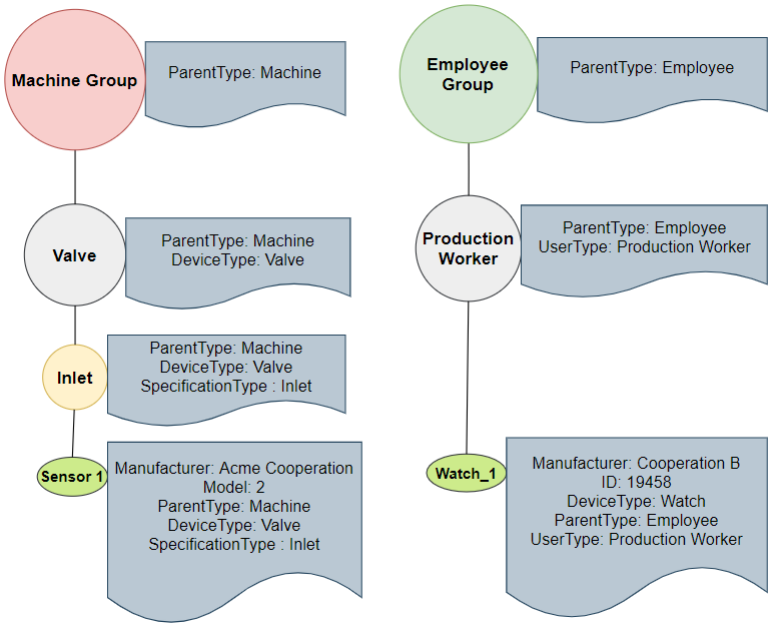


FIGURE 7. Attribute inheritance from group.

```
{
  "Machine": {
    "Oil_Tank": [
      {"GPM":"0", "Operation": "==","Action": []},
      {"GPM":"1", "Operation":"<=","Action": [{"Off":["Outlet"]}, {"Notify":{"Destination": ["Maintenance"],"MSG": "Small Leakage"}}]},
      {"GPM":"1", "Operation":">","Action": [{"Off":["Outlet","Inlet","Pump"]}, {"Notify":{"Destination": ["Employee"],"MSG": "Critical Leakage"}}]},
      {"Oil_Level":"95", "Operation":">=","Action": [{"Off":["Inlet"]},{"On": ["Outlet"]}, {"Notify":{"Destination": ["Production_Worker"],"MSG": "High Oil Level"}}]},
      {"Oil_Level":"20", "Operation":"<","Action": [{"On":["Inlet"]}, {"Notify":{"Destination": ["Production_Worker"],"MSG": "Low Oil Level"}}]},
      {"Oil_Level":"20", "Operation":">=","Action": [{"On":["Outlet"]}]}
    ],
    "Valve": [
      {"Pressure":"50", "Operation":"<=","Action": [{"On":["Pump"]}, {"Notify":{"Destination": ["Production_Worker"],"MSG": "Low Pressure"}}]},
      {"Vibration": "100","Operation": ">","Action": [{"Off":["Pump"]}, {"Notify":{"Destination": ["Manager","Maintenance"],"MSG": "Abnormal Vibration"}}]}
    ]
  },
  "Employee": {
    "Watch": [
      {"Heart_Rate":"120", "Operation":">=","UserType": ["Employee"],"Action": [{"Notify":{"Topic":["Medical"],"MSG": "High Heart Rate Employee"}}]},
      {"UserType": ["Manager"] ,"Action":[{"Read":["Machines"]},{"Publish":["Machines"]}]},
      {"UserType": ["Production_Worker","Maintenance"] ,"Action":[{"Read":["Machines"]},{"Publish":["Valve","Oil_Tank"]}]}
    ]
  }
}
```

FIGURE 8. Attribute-based custom policy.

After being assigned to the Production Worker group, *Watch_1* attributes will be updated as follows.

```
'Manufacturer':'Cooperation B',
'ID': '19456 ', 'DeviceType': 'Watch',
'ParentType': 'Employee', 'UserType':
'Production Worker'
```

Once things, groups, and attributes are defined, we specify authorization policies. For our smart oil refinery, we define the custom ABAC policy based on our ABAC AWS-IoTAC model to apply ABAC policies in AWS IoT. The custom ABAC security policy is shown in Figure 8.

Generally in AWS, authorization policies are in JSON format with a key and its value(s). We also utilize the JSON format for specifying our custom ABAC policy for the smart oil refinery. Figure 8 shows the policy file with specific keys and their values. The top-level keys are Machine and Employee which determine the type of devices (connected machines or user's wearable devices). These keys represent the Machine Group and Employee Group and allow to check whether a machine is sending its state or a smart watch of a factory worker is sending messages. The next key level represents the DeviceType attribute, which is Oil Tank, Valve, and Watch. For these keys, the values represents conditions and actions that are generated based on the attributes of these devices and their associated data. By separating the "Machine" and "Employee" as the top-level keys followed by other various sub-keys, it is easier and more efficient to parse the policy and determine the authorized action(s) to be performed. There are different conditions based on the type of device and attributes. For example, if it is determined that a device is in the Machine Group and the device is Oil

Tank (Source) which sends a message with GPM (Attribute) equal to 0, then there is no leakage and no action is needed. However, if the GPM is less than 1 but greater than 0, it indicates there is a small leakage and a notification is sent to the destination which could be the Maintenance group of employees with the message: ''Small Leakage, along with the name of the device.'' Additionally, it would turn off associated Outlet Valve for inspection. If the GPM is greater than 1 which indicates that there is a major leakage, then device notifies all the employees including Manager, Production Worker, and Maintenance. At the same time, close operation is sent to both the Inlet Valve and Outlet Valve and turn off the Pump. In the real world, this is similar to the shutdown process of a smart oil refinery to prevent an explosion. Similarly notifications and actions are specified for Oil Tank based on the Oil_Level. For the Valve, there are two conditions specified based on Pressure and Vibration levels in the pipelines where actions are to turn the Pump ''ON'' or ''OFF'' respectively, and notifications are sent to respective destinations which are specific employees.

