KRATOS: Multi-User Multi-Device-Aware Access Control System for the Smart Home

Amit Kumar Sikder[1], Leonardo Babun[1], Z. Berkay Celik[2], Abbas Acar[1], Hidayet Aksu[1], Patrick McDaniel[3], Engin Kirda[4], A. Selcuk Uluagac[1]
[1]Florida International University - {asikd003, lbabu002, aacar001, haksu, suluagac}@fiu.edu
[2]Purdue University - zcelik@purdue.edu
[3]Penn State University - mcdaniel@cse.psu.edu
[4]Northeastern University - ek@ccs.neu.edu

ABSTRACT

In a smart home system, multiple users have access to multiple devices, typically through a dedicated app installed on a mobile device. Traditional access control mechanisms consider one unique trusted user that controls the access to the devices. However, multi-user multi-device smart home settings pose fundamentally different challenges to traditional single-user systems. For instance, in a multi-user environment, users have conflicting, complex, and dynamically changing demands on multiple devices, which cannot be handled by traditional access control techniques. To address these challenges, in this paper, we introduce KRATOS, a novel multi-user and multi-device-aware access control mechanism that allows smart home users to flexibly specify their access control demands. KRATOS has three main components: user interaction module, backend server, and policy manager. Users can specify their desired access control settings using the interaction module which are translated into access control policies in the backend server. The policy manager analyzes these policies and initiates negotiation between users to resolve conflicting demands and generates final policies. We implemented KRATOS and evaluated its performance on real smart home deployments featuring multi-user scenarios with a rich set of configurations (309 different policies including 213 demand conflicts and 24 restriction policies). These configurations included five different threats associated with access control mechanisms. Our extensive evaluations show that KRATOS is very effective in resolving conflicting access control demands with minimal overhead, and robust against different attacks.

CCS CONCEPTS

• **Security and privacy → Access control**.

KEYWORDS

Smart Home Security, Access Control, Internet of Things.

ACM Reference Format:
Amit Kumar Sikder[1], Leonardo Babun[1], Z. Berkay Celik[2], Abbas Acar[1], Hidayet Aksu[1], Patrick McDaniel[3], Engin Kirda[4], A. Selcuk Uluagac[1]. 2020. KRATOS: Multi-User Multi-Device-Aware Access Control System for the Smart Home. In *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '20), July 8–10, 2020, Linz (Virtual Event), Austria.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3395351.3399358

WiSec '20, July 8–10, 2020, Linz (Virtual Event), Austria

1 INTRODUCTION

Cyberspace is expanding fast with the introduction of new smart home technologies dedicated to make our homes automated and smarter [28, 35]. This trend will only continue, and billions of smart devices will dominate our everyday lives by the end of this decade [27, 32]. The smart home systems (SHSs) allow multiple devices to be connected to automate daily activities and increase the overall efficiency of the home. Devices as simple as a light bulb to ones as complicated as an entire AC system can be connected and exposed to multiple users. The users then interact with the devices through different smart home applications installed through a mobile host app provided by the smart home vendors. Traditional access control mechanisms proposed for personal devices such as computers and smartphones primarily target single-user scenarios. However, in a SHS, multiple users access the same smart device, typically via a common controller app (e.g., SmartThings App), which can cause conflicting device settings. For instance, a homeowner may want to lock the smart door lock at midnight while a temporary guest may want to access the lock after midnight. Also, current smart home platforms do not allow the conflicting demands of the users to be expressed explicitly. Finally, the current access control mechanism in smart home platforms offer coarse-grained solutions that might cause safety and security issues [2, 4, 16]. For instance, smart home platforms often give automatic full access to every user added to the SHS [25]. With full access, a new user can easily add new unauthorized users, remove existing users, or reconfigure the connected devices [10, 36]. This benign, yet undesired action from the new user can lead to several safety issues [5, 20, 33]. In these real-life scenarios, current smart home platforms cannot fulfill such complex, asymmetric, and conflicting demands of the users as they can only handle primitive and broad controls with static configurations.

In this paper, we introduce KRATOS, a multi-user multi-device-aware access control system designed for the SHSs. KRATOS introduces a formal policy language that allows users to define different policies for smart home devices, specifying their needs. It also implements a policy negotiation algorithm that automatically solves and optimizes the conflicting policy requests from multiple users by leveraging user roles and priorities. Lastly, KRATOS governs different policies for different users, reviewing the results of the policy negotiation and enforcing the negotiation results over the smart home devices and apps. We implemented KRATOS in a real a multi-user multi-device SHS that include 17 different sensors and actuators. We further evaluated KRATOS performance on 219 different policies including 146 demand conflicts and 33 restriction policies collected from real-life smart home users. We also assessed the performance

of Kratos against five different threat models. Our extensive evaluation shows that, Kratos can resolve demand conflicts and detect different threats with 100% success rate in a multi-user multi-device SHS with minimal overhead.

Contributions: The main contributions of this work are as follows:

- We introduced Kratos, a multi-user multi-device-aware access control system for SHS. Kratos provides a flexible policy-based user controls to define user roles and understand users' demands on smart home, a formal policy language to express users' desires, and a policy negotiation mechanism to automatically resolve and optimize conflicting demands and restrictions in a multi-user SHS.
- We implemented Kratos on a real SHS using 17 different smart home devices and sensors. Further, we evaluated its performance with 309 different policies provided by real users. Our evaluation results show that Kratos effectively resolves conflicting demands with minimal overhead.
- We tested Kratos against five different threats arising from inadequate access control system. Our evaluation shows that Kratos can detect different threats with 100% success rate.

Organization: In section 2, we present the needs of access control in SHS. In Section 3, we articulate the problem space and explain the threat model. We detail the design of Kratos in Section 4. Section 5 articulates the implementation of Kratos in a real-life setting. In Section 6, we evaluate the performance of Kratos. Finally, Section 8 discusses the related work and Section 9 concludes the paper.

2 MOTIVATION AND DEFINITIONS

2.1 Access Control Needs in a Smart Home

Access control in multi-user SHSs pose unique challenges in terms of device sharing and conflict resolution. People sharing smart devices in the same environment may have different needs and usage patterns which can lead to conflict scenarios [39]. However, existing smart home platforms mostly offer binary control mode where a user gets all the control or no control at all. For instance, *Samsung SmartThings* provides full access to all the connected devices to a user. Unfortunately, this all-inclusive access permits any authorized users to control the smart devices which can lead to conflicting demands, privacy violations, and undesired app installations [9]. To protect smart devices from unauthorized app installation and device settings, some smart home platforms (e.g., *Apple HomeKit*) offers two device access options: remote access, and editing. In remote access, a new user gets access only privilege to the connected devices. In the editing option, a new user obtains permission for adding or removing any app, device, or user in the system. Additionally, some smart home devices (e.g., August smart lock, Kwikset Kevo Smart Lock, etc.) offers temporary user access or guest access to limit undesired access after an expiry time [18, 23]. These solutions, however, are vendor- and device-specific, thus, are not ready and applicable in a multi-device multi-user smart home system. In summary, *existing access control mechanism in smart home technologies fail to deliver the diverse and complex user demands in a multi-device multi-user setting.*

As conflicting scenarios in a multi-user SHS depend on users' relationships, social norms, and personal preferences, it is important to understand these dynamics before designing a fine-grained access control system. For instance, parents may want to restrict smart TV access for the kids, roommates may want privacy for bedroom locks in a sharing apartment, or owner may want to give temporary access to Airbnb guests. Hence, a fine-grained access control system should address diverse needs of the users in a multi-user multi-device smart home ecosystem. Several prior works have focused on understanding the user preferences and needs by conducting user studies among smart home users [13, 40, 41]. These prior works have established the needs and design requirements of access control systems in SHSs. He et al. conducted a user study among 425 smart home users to understand how relationships among the users affect the access control needs in a SHS [13]. In a recent work, Zeng et al. developed an early prototype of access control system and conduct a usability study among 8 households to identify the access control needs in a real-life SHS [41]. These user studies reported the following preferences and needs among smart home users in terms of access control.

- Majority of the users expressed the need of a fine-grained access control system in a SHSs.
- Users suggested role-based access control (RBAC) in a home environment for limiting access to the devices and applications.
- In a shared environment, users agreed for per-device roles for private rooms.
- For temporary users in a shared environment, users suggested location-based and time-based access control.
- For conflicting scenarios, users expressed the need of automation rules to resolve the conflicting demands and configure the devices at an optimum operational value.

In our work, we consider the users' needs and suggestions reported in prior works to design a fine-grained access control system for multi-user multi-device SHSs.

2.2 Terminology

We define several important terms that we use in this work.

Policy. We consider *Policy* as the group of requests made by the users to control device usage in a multi-user smart environment. Based on the nature of request, there are three types of policies.

(1) *Demand Policy.* We consider *Demand Policy* as the group of requests made by a user that define the control rules for a specific device or group of devices in the smart home system. Demand policies can be general (i.e., created by the admin and applied to all the users in the system) or specific to a certain user. If a demand policy is general to all users, we define that as *General Policy.*
(2) *Restriction Policy.* We consider *Restriction Policy* as the set of rules that govern the accessibility and level of control of a user or group of users to a certain device or group or devices in the smart home system. Restriction policies regulate (1) what devices the user has access to, (2) the time frame in which the user is authorized to use/control the device, and (3) the control setting limits.
(3) *Location-based Policy* We consider *Location-based Policy* as the set of automation rules enforced by the user that are only applicable if the user is connected in the system network. Location-specific policies regulate (1) what devices the user has remote access to and (2) the control setting limit if a specific user is not present in the smart home network.

Priority. We call *Priority* as the importance level of a user that may be used to create preferences for users of higher priority over users with lower priority during new user addition, restriction, and

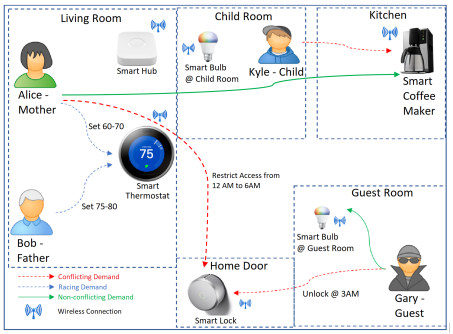


Figure 1: Sample smart home with multiple users attempting to control multiple devices with conflicting demands.

demand negotiation processes. In Section 4, we detail the different priority levels considered in this work.

Conflict. For the purpose of this work, *Conflict* is defined as the dispute process that is generated from two or more demand policies that interfere or contradict based on the specific requests of the policies. Based on the nature of the demand and restriction policies, three types of conflict can occur.

(1) *Hard conflict.* A hard conflict occurs when demand policies of a specific device enforced by two different users do not have any overlapping device condition.
(2) *Soft conflict.* A soft conflict occurs when demand policies enforced by different users for a specific device have overlapping device conditions.
(3) *Restriction conflict.* Restriction conflict occurs when the restriction policy for a device is being disputed by the restricted user.

Device condition. We consider *device condition* as the set of rules assigned to a device by the user to perform a specific task in a SHS. For instance, if the user configures a smart light to switch on at sunset, the specified time is considered as the device condition.

3 PROBLEM AND THREAT MODEL

We introduce the challenges of an access control mechanism and articulate the threat model.

3.1 Problem Definition

We assume a SHS (S) similar to the one depicted in the Figure 1. In S, four different users – Bob (father), Alice (mother), Kyle (child), and Gary (guest) interact with the smart devices. Bob and Alice are the owners and all four users control the devices via the controller app. Here, the term *access to the SHS* refers to the ability of controlling the devices, configuring the system (add/delete devices), and adding new users. We assume the users perform the following conflicting activities- (1) Bob and Alice configure the smart thermostat to different overlapping values at the same time (soft conflict), (2) Alice wants to limit access of smart lock after midnight while Gary wants to have access (hard conflict), (3) Alice wants Kyle to have access to the smart light only while Kyle is present and restrict other devices (restriction conflict and location access). To address these, we propose KRATOS, a fine-grained access control system for the smart home that allows users to resolve the conflicting access control demands automatically, add new users, select specific devices to share, limit the access to specific users, and prevent undesired user access based on user location in the system.

3.2 Threat Model

KRATOS considers undesired access control decisions that may arise from existing coarse-grain solutions. For instance, a new user automatically gets full access to the system (i.e., over privileged control) which may lead to *undesired device access*. Also, KRATOS considers legitimate smart home users *trying to change the system settings without authorization* (e.g., overriding existing system by installing new apps) that may result in undesired device actions such as installing unknown apps and overriding device conditions (i.e., privilege abuse), even deleting device owners from the system (i.e., privilege escalation). Furthermore, KRATOS considers threats that arise from *inadequate*, *inaccurate*, or *careless access control* to multi-user multi-device smart home (i.e., transitive privilege). In fact, access to a SHS granted to unknown parties by an authorized user other than the owner may escalate to additional threats (i.e., unauthorized device access), that KRATOS also considers as malicious activity. Also, if a temporary guest is not timely removed from the system by the authorized user, it may lead to malicious activities such as sensitive information leakage.

We do not consider any unauthorized user access due to malicious apps installed in the system. We also assume that the SHS is not compromised, which means no malicious user is added automatically at the time of system installation as they are different problems from the contributions of KRATOS.

4 KRATOS DESIGN

In this section, we present the architecture of the KRATOS and its main components. KRATOS is a comprehensive access control system for multi-user SHS where users can express their conflicting demands, desires, and restrictions through policies. KRATOS allows an authorized user to add new users and enforce different policies to smart devices based on the needs of users and the environment. KRATOS considers all the enforced policies from authorized users and includes a policy negotiation algorithm to optimize and solve conflicts among users. In designing the KRATOS framework, we consider the following design features and goals.

User-friendly Interface. An access control system should have a user-friendly interface to add or remove users and assign policies in the SHS. We integrate KRATOS into the mobile app provided by smart home vendors to provide a single user interface to manage users and assign policies in the connected devices.

Diverse User Roles/Complex Relations. In a SHS, users have different roles that an access control system needs to define. For example, a user having a parent role should be able to express controls on a user with a child role, while adults in the same priority class should be able to negotiate the access control rules automatically. To address this design feature, KRATOS introduces user priority in the system to define user roles.

Conflict Resolution. As discussed earlier, diverse needs in device usage result in conflicts among users in a shared SHS. The main challenge of an access control system in a SHS is to resolve these conflicts in a justified way. In addition, users in a multi-user SHS should agree with the outcome of conflict resolution provided by the access control system. KRATOS uses a novel policy negotiation system to automatically optimize and resolve the conflicting demands among users and institute a generalized usage policy reflecting the needs of all the users.

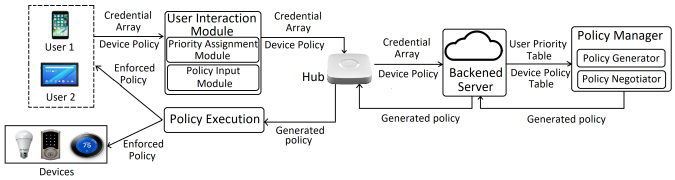


Figure 2: Architecture of Kratos system.

Policy Expiration. In a SHS, temporary access for different smart devices might be needed for guests or occasional users. To automate the temporary access, Kratos offers policy expiration time for specific users. Kratos automatically deletes the devices access and the user from the system after the expiration period to avoid undesired access in the system.

Location-based Access. SHSs offer remote access and users can control the devices within the smart home. Kratos uses the location of the users to trigger a device policy to limit device usage and meet diverse needs of the users. For example, the parents may want to restrict remote access of the kids for specific devices. Kratos only allows users to access location-restricted devices if he/she is connected to the home network.

Expressive Control. In a SHS, a user should be able to express the desired device settings easily. An access control system should provide a simple method to the users to express their diverse needs. Kratos introduces a unified policy language that covers different control parameters (e.g., role, environmental, time, device, location-specific expressions) in a SHS to understand the users' needs and control the devices accordingly.

Unified Policy Enforcement. All user commands [3] to the devices should go through an *access control enforcement* layer to provide fine-grained access control in SHS. Kratos uses an execution module that checks all enforced policies before executing a user command in the SHS.

Figure 2 shows the architecture of the Kratos system. Kratos includes three main modules: (1) user interaction module, (2) back-end module, and (3) policy manager. First, the *user interaction module* provides a user interface to add new users and assign priorities based on the user's role. This module also collects user-defined device policies for smart home devices. These device policies and priority assignment data are forwarded to the *back-end module* via the smart home hub. back-end module captures these data and creates user priority and device policy list for the users. Lastly, the *policy manager* module gathers user priorities and device policies from the generated lists and triggers policy generation and negotiation process. The following subsections details each module in Kratos and explains how policy generation and negotiation processes are initiated by Kratos.

4.1 User Interaction Module

The user interaction module collects priority assignment data and device policies from the users. It includes two sub-modules: priority assignment and policy input .

Priority Assignment Module. The priority assignment module operates as a user interface to add new users and assign priorities. Kratos introduces a formal format to specify new users, illustrated as follows: $U_a = [A_{id},\ N_{id},\ P,\ D,\ T]$, where, A_{id} is the unique ID of the commanding user, N_{id} is the new user ID that is added in the system, P is the priority level of the new user, D is the permission to add or remove devices from the system, and T is the validity

$@U_1$: Alice
U_2- restrict smart thermostat between 60-70 degrees.
U_3- restrict access to smart coffee maker.
U_3- allow access to smart bulb at the child room if U_4 is home.
U_4- allow access to smart bulb at the guest room.
U_4- restrict access to smart lock at the home door from 12:00 AM to 6:00 AM.
$@U_2$: Bob
U_1- restrict smart thermostat between 75-80 degrees.
U_3- allow access to smart bulb in the child's room between 7:00 PM and 7:00 AM
U_4- allow access to smart lock at the guest room and the home door
$@U_3$: Kyle
U_1- desires to access smart bulb at the child room
U_1- desires to access coffee maker
$@U_4$: Gary
U_1- desires to access smart lock at the guest room and the home door at 3 AM
U_1- desires to access the smart bulb in guest room

Figure 3: An example demand and restriction requirements of users in Figure 1.

time of the new user in the system. The user priority level is used in the policy generation module to resolve conflicting demands. For adding a new user and assigning priorities, we consider the following rules to avoid conflicts in the priority assignment.

- Each user has an authority to add new users and assign a priority.
- The Owner of the SHS will have the highest priority in the system.
- Priority in the system is depicted with a numerical value. The lower the priority of a user, the higher is the level of priority. For example, the owner of the hub has the priority of "0".
- Each user can only assign the same or higher value of the priority to a new user, e.g., a user with a priority of "1" can only assign priority of "1" or higher to a new user.
- If two existing users add the same new user with a different priority level, the user with a higher priority level gets the privilege to add the new user.
- If two existing users with the same priority level assigned different priority levels to a new user, the system notifies the existing users to fix a priority level of the new user.
- Each user can assign permissions for adding or removing devices to a new user if the commanding user has the same permission.

The priority assignment of Kratos can also be configured to define the roles of the users. For example, the SHS in Figure 1, Alice and Bob (parents) can be assigned to priority 0, Gary (guest) can be assigned to priority 2, and Kyle (child) can be assigned to priority 3. We use this priority list to explain the functions of Kratos throughout the paper. In Kratos, administrator or homeowner obtains the privilege to define the priority-role mappings in the system. Kratos also allows the users to add temporary users by specifying validity time (T) of a user in the system. After the specified validity time, Kratos removes the user from the system automatically preventing any unauthorized access from a temporary guest.

Policy Input Module and Access Policy Language. Policy input module provides an interface to the users for assigning policies in smart home devices. All the authorized users can choose any installed device and create a device policy using this module. To define the device policies, Kratos introduces a formal access control policy language for the SHS to express complex user preferences (e.g., users' demands, desires, and restrictions) by utilizing an existing open-source smart home ecosystem (e.g., Samsung SmartThings). Each user defines a policy about their preferences for each smart device and any restriction over others' accesses in the SHS. For instance, sample policies for the smart home of four users shown in Figure 1, where each user defines her requirements for other users in a smart home with the thermostat, bulbs, lock, and coffee maker, are shown in Figure 3. The criteria defined by the users are used throughout this sub-section to construct their policies.

$@U_1$ restrict :: : $thermostat_1$: temperature $\notin [60 - 70]$;
restrict :: U_3 : coffeemaker : ;
location :: U_3 : $bulb_3$: location $\in [Home]$;
demand :: U_4 : $bulb_4$: ;
restrict :: U_4 : $lock_1$: time $\notin [6:00am - 9:00pm]$;
$@U_2$ restrict :: : $thermostat_1$: temperature $\notin [75 - 80]$;
demand :: U_3 : $bulb_3$: time $\in [7:00pm - 7:00am]$;
demand :: U_4 : $lock_1$, $lock_4$: ;
… …

Figure 4: Sample policy clauses to partially implement demands and restrictions shown in Figure 3.

Policy Structure. Kratos represents the policies as collections of clauses. The clauses allow each user to declare an independent policy for their demands and other users. The clauses have the following structure: ⟨users⟩ : ⟨devices⟩ : ⟨conditions⟩ : ⟨actions⟩. The first part of the policy is *users*, which includes the information of both policy assigner and assignee. The second part, *devices*, specifies the device or a list of devices included in this statement. Kratos uses device ID assigned by the SHS to distinguish device-specific policies in a multi-device environment. The third part, *conditions*, is a list of device conditions defining different control parameters (time-based operation, values, etc.) based on the capabilities of the smart devices. For instance, a user may define a condition where only a pre-defined range of commands or only a certain time-window is matched. The final part of the policy is ⟨action⟩ which states the clause type, *demand*, *restrict*, or *location*. We note that the Kratos's policy language allows users to define multiple clauses. For instance, a user may restrict a distinct subset of smart home devices for different conditions and different users. A sample policy scenario is illustrated in Figure 4.

4.2 Back-end Module

The user interaction module collects the user credentials and device policies generated using the access policy language. It then forwards them to the back-end module where these data are stored and formatted for policy generation and negotiation. The back-end module has two functionalities: (1) generating user priority list, and (2) generating device policy list.

User Priority List. The back-end module collects the credential arrays and creates a database for authorized users and their assigned priorities. All the credential arrays are checked with the priority assignment rules (explained in Section 4.1) and sorted as valid and invalid priority assignments. For each invalid priority assignment, the back-end module notifies the users who initiated the priority assignment. The back-end module also checks the validity of the users added in the user priority list based on the specified time in the credential arrays. The back-end module automatically removes user with expired validity and updates the user priority table.

Device Policy List. The back-end module accumulates all the policies assigned by the users and creates a database based on the device ID. As explained in Section 4.1, the access policy language assigns a device ID to determine the intended policy for each device. This list is updated each time a user generates a new policy.

4.3 Policy Manager Module

The policy manager module collects the user priority list and device policy list from the back-end module and compares different user policies. This module consists of two sub-modules (policy negotiation module and policy generation module) to initiate the policy negotiation and generation processes.

Policy Negotiation Module and Negotiation Algorithm. The policy negotiation module compares all the user-defined policies and detects different types of conflicts based on user priorities and demands. Similar to traditional RBAC, Kratos uses assigned user roles and priorities to understand the user needs in a smart home hierarchy. However, a smart home environment needs a more fine-grained approach than RBAC to address the conflicting scenarios based on users' relationship, social norms, and personal preferences. To address these diverse needs, Kratos uses an automatic policy negotiation module to resolve conflicts in multi-user smart home environment. The policy negotiation module identifies types of conflicts based on user roles and priorities, categorizes the conflicts based on implemented policies, automatically decides whether a policy should be executed or not, starts a negotiation method between conflicting users using notification methods, and chooses an optimum operating point for both users upon mutual agreement. For policy negotiation, Kratos considers two essential research questions: (1) How does Kratos handle the policy conflicts between users with the same and different priority levels?, and (2) How does Kratos handle restriction policies without affecting smart home operations?, In the following, we address these questions.

The policy negotiation algorithm of Kratos processes all the policies and computes the negotiated results by modeling the users' authorities (classes, roles) in a multi-layer list. User authorities are split into ordered classes. Class 0 has the highest priority, and a higher class number means a lower priority. Each class may include a list of users (or roles as roles are just a set of users). Users at the same priority class shares the same priority. Kratos considers three types of conflicts (hard, soft, and restriction conflicts) between user policies after users are classified into authorities.

When two different policies include clauses of the same user's access for the same device, there can be an *interference* between those clauses. Any such possible interference is further checked to disclose the potential conflicts. In this, hard conflicts can happen when two interfering clauses dictate different actions for some overlapping cases or dictate the same action for never overlapping cased. In other words, when policies have no possible way of cooperation or compromising, Kratos detects a hard conflict. However, if the same action exists with some common overlap while opposite actions never occur together, such interference is a soft conflict. Moreover, hard and soft conflicts are further categorized as *Priority Conflicts* or *Competition Conflicts* based on the priority of policy owners. When the conflict happens between users' policies who have different priority classes, Kratos defines a *hard or soft priority conflict*. However, if the users have the same priority, *hard or soft competition conflicts* happens. For hard priority conflicts, Kratos enforces the policy defined by the user with higher priority. In hard competition conflicts, Kratos initiates a negotiation process between the users with an average operational condition calculated from both policies. If the users mutually agree with an average operational condition, Kratos creates a new policy for the targeted device. In case of no mutual operation condition, Kratos notifies the higher priority user/admin to resolve the dispute with a common policy. In the case of both soft priority and soft competition conflicts, the result of the negotiation process of Kratos is a new

clause with common set of conditions. If any interference is caused by the nature of action requested in two different policies, KRATOS detects a restriction conflict in the system. By incorporating these with hard, soft, and restriction conflicts, KRATOS overall implements five distinct conflict types (details in Appendix A and B).

Policy Generation Module. The goal of the policy generation module is to construct valid policies that reflect the demands and restrictions of all authorized users based on the device policies generated in the user interaction module. The generated policies are passed to the back-end module and stored in a database. Thereafter, these policies are enforced in smart devices. The negotiated policies computed by the policy negotiation algorithm are converted into enforceable access control rules. The negotiated policy clause, $\Psi = \{P, U, D, C, A\}$, has a 5-tuple format and is indeed well suited for existing attribute-based access control (ABAC) systems. Thus, KRATOS uses an ABAC-like enforcement for the final generated rules. Here, the policy, B, is the set of {action, subject, resource, constraints} tuples for a negotiated device policy. An example of mapping a sample policy to ABAC rule through a transformation function can be illustrated as follows:

$$\begin{aligned} ABAC(\Psi_i) \coloneqq \{B \mid \text{action}\,(B) = \mathrm{A_i} \wedge \text{subject}\,(B) = \mathrm{U_i} \\ \wedge\ \text{resources}\,(B) = \mathrm{D_i} \\ \wedge\ \text{constraints}(B) = \mathrm{T_{C_i}}\} \end{aligned} \tag{1}$$

where $T_C \coloneqq \{c \mid \text{c satisfies the same conditions of C in mapped attributes into ABAC policy. Here, } ABAC(\Psi_i) \text{ holds a direct translation of actions, subjects, resources, and constraints. We develop an ABAC-like rule generator that enforces the rules in a control device. The generator is integrated into the hub device as a unified enforcement point.}$

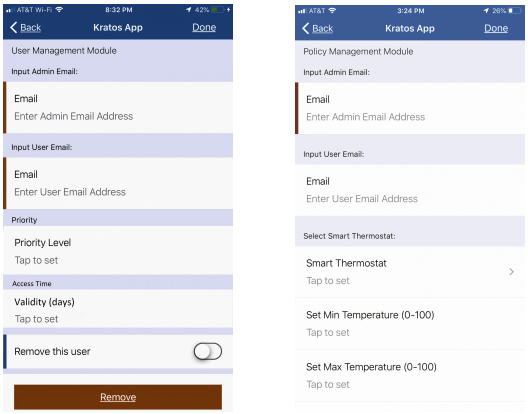
4.4 Policy Execution Module

Policy execution module enforces the final policies generated from the policy negotiation process. Smart home devices can be controlled through a mobile phone controller app or by installing different device-specific apps in the system (e.g., Samsung SmartThings). Policy execution process appends the generated policies in the smartphone controller app or the installed smart home apps. To append the policies, KRATOS adds conditional statements to the app source code to enforce the policies. When a user tries to change the state of the device, the app asks the policy execution module to check in the policy table generated by a policy generator. If an acceptable condition is matched, the policy execution engine returns the policy to the app and creates a binary decision (true for the accepted policy and false for the restricted policy) in the conditional branches. Based on the decision enforced by the policy execution engine, the user command in a smart home app is executed.

5 KRATOS IMPLEMENTATION

We implemented KRATOS in Samsung SmartThings platform which has the largest market share in consumer IoT, supports highest number of off-the-shelf smart home devices, and open-source apps [11].

Implementation and Data Collection. We setup a SHS to test the effectiveness of KRATOS. We used Samsung SmartThings hub and connected multiple smart devices and sensors to the hub. The complete list of devices in our SHS is provided in Appendix C. The setup included four different types of devices: smart light, smart lock, smart thermostat, and smart camera, which are some of the most common smart home devices used in SHSs [34]. We also used



(a) User Management **(b) Policy Management**

Figure 5: User interfaces of KRATOS.

three different types of sensors: motion, temperature, and contact sensors to provide autonomous control. Further, we collected data from 43 different smart home users. We selected the participants by conducting an institution-wide open call for participation and flyers for community outreach. We obtained the necessary Institutional Review Board approval for collecting data from real-life smart home users. While selecting participants for our study, we considered several features: (1) owns more than one smart home devices, (2) shares smart home environment with multiple users (e.g., parents, partners, friends, or housemates), (3) diverse user roles (working adults, housewives, young adults, student, etc.), and (4) beginner level knowledge on using smart home devices. The participants were grouped into 14 different groups and asked to choose their roles in a SHS. First, we recorded different conflicting scenarios experienced by the users and asked them to use KRATOS to assign device policies. We investigated several multi-user scenarios for the policy generation and negotiation processes as detailed below:

Scenario 1: Multiple policies for the same device. We selected common devices (e.g., smart thermostat) and enforced different policies set by multiple users. Users assigned demand and restriction policies in the system for the same device. We collected 44 sets of policies (a set of policy includes at least two policies from multiple users) which included 13 hard, 17 soft, and 8 restriction conflicts.

Scenario 2: Multiple policies for different devices. We used multiple devices from the same device category (e.g., smart light, smart lock, smart thermostat) to enforce different policies over the same type of devices. Here, we collected 48 sets of policies from 43 users which resulted in 15 hard, 22 soft, and five restriction conflicts.

Scenario 3: Multiple apps for the same device. In the SHS, we allowed users to install different apps to control the same device (e.g., smart light). For example, multiple users can configure a smart light with both motion and door sensors using different apps. We chose three different smart light apps available in SmartThings marketplace (light control with motion sensor, door sensor, and luminance level, respectively) and asked the users to install preferable apps and assign device policies accordingly. Here, we collected 35 sets of policies including 8 hard, 18 soft, and five restriction conflicts.

Scenario 4: Single app for multiple devices. We considered an individual app controlling multiple same types of devices in the SHS. We chose a single light controlling app to control four different lights and asked users to enforce device policies in different devices using

one single app. We collected 32 sets of policies in this scenario which includes 12 hard, 15 soft, and 3 restriction conflicts.

Scenario 5: Temporary users in the system. We considered a temporary user is added in the system and trying to access a smart light and smart lock after the access is expired for that specific user. We collected 30 sets of policies in this scenario.

Scenario 6: Location-based access in the system. In the location-based access control, we allowed multiple users to set location-based policies for a smart thermostat. Here, users are allowed to define both location-based restriction and demand policies. We collected 30 sets of policies in this scenario.

Malicious scenarios. We also implemented five real-life threats in our SHS to generate malicious data and further evaluate the effectiveness of Kratos (more details in Section 6) . For Threat-1 (Over privileged controls), we asked the users to add restriction clauses to the smart thermostat and asked the restricted users to change the temperature. For Threat-2 (Privilege abuse), we asked a newly added user with lower priority to install a new app in the smart home and trigger a smart camera. Threat-3 (Privilege escalation) is presented by a scenario where a new user changed the lock code of a smart lock and removed the smart lock from the environment. For Threat-4 (Unauthorized access), we added a temporary authorized user with limited priority and asked the users to control a smart thermostat outside their accepted time range. For Threat-5 (Transitive privilege), we asked the user with lower priority to add a new user with higher priority in the system.

User Interface. We built a SmartThings app that represents the user interaction module described in Section 4. This app has two modules: user management and policy management. The user management module allows users to add new users and assign priorities. We define five different roles and priority levels in Kratos (i.e., father/owner - priority 0, mother/owner - priority 0, adult - priority 1, guest - priority 2, child - priority 3). These roles and priorities can be assigned by the smart home owner or by authorized users with the same or higher priority to the one being assigned. Upon created a new role/priority, the information is sent and stored in the backend server. In the policy management module, users select devices and create new policies. Kratos provides options to add either general device policies (intended for all existing users) or policies that apply only to specific users. Kratos allows users to use different device conditions (operation-based, time-based, value-based, etc.) to define the policies. As our implementation environment had devices that only allows time-based and value-based conditions, we classified the policies in three different possible categories: (1) time-based device policy, (2) value-based device policy, and (3) time-value-based policy. The policies for different devices in our implementation can be represented by a device policy array: Device Policy, P = $\{U, D, C_1, C_2, R\}$.

- *User ID (U):* The first element of the policy array is to identify the policy assignee. We utilized the user email as a personal identifier in our implementation.
- *Device ID (D):* SmartThings assigns a unique device ID for each installed device which was used for the devices and policies.
- *Time conditions (C_1):* Users could assign a start time and an end time for any device action in the policies. For example, a smart light can be accessed from sunset to sunrise only.
- *Value conditions (C_2):* Users could assign a maximum and minimum value to specify an acceptable range to control a device functionality. For example, a user can set the operational range of a thermostat from 68°F to 70°F.
- *Restricted User (R):* High-priority Users could define the restriction policy for a specific lower-priority user by adding the user ID to the restricted user's list. Users could also assign general policies (Section 2.2) for the devices by assigning '0' in this field.

Figure 5 shows the user interface of Kratos. We implemented Kratos as a customized smart app in Samsung SmartThings platform. We built the Kratos app in *Groovy* platform and installed the app using SmartThings web interface. As Samsung allows each users to install customized apps in same smart environment using the web interface, Kratos app can be easily installed in each user's controller device in multi-user smart environment. Each user can use official SmartThings app in the controller device to use Kratos app to asign new users and device policies. The information of new users and device policies are forwarded to the policy generator via the backend server for generating final device policies.

Policy Enforcement. The final step during implementation is to enforce the generated policies by Kratos. We utilized 10 different official SmartThings apps that control 17 different devices and installed them in the SHS. We installed all the apps and observed the user-specific policies generated in the policy generation module. We modified these apps to connect with the backend server and capture the generated policies from the policy generator. These policies were appended to the conditional statements inside the app to execute the policies. A sample modified app is given in Appendix D to illustrate the steps to enforce policies in a SmartThings app.

6 PERFORMANCE EVALUATION

We evaluate Kratos by focusing on the following research questions:

RQ1 How effective is Kratos in enforcing access control in multi-user scenarios while handling different threat models? (Sec 6.1)

RQ2 What is the overhead introduced by Kratos on the normal operations of the SHS? (Sec. 6.2)

6.1 Effectiveness

In this sub-section, we present the experimental results of Kratos while enforcing access control in different multi-user smart home scenarios and threat models. We first considered a use case scenario to explain the results of Kratos in different smart home operations. Then, we considered six different utilization scenarios (explained in Section 5) to evaluate the effectiveness of Kratos.

To understand the performance of Kratos, we assume two users Alice and Bob using the same smart thermostat and assigning different policies according to their needs. This usage scenario may lead to conflicts in which case Kratos uses policy negotiation module to solve the conflicts. For instance, let us assume Alice and Bob has the same priority level which is 2 and assign temperature range 60-70 and 75-80 respectively. Kratos considers this as a hard competition conflict and starts the negotiation process with average range 67-75. If Alice and Bob both agree with the range, Kratos generates a new policy for the thermostat with the temperature range 67-75 and enforces this in the device. On the other hand, if Alice and Bob cannot agree, Kratos notifies a higher level user/admin to resolve this conflict by assigning a new policy for the device. We also consider a temporary user scenario in evaluating Kratos where Alice (priority-1) adds a temporary user Gary (priority-4) in the system for 2 days. After the validity period (2 days), Gary tries

Conflict type	Policy example	Kratos outcome
Hard priority conflict	Alice (priority-1) and Bob (priority-2) set up the temperature range 60-70 and 75-80 respectively in the smart thermostat.	As Alice has higher priority, Kratos sets the thermostat to 60-70 and notifies the users with the decision
Soft priority conflict	Alice (priority-1) and Bob (priority-2) set up the temperature range 60-70 and 65-75 respectively in the smart thermostat.	• As Alice has the higher priority, Kratos sets the thermostat to 60-70 and notifies Alice with common range (65-70). • If Alice agrees with common range, Kratos sets the temperature range 65-70.
Hard competition conflict	Alice (priority-2) and Bob (priority-2) set up the temperature range 60-70 and 75-80 respectively in the smart thermostat.	• Kratos starts the negotiation with average range (67-75) and upon mutual agreement from the users set the range. • If the users fail to agree, Kratos notifies higher level user/admin to decide the policies.
Soft competition conflict	Alice (priority-2) and Bob (priority-2) set up the temperature range 60-70 and 65-75 respectively in the smart thermostat.	Kratos sets the temperature range 65-70 and notifies the users with updated policy.
Restriction conflict	Alice (priority-1) set the temperature range 60-70 and restrict Bob (priority-2) to change the thermostat. Bob sets the temperature range 75-80.	Kratos sets the temperature range 60-70 and notifies Bob regarding restriction.
Temporary access	Alice (priority-1) added Gary (priority- 4) as a temporary user for 2 days. After 2 days, Gary tries to unlock the smart lock.	Kratos automatically detects the expired validity for smart home access and deletes Gary from authorized user list to prevent any undesired access.
Location-based access	Alice (priority-1) set up the temperature range 70-72 and restrict Kyle (priority-3) from using the smart thermostat remotely. Kyle sets the temperature range 74-76.	• If Kyle is not in the home network, Kratos disregard Kyle's access policy. • Kratos checks the location of both Kyle and Alice. If only Kyle is home, Kratos sets the temperature range 74-76. If both Kyle and Alice are home, Kratos sets the temperature range 70-72.

Table 1: Different usage scenarios and outcomes of Kratos.

to access the smart home devices. However, Kratos automatically detects any expired validity of the users in the system and restricts the temporary users to access the system. Table 1 summarizes the outcome of Kratos in different usage scenarios. Table 2 also shows the summary of policy conflicts and negotiations between smart home users in different multi-user scenarios explained in Section 5. In Scenario-1, Kratos successfully negotiated 44 sets of policies collected from 43 users and executed the generated policies in the SHS. Average policy generation time including the policy negotiation was 0.68 seconds. In Scenario-2, Kratos evaluated 48 sets of policies in total with an average policy generation time of 1.2 seconds. In Scenario-3 and 4, Kratos manages 35 and 32 sets of policies with an average generation time of 0.86 and 0.48 seconds respectively. In Scenario-5, Kratos successfully manages 20 sets of policies and automatically detects unauthorized access for expired temporary access. For location-based access in Scenario-6, Kratos successfully manages 30 sets of policies and provides location-based acess to multiple users. Kratos also successfully resolves all the conflicts generated in different scenarios. In summary, Kratos successfully resolved the policy conflicts and created optimized final policies that could be executed within different smart home apps.

We also evaluated the effectiveness of Kratos in preventing different threats in the SHS. We considered five different threats presented in Section 5. We collected data from fifty malicious occurrences in total to evaluate Kratos against these threats. Table 3 summarizes the performance of Kratos in identifying different threats. In each of these scenarios, Kratos detected the policy violation with 100% accuracy and effectively notified the smart homeowner/policy assigner via push notifications. For Threat-1, Kratos achieves the lowest average detection and notification time 0.25 and 0.4 seconds respectively. To identify Threat-2 and 3, Kratos takes 0.4 and 0.47 seconds on average with average notification time 0.6 seconds. For Threat-4 and 5, the average detection time is 0.35 and 0.28 seconds, respectively. In summary, Kratos can detect different threats with 100% accuracy and notify users with minimum delay.

Usage Scenario	No. of policies	No. of hard conflicts	No. of soft conflicts	Restriction policies	No conflicts	Average time (s)	Success rate (s)
Scenario-1	44	13	17	8	6	0.68	100%
Scenario-2	48	15	22	5	6	1.2	100%
Scenario-3	35	8	18	5	4	0.86	100%
Scenario-4	32	12	15	3	2	0.48	100%
Scenario-5	30	-	12	6	12	0.2	100%
Scenario-6	30	10	8	8	4	0.32	100%

Table 2: Kratos's performance in different scenarios.

Threat model	No. of occurances	Success rate	Average Detection time (s)	Average Notification time (s)
Threat-1	10	100%	0.25	0.4
Threat-2	10	100%	0.4	0.6
Threat-3	10	100%	0.47	0.6
Threat-4	10	100%	0.35	0.52
Threat-5	10	100%	0.28	0.45

Table 3: Performance of Kratos against different threats.

6.2 Performance Overhead

We considered the following research questions to measure the performance overhead of Kratos:

RQ3 What is the impact of Kratos in normal operations of the SHS? (Table 4)

RQ4 What is the impact of Kratos in executing a user command in the SHS via the smart home apps? (Table 5)

RQ5 How does the impact of Kratos change with different parameters in the SHS? (Figure 6)

For different multi-user scenarios, we considered four different scenarios as explained in Section 5.

Latency Introduced by Kratos. Kratos considers three different types of conflicts (hard conflicts, soft conflicts, and restriction policy) during policy generation and negotiation based on user priorities and policy types. These policy generation and negotiation processes normally introduce latency in the normal operations of a SHS and the smart apps to analyze given policies and solving conflicts. Table 4 illustrates the delay introduced by Kratos while handling the policy conflicts and negotiations. We note that the average negotiation time increases with the number of policies for all types of policy conflicts. For hard conflicts, the average negotiation time is 0.403 seconds for ten policies, which increases to 1.21 seconds for 30 policies. Because the hard conflicts require all the conflicted users to interact with the system to resolve the conflicts, it takes more time than soft conflict and restriction policies. For soft conflicts, the average negotiation time is 0.27 seconds for ten policies which increases to 0.73 seconds for 30 policies. For the restriction policies, the latency is introduced only when a low-priority user tries to assign policies to high-priority users. In this case, average negotiation times vary from 0.102 seconds to 0.25 seconds from 10 to 30 policies.

Conflict types	No. of Policies	Average negotiation time (s)
Hard conflict	10	0.403
	20	0.715
	30	1.21
Soft conflict	10	0.27
	20	0.53
	30	0.73
Restriction Policy	10	0.102
	20	0.117
	30	0.25

Table 4: Overhead of Kratos in handling policy negotiations.

Impact of Kratos on Executing User Commands. As the policies in Kratos are enforced in the smart apps installed via the controller device (e.g., smartphone and smart tablet), it introduces overhead in the controller devices while installing the apps and executing users' command. Table 5 depicts the impact of Kratos on executing user commands based on generated policy. Here, we

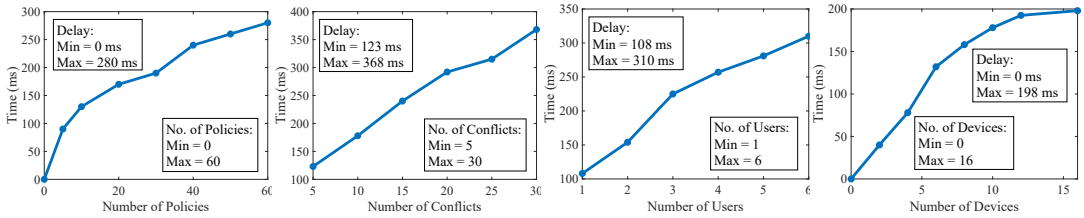


Figure 6: Impact of different evaluation parameters on Kratos's performance: (a) number of policies, (b) number of conflicts, (c) number of users, and (d) number of devices.

used eight different apps to measure the performance overhead of Kratos. We also considered three types of constraints on the policies: time constraint, value constraint, and both time and value constraints. Time constraint refers to the specific time range for the desired action of a smart device (e.g., turning on lights at sunset) while value constraint refers to the specific range of inputs to a smart device (e.g., the temperature of the smart thermostat). With no policy enforced on a device, the average time to install an app and execute user command is 1.3 seconds with 1.75% and 1.6% of CPU and RAM utilization, respectively. For time constraints and value constraints, the average time is 1.72 and 1.46 seconds, respectively. Average CPU and RAM utilization are almost similar for both time and value constraints (2.1-2.2% and 2.25-2.6%, respectively). For both time and value constraints, the average execution time increases to 1.92 seconds. The CPU and RAM utilization also increases to 2.5% and 2.82%, respectively. Considering the CPU and RAM available in modern smartphones and tablets, the overhead introduced by Kratos can be considered negligible [29–31].

Type of policy	Avg. time (s)	Avg. CPU usage	Avg. RAM usage
No policy	1.3	1.75%	1.6%
Time constraint	1.72	2.2%	2.6%
Value constraint	1.46	2.1%	2.25%
Time and Value constraint	1.92	2.5%	2.82%

Table 5: Overhead of Kratos in policy executions.

Impact of Different Parameters on Performance Overhead. Kratos considers different parameters in SHSs to define and execute device policies reflecting diverse user demands. Here, we observed the performance overhead of Kratos by changing various parameters. As policy generation and negotiation are executed at the backend server, Kratos does not pose any performance overhead to computational parameters (CPU and RAM utilization). The only noticeable change is observed in delay imposed by Kratos in the normal operation of the SHS. In Figure 6, the delay introduced by Kratos is shown based on the number of policies, conflicts, users, and devices. One can notice from Figure 6a, the delay introduced by Kratos increases with the number of policies generated by the users. Kratos introduces 90 ms delay in the SHS for five policies to execute a user command which increases to 280 ms delay for 60 policies. The delay increases linearly with the number of conflicts and users in the system (Figure 6b and Figure 6c). The highest delay to execute a user command is 368 ms, which occurs when the system includes 30 different policy conflicts. Kratos also takes 310 ms to execute a command with six different users presents in the system. This delay is the result of the overhead introduced by notifying different users about executing the command. For the number of devices, the delay introduced by Kratos becomes steady after adding 12 different devices in the SHS (Figure 6d).

7 BENEFITS AND FUTURE WORK

7.1 Benefits of Kratos

Consider a user, Bob, who defines himself as a technology savvy and enthusiastic entrepreneur homeowner. Bos's house is set with devices such as smart lock, thermostat, and fire alarm. Bob's is the head of a family of three members, including his wife Alice, and his teenager son Matt. Finally, Bob offers high-quality vacation rentals to Airbnb users.

Efficient Conflict Resolution. With several devices shared among all household members (including the Airbnb tenant), Bob feels that there is an immediate need for some control mechanism that defines how all the smart devices are being set up and managed among the different users. However, despite trying devices and smart apps from different platforms (e.g., Samsung SmartThings, Google Home, etc.), Bob cannot find a feasible and user-friendly solution that consider the needs of the different users (e.g., Bob and Alice's priority is to keep the thermostat temperature as high as possible while Matt's idea is to have cooler temperature). Kratos offers a fine-grained access control mechanism for the SHS that allows Bob to provide access control based on the users' needs, demands, and priorities.

Multi-users/Multi-devices. As mentioned before, Bob's setup comprises several devices with different levels of usability based on their impact on the quality of life of users and their contribution to the general security of the household. Additionally, users may have different levels of access based on Bob's and the household's best interests. Based on these scenarios, Bob expects an smart home access control system capable of managing multi-user and multi-device environments. Kratos realizes and offers an access control system where the administrator (i.e., Bob) can assign priority levels to the different devices and users. This allows control mechanisms to consider the importance of the various devices, but also the needs of the users based on admin's pre-defined priorities.

Suitability for Complex User Demands. Users' demands can be very complex at times. For instance, in addition to the demands and interests of Bob, Alice, and Matt, new access control policies can be generated in case Bob decides to give some control to his Airbnb tenant Ed. Adding new users and devices to an already configured SHS increase the complexity due to new conflicts between users and device policies. To solve these issues, Kratos can actively analyze and solve policy conflicts through negotiations in an optimized fashion based on the different user and device priorities.

Inherent Security. Bob has certain rules to protect his smart home ecosystem. First, security-related devices (e.g., smart lock) have the highest priority. Second, he would like to have strict and unique control over these devices, so no other user can change their settings or expected behaviors. Finally, users with the lowest priority (e.g.,

Prior Work	Domain	Multi-user Multi-device environment	A.C. Threat Model	User Interface	Conflict resolution	Overhead analysis	A. C. Language
xShare [17]	Smartphone	○	○	●	○	●	○
DiffUser [21]	Smartphone	○	○	●	○	●	○
Capability-based A. C. [12]	IoT network	●	○	●	○	○	○
Situation-based A. C. [26]	Smart home	●	○	○	○	○	●
Expat [38]	Smart home	●	○	○	○	●	●
Zeng et al. [41]	Smart home	●	○	●	○	○	○
KRATOS	Smart home	●	●	●	●	●	●

Table 6: Comparison between KRATOS and other access control mechanisms (A.C. stands for Access Control).

Ed) should not be able to add new devices, change SHS settings, etc. Our framework was designed to provide inherent security based on the specific user's needs. Specifically, KRATOS offers the means to provide complex control and demands through comprehensive policy negotiation and conflict resolution.

7.2 Future Work

User and Usability Study. To understand the users' access control needs in a smart home environment, we will conduct a detailed user study considering user characterization, device sage patterns, smart home configuration references, and user preferences Also, as KRATOS is a user-centric access control solution, usability of KRATOS should be tested in a real-life smart home environment. We will conduct a usability study among smart home users to test KRATOS based on different parameters such as user interface, acceptability of use, availability, user friendliness, notification system, and effectiveness. We will also develop tutorials and detailed user guides to assist the participants (both experienced and inexperienced users) to properly evaluate the usability of KRATOS in the real-life smart home.

8 RELATED WORK

Rather than providing fine-grained user access control, most of the prior works emphasize on limiting malicious activities via controlling app access [7, 8]. Moreover, several works focus on device access control and authentication on an IoT network for single-user scenarios [1, 6, 14, 22, 24]. In a recent work, He et al. present a detailed smart home user study that portrays users' concerns of fine-grained access control in multi-user smart environments [13]. Here, authors conducted the user study among 425 smart home users and outlines access control needs based on users preferences, social norms, and mutual relationships. Zeng et al. discuss their findings related to security and privacy concerns among smart home users [39]. Authors selected 15 smart home users and outlined their security and privacy concerns and summarizes users' actions in security-affected scenarios. In both works, smart home users raise their concerns regarding the need of access control mechanism in SHS. In addition, these studies also summarize design specifications to reflect users' needs in an access control mechanism. Matthews et al., also points out relevant issues with smart home users that share the same devices and accounts [19]. However, no explicit solution for multi-user access is proposed in any of these works.

In other works, researchers explore different access control strategies when multiple users share a single IoT device. Liu et al. suggested a user access framework for the mobile phone ecosystem called *xShare*, which provides policy enforcement on file level accesses [17]. Ni et al. presented *DiffUser*, a user access control model for the Android environment based on access privileges [21], which is only effective for a single device. Tyagi et al. discussed several design specification needed for multi-party access control in a shared environment [37]. Aside from these works, there are few prior works proposing access control systems for multi-user multi-device SHS. Gusmeroli et al. suggested a capability-based access control for users in a multi-device environment [12]. However, this system is not flexible enough to express the real needs of the users. Jang et al. presented a set of design specification for access control mechanism based on different use scenarios of multi-user SHS [15]. Schuster et al. proposed a situation-based access control in the SHS which considers different environmental parameters [26]. Here, the authors considered state of the device along with the location of the users to determine a valid access request. However, this work does not solve the conflicting demands of multiple users. Yahyazadeh et al. presented *Expat*, a policy language to define policies based on user demands [38]. In a recent work, Zeng et al. built an access control prototype with different access control options for smart home users [41]. Here, the authors considered four different access control mechanisms and assessed in a month-long user study among seven households to understand the users' needs and improve the design. Although authors built a proof-of-concept framework to perform a detailed user study and outlined the access control needs in smart home, they did not implement the framework in real-life systems and did not consider user conflicts while operating in a multi-user smart environment.

Differences from existing works. KRATOS was built upon considering prior user studies [13, 41]. KRATOS presents an access control system designed for multi-device multi-user smart home systems that provides a fine-grained access control to the users considering (1) easy new user addition with priority levels, (2) device restrictions for specific users, (3) automatic policy negotiation for conflicts, (4) easy policy assignment for multiple users, (5) different threats arising from over-privileged users, (6) Real-life implementation in smart home platform, (7) Effectiveness evaluation with real-life users, and (8) minimum overhead in real-life deployment. Table 6 summarizes the differences of KRATOS from other existing solutions.

9 CONCLUSION

In smart home systems, multiple users have access to multiple devices simultaneously. In these settings, users may want to control and configure the devices with different preferences which give rise to complex and conflicting demands. In this paper, we proposed KRATOS, an access control system that addresses the diverse and conflicting demands of different users in a shared multi-user smart home system. KRATOS implements a priority-based policy negotiation technique to resolve conflicting user demands in a shared smart home system. We implemented KRATOS on real settings with multiple users and evaluated its performance via real devices. KRATOS successfully covers the users' needs, and our extensive evaluations showed that KRATOS is effective in resolving the conflicting requests and enforcing the policies without significant overhead. Also, we tested KRATOS against five different threats and found that KRATOS effectively identifies the threats with high accuracy.

10 ACKNOWLEDGMENT

This work is partially supported by the US National Science Foundation (Awards: NSF-CAREER-CNS-1453647, NSF-1663051, NSF-1705135), US Office of Naval Research grant Cyberphysical Systems, and Cyber Florida's Capacity Building Program. The views expressed are those of the authors only, not of the funding agencies.

REFERENCES

[1] Ioannis Agadakos, Per Hallgren, Dimitrios Damopoulos, Andrei Sabelfeld, and Georgios Portokalidis. 2016. Location-enhanced Authentication Using the IoT: Because You Cannot Be in Two Places at Once. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications*. ACM.

[2] Leonardo Babun, Amit Kumar Sikder, Abbas Acar, and A Selcuk Uluagac. 2018. IoTDots: A Digital Forensics Framework for Smart Environments. *arXiv preprint arXiv:1809.00745* (2018).

[3] Z. Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick McDaniel, and A. Selcuk Uluagac. 2018. Sensitive Information Tracking in Commodity IoT. In *27th USENIX Security Symposium*. Baltimore, MD.

[4] Z. Berkay Celik, Patrick McDaniel, and Gang Tan. 2018. Soteria: Automated IoT Safety and Security Analysis. In *USENIX Annual Technical Conference (USENIX ATC)*.

[5] Z. B. Celik, P. McDaniel, G. Tan, L. Babun, and A. S. Uluagac. 2019. Verifying Internet of Things Safety and Security in Physical Spaces. *IEEE Security Privacy* 17, 5 (Sep. 2019), 30–37. https://doi.org/10.1109/MSEC.2019.2911511

[6] S. Cirani, M. Picone, P. Gonizzi, L. Veltri, and G. Ferrari. 2015. IoT-OAS: An OAuth-Based Authorization Service Architecture for Secure Services in IoT Scenarios. *IEEE Sensors Journal* 15, 2 (Feb 2015), 1224–1234.

[7] Soteris Demetriou, Nan Zhang, Yeonjoon Lee, XiaoFeng Wang, Carl A Gunter, Xiaoyong Zhou, and Michael Grace. 2017. HanGuard: SDN-driven protection of smart home WiFi devices from malicious mobile apps. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*.

[8] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. 2016. Flowfence: Practical data protection for emerging iot application frameworks. In *25th {USENIX} Security Symposium*.

[9] Christine Geeng and Franziska Roesner. 2019. Who's In Control? Interactions In Multi-User Smart Homes. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–13.

[10] Chaowen Guan, Aziz Mohaisen, Zhi Sun, Lu Su, Kui Ren, and Yaling Yang. 2017. When smart tv meets crn: Privacy-preserving fine-grained spectrum access. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1105–1115.

[11] Rachel Gunter. 2017. Making Sense of Samsung's SmartThings Initiative. (2017). https://marketrealist.com/2017/12/making-sense-samsungs-smartthings-initiative

[12] Sergio Gusmeroli, Salvatore Piccione, and Domenico Rotondi. 2013. A capability-based security approach to manage access control in the Internet of Things. *Mathematical and Computer Modelling* 58, 5 (2013), 1189 – 1205.

[13] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. 2018. Rethinking Access Control and Authentication for the Home Internet of Things (IoT). In *27th USENIX Security Symposium*. Baltimore, MD.

[14] Maia Jacobs, Henriette Cramer, and Louise Barkhuus. 2016. Caring About Sharing: Couples' Practices in Single User Device Access. In *Proceedings of the 19th International Conference on Supporting Group Work*. ACM.

[15] William Jang, Adil Chhabra, and Aarathi Prasad. 2017. Enabling Multi-user Controls in Smart Home Devices. In *Proceedings of the Workshop on Internet of Things Security and Privacy*. ACM.

[16] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Z Morley Mao, Atul Prakash, and Shanghai JiaoTong Unviersity. 2017. ContexIoT: Towards providing contextual integrity to appified IoT platforms. In *Proceedings of The Network and Distributed System Security Symposium*.

[17] Yunxin Liu, Ahmad Rahmati, Yuanhe Huang, Hyukjae Jang, Lin Zhong, Yongguang Zhang, and Shensheng Zhang. 2009. xShare: supporting impromptu sharing of mobile phones. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*. ACM.

[18] August Smart Lock. 2018. How August Smart Lock Works? (2018). https://august.com/pages/how-it-works

[19] Tara Matthews, Kerwell Liao, Anna Turner, Marianne Berkovich, Robert Reeder, and Sunny Consolvo. 2016. "She'll Just Grab Any Device That's Closer": A Study of Everyday Device & Account Sharing in Households. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. ACM.

[20] AKM Iqtidar Newaz, Amit Kumar Sikder, Mohammad Ashiqur Rahman, and A Selcuk Uluagac. 2019. Healthguard: A machine learning-based security framework for smart healthcare systems. In *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. IEEE, 389–396.

[21] Xudong Ni, Zhimin Yang, Xiaole Bai, Adam C Champion, and Dong Xuan. 2009. DiffUser: Differentiated user access control on smartphones. In *6th International Conference on Mobile Adhoc and Sensor Systems*. IEEE.

[22] Sarah Rajtmajer, Anna Squicciarini, Jose M Such, Justin Semonsen, and Andrew Belmonte. 2017. An Ultimatum Game Model for the Evolution of Privacy in Jointly Managed Content. In *International Conference on Decision and Game Theory for Security*. Springer, 112–130.

[23] RemoteLock. 2018. Smart Locks by RemoteLock. (2018). https://www.remotelock.com/smart-locks

[24] H. Ren, Y. Song, S. Yang, and F. Situ. 2016. Secure smart home: A voiceprint and internet based authentication system for remote accessing. In *2016 11th International Conference on Computer Science Education (ICCSE)*. 247–251.

[25] Samsung. 2018. How do I share my Location and manage users in SmartThings Classic? (2018). https://tinyurl.com/y86unolb

[26] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Situational Access Control in the Internet of Things. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1056–1073.

[27] Nicholas Shields. 2017. THE US SMART HOME MARKET REPORT: Systems, apps, and devices leading to home automation. http://www.businessinsider.com/the-us-smart-home-market-report-systems-apps-and-devices-leading-to-home-automation-2017-4. (2017). [Online; accessed 9-November-2017].

[28] Amit Kumar Sikder, Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, Kemal Akkaya, and Mauro Conti. 2018. IoT-enabled smart lighting systems for smart cities. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 639–645.

[29] Amit Kumar Sikder, Hidayet Aksu, and A. Selcuk Uluagac. 2017. 6thSense: A Context-aware Sensor-based Attack Detector for Smart Devices. In *26th USENIX Security Symposium*. Vancouver, BC.

[30] Amit Kumar Sikder, Hidayet Aksu, and A Selcuk Uluagac. 2019. A Context-aware Framework for Detecting Sensor-based Threats on Smart Devices. *IEEE Transactions on Mobile Computing* (2019).

[31] Amit Kumar Sikder, Hidayet Aksu, and A Selcuk Uluagac. 2019. Context-aware intