2022 IEEE 8th International Conference on Collaboration and Internet Computing (CIC) | 978-1-6654-7300-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/CIC56439.2022.00021

Attributes Aware Relationship-based Access Control for Smart IoT Systems

Lopamudra Praharaj*, Safwa Ameer†, Maanak Gupta§, and Ravi Sandhu¶
*§Dept. of Computer Science, Tennessee Technological University, Cookeville, Tennessee 38505, USA
†¶Institute for Cyber Security and NSF C-SPECC Center, The University of Texas at San Antonio, TX, USA
*lpraharaj42@tntech.edu, †safwa.ameer@gmail.com, §mgupta@tntech.edu, ¶ravi.sandhu@utsa.edu

***Abstract*—The pervasive nature of smart connected devices has intruded on our daily lives and has become an intrinsic part of our world. However, the wide use of the Internet of Things (IoT) in critical application domains has raised concerns for user privacy and security against growing cyber threats. In particular, the implications of cyber exploitation for IoT devices are beyond financial losses and could constitute risks to human life. Most deployed access control solutions for smart IoT systems do not offer policy individualization, the ability to specify or change the policy according to the individual user's preference. As a result, currently deployed systems are not well suited to specify access control policies in a multi-user environment, where users access the same devices to perform different operations. The system's security gets tricky when the smart ecosystem involves complicated social relationships, much like in a smart home. Relationship-based access control (ReBAC), widely used in online social networks, offers the ability to consider user relationships in defining access control decisions and supports policy individualization. However, to the best of our knowledge, no such attempt has been made to develop a formal ReBAC model for smart IoT systems. This paper proposes a $\mathrm{ReBAC_{IoT}}$ dynamic and fine-grained access control model which considers the social relationships among users along with the attributes to support an attributes-aware relationship-based access control model for smart IoT systems. $\mathrm{ReBAC_{IoT}}$ is formally defined, illustrated through different use cases, implemented, and tested.**

***Index Terms*—IoT, Access Control, Relationship-based access control, Neo4j, Amazon Web Services (AWS)**

I. Introduction and Motivation

The Internet of Things (IoT) refers to the smart and automated devices that communicate with each other to enable convenience and remote services for users. It has proliferated in multiple spheres of human lives and has become an intrinsic part of our smart lifestyle. The rapid progress of IoT has led to a technological revolution. This enables a connected ecosystem that includes novel consumer applications in smart homes, elder care, organizational applications (medical and health care, vehicular communication systems), industrial applications (manufacturing, agriculture), infrastructure applications (smart cities, energy management), and military applications including Internet of Battlefield Things.

Although the functional aspects and the novel IoT applications completely unimaginable, the security issues involved in this technology are often overlooked. The deployment of resource constrained devices, along with the adoption of a plethora of technologies in IoT, enlarges the attack surface and introduces new security vulnerabilities [1], [2]. Open source and proprietary IoT frameworks like SmartThings [3], Nest, and IoTivitiy [4] help connectivity over the cloud and the edge to enable communication and operations by devices and third-party applications. In such a setting, IoT devices are accessed by other smart devices and third-party applications running remotely in the cloud and operated using mobile devices held by end users. If such complicated systems are compromised, an adversary can remotely issue commands to switch ON your home thermostat or remotely issue commands to TURN OFF your car engine [5]. To prevent such security attacks and to make these distributed smart devices resilient against such misuse, it is essential to deploy security mechanisms, including access control, at different interfaces in this ecosystem without curtailing user experience. Current access control policies fall short when multiple users use the same devices [6]. Real-world examples of lapses in access control models have started to surface; for instance, a burger commercial triggered a home assistant, and a cartoon mischievously triggered an Amazon Echo voice assistant to fill an Amazon cart with items [7]. Although these examples have no financial losses, other scenarios may involve an adversary asking a voice assistant to open the smart front door lock.

Many access control models have been proposed in the literature for different IoT application domains. The majority of them are based on role-based access control (RBAC) [8], [9], or attribute-based access control (ABAC) [10], [11]. Several other access control models for IoT that are built on different technologies, such as UCON [12]–[14] and blockchain technology [15]–[18] have been proposed. However, most of these models support a system-wide access control policy defined by the security administrator. On the other hand, they do not support policy individualization, where different users can express their preferences on how their own or related devices can be used. In an IoT system that involves multiple users with complex social relationships, even though one user can own a device, his/her actions may impact other related users in the system. For example, in smart home IoT use case, changing the thermostat temperature will also affect other users in the system. Therefore, policy individualization must exist, and different users should be able to specify their 'relationship-based' policies on the devices they own or for the devices which affect them. Moreover, users should be able to generate policies that regulate their usage according to their preferences. The system must then collectively utilize these

978-1-6654-7300-2/22/$31.00 ©2022 IEEE
DOI 10.1109/CIC56439.2022.00021

individualized policies from related users and system-defined centralized policies for access control decisions. Furthermore, in some IoT applications, systems' users have complex social relationships between them. Traditional access control models don't consider the relationship between IoT system users when deciding on access. This is particularly critical in the social IoT (SIoT) application domain [19], [20]. The objectives being pursued by the SIoT paradigm are clear: to keep separate the two levels of people and things; to allow objects to have their own social networks; to enable humans to impose rules to protect their privacy and only access the result of autonomous inter-object interactions occurring on the objects social network. However, how social relationships can be used for access control in the way used in online social networks (OSNs) [21] has not been widely studied.

Relationship-based access control model (ReBAC) models [21]–[24] were first introduced to address access control in online social networks (OSNs). ReBAC models' basic idea is to consider the social relationship between the subject user and the object owner when deciding on different access requests. Moreover, ReBAC models support policy individualization, where in addition to system administrators, different users in the system can customize their policies on their related users or resources. However, the existing core ReBAC models [21], [22], [24] do not capture different characteristics of users, devices, operations, and environments, to develop fine-grained policies and rules for socially driven authorization. Some proposed ReBAC models [23], [25] incorporate users or/and relationship attributes in addition to the relationships between different users in the system when deciding on access requests. Nevertheless, these models were designed for social networks, which have different policy needs, and only require a limited number of relationships, such as *friend* or *friend-of-friend*. Furthermore, The dynamic nature of IoT systems necessities incorporating devices', operations', and environment's attributes when deciding on an access request. He et al [6] discussed the need for a social relationship-centric access control model for multi-user IoT devices. However, no formal model has been proposed so far.

This paper proposes an operational relationship-based access control model for smart IoT systems, referred to as $\mathrm{ReBAC_{IoT}}$. Our model is inspired by the attribute-aware relationship-based access control model designed for online social networks [23]. However, unlike the attribute-aware ReBAC model [23] which only captures users and relationships attributes when deciding on an access request, $\mathrm{ReBAC_{IoT}}$ is a dynamic and fine-grained model that captures different users, sessions, devices, operations, and environmental attributes. Moreover, unlike other access control models, $\mathrm{ReBAC_{IoT}}$ supports policy individualization and considers the complex social relationships between the subjects (users) and the objects (resources) in the authorization process. The key contributions of this paper are as follows:

- We motivate the need for an attribute-aware relationship-based access control model for socially-driven IoT.
- We propose $\mathrm{ReBAC_{IoT}}$, a formal relationship-based access control policy model for smart IoT systems.
- We present multiple use cases to demonstrate and highlight the need for $\mathrm{ReBAC_{IoT}}$ formal policy model.
- We provide a proof of concept implementation using Neo4j [26] graph database in AWS IoT [27] to illustrate $\mathrm{ReBAC_{IoT}}$ applicability in commercial technologies.

The rest of the paper is organized as follows. Section II discuss relevant background and related work. Section III propose the $\mathrm{ReBAC_{IoT}}$ formal model, followed by policy specification in Section IV. Different smart applications use-cases and detailed proof of concept implementation are discussed in Sections V and VI respectively. Our work is concluded with some future directions in Section VII.

II. Related Work

Multiple access control models have been proposed in the literature for IoT application domains. Some models are based on ABAC [10], [11] as in [28]–[33], while other models are based on RBAC [8], [9] as in [34]–[38]. UCON based access control models [12]–[14] were also utilized for different IoT application domains [39]–[41]. Moreover, some of the proposed models are built on blockchain technology [15]–[18]. Several other access control models for IoT have been proposed. For example, authors in [42] presented a certificate-based device access control scheme in an IoT environment. The authors in [1], [2], [43], [44] provided surveys on different access control models in the literature.

ReBAC have been widely used in the OSNs. It enables different content owners to specify their preferences on how related users can access their objects (i.e., photos, posts, etc) based on the relationship between the content owner and the requesting user. The authors in [22], [24] proposed a ReBAC model that utilizes user-to-user relationships (the relationship between the requesting user and the target resource owner) to decide on different access requests in a social network. Cheng et al [21] has extended the ReBAC model proposed in [22] to incorporate user-to-resource and resource-to-resource relationships in addition to user-to-user relationships when deciding on access requests. Carminati et al [25] [45] [46] [47] have introduced the notion of trust to make an access decision based on the trust level, type and depth of user to user relationship using a centralized certificate authority which asserts the validation of relationship path. Besides OSNs, ReBAC models have also been proposed in other application domains, including healthcare and education. In the same line, Fong et al. [48] have also demonstrated the use of ReBAC in electronic health record use cases.

In general, pure ReBAC models don't capture different characteristics of users, devices, operations, and environments. Hence, they are not fine-grained and dynamic models. Accordingly, the authors in [22] proposed an attribute-aware ReBAC model. This model utilizes the relationship type between the requesting user and target resource owner, as well as users' attributes and relationships' attributes, to decide on access requests. Attribute-aware ReBAC offers more flexi-

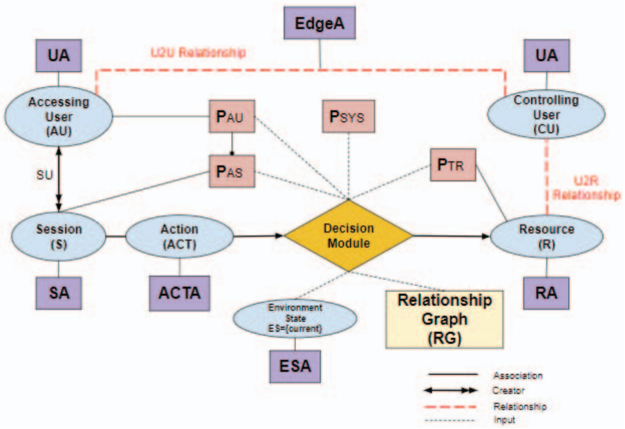

Fig. 1: Conceptual $\mathrm{ReBAC_{IoT}}$ Model

ble and expressive policies. However, unlike our proposed model ($\mathrm{ReBAC_{IoT}}$), attribute-aware ReBAC doesn't capture sessions, resources, operations, and environmental attributes.

Recently, some works on ReBAC for IoT have been proposed [49], [50]. In [49], the authors proposed a blockchain architecture to enforce ReBAC in a smart city use case. However, they neither provided a formal authorization policy model nor implemented or tested their architecture. In [50], the authors focused on representing how relationships actualize and dissolve over time in relationship graphs.

III. Formal Model

This section introduces the $\mathrm{ReBAC_{IoT}}$ model. It is a user-to-user ReBAC model. In other words, it decides on access requests based on the relationship between the subject user and the owner of the requested resource. The model is conceptually illustrated in Figure 1, and formally specified in Table I.

A. Basic Components

Users (U): This set refers to the system's human users. It consists of accessing users (AU) and controlling users (CU). Accessing user is a user who tries to access a resource in the system. Each resource in the system is owned by a specific user; we refer to the resource owner as the controlling user.

Relationships (Σ): This set defines the relationship types between users (U2U Relationship). Given a relationship type $\sigma_i \in \Sigma$, the inverse of the relationship is $\sigma_i^{-1} \in \Sigma$. This model supports only one relationship between resources and users (U2R Relationship), which is ownership.

Sessions (S): Each user creates one or more sessions during which he may initiate an action on a resource. Each session is linked to a unique, controlling user through the many to one SU relation. The user-session distinction allows sessions to partially inherit some of its unique creator user's policies. A user might have multiple sessions with different inherited access control policies active concurrently and asynchronously.

Resources (R): Resources is the set of targets in the smart system. We have two types of resources, devices (D) and information ($INFO$). Devices is the set of smart devices, and information is the set of different information on different devices (e.g., the video feeds file on the smart camera). The function $info$ maps each device to the set of information available on that device.

Actions (ACT): The actions set refers to the actions allowed to be performed on resources as specified by resource manufacturers. We have two types of actions: (a) Actions on devices (ACT_d), and (b) Actions on information (ACT_{info}).

Policies (P): We have two types of policies, user-specified policies and system-specified policies. In user-specified policies, ReBAC_{IoT} allows users to express their preferences concerning themselves or their related users and resources. Accessing user policy (P_{AU}), accessing session policy (P_{AS}), and target resource policy (P_{TR}) are policies specified by different users in the system and are applied to accessing user, accessing session, and target resource respectively. Accessing user policies (P_{AU}) is the set of policies specified by different users in the system and include policies that regulate the accessing user access rights granted by different users in the system. Accessing session policies (P_{AS}) is a subset of the accessing user policies set (P_{AU}). A user creates sessions during which he/she initiates some actions on specific resources. For a session s_i, the session policies set P_{s_i} is inherited from the user u_j, where $u_j = user(s_i)$, and $P_{s_i} \subseteq P_{u_j}$. Two different sessions initiated by the same user may inherit different subsets of policies. How different sessions inherit policies from users is considered part of administrative access control which is outside the scope of this model. Target resource policies (P_{TR}) is the set of policies specified by different users, including the target's controlling user, and regulate the target resource access rights. On the other hand, system-specified policies P_{SYS} is the set of policies defined by the system administrator and applied system-wide. It is categorized into two types: system authorization policies ($P_{SYSauth}$) and conflict resolution policies (P_{SYScr}). System authorization policies enable the administrator to decide on authorization rights for users on resources. Authorization policies written by multiple users may conflict and require conflict resolution policies. System conflict resolution policies are outside this model's scope and are part of administrative access control models.

Environment State (ES): Environment state is a singleton set, where current denotes the posture of the environment at the current time. This can be described by different environment attributes as shown in Section III-B

Relationship Graph (RG): The relationship graph is depicted similarly to the OSN's social graph proposed in [22], where the relationship between different smart system users can be depicted as a directed labeled simple graph. Each user is represented as a node, and the edges between nodes represent the U2U relationship between different users. For every relationship $\sigma_i \in \Sigma$, there exist an inverse relationship $\sigma_i^{-1} \in \Sigma$. We do not explicitly always show the inverse relationships on the relationship graph, but we assume the original relationship and its inverse twin always exist simultaneously.

Access Decision Module (ADM): The access decision module receives the request, converges all required policies as well as the relationships on the relationship graph, and decides on

TABLE I: ReBAC$_{IoT}$ Model Formalization Part I: Basic Components

Users, Relationship, and Sessions

$-U$ is the set of users, which include accessing users ($AU \subseteq U$) and controlling users ($CU \subseteq U$).
$-\Sigma$ is the set of relationships between users (i.e., relationship types).
$-\Sigma = \{\sigma_1, \sigma_2, .., \sigma_n, \sigma_1^{-1}, \sigma_2^{-1}, .., \sigma_n^{-1}\}$ denotes a finite set of relationship types, where each type specifier denotes a relationship type supported in the system between two users.
$-S$ is the set of sessions (each session is created, terminated and controlled by an individual user).
$-SU \subseteq S \times U$ is a many to one relation assigning each session to its single controlling user. We define the derived function $user(s) : S \to U$, where: $user(s_i) = u_j$ such that $(s_i, u_j) \in SU$.

Resources and Actions

$-R$ is the set of resources. Resources can be devices (D) or information ($INFO$), $R = D \cup INFO$.
$-D$ is the set of devices deployed in the smart system.
$-INFO$ is the set of possible information on devices (device manufacturers specified)
$-$The function $info : D \to 2^{INFO}$ specifies the valid information for each device (device manufacturers specified)
$-ACT$ is the set of actions, $ACT = \{act_1, act_2, \ldots, act_n\}$. Actions are initiated by sessions on resources.
$-ACT = ACT_d \cup ACT_{info}$.
$-ACT_d$ is the set of possible actions on devices (device manufacturers specified).
$-ACT_{info}$ is the set of possible actions on devices information (device manufacturers specified).

Environment State

$-ES = \{current\}$ is a singleton set where $current$ denotes the environment at the current time instance

Policies

$-P$ is the set of policies that govern the ability of accessing users to access target resources.
$-P_{AU}$ is the set of accessing user policies.
$-P_{AS}$ is the set of accessing session policies. We define $P_{AS} \subseteq P_{AU}$.
$-P_{TR}$ is the set of target resource policies.
$-P_{SYS}$ is the set of system-specified policies. This set is furthered divided into system authorization policies ($P_{SYSauth}$) and conflict resolution policies(P_{SYScr}). We have $P_{SYS} = P_{SYSauth} \cup P_{SYScr}$
- We have $P = P_{AU} \cup P_{TR} \cup P_{Sys}$.

Relationship Graph

$-RG$ is the relationship graph of an IoT smart system users. It is modeled as a triple $RG = \langle U, E, \Sigma \rangle$ where:

- U: is the set of users in the smart system.
- Σ is the set of relationships between users (i.e., relationship types).
- E is the set of graphs edges. We have $E \subseteq U \times U \times \Sigma$.

$-RG$ is the directed labeled relationship graph, where $(u_i, u_j, \sigma_k) \in RG$ refers to a relationship from user u_i to user u_j with the relationship name σ_k.

TABLE II: ReBAC$_{IoT}$ Model Formalization Part II: Attribute Functions and Values

$-UA, SA, RA, ACTA, ESA, EdgeA$ and $CountA$ are sets of user, session, resource, action, environment state, edge, and count attribute functions receptively.
– Session attribute functions can be inherited attribute functions from the session's unique user creator or it can be unique session attribute functions.
– Each session s_i inherits a subset of the attribute functions in UA from its unique user creator (controlled by the session creator $user(s_i)$). For every inherited attribute function $att \in UA$, $att(s_i) = att(user(s_i))$ at all time.
– S_uA is the unique session attribute functions set.
– Resource attribute functions set RA can be divided into two subsets: (a)Device attribute functions DA. (b)Information attribute functions $INFOA$. $RA = DA \cup INFOA$.
– $EdgeA$ and $CountA$ are attributes related to the relationship graph.
– Edge attribute functions set $EdgeA$ describe edges in the relationship graphs.
– Count attribute function set $CountA$ is a singleton set, where $CountA = \{count\}$.
– Count attribute $count$ captures the number of occurrence for the attribute-based relationship graph path specification as described in Section IV-A and Section III-B .
– For each attribute att in $UA \cup SA \cup DA \cup INFOA \cup ACTA \cup ESA \cup EdgeA \cup CountA$, $Range(att)$ is the attribute range, a finite set of atomic values
– $attType : UA \cup SA \cup DA \cup INFOA \cup ACTA \cup ESA \cup EdgeA \to \{set, atomic\}$.
– $attType : CountA \to \{atomic\}$.
– The attribute $count \in CountA$ maps the occurrence of a specific path in the relationship graph to a specific number in $Rang(count)$.
– Each $att \in UA \cup SA \cup DA \cup INFOA \cup ACTA \cup ESA \cup EdgeA$ correspondingly maps users in U, sessions in S, devices in D , information in $INFO$, actions in ACT, the environment state $current$, or edge in E to atomic or set attribute values. Formally:

$$att : U \text{ or } S \text{ or } D \text{ or } INFO \text{ or } ACT \text{ or } \{current\} \text{ or } E \to \begin{cases} Range(att), & \text{if } attType(att) = atomic \\ 2^{Range(att)}, & \text{if } attType(att) = set \end{cases}$$

the access. Policy conflicts are resolved using conflict resolution policies in P_{SYS}. An *access request* is a triple (s, r, act), whereby an accessing user session s requests to perform an action act on a resource r. For example, if Alice, the parent at home, wishes to turn on ($turn\_on$) the home security alarm ($home\_alarm$). This request can be modeled as (s_{Alice}, $home\_alarm$, $turn\_on$) where $user(s_{Alice}) = Alice$.

B. Attributes in ReBAC$_{\text{IoT}}$

In IoT smart systems, the dynamism of communication between people, connected devices, data, utility, and the changing nature of the environmental characteristics in IoT smart systems requires that actors' rights change accordingly. Therefore, it is critical to capture users', sessions', resources', and environment's attributes when deciding on an access control request. See Table II for attribute functions definition. Attributes are functions that take an entity, such as a user, and determine a specific value from its range. For each attribute function att_j, there is a range of possible values ($Range(att_j)$) that att_j can be evaluated to. An atomic valued attribute will return one value from its range, while a set valued attribute will return a subset of its range. UA, SA, $ACTA$, RA, and ESA are sets of attributes associated with users, sessions, actions, resources, and environment state, respectively. User attributes (UA) define users' characteristics, such as name, age, etc. Session attributes (SA) capture session characteristics. Some session attributes are inherited from the session's user creator, for instance, user age, user gender, etc. Other session attributes (S_uA) are unique to the session, for example, the type of connection, the device used for the access, etc. Action attributes ($ACTA$) is the set of attributes associated with the requested action, for instance, action level of danger. Resource attributes RA is the set of attributes associated with resources. We have two types of resource attributes, device attributes DA, and information attributes $INFOA$, that describe different devices and information characteristics, respectively. Hence, $RA = DA \cup INFOA$. Different environment characteristics are captured through the environment state attributes ESA. Examples of environment states attributes may include weather, time, etc. Moreover, we define two relationship graph related attributes , Edge attributes($EdgeA$) and count attributes ($CountA$). $EdgeA$ are characteristics that describe edges in relationship graphs. For instance, relationship weights, relationship types, and so on. $CountA$ describes the occurrence requirement for the attribute-based path specification; it specifies the lower bound of the occurrence of such bath, as described in Section IV-A.

IV. Policy specification

In IoT, attribute-based policies should capture the requesting user and session attributes, the requested action attributes, the requested resource attributes, and the current environment attributes. The authors in [23] proposed an attribute aware policy specification language for social networks ReBAC models. However, their policy only captures users' and relationships' attributes. In this paper, we adapt this policy language for IoT application use cases. Moreover, we extend it to capture resources', actions', and environmental attributes in addition to users' and relationship attributes. Towards this goal, in Section IV-A, we introduce the attribute-based policy language for relationship graph related attributes, which is adapted from [23] and specifies access control requirements on relationship paths (nodes and relationship) between accessing user and the target's controlling user in the relationship graph. Then, in Section IV-B, we introduce the authorization function grammar, which specifies access control requirements on the requesting session, the requested device, the requested action, and the environmental context. Finally, in Section IV-C we utilize the attribute-based policy language for relationship graph attributes and the authorization function to present our formal policy language for ReBAC$_{IoT}$. This policy language governs accessing user policies (P_{AU}), accessing session policies (P_{AS}) target resource policies (P_{TR}), and system authorization policies ($P_{SYSauth}$). System conflict resolution policies (P_{SYScr}) are outside the scope of this model.

A. Attribute-Based Policy Language for Relationship Graph Features

TABLE III: Attribute quantifiers [23]

$\forall[+m,-n]$	All entities from the m^{th} to the n^{th} last, $m+n \leq h$ where m & n are non-negative integers and h is a hop-count limit
$\forall[+m,+n]$	All entities from the m^{th} to the n^{th}, $m \leq n \leq h$
$\forall[-m,-n]$	All entities from the m^{th} last to the n^{th} last, $h \geq m \geq n$
$\exists[+m,-n]$	One entity from the m^{th} to the n^{th} last, $m+n \leq h$
$\exists[+m,+n]$	One entity from the m^{th} to the n^{th}, $m \leq n \leq h$
$\exists[-m,-n]$	One entity from the m^{th} last to the n^{th} last, $h \geq m \geq n$
$\forall\{2^{\{\pm N\}}\}$	All entities in this set
$\exists\{2^{\{\pm N\}}\}$	One entity in this set

Access control requirements related to different attributes are specified using attribute-based policies. Many users may exist on the relationship path between two users in the relationship graph (RG). Each user (node) or relationship (edge) has attributes that can be used for specifying access control rules. Sometimes, when a user tries to access a specific resource, the attributes of all users or relationships on the path between the requesting user and the resource controlling user need to be considered. In some cases, attributes of only certain users or relationships are used. To capture these cases, we need attribute quantifiers. In this model, we use the attribute quantifiers proposed in [23] and shown in Table III. As illustrated in Table III, the universal quantifier $\forall$ denotes "all" user(s) or relationship(s), while the existential quantifier $\exists$ denotes "at least one" user or relationship. The notations $[]$ and $\{\}$ denote ranges and a set of users/relationships, respectively. These ranges and sets are located at a specific distance on the relationship path between accessing user and the target resource's controlling user. Plus and minus signs express the forward (from the start) and backward directions (from the end), followed by a number that denotes the position from the front or the back. The indicator for users starts from 0. On the other hand, the indicator for relationships begins from 1. For instance, for a relationship, $+1$ indicates the first relationship on the path, while -2 means the second last. However, for users, $+0$ denotes the starting user, and -1 represents the second last user on the path. The attribute-based policy for the attributes related to the relationship graph, which are nodes (users) and edges (relationships) attributes, is defined as follows:

$$\text{A relationship graph attributes-based policy rule} \\ (RGAttPolicy) \text{ is a triple,} \\ \langle quantifier, f(UA, EdgeA), f(CountA) \rangle \ .$$

In a relationship graph attributes-based policy rule, a quantifier denotes the quantity and the position of specific node/edge attributes. It is applied to a user and edge attribute function ($f(UA, EdgeA)$ but not to the count attribute function ($f(CountA)$. $f(UA, EdgeA)$ is a boolean function of the quantified user and/or edge attributes. For instance, consider the following three rules:

- *Ra* : $\langle \exists[+1,-1], familyMember(u) = True, count \geq 3 \rangle$
- *Rb* : $\langle \exists[+1,-1], familyMember(u) = True \wedge adult(u) = True, - \rangle$
- *Rc* : $\langle \forall[+1,-1], TrustLevel(e) \geq 7, - \rangle$

Ra defines a rule stating that "there must be at least three common connections (paths) between the requester and the resource owner, which contains a family member". In *Rb* and *Rc*, the count attribute predicate is not used and is shown as $'-'$, which means $count \geq 1$ in default. *Rb* defines a rule

TABLE IV: $\mathrm{ReBAC_{IoT}}$ Model Formalization Part III: Authorization Policy

− Accessing user policies P_{AU}, accessing session policies P_{AS}, target resource policies P_{TR}, and system authorization policies $P_{SYSauth}$ are defined using the following formula:

$$< Authorization(s : S, act : ACT, r : R, current : ES), GraphRule >$$

− The graph rule $GraphRule$ is described using the grammar of Table V.

Attributes Authorization Function

−$Authorization(s : S, act : ACT, r : R, current : ES)$ is a propositional logic formula returning true or false specified using the following grammar.

- $\alpha ::= term \mid term \wedge term \mid term \vee term \mid (term) \mid \neg term \mid \exists x \in set.\alpha \mid \forall x \in set.\alpha$
- $term ::= set\ setcompare\ set \mid atomic \in set \mid atomic \notin set \mid atomic\ atomiccompare\ atomic$
- $setcompare ::= \subset \mid \subseteq \mid \nsubseteq$
- $atomiccompare ::= < \mid = \mid \leq$
- $set ::= sa(s) \mid acta(op) \mid da(d) \mid infoa(info) \mid esa(current)$, where $attType(sa) = attType(acta) = attType(da) = attType(infoa) = attType(esa) = set$
- $atomic ::= sa(s) \mid acta(op) \mid da(d) \mid infoa(info) \mid esa(current) \mid value$, where $attType(sa) = attType(acta) = attType(da) = attType(infoa) = attType(esa) = atomic$

−For a specific session s_i, resource r_j and action act_k, the authorization function $Authorization(s_i, act_k, r_j, current)$ is evaluated by substituting the actual attribute values of $sa(s_i)$, $acta(act_k)$, $esa(current)$, and $da(r_j)$ if the requested resource is a device ($r_i \in D$) or $infoa(r_j)$ if the requested resource is an information ($r_j \in INFO$) for the corresponding symbolic placeholders and evaluating the resulting logical formula to be True or False. Any term that references an undefined attribute value is evaluated as False.

TABLE V: Grammar for graph rules

$GraphRule \rightarrow$ “(”$StartingNode$“,”$PathRule$“)”
$PathRule \rightarrow AttPathSpecExp \mid AttPathSpecExp$
Connective PathRule
$AttPathSpecExp \rightarrow PathSpecExp \mid PathSpecExp$
” : ”$RGAttPolicy$
$Connective \rightarrow \vee \mid \wedge$
$PathSpecExp \rightarrow PathSpec \mid$ “$\neg$”$PathSpec$
$PathSpec \rightarrow$ “(”$AttPath$“,”$HopCount$“)” |
“(”EmptySet“,” HopCount“)”
$HopCount \rightarrow Number$
$AttPath \rightarrow Path \mid Path$” : ”$RGAttPolicy$
$Path \rightarrow TypeSeq \mid TypeSeqPath$
$EmptySet \rightarrow \emptyset$
$TypeSeq \rightarrow AttTypeExp \mid AttTypeExp$“ $\cdot$ ”$TypeSeq$
$AttTypeExp \rightarrow TypeExp \mid TypeExp$“ : ”$RGAttPolicy$
$TypeExp \rightarrow TypeSpecifier \mid TypeSpecifier\ Wildcard$
$RGAttPolicy \rightarrow$ use dedicated parser to process
$StartingNode \rightarrow u_a \mid u_c$
$TypeSpecifier \rightarrow \sigma_1 \mid \sigma_2 \mid \ldots \mid \sigma_n \mid \sigma_1^{-1} \mid \sigma_2^{-1} \mid \ldots \mid \sigma_n^{-1} \mid \Sigma$
$Wildcard \rightarrow$ “ $*$ ”|“?”|“ $+$ ”
$Number \rightarrow [0-9]+$

stating that ”there must be at least one common path between the requester and the resource owner, which contains an adult family member”. *Rc* defines a rule saying that ”there must be at least one common bath between the requester and the resource owner, in which the trust level of each edge in the bath is greater than or equal to 7”. Hence, the system will check each edge on the path to ensure its trust value meets the requirement before granting access.

B. Attributes Authorization Function

An attribute authorization function is a boolean function. It is inspired by the work of [32], and defined using the grammar of Table IV. For a specific session s_i, action act_k, and a resource r_j the authorization function $Authorization(s_i, act_k, r_j, current)$ is evaluated by substituting the actual attribute values of $sa(s_i)$, $acta(act_k)$, $da(r_j)$ (if the requested resource is a device in the smart system $r_j \in D$) or $infoa(r_j)$ (if the requested resource is an information on a smart device in the smart system $r_j \in INFO$), and $esa(current)$ for the corresponding symbolic placeholders and evaluating the resulting logical formula to be True or False. Any term that references an undefined attribute value is evaluated as False. Term refers to any atomic logical declarative sentence. An atomic sentence is a type of declarative sentence that is either true or false and cannot be broken down into other sentences [51].

C. Policy Formalization

As shown in Table IV, each access control authorization policy is composed of two parts: the authorization function and the graph rule. It is represented as a pair $< Authorization(s : S, act : ACT, r : R, current : ES), GraphRule >$. The attributes' authorization function $Authorization(s : S, act : ACT, r : R, current : ES)$ set attributes specifications on the requesting session s, requested action act, requested resource r, and the current environment state $current$. On the other hand, the graph rule $GraphRule$ specifies the type and attributes of the relationship path between the access requester and the controlling user of the target resource.

Graph Rules. Table V defines the syntax for the graph rules using Backus-Naur Form (BNF). This syntax is adapted from [23]. Each graph rule specifies a starting node ($startingnode$) and a path rule ($pathrule$). Starting node stands for the user where the policy evaluation begins, which can be the accessing user (u_a) or the resource's controlling user (u_c). A path rule comprises one or more attribute path spec expressions ($AttPathSpecExp$). Each attribute path spec expression consists of one or two parts, the path spec expression only ($PathSpecExp$) or the path specifier expression and the relationship graph attribute policy ($RGAttPolicy$). Since this is an attribute-aware U2U ReBAC, we add the term $RGAttPolicy$ similarly to the addition of $AttPolicy$ in [23]. The $RGAttPolicy$ is defined in Section IV-A.It facilitates policies capable of capturing relationship graph attributes (RGA). The $RGAttPolicy$, in this case, is called a global relationship graph attribute-based policy. It denotes the relationship graph attributes that need to be applied on the entire $PathSpecExp$. The $PathSpecExp$ can be expressed as a path specifier ($PathSpec$) with or without negation. The $PathSpec$ state the required sequence of relationship types and the corresponding maximum number of edges on the graph (the hop count limit for the sequence). Users can specify a more complicated and fine-grained policy for an action against a target by connecting multiple path spec expressions with conjunctive connective “$\wedge$” and disjunctive connective “$\vee$”. Also, negation “$\neg$” over path specs is used to imply the absence of the specified pattern of relationship types and hop count limit as authorization requirements. The path specifier consists of two parts, the attribute path ($AttPath$) and the hop count ($HopCount$) or the empty set and the hop count. The attribute path consists of the path ($path$) or the path and a relationship graph attribute policy ($RGAttPolicy$). The path $path$ is a sequence of characters, denoting the pattern of relationship path between two users that must be satisfied, the $RGAttPolicy$ (if it is there) is called a local relationship graph attribute-based policy that applies only to this $path$ segment, and the hopcount limits the maximum number of edges on the path. The pattern of relationship path ($path$) represents a sequence of type sequence ($TypeSeq$) . $TypeSeq$ can be one attribute type expressions ($AttTypeExp$) or multiple attribute type expressions concatenated together where, $AttTypeExp$“ · ”$TypeSeq$ denotes multiple $AttTypeExp$ concatenated together. The $AttTypeExp$ consists of one or two parts, the type expression only ($TypeExp$), or the type expression and a local $RGAttPolicy$, which needs to be applied to this type expression only. The $TypeExp$ consists of one type specifier ($TypeSpecifier$), or one $TypeSpecifier$ and a wildcard ($Wildcard$). The ($TypeSpecifier$) denotes a relationship in the set of relationships. Hence, the $path$ is basically a sequence of $TypeSpecifier$ (relationships) denoting the pattern of relationship types required between the starting node and the evaluating node in the relationship graph to denote the access. We use three kinds of wildcard notations representing different occurrences of relationship types: an asterisk (*) for 0 or more, plus (+) for 1 or more, and a question mark(?) for 0 or 1. The hop count ($HopCount$) describes the maximum distance between the starting node and the evaluated node in the graph. The $AttPath$ can be the empty set, and $HopCount = 0$, indicating that only the policy writer can access the resource under the specified conditions.

V. Use Case Scenarios

The following section describes two use cases that utilize $\mathrm{ReBAC_{IoT}}$ authorization policies in different IoT domains.

Use Case 1: Smart Home. The child John, would like to grant his direct friends who are older than nine years access to entertainment devices during weekend evenings only.

$$P1 :\langle \text{ (day(current)} \in \{\text{Sa,S}\} \wedge \{\text{17:00}\} \\ \leq time(current) \leq \{19:00\} \wedge \text{EntertainmentDevices(r)=} \\ \text{True)}, (\text{u}_a, ((friend, 1){:}\exists \; \{\text{+0}\}, \text{ age(u)} \geq 9, -))\rangle$$

In this use case, the authorization policy has two parts, the authorization function ($day(current) \in \{Sa, S\} \wedge$ 17:00 $\leq time(current) \leq$ 19:00 $\wedge EntertainmentDevices(r) = True$) and the graph rule. The graph rule consists of two parts: (1) A starting node equal to u_a indicating that the graph rule should be calculated starting from the accessing user in the relationship graph. (2) One attribute path specifier expression. The attribute path specifier expression has two parts a path specifier ($(friend, 1)$) and a relationship graph attribute policy ($\exists\{+0\}, age(u) \geq 9, -$). The relationship graph attribute policy $\exists\{+0\}, age(u) \geq 9, -$ indicates that there must be at least one connection between the requesting user and the target controlling user, where the requesting user's age is greater than or equal to nine. The path specifier $(friend, 1)$ indicates that this connection path has one edge with a direct relationship friend. The authorization function ($day(current) \in \{Sa, S\} \wedge$ 17:00 $\leq time(current) \leq$ 19:00 $\wedge EntertainmentDevices(r) = True$) captures the required environment and resource attributes. In other words, to grant the access request, the day needs to be Saturday or Sunday, the access time between 5:00 pm and 7:00 pm, and the requested resource needs to be an entertainment device.

Use Case 2: Smart Health. Doctor Alex wants to grant each nurse that he supervises access to the medical records of the patients under the supervision of both the doctor and the nurse. Nurses are not allowed to access the medical records of patients who are not under their supervision but under the doctor's supervision. Nurses can access those medical records only during their shifts from 8:00 am to 5:00 pm.

$$P2 : \langle \; (\; 08:00 \leq time(current) \leq \text{05:00} \wedge \\ MedicalRecords(r) = True), \\ (\quad u_a, (Responsiblenurse, 1) \wedge \\ (SupervisorResponsibledoctor, 2) \quad) \; \rangle$$

In this use case, the graph rule consists of two parts: (1) The starting node u_a indicating that the graph rule should be calculated starting from the accessing user in the relationship graph. (2) Two attribute path specifier expressions with a connective logical and ($\wedge$) between them. Each attribute path specifier expression consists of one path specifier. The first path specifier is $(Responsiblenurse, 1)$ indicating that there must be a connection path of one edge in the relationship graph between the requesting user and the target resource owner (which is the patient) with a relationship $Responsiblenurse$. In other words, the requesting user needs to be the responsible nurse of the target owner (The patient). The second path specifier is $(SupervisorResponsibledoctor, 2)$ indicating that there must be another connection path of two edges in the relationship graph between the requesting user and the target resource owner with a relationship sequence $Supervisor$ and $Responsibledoctor$, in other words the requesting user need to be a nurse reported to a doctor who in turns the responsible doctor of the target resource owner. Note here that the two attribute path specifier expressions don't contain a relationship graph attribute policy ($RGAttPolicy$) indicating that there is no graph related attributes that need to be satisfied on the two specified paths ($(Responsiblenurse, 1)$ and $(SupervisorResponsibledoctor, 2)$). Moreover, the authorization function ($08:00 \leq time(current) \leq$ 05:00 $\wedge$ $MedicalRecords(r) = True$) indicates that to grant the access the time of request should be between 8:00 am and 5:00 pm and the resource should be a medical record.

VI. Proof Of Concept Implementation

In this section, we provide a proof of concept implementation of the $\mathrm{ReBAC_{IoT}}$ model demonstrating its practicality. Section VI outlines our implemented use case while section VI-B illustrates our architecture and underlying details.

A. Use Case Outline

We modeled the use case based on the $\mathrm{ReBAC_{IoT}}$ components, as shown in Table VI. The goal is to evaluate the user access request based on the policies at the end of Table VI

TABLE VI: $\text{ReBAC}_{\text{IoT}}$ Implementation Use Case

$U = \{Alex, Bob, John, Juliet, Andrew\}$
$UA = \{UserAge, Admin\}$
$UserAge : U \rightarrow \{\text{x : x is an integer }\}$
$Admin : U \rightarrow \{True, False\}$
$UserAge(Alex) = 36, UserAge(Bob) = 32$
$UserAge(John) = 14, UserAge(Juliet) = 9$
$UserAge(Andrew) = 14$
$Admin(Alex) = Admin(Bob) = True$
$Admin(John) = Admin(Juliet) = Admin(Andrew) = False$

$S = \{...\}$
$SA = \{UserAge, Admin, SessionTimeOut\}$
$UserAge : S \rightarrow \{\text{x : x is an integer }\}$
$Admin : S \rightarrow \{True, False\}$
$SessionTimeOut : s : S \rightarrow \{\text{x : x is an integer }\}$

$\Sigma = \{Spouse, Child, Friend\}$

$R = D \cup INFO$
$INFO = \{\}, D = \{SmartDoor, SmartLight, SmartTV, PlayStation\}$
$RA = DA \cup INFOA$
$INFOA = \{\}, DA = \{EntertainmentDevices, DeviceOwner\}$
$EntertainmentDevices : D \rightarrow \{True, False\}$
$DeviceOwner : D \rightarrow U$
$EntertainmentDevices(SmartDoor) = False$
$EntertainmentDevices(SmartLight) = False$
$EntertainmentDevices(SmartTV) = True$
$EntertainmentDevices(PlayStation) = True$
$DeviceOwner(PlayStation) = John$
$DeviceOwner(SmartDoor) = Alex$
$DeviceOwner(SmartLight) = Alex$
$DeviceOwner(SmartTV) = Alex$

$ACT = ACT_{SmartDoor} \cup ACT_{SmartLight} \cup$
$\quad ACT_{SmartTV} \cup ACT_{PlayStation}$
$ACT_{SmartDoor} = \{lock, unlock\}$
$ACT_{SmartLight} = \{turn\_on, turn\_off\}$
$ACT_{SmartTV} = \{turn\_on, turn\_off\}$
$ACT_{PlayStation} = \{turn\_on, turn\_off\}$
$ACTA = \{\}$

$ES = \{Current\}$
$EA = \{Time\}$
$Time : ES \rightarrow \{\text{x : x is an hour of a day }\}$
$P1 : \langle\ (\text{day(current)} \in \{\text{Sa,S}\} \wedge \{\text{17:00}\} \leq time(current) \leq \{19:00\}$
$\wedge EntertainmentDevices(r) = True),$
$(u_a, ((Friend, 1) : \exists\{+0\}, age(u) \geq 9, -))\rangle$
$P2 : \langle(DeviceOwner(r) = Alex), (u_a, (\emptyset, 0))\rangle$
$P3 : \langle(DeviceOwner(r) = Alex), (u_a, (Spouse, 1))\rangle$

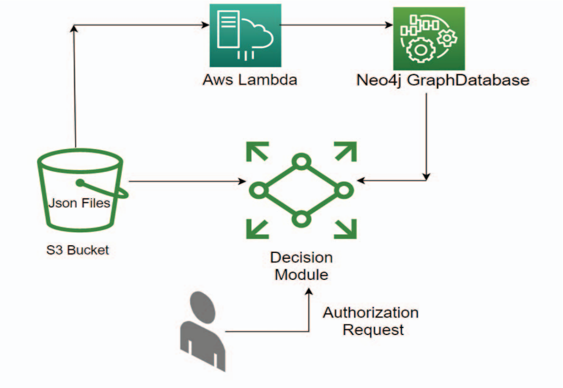

Fig. 2: Deployed System Architecture using AWS

(*P1*, *P2*, and *P3*). Our smart IoT system has five users: *Alex*, *Bob*, *John*, *Juliet*, and *Andrew*. We have two user attributes:(1) $UserAge$ determining users' age. (2) $Admin$ determining if the user is a system admin. We have three session attributes, $UserAge$ and $Admin$ inheriting values from the user creator, while $SessionTimeOut$ is a unique session attribute function. The session set is not defined here since it is a dynamic set which will be defined during run time. We defined three relationships *Spouse*, *Child*, and *Friend*. *Bob* and *Alex* are Spouse, *John* and *Juliet* are children of Bob and Alex, and *Andrew* is John's friend. In this use case, we don't have any information resources, hence, the information set ($INFO$) is empty and the resources set (R) is equal to the devices set. We also created four devices: *SmartDoor* (owner *Alex*), *SmartLight* (owner *Alex*), *SmartTV* (owner *Alex*), and *PlayStation* (owner *John*). We have two device attribute functions: (a) $EntertainmentDevices$ determines if the device is for entertainment, and (b) $DeviceOwner$ specifies the owner of each device. Each device *d* has a set of supported actions, referred as ACT_d. In our use case, we don't have action attribute functions, accordingly, the set of action attributes ($ACTA$) is empty set. We have one environment attribute ($Time$), which takes an environment state as an input, and return the current time. Finally, we have three user defined policies: (1) *P1* is defined by user *John*, as explained in use case 1 in Section V. (2) *P2* is written by *Alex*, and is explained as following. The authorization function part ($(DeviceOwner(r) = Alex)$) implies that this policy applies on the devices owned by Alex. The graph rule part ($(u_a, (\emptyset, 0))$ indicates that only the policy writer (Alex) can access the devices specified by this policy. (3) *P3* is written by *Alex*, where the authorization function part ($(DeviceOwner(r) = Alex)$) implies that this policy applies on the resources owned by Alex. The graph rule part ($(u_a, (Spouse, 1))$) indicates that in the relationship graph starting from the requesting user, there should be an edge between the requesting user and the requested resource's owner with a $Spouse$ relationship.

B. Enforcement Architecture

Our deployed system architecture is shown in Figure 2. We simulated the environment with AWS Lambda [52], AWS S3 bucket [53] and Neo4j Graph database [26]. We created four JSON files uploaded to AWS S3 bucket: *users_attributes.json*, *env_attributes.json*, *devices_attributes.json,* and *user_policy.json* to capture attributes of users, environment, and devices, together with user defined policies respectively. We defined a Lambda function to analyze the *users_attributes.json* file and insert the user information into the Neo4j graph database.

Neo4j is a graph database that has nodes and relationships instead of tables or documents, as shown in Figure 3. Relationships are kept locally alongside the nodes to offer more flexible format [54]. The system is optimized for traversing through the data quickly, with millions of edges per second. The data from the Neo4j graph database can be retrieved through a Cypher Query Language [55]. In our use case, the Neo4j graph database (shown in Figure 3) is used (deployed in AWS EC2 [56]) to build the relationship graph between

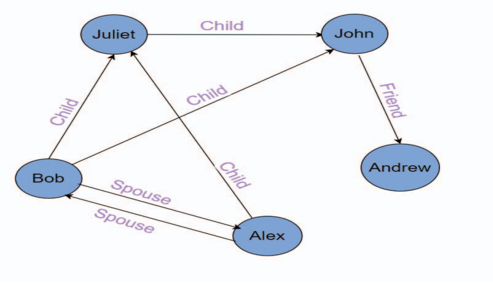

Fig. 3: Relationship Graph in Neo4j

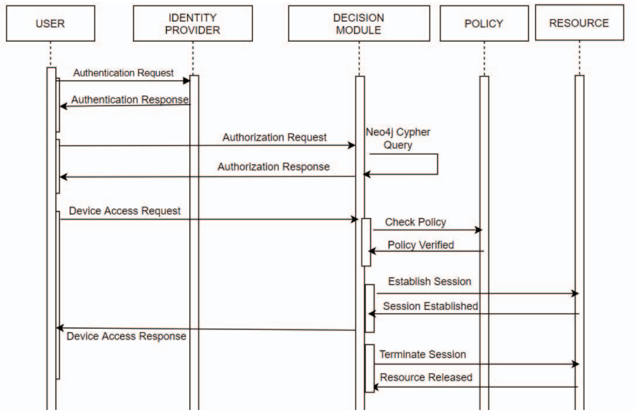

Fig. 4: Access Request Handling Sequence Diagram

the system's users and to calculate the total number of edges between the requesting user and the target device owner. The decision module (written in Python 3.9) receives the access request, and then analyzes it according to the contents of the JSON files and the Neo4j relationship graph.

We developed a web application in the Flask [57] to create our use case. The details of user access request handling are formulated in Figure 4. A user sends the authentication request through the IDP (Identity Provider). If the authentication is valid, the user sends the access request (user, device, action) through the web application. The decision module receives the access request from the application. Then for each defined policy it performs the following two checks: (1) First, using the Cypher Query Language it checks the accessing user node's validation in the Neo4j graph. If the accessing user exists in the graph, the decision module finds the paths between the accessing user and the requested device owner, and checks whether one of the paths satisfy the graph rule of the tested policy. If the graph rule is satisfied by one of the paths, then the decision module performs the second check, if the graph rule is not satisfied for any of the paths between the accessing user and the device owner, then the decision module will check the next policy. (2) In the second step, the decision module extracts the attributes of the requesting session, device, action, and the current environment state from the attributes' JSON file, and checks whether they satisfy the authorization function part of the tested policy. If the two checks are satisfied, the session is established and the user can access the requested device within that session. If none of the policies is satisfied, then the decision module will reject the request.

TABLE VII: Multiple users requesting one device

Number Of Users	Number Of Devices	Average Processing Time in ms
1	1	266.200
2	1	290.597
3	1	315.887
5	1	461.493

TABLE VIII: Multiple users requesting multiple devices

Number Of Users	Number Of Devices	Average Processing Time in ms
1	1	249.344
2	2	464.507
3	3	514.615
4	4	533.962

C. Performance Analysis

In this section, the performance of our implementation is evaluated by conducting multiple test cases. We examined situations with different sets of requests. Each set of authorization requests is processed ten times to determine the average processing time. The multi-threading paradigm of Python is used to evaluate multiple user requests simultaneously. Table VII shows the average processing time of the decision module when multiple users send access request for one device (*PlayStation*). From these results, we can notice that when the number of requests increase, the average processing time of the decision module also increases.

Table VIII shows the decision module average processing time when multiple users send access requests for multiple devices simultaneously. The first row the average processing time when *Alex* requests to $unlock$ the $SmartDoor$. The second row appends one more request to the previous, and checks if *Bob* can $turn\_on$ the $SmartLight$. Third row has three simultaneous requests, appending *John* request to $turn\_on$ the $PlayStation$ with earlier two. Finally, the last row shows the decision module average processing time with four requests, adding *Juliet* request to $turn\_on$ the $SmartTV$. The system decided correctly according to our defined policies (*P1, P2,* and *P3*), where all the requests were approved except *Juliet* request to $turn\_on$ the $SmartTV$. As can be noticed that with the increase in number of requests for multiple users and different devices (one user per device), the average processing time of the decision module also increases.

VII. Conclusion and Future Work

In this paper, we proposed an attribute aware relationship based access control model for socially driven smart IoT systems. The model supports policy individualization allowing system users to define their own policies, and further captures social relationships among users to define a dynamic and fine-grained access control approach. We developed a formal $\mathrm{ReBAC}_{\mathrm{IoT}}$ policy model and attributes aware relationship based policy specification language. Additionally, we presented two use case scenarios for our model in different IoT environments. We demonstrated the applicability of one of the use cases through a proof-of-concept implementation using the Neo4j graph database and AWS. Furthermore, we provided a performance test to show how our system responds in different scenarios. We can conclude that our model is applicable and functional based on the evaluation. In the future, we plan to use user-device and device-device relationships to cover some more practical real-life scenarios. We will also conduct a comprehensive theoretical and empirical analysis comparing our proposed approach with other existing approaches.

Acknowledgement

This paper is partially supported by NSF grants 2025682 at TTU, and 1736209, 2112590 at UTSA.

References

[1] A. Ouaddah *et al.*, "Access control in the internet of things: Big challenges and new opportunities," *Computer Networks*, 2017.

[2] S. Ravidas, A. Lekidis *et al.*, "Access control in internet-of-things: A survey," *Journal of Network and Computer Applications*, 2019.

[3] "Samsung SmartThings," [Accessed:2019-05-05]. [Online]. Available: https://www.smartthings.com/

[4] "Iotivity," [Accessed: 2019-06-08]. [Online]. Available: https://iotivity.org/

[5] W. Zhang *et al.*, "Homonit: Monitoring smart home apps from encrypted traffic," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*.

[6] W. He, M. Golla *et al.*, "Rethinking access control and authentication for the home internet of things (iot)," in *27th USENIX Security Symposium*.

[7] V. Wong, "Burger king's new ad will hijack your google home," in *CNBC*, 2017.

[8] D. F. Ferraiolo *et al.*, "Proposed NIST standard for role-based access control," *ACM TISSEC*, 2001.

[9] R. Sandhu, "Role-based access control," in *Advances in computers*, 1998, vol. 46.

[10] X. Jin, R. Krishnan, and R. Sandhu, "A unified attribute-based access control model covering DAC, MAC and RBAC," in *IFIP Annual Conf. on Data and App. Sec.*, 2012.

[11] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas, "Attribute-based access control," *Comp.*, 2015.

[12] J. Park and R. Sandhu, "Towards usage control models: beyond traditional access control," in *ACM SACMAT*, 2002.

[13] J. Park, "Usage control: A unified framework for next generation access control," Ph.D. dissertation, George Mason University, 2003.

[14] X. Zhang, F. Parisi-Presicce *et al.*, "Formal model and policy specification of usage control," *ACM TISSEC*, 2005.

[15] G. Ali *et al.*, "Blockchain based permission delegation and access control in internet of things (BACI)," *Elsevier Computers & Security*, 2019.

[16] A. Ouaddah *et al.*, "Towards a novel privacy-preserving access control model based on blockchain technology in IoT," in *Europe and MENA Coop. Adv. in Inf. and Comm. Tech.* Springer, 2017.

[17] O. Novo, "Blockchain meets IoT: An architecture for scalable access management in IoT," *IEEE IoT Journal*, 2018.

[18] S. Ding, J. Cao *et al.*, "A novel attribute-based access control scheme using blockchain for IoT," *IEEE Access*, 2019.

[19] M. Nitti *et al.*, "Trustworthiness management in the social internet of things," *IEEE Trans. on knowledge and data engineering*.

[20] L. Atzori *et al.*, "The social internet of things (siot)–when social networks meet the internet of things: Concept, architecture and network characterization," *Computer networks*.

[21] Y. Cheng, J. Park, and R. Sandhu, "Relationship-based access control for online social networks: Beyond user-to-user relationships," in *IEEE Conference on Privacy, Security, Risk and Trust*, 2012, pp. 646–655.

[22] ——, "A user-to-user relationship-based access control model for online social networks," in *IFIP Annual conference on data and applications security and privacy*. Springer, 2012.

[23] ——, "Attribute-aware relationship-based access control for online social networks," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2014, pp. 292–306.

[24] P. W. Fong, "Relationship-based access control: protection model and policy language," in *Proc. of the ACM CODASPY*, 2011.

[25] B. Carminati and E. Ferrari, "Access control and privacy in web-based social networks," *International Journal of Web Information Systems*.

[26] *Neo4j Graph Database*. [Online]. Available: https://neo4j.com/

[27] "Amazon AWS IoT." [Online]. Available: https://aws.amazon.com/iot/

[28] M. Gupta, J. Benson *et al.*, "Dynamic groups and attribute-based access control for next-generation smart cars," in *ACM CODASPY*, 2019.

[29] R. Zhang *et al.*, "ABSAC: attribute-based access control model supporting anonymous access for smart cities," *Security and Communication Networks*, 2021.

[30] B. Bezawada *et al.*, "Securing home IoT environments with attribute-based access control," in *ACM ABAC*, 2018.

[31] M. Gupta and R. Sandhu, "Towards activity-centric access control for smart collaborative ecosystems," in *ACM Symposium on Access Control Models and Technologies*, 2021, pp. 155–164.

[32] S. Ameer, J. Benson, and R. Sandhu, "An Attribute-Based Approach toward a Secured Smart-Home IoT Access Control and a Comparison with a Role-Based Approach," *Information*.

[33] S. Ameer and R. Sandhu, "The HABAC model for smart home IoT and comparison to EGRBAC," in *ACM SAT-CPS Workshop*, 2021.

[34] G. Zhang and J. Tian, "An extended role based access control model for the internet of things," in *IEEE ICINA*, 2010.

[35] E. Barka, S. S. Mathew *et al.*, "Securing the web of things with role-based access control," in *Springer C2SI*, 2015.

[36] S. Kaiwen and Y. Lihua, "Attribute-role-based hybrid access control in the internet of things," in *APWeb*. Springer, 2014.

[37] S. Ameer, J. Benson, and R. Sandhu, "The EGRBAC model for smart home IoT," in *IEEE Conference on Information Reuse and Integration for Data Science*, 2020.

[38] M. J. Covington *et al.*, "Generalized role-based access control for securing future applications," Georgia Tech, Tech. Rep., 2000.

[39] Z. Guoping and G. Wentao, "The research of access control based on UCON in the internet of things," *Journal of Software*, 2011.

[40] A. La Marra *et al.*, "Implementing usage control in internet of things: A smart home use case," in *IEEE Trustcom/BigDataSE/ICESS*, 2017.

[41] F. Martinelli *et al.*, "Too long, did not enforce: a qualitative hierarchical risk-aware data usage control model for complex policies in distributed environments," in *ACM CPSS*, 2018.

[42] S. Malani *et al.*, "Certificate-based anonymous device access control scheme for iot environment," *IEEE Internet of Things Journal*, 2019.

[43] M. Alramadhan and K. Sha, "An overview of access control mechanisms for internet of things," in *IEEE ICCCN*, 2017.

[44] Y. Zhang and X. Wu, "Access control in internet of things: A survey," *arXiv preprint arXiv:1610.01065*, 2016.

[45] B. Carminati, E. Ferrari, and A. Perego, "Rule-based access control for social networks," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2006.

[46] B. , E. Ferrari, and A. Perego, "Enforcing access control in web-based social networks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, pp. 1–38, 2009.

[47] B. Carminati *et al.*, "A semantic web based framework for social network access control," in *ACM SACMAT*, 2009, pp. 177–186.

[48] S. Z. R. Rizvi *et al.*, "Relationship-based access control for an open-source medical records system," in *Proceedings of the ACM Symposium on Access Control Models and Technologies*, 2015, pp. 113–124.

[49] F. Sabrina, "Blockchain and structural relationship based access control for iot: a smart city use case," in *IEEE Conf. on Local Computer Networks*, 2019.

[50] C. Arora *et al.*, "Higher-order relationship-based access control: A temporal instantiation with iot applications," in *ACM on Symposium on Access Control Models and Technologies*, 2022, pp. 223–234.

[51] "Atomic sentence." [Online]. Available: https://en.wikipedia.org/wiki/Atomicsentenc

[52] *AWS Lambda*. [Online]. Available: https://aws.amazon.com/lambda/

[53] *Amazon S3 bucket*. [Online]. Available: https://aws.amazon.com/s3/

[54] J. J. Miller, "Graph database applications and concepts with neo4j," in *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*.

[55] *Cypher Query Language*. [Online]. Available: https://neo4j.com/developer/cypher/

[56] *Amazon EC2*. [Online]. Available: https://aws.amazon.com/ec2/

[57] *Flask Web Developement*. [Online]. Available: https://flask.palletsprojects.com/en/2.2.x/