Another group is Employee Group which has the smart watch, there are three rules–first, based on the heart rate of the employee which represents the health status of the employee. For example, when the watch detects that the employee has a heart rate higher than 120, it will send a notification message to Topic ''Medical''. Users and/or devices who are subscribed to the *Medical* topic will receive the message about the high heart rate of the employee. The second case is for watch in Employee Group and the policy rules are based on UserType attribute. The possible values for the UserType attribute are Manager, Production Worker, and Maintenance. For example, if an access request is from a watch whose UserType attribute value is *Production Worker* or *Maintenance*, then it is allowed to read all the machines states including Oil Tank, Valve, Pump, and other connected devices in the factory. However, it can only publish messages to specific devices– Valve and Oil Tank within the same section (section is an attribute for all devices which represents the section (e.g., 1, 2, 3, etc.) they belong to). The Production Worker can't publish to the Valve or Oil Tank in other sections or other devices from the same section if they are not authorized, i.e., there is no rule for it in the policy file. If the UserType is *Manager* for the watch, then it can read the status of all machines and can also publish messages to them. Similarly, administrators can specify various actions and notifications on different types of smart devices in various IoT domains based on their attributes and specific conditions.

As discussed earlier, the policy along with policy decision and enforcement points are deployed at the edge on the AWS Greengrass service. However the attributes are defined in the AWS IoT service while creating things and groups in cloud. Therefore, for local authorization policy evaluation when the Internet is not available, attributes need to be queried from AWS IoT Core and stored locally at Greengrass. This also allows fast and efficient authorization decisions. In our implementation, we cache the devices names and attributes locally from the cloud to guarantee the data is available. We store them in two JSON files as follows.

• **data.json**: This file stores the thing name and their corresponding attributes (their value(s)) in the smart oil refinery factory.

• **section.json**: This file stores a list of thing names of specific smart watch devices that are corresponding to UserType *Manager, Production Worker, Maintenance* in each section and keep a track of section attribute values for the watches.

Both of these locally cached files need to be updated periodically either manually upon request or automatically at certain intervals.

2) OPERATIONAL PHASE

In this phase, we present two scenarios within the smart oil refinery factory. First scenario depicts how devices are authorized to report their states to their shadows and actions are triggered based on those updates on other devices as per the specified policy. Second scenario shows how a user's request access using their smart watch on specific devices in the refinery are evaluated and authorized (allowed or denied) as per the ABAC policy.

Scenario 1: Devices state updates on Shadows

Figure 9 shows a notification scenario based on the update message from *Oil_Tank1*. The following message, along with the device's attributes is sent from *Oil_Tank1* to its shadow in Greengrass which then triggers the Lambda function using a defined rule.

```
{"state":{"reported": {"Oil Level":
"95.1278011", "GPM": "0", "Time":
"2020-12-19 14:11:40.930681" }}}
```

In addition, the *Oil_Tank1* has the following attributes:

```
"Factory_Location":"A", "Section" : "0",
"ParentType":"Machine",
"Manufacturer": "CompanyA"
"DeviceType":"Oil_Tank",
"Correspond_Pump": "Pump1",
"Inlet": "Valve1",
"Outlet": "Valve11, Valve12"
```

In the Policy JSON document in Figure 8, there are different conditions for *Oil_Tank1* based on Oil Level and GPM. For instance, if the Oil Level is greater than or equal to 95, then two actions will be triggered, first to close the corresponding Inlet Valve, and second, to open the Outlet Valve. It also sends a notification to the production worker with the message – ''High Oil Level''. Now identifying the specific values depends on the attributes of the tank and valves. The attributes of the *Oil_Tank1* and valves can be accessed immediately by accessing the data.json file (which stores all the attributes of things). The policy and attributes allow us to find exactly what Outlet Valve and Inlet Valves needs to be opened or closed. Based on the action policy and attributes of the *Oil_Tank1*, the Outlet Valve named *Valve11* and *Valve12* will be opened, and the Inlet Valve named *Valve1*

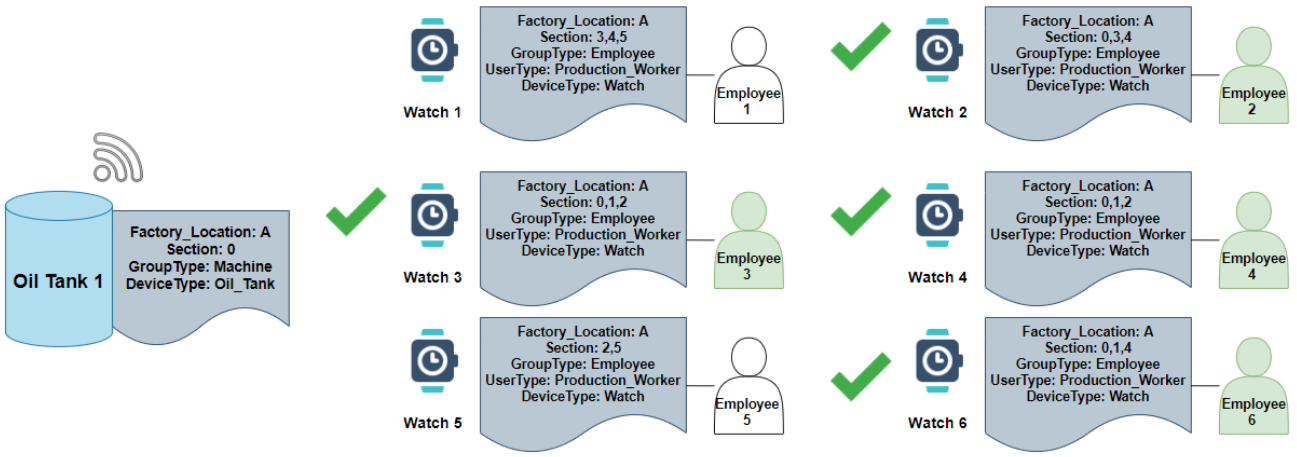


FIGURE 9. Employee notification.

will be closed. Below is the sample payload message that will be sent to *Valve11* and *Valve12* at their corresponding shadow topics `aws/thing/valve11/shadow/update` and `aws/thing/valve12/shadow/update`:

```
{"state":{"desired":{"state":"on"}}}
```

The shadow topics for each thing in AWS IoT are reserved topics. Besides the reserved topics, there are also other topics that can be created on the fly and used in different scenarios. Additionally, for the same condition, the policy also evaluates that the Production Worker needs to be notified about it. The notification with a specific payload (notification message) is sent to the smart watch of the respective employee based on the smart watch attributes to check if the employee is a Production Worker and is working in the same section or not (identified using section.json file).

Figure 9 represents an example of employee notification scenario. The Oil Tank *Oil_Tank1* has the following attributes: `Factory_Location`, `Section`, `GroupType`, `DeviceType` and Watch devices has the following attributes: `Factory_location`, `Section`, `GroupType`, `UserType`, `DeviceType`. Generally, an employee can work in one or more sections, and multiple sections are separated by a commas in Section attribute values. For example, Employee 1's Watch device is `Watch1` which has attribute `Section` with value `3,4,5`. It means that this employee works at `Section 3`, `Section 4`, `Section 5` at the same time. Assume that *Oil_Tank1* has a low Oil Level of oil storage, thus, it will notify to each employee who works and/or has responsibility in that `Section 0`. *Watch2, Watch3, Watch4,* and *Watch6* will be notified by publishing an update message with specific payload to their own corresponding reserved shadow topics which are as follows.

```
$aws/things/Watch2/shadow/update
$aws/things/Watch3/shadow/update
$aws/things/Watch4/shadow/update
$aws/things/Watch6/shadow/update
```

Another scenario is when the smart watch sends a message reporting employees' health parameters. Suppose that an employee has a heart rate of 130 and according to the policy shown in Figure 8, it will identify that the Employee has a high heart rate. However, in this critical situation, the message will be published to a general Topic = `notify/Medical` and the healthcare providers and emergency services are subscribed to this topic. Once the message is published to a topic, it is forwarded to all devices who have subscribed to that topic. There are two main benefits in this case. First, messages can still be sent to users (employees) watches if they are subscribed to `notify/Medical` topic. Furthermore, using general topics, messages can be routed beyond a smart factory and could be sent to other services, such as Simple Notification Service (SNS) in AWS cloud which would send SMS message to an on-site doctor or hospital professional for immediate assistance. However, employee's health related data is highly privacy sensitive, this service need to use secure channels, such as private topics with specific attributes. Using customized topics with specific attributes, we can secure access to these topics and their associated data from specific users, devices, or clients by specifying fine-grained ABAC policies using Topic attributes and attributes of other entities.

Scenario 2: Request from Users to access Devices

Another scenario is when users request to access a device/machine in the refinery. It is critical to check and authorize each of these requests.

Figure 10 represents an example of user authorization. Suppose that the user Anna wants to read the states of the *Oil_Tank1*. Using her smartwatch (*Watch1*), she will publish

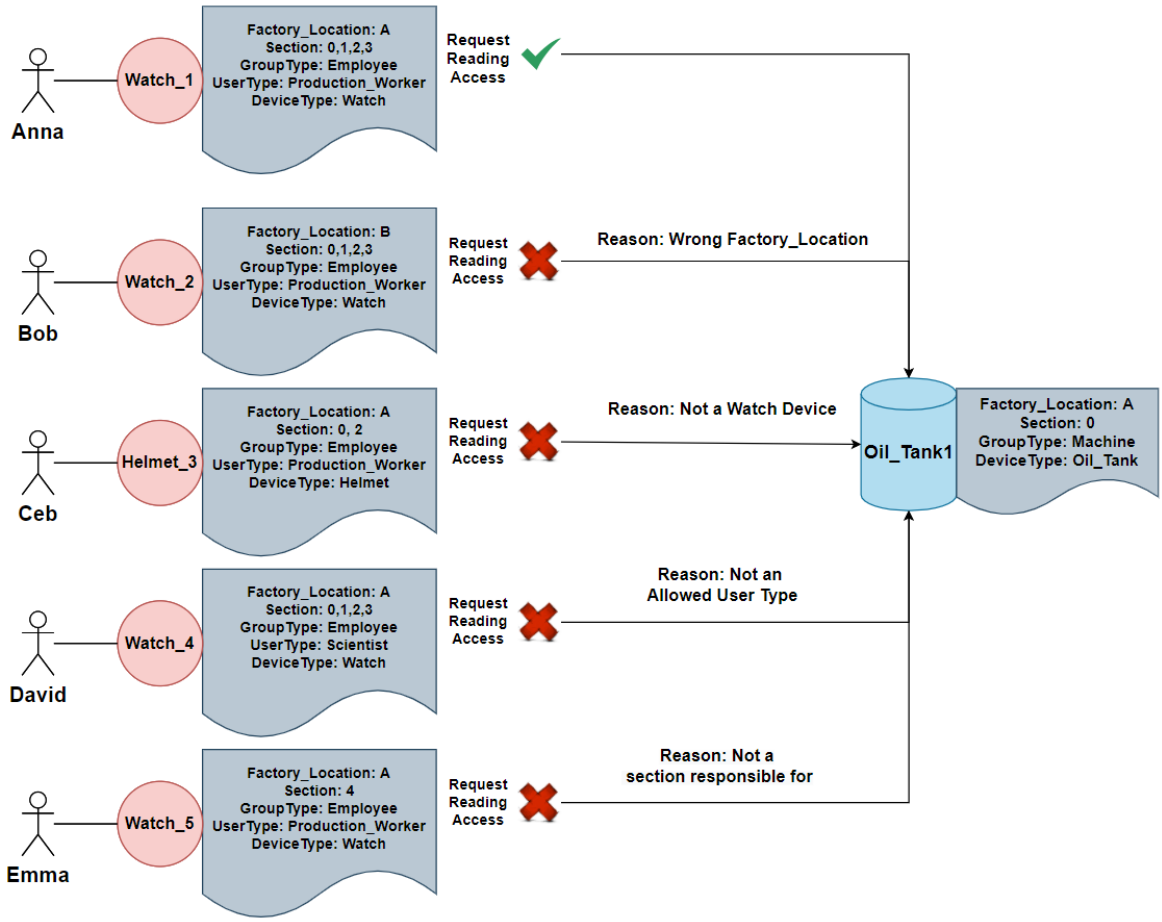


FIGURE 10. User authorization request cases.

her request as a message to the watch's shadow topic as shown below:

```
{"state":{"reported":{"Action":"Read",
"Target":"Oil_Tank1"}}}
```

This request will be evaluated by the Policy Engine and Lambda in Greengrass based on the policy and determine if the request from the user Anna should be accepted or denied. As shown in Figure 10, the request is evaluated based on the Watch's attributes and policy. As per the policy (shown in Figure 8), if the watch device has UserType as Production Worker, then it can read *Machines* (which is any kind of smart machine, e.g., Oil Tank, Pump, or Valve) and publish messages to any of the two devices: [Oil Tank, Valve]. In this case, the source is a smart watch whose UserType is *Production_Worker*, and the section of the *Oil_Tank1* is included in the section attribute value of the *Watch_1*, so Anna is authorized to perform desired action.

However, other users who have different attributes which do not satisfy the policy will not be authorized to read the device's states. For example, Bob is not authorized because the Factory_Location attribute for Bob and *Oil_Tank* are different. Another user, Ceb is not authorized to read machine states since he tried to read it through another smart wearable device, such as Helmet. The request might have been sent by a malicious user in the factory or through a compromised device. David is not authorized because his UserType is Scientist and not Production Worker, therefore rejected. Emma is not allowed to read *Oil_Tank1* since she is not working in that section, i.e., *Section 0*.

To summarize and depict the sequence of IoT operations in the above scenarios, we present a general sequence of events for any device sending its state to lambda through its device shadow in Figure 11. Initially, Device 1 publishes its sensor information to its Shadow (1). The sensor information update message triggers the Lambda function with Policy Engine (2), and once authorization decision is made, a specific message is sent to a particular topic specified in the policy (3.1). When a specific message is published to a topic (3.1), all the devices and clients (e.g., applications) that have subscribed to that topic would receive the message (3.2), which is Device 3 here. For example, when an employee in the refinery is detected to have a high heart rate, a message will be published to the topic `Medical` by the Policy Engine. The published message

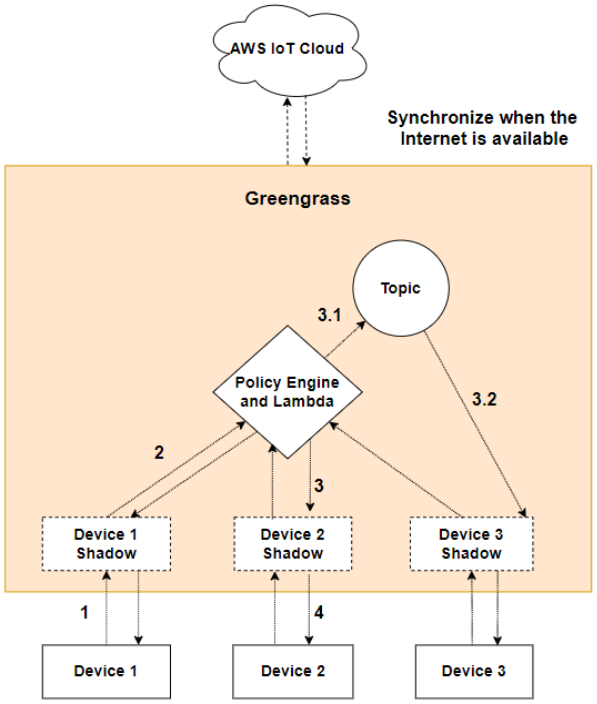


FIGURE 11. External services authorizing sequences.

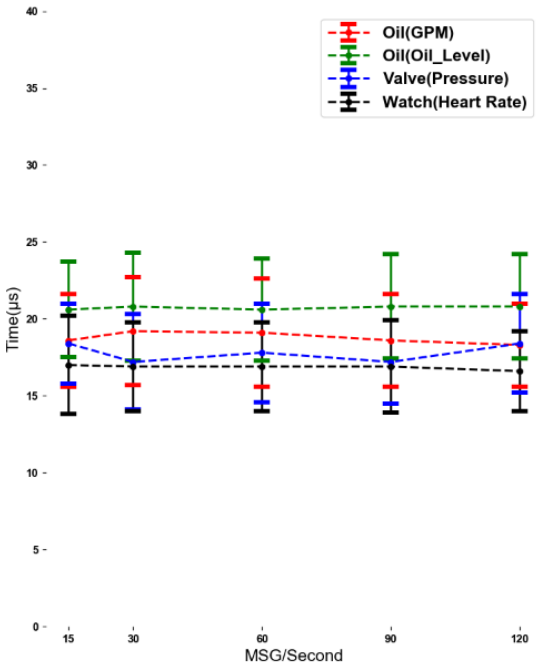


FIGURE 12. Separate payload comparison.

TABLE 3. AWS set up parameter information.

Payload size range (Byte)	94 - 162
Number of AWS Greengrass group	1
Number of devices	200
Range of message sent to Lambda (Message per second)	1 - 120
Greengrass server configuration	1 VCPUs, 2GB RAM

on this topic can trigger external applications, such as SNS service in cloud that sends messages to an onsite Medical clinic. Simultaneously, the message will be published to some smart watches that subscribe to that topic. For instance, if a message is published on `Medical` topic, then Managers and Production Worker are notified about the medical issue through their smart watches in the refinery.

C. PERFORMANCE EVALUATION

We now present the performance evaluation and results of our proof-of-concept implementation. In our proof-of-concept implementation, we created the policy decision point (PDP) and policy enforcement point (PEP) using the Lambda on AWS Greengrass. Table 3 lists the AWS system parameters to provide better understanding of the performance metrics shown here. We deployed our AWS Greengrass server on a local VM with 1 VCPU and 2GB RAM. As per the AWS Greengrass documentation,[9] a device with a minimum of 256 MB disk space, 96 MB RAM, Java 8 or greater, GNU C Library (glibc) version 2.25 or greater, and have root permissions should be able to run the AWS Greengrass server, and hence our proposed scheme. The number of devices mentioned in the table is the number of actively involved devices with the lambda function. A device sends MQTT messages to its own shadow and forwards these messages to lambda function, or the device can receive MQTT messages published from the lambda function and/or other devices. To evaluate the performance of our ABAC AWS-IoTAC model, we conducted tests and experiments on AWS Greengrass which includes the PDP and PEP. We calculated the time starting from the IoT device message arrival on Greengrass till the policy enforcer returns the action and notification to specific entities (devices, applications, or other clients). In our experiments, we had some outliers due to a cold start [55], which is a time penalty due to creating a new container and loading libraries. We discarded these outliers from our evaluation, and once a Greengrass device is at constant load, the time penalty from cold start diminishes as resources are reused and libraries stay loaded in memory. We measure the time performance by taking a sample of 5000 messages. Specifically, each time a device sends messages at a specific rate to lambda service, a total of 5000 messages will be sent. The result is shown in Figure 12, Figure 13, and Table 4. The sample size guarantees that the result is within 99% confidence interval with a margin of error of 2%, where we excluded no more than 3% of the sample due to the cold start.

In order to simulate our devices, we sent a certain amount of messages per second from one device to lambda. The parameter values in device messages are continuously varied

[9]https://docs.aws.amazon.com/greengrass/v2/developerguide/setting-up.html#greengrass-v2-requirements

TABLE 4. Policy enforcer evaluation time.

MSG/sec	GPM(Oil Tank)	Oil Level(Oil Tank)	Pressure(Valve)	Heart Rate(Watch)	Request(Watch)	GPM + Oil Level(Oil Tank)
15	18.6 ± 3.0	20.6 ± 3.1	17.1 ± 2.6	17.0 ± 3.2	21.3 ± 4.0	19.7 ± 2.6
30	19.2 ± 3.5	20.9 ± 3.5	17.7 ± 3.1	16.9 ± 2.9	21.8 ± 4.5	20.5 ± 3.4
60	19.1 ± 3.5	20.6 ± 3.3	17.8 ± 3.2	16.9 ± 2.9	22.1 ± 4.7	20.2 ± 3.3
90	18.6 ± 3.0	20.8 ± 3.4	17.2 ± 2.7	16.9 ± 3.0	22.8 ± 5.7	19.8 ± 2.7
120	18.3 ± 2.7	20.8 ± 3.4	18.4 ± 3.2	16.6 ± 2.6	23.2 ± 6.1	19.6 ± 2.5

Note: The time unit is in μs.

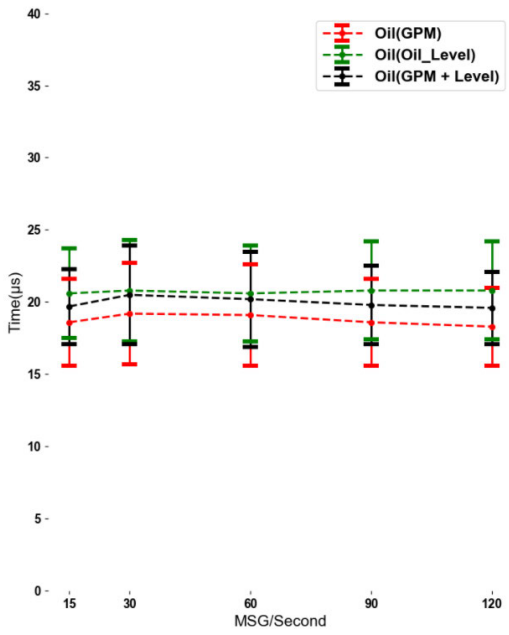


FIGURE 13. Combine payload comparison.

to guarantee that all the policies are tested and evaluated during the execution. We examined how different sizes (with one key, two keys, or more than two keys) of upload messages from devices would impact the policy evaluation time. For example, the Oil Tank will publish to its shadow the GPM's data, or the Oil Level, but not both at the same time. The results are shown in Figure 12. The time taken by policy enforcer in all cases on average is less than 25 μs. The average time for policy evaluation seems to be consistent regardless of number of messages sent per second based on our experiments. In case there are several keys and many associated conditions for those keys in the JSON policy file, the policy evaluation may take more time. For example, there are more policy decision points associated with Oil Level and GPM values of Oil Tank compared to the Watch and Valve, therefore, the average time it takes to evaluate the policy would be higher. However, there isn't a significant difference as reflected by the standard deviations shown in Table 4.

We also examined test cases with varying payload message sizes being sent to the shadows. In our use case scenarios, Oil Tank contained two parameters, GPM and Oil Level, though in a real-world scenario, there can be multiple integrated sensors which could measure multiple parameters, e.g., temperature, pressure, gas, vibration, etc. The Figure 13 shows how increased payload size affected the performance of the policy engine. In this scenario, there were 15 to 120 messages sent from a simulated device but the messages contained only two parameters, which were separately evaluated, thus resulted in two policy decisions which were independently evaluated and averaged. However, the average processing time over the number of decisions made was unchanged. Based on our experiments, there was no evidence that increasing the payload size would increase the time to evaluate and enforce each policy. For the smart watch requests, we check source and target devices' attributes to match the attributes as per the policy to make authorization decisions. Even with this case, there is no significant impact on the performance as shown in Table 4. The average time for a request to be processed is ranged from 19 μs - 23 μs with standard deviation ranged from 4 μs - 6 μs. This could be explained by the process of caching devices' attributes locally in Greengrass which enabled quick access to the source attributes and target attributes for evaluating the policy and making authorization decisions. It should be noted that message payloads from different sensors and devices are independent and do not impact one another.

D. DISCUSSION AND LIMITATIONS

The main advantage of the AWS IoT ABAC model is to enable more flexible access control based on various attributes of entities including devices, things and shadows, and device groups. With ABAC approach, we can also utilize environmental attributes (e.g., time of day, device location, user location, system risk level, etc.) in making decision for allowing or denying source actions on specific targets. However, in ABAC model implementation, a common challenge is managing entity attributes and their values. Our model is developed specifically for AWS IoT and attributes and policies can be defined within AWS cloud through its existing services, such as AWS Lambda, thus it allows to manage the attributes and policies more efficiently. We demonstrated it with our proof-of-concept model implementation in a smart oil and gas factory.

Overall, the experimental results show that our model is feasible and applicable to be implemented in real-world scenarios. There is minimal performance overhead with the

addition of custom ABAC policy engine including PDP and PEP at the edge of the network (AWS Greengrass). All the devices connect to Greengrass and policies are defined and enforced in the Greengrass enabling first point of access control in the Cloud-enabled IoT platform. Edge computing is essential for enabling industries of the future and provide real-time communication (machine-to-machine, users-to-machines) and response to the users. Here, we demonstrated two specific scenarios in smart oil refinery simulated using the AWS IoT, Greengrass, and Lambda. However, these services have their own limitations. AWS Greengrass has a limitation of 2500 devices per Greengrass group. Therefore, in a real-world implementation, this limitation may require multiple deployments and coordination between various Greengrass groups to completely encompass the large smart oil refinery. Initially Greengrass was limited to 250 devices and recently expanded to 2500, this is a limitation imposed by AWS and may expand further in the future.

Besides, the ABAC policy engine also needs to be further optimized to be scalable as per the growing needs of smart industries. We also envision that ABAC policy engine may be built as a separate service in the future that can enable ABAC policies to be used with any real-world cloud enabled IoT platform given there is a way to define and store attributes for different entities. In the future, we plan to study other cloud-IoT platforms and apply our ABAC approach to enable dynamic and more fine-grained access control for them.

It must be noted that, there are several other hardware or software specific limitations on IoT devices, such as power and energy constraints. In this paper, we mainly focused on the security aspects, and developed a fine grained AWS IoT access control model definition and implementation in a cloud enabled IoT architecture with edge computing capabilities, and no on-device policy evaluation, or computation as discussed in the performance evaluation Subsection V-C. Although, power evaluation aspects are very important and must be considered when deploying these security solutions in the real world, these are not considered and are outside the scope of this research. This is one limitation of this work, which we plan to explore in our future research that will primarily focus on IoT devices capabilities and efficiency when implementing such mechanisms in real world settings.

VI. CONCLUSION

In this paper, we developed an ABAC model for a real-world cloud-enabled AWS IoT platform. We extended the AWS-IoTAC model with ABAC capabilities including its existing capabilities to enable more fine-grained access control in AWS IoT platform. We demonstrated our ABAC AWS-IoTAC model in a future industry use case, a smart oil refinery. With rapidly widening IoT ecosystem, we discussed the vision of Industries of the future and also designed our use case to be compliant with this vision. For this purpose, we developed different scenarios and simulated relevant IoT devices in the oil refinery. We implemented a proof-of-concept implementation of our model in AWS IoT. Our performance evaluation results demonstrate the viability of our model and are promising for realizing and implementing our ABAC AWS-IoT model in real-world cloud-IoT platforms. Ultimately, lessons learned from developing an ABAC model based on AWS IoT will be valuable for similar development in other platforms and further benefit studies on a platform-independent model as well.

REFERENCES

[1] M. Gupta, M. Abdelsalam, S. Khorsandroo, and S. Mittal, "Security and privacy in smart farming: Challenges and opportunities," *IEEE Access*, vol. 8, pp. 34564–34584, 2020.

[2] S. Sontowski, M. Gupta, S. S. L. Chukkapalli, M. Abdelsalam, S. Mittal, A. Joshi, and R. Sandhu, "Cyber attacks on smart farming infrastructure," in *Proc. IEEE 6th Int. Conf. Collaboration Internet Comput. (CIC)*, Dec. 2020, pp. 135–143.

[3] S. Bhatt, F. Patwa, and R. Sandhu, "An access control framework for cloud-enabled wearable Internet of Things," in *Proc. IEEE 3rd Int. Conf. Collaboration Internet Comput. (CIC)*, Oct. 2017, pp. 328–338.

[4] *Amazon Web Service*. Accessed: Jan. 4, 2021. [Online]. Available: https://aws.amazon.com/

[5] *Google Cloud Platform*. Accessed: Dec. 10, 2020. [Online]. Available: https://cloud.google.com/docs

[6] *Microsoft Azure*. Accessed: Dec. 24, 2020. [Online]. Available: https://azure.microsoft.com/en-us/

[7] *AWS IoT*. Accessed: Jun. 24, 2021. [Online]. Available: https://aws.amazon.com/iot/

[8] 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, 1996.

[9] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The NIST model for role-based access control: Towards a unified standard," in *Proc. 5th ACM Workshop Role-Based Access Control (RBAC)*, vol. 10, 2000, pp. 1–17.

[10] Y. Zhang, F. Patwa, and R. Sandhu, "Community-based secure information and resource sharing in AWS public cloud," in *Proc. IEEE Conf. Collaboration Internet Comput. (CIC)*, Oct. 2015, pp. 46–53.

[11] S. Bhatt, F. Patwa, and R. Sandhu, "Access control model for AWS Internet of Things," in *Proc. Int. Conf. Netw. Syst. Secur.* Cham, Switzerland: Springer, 2017, pp. 721–736.

[12] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to attribute based access control (ABAC) definition and considerations (draft)," *NIST Special Publication*, vol. 800, no. 162, pp. 1–54, 2013.

[13] X. Jin, R. Krishnan, and R. Sandhu, "A unified attribute-based access control model covering DAC, MAC and RBAC," in *Proc. IFIP Annu. Conf. Data Appl. Secur. Privacy*. Berlin, Germany: Springer, 2012, pp. 41–55.

[14] M. Nitti, V. Pilloni, G. Colistra, and L. Atzori, "The virtual object as a major element of the Internet of Things: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1228–1240, 2nd Quart., 2015.

[15] A. Ouaddah, H. Mousannif, A. A. Elkalam, and A. A. Ouahman, "Access control in the Internet of Things: Big challenges and new opportunities," *Comput. Netw.*, vol. 112, pp. 237–262, Jan. 2017.

[16] J. L. Hernández-Ramos, A. J. Jara, L. Marın, and A. F. Skarmeta, "Distributed capability-based access control for the Internet of Things," *J. Internet Services Inf. Secur.*, vol. 3, nos. 3–4, pp. 1–16, Nov. 2013.

[17] 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.*, 2012, pp. 604–608.

[18] 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 Secur. Mobility*, vol. 1, no. 4, pp. 309–348, Oct. 2014.

[19] R. Xu, Y. Chen, E. Blasch, and G. Chen, "BlendCAC: A BLockchain-enabled decentralized capability-based access control for IoTs," in *Proc. IEEE Int. Conf. Internet Things (iThings), IEEE Green Comput. Commun. (GreenCom), IEEE Cyber, Phys. Social Comput. (CPSCom), IEEE Smart Data (SmartData)*, Jul. 2018, pp. 1027–1034.

[20] R. Sandhu, E. Coyne, H. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Comput.*, vol. 29, no. 2, pp. 38–47, Feb. 2013.

[21] 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. 51–319.
[22] J. Liu, Y. Xiao, and C. L. P. Chen, "Authentication and access control in the Internet of Things," in *Proc. 32nd Int. Conf. Distrib. Comput. Syst. Workshops*, Jun. 2012, pp. 588–592.
[23] M. Gupta, F. Patwa, and R. Sandhu, "POSTER: Access control model for the Hadoop ecosystem," in *Proc. 22nd ACM Symp. Access Control Models Technol.*, Jun. 2017, pp. 125–127.
[24] J. Wang, H. Wang, H. Zhang, and N. Cao, "Trust and attribute-based dynamic access control model for Internet of Things," in *Proc. Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discovery (CyberC)*, Oct. 2017, pp. 342–345.
[25] S. Ding, J. Cao, C. Li, K. Fan, and H. Li, "A novel attribute-based access control scheme using blockchain for IoT," *IEEE Access*, vol. 7, pp. 38431–38441, 2019.
[26] L. Song, M. Li, Z. Zhu, P. Yuan, and Y. He, "Attribute-based access control using smart contracts for the Internet of Things," *Procedia Comput. Sci.*, vol. 174, pp. 231–242, Jan. 2020.
[27] A. Alshehri and R. Sandhu, "Access control models for virtual object communication in cloud-enabled IoT," in *Proc. IEEE Int. Conf. Inf. Reuse Integr. (IRI)*, Aug. 2017, pp. 16–25.
[28] S. Kaiwen and Y. Lihua, "Attribute-role-based hybrid access control in the Internet of Things," in *Proc. Asia–Pacific Web Conf.* Cham, Switzerland: Springer, 2014, pp. 333–343.
[29] D. Gupta, S. Bhatt, M. Gupta, O. Kayode, and A. S. Tosun, "Access control model for Google cloud IoT," in *Proc. IEEE 6th Int. Conf. Big Data Secur. Cloud (BigDataSecurity), Int. Conf. High Perform. Smart Comput. (HPSC), IEEE Int. Conf. Intell. Data Secur. (IDS)*, May 2020, pp. 198–208.
[30] A. Thakare, E. Lee, A. Kumar, V. B. Nikam, and Y.-G. Kim, "PARBAC: Priority-attribute-based RBAC model for azure IoT cloud," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 2890–2900, Apr. 2020.
[31] B. Ur, J. Jung, and S. Schechter, "The current state of access control for smart devices in homes," in *Proc. Workshop Home Usable Privacy Secur. (HUPS)*, vol. 29, 2013, pp. 209–218.
[32] Y. Zhang, D. Zheng, and R. H. Deng, "Security and privacy in smart health: Efficient policy-hiding attribute-based access control," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 2130–2145, Jun. 2018.
[33] S. Ameer, J. Benson, and R. Sandhu, "The EGRBAC model for smart home IoT," in *Proc. IEEE 21st Int. Conf. Inf. Reuse Integr. Data Sci. (IRI)*, Aug. 2020, pp. 457–462.
[34] S. Ameer and R. Sandhu, "The HABAC model for smart home IoT and comparison to EGRBAC," in *Proc. ACM Workshop Secure Trustworthy Cyber-Phys. Syst.*, Apr. 2021, pp. 39–48.
[35] M. Shakarami and R. Sandhu, "Role-based administration of role-based smart home IoT," in *Proc. ACM Workshop Secure Trustworthy Cyber-Physical Syst.*, Apr. 2021, pp. 49–58.
[36] M. Gupta, J. Benson, F. Patwa, and R. Sandhu, "Dynamic groups and attribute-based access control for next-generation smart cars," in *Proc. 9th ACM Conf. Data Appl. Secur. Privacy*, Mar. 2019, pp. 61–72.
[37] M. Gupta, F. M. Awaysheh, J. Benson, M. Alazab, F. Patwa, and R. Sandhu, "An attribute-based access control for cloud enabled industrial smart vehicles," *IEEE Trans. Ind. Informat.*, vol. 17, no. 6, pp. 4288–4297, Jun. 2021.
[38] M. Gupta, J. Benson, F. Patwa, and R. Sandhu, "Secure V2V and V2I communication in intelligent transportation using cloudlets," *IEEE Trans. Services Comput.*, early access, Sep. 22, 2020, doi: 10.1109/TSC.2020.3025993.
[39] M. Gupta and R. Sandhu, "Towards activity-centric access control for smart collaborative ecosystems," in *Proc. 26th ACM Symp. Access Control Models Technol.*, Jun. 2021, pp. 155–164.
[40] S. S. L. Chukkapalli, A. Piplai, S. Mittal, M. Gupta, and A. Joshi, "A smart-farming ontology for attribute based access control," in *Proc. IEEE 6th Int. Conf. Big Data Secur. Cloud (BigDataSecurity), Int. Conf. High Perform. Smart Comput. (HPSC), IEEE Int. Conf. Intell. Data Secur. (IDS)*, May 2020, pp. 29–34.
[41] S. Xu, Y. Li, R. Deng, Y. Zhang, X. Luo, and X. Liu, "Lightweight and expressive fine-grained access control for healthcare Internet-of-Things," *IEEE Trans. Cloud Comput.*, early access, Aug. 20, 2019, doi: 10.1109/TCC.2019.2936481.
[42] S. Bhatt and R. Sandhu, "Convergent access control to enable secure smart communities," in *Proc. 2nd IEEE Int. Conf. Trust, Privacy Secur. Intell. Syst. Appl. (TPS-ISA)*, Oct. 2020, pp. 148–156.
[43] P. W. L. Fong, "Relationship-based access control: Protection model and policy language," in *Proc. 1st ACM Conf. Data Appl. Secur. Privacy (CODASPY)*, 2011, pp. 191–202.
[44] Y. Cheng, J. Park, and R. Sandhu, "Relationship-based access control for online social networks: Beyond user-to-user relationships," in *Proc. Int. Conf. Privacy, Secur., Risk Trust Int. Conf. Social Comput.*, Sep. 2012, pp. 646–655.
[45] X. Zhang, F. P. Presicce, and R. Sandhu, "Formal model and policy specification of usage control," *ACM Trans. Inf. Syst. Secur.*, vol. 8, no. 4, pp. 351–387, Nov. 2005.
[46] S. Bhatt, A. T. Lo'ai, P. Chhetri, and P. Bhatt, "Authorizations in cloud-based Internet of Things: Current trends and use cases," in *Proc. 4th Int. Conf. Fog Mobile Edge Comput. (FMEC)*, Jun. 2019, pp. 241–246.
[47] S. Bhatt and R. Sandhu, "ABAC-CC: Attribute-based access control and communication control for Internet of Things," in *Proc. 25th ACM Symp. Access Control Models Technol.*, Jun. 2020, pp. 203–212.
[48] M. Gupta and R. Sandhu, "Authorization framework for secure cloud assisted connected cars and vehicular Internet of Things," in *Proc. 23nd ACM Symp. Access Control Models Technol.*, Jun. 2018, pp. 193–204.
[49] M. Gupta and R. Sandhu, "The GURA_{G} administrative model for user and group attribute assignment," in *Proc. Int. Conf. Netw. Syst. Secur.* Cham, Switzerland: Springer, 2016, pp. 318–332.
[50] J. Chen, "Oil refinery," Tech. Rep., 2020. [Online]. Available: https://www.investopedia.com/terms/o/oil-refinery.asp
[51] M. Alkhaldi, C. Pathirage, and U. Kulatunga, "The role of human error in accidents within oil and gas industry in Bahrain," in *Proc. 13th Int. Postgraduate Res. Conf. (IPGRC), Conf.* Salford, U.K.: Univ. of Salford, 2017, pp. 822–834.
[52] A. Neal, "How IoT is enhancing the performance of control valves," Tech. Rep., 2018. [Online]. Available: https://www.controlglobal.com/articles/2018/how-iot-is-enhancing-the-performance-of-control-valves/
[53] H. Patel, S. Patel, M. Shah, and L. Sharma, "Real time multi sensor technique for false alarm reduction using IoT," Tech. Rep., Mar. 2018, vol. 12. [Online]. Available: http://www.ijcea.com/real-time-multi-sensor-technique-false-alarm-reduction-using-iot/
[54] R. Atoui, "IoT security concerns for oil & gas," Tech. Rep., 2020.
[55] A. Das, S. Imai, S. Patterson, and M. P. Wittie, "Performance optimization for edge-cloud serverless platforms via dynamic task placement," in *Proc. 20th IEEE/ACM Int. Symp. Cluster, Cloud Internet Comput. (CCGRID)*, May 2020, pp. 41–50.

SMRITI BHATT received the Ph.D. degree in computer science from The University of Texas at San Antonio. She has done the Ph.D. research at the Institute for Cyber Security (ICS) and the Center for Security and Privacy Enhanced Cloud Computing (C-SPECC). She has been an Assistant Professor with the Department of Computer and Information Technology, Purdue University, since 2021. She has been actively publishing her work on well-regarded conferences and journals in the field. Her research interests include the field of cyber security, mainly focused on access control and communication control models, and security and privacy in cloud computing, and the Internet of Things (IoT). Her current research projects focus on developing secure access control and communication control models for cloud-enabled IoT architecture applicable to various IoT domains, such as smart home, smart health, and wearable IoT. Her research work also expands into deep learning for the IoT security with applications in access control and anomaly detection. She serves as an Expert Reviewer for journals, namely IEEE Transactions on Cloud Computing, IEEE Access, and IEEE Transactions on Dependable and Secure Computing, and technical program committee on several conferences and workshops.

THANH KIM PHAM is currently pursuing the bachelor's degree in computer science with Tennessee Technological University. He plans to join graduate studies and pursue his interest in future computing technologies. His primary research interests include software development methodologies, the IoT, and data analytics.

MAANAK GUPTA (Member, IEEE) received the B.Tech. degree in computer science and engineering, India, the M.S. degree in information systems from Northeastern University, Boston, USA, and the M.S. and Ph.D. degrees in computer science from The University of Texas at San Antonio (UTSA). He has worked as a Postdoctoral Fellow with the Institute for Cyber Security (ICS), UTSA. He is currently an Assistant Professor in computer science with Tennessee Technological University, Cookeville, USA. He has worked in developing novel security mechanisms, models and architectures for next generation smart cars, smart cities, intelligent transportation systems, and smart farming. His primary research interests include security and privacy in cyber space focused in studying foundational aspects of access control and their application in technologies, including cyber physical systems, cloud computing, the IoT, and big data. He is also interested in machine learning-based malware analysis and AI assisted cyber security solutions. His research has been funded by the U.S. National Science Foundation (NSF), NASA, the U.S. Department of Defense (DoD), and private industry.

JAMES BENSON received the B.Sc. and M.Sc. degrees in physics from Clarkson University, in 2007 and 2009, respectively, and the M.Sc. degree in electrical engineering from The University of Texas at San Antonio (UTSA), in 2016. He has worked at the Texas Renewable Energy Institute (TSERI) and the Open Cloud Institute (OCI), UTSA, where he was assisting with data analytics and various research projects. He is currently working as a Technology Research Analyst II with the Institute for Cyber Security (ICS) and the Center for Security and Privacy Enhanced Cloud Computing (C-SPECC), UTSA. His research interests include cyber physical systems, cloud computing, and automation.

JAEHONG PARK (Member, IEEE) received the B.B.A. degree in information management from Dongguk University, Seoul, South Korea, the M.S. degree in information systems technologies from George Washington University, and the Ph.D. degree in information technology from George Mason University. He was a Research Associate Professor with the Institute for Cyber Security, The University of Texas at San Antonio. He is currently an Associate Professor of management, marketing and information systems with The University of Alabama in Huntsville. He pioneered the area of usage control (UCON), and his seminal works on UCON models have been well-regarded and highly cited. His research interests include data and application security and privacy, access and usage control, secure collaboration, cloud computing security, smart and connected system security, the IoT security, secure provenance, and social computing. He is a member of ACM and ACM SIGSAC. He has served as a Program Committee Member, the General Chair, and the Program Chair for the ACM Conference on Data and Application Security and Privacy (CODASPY). He has also served as a program committee member for many conferences and workshops.

RAVI SANDHU (Fellow, IEEE) received the B.Tech. degree from IIT Bombay, the M.Tech. degree from IIT Delhi, and the M.S. and Ph.D. degrees from Rutgers University. He served on the Faculty at George Mason University, from 1989 to 2007, and The Ohio State University, from 1982 to 1989. He is currently a Professor of computer science, the Executive Director of the Institute for Cyber Security, and the Lead PI of the NSF Center for Security and Privacy Enhanced Cloud Computing, The University of Texas at San Antonio, where he holds the Lutcher Brown Endowed Chair in cyber security. He is a fellow of ACM and AAAS, and has received numerous awards from IEEE, ACM, NSA, NIST, and IFIP. A prolific and highly cited author, his research has been funded by NSF, NSA, NIST, DARPA, AFOSR, ONR, AFRL, ARO, and private industry. He was the Chairman of ACM SIGSAC, and founded the ACM Conference on Computer and Communications Security, the ACM Symposium on Access Control Models and Technologies, and the ACM Conference on Data and Application Security and Privacy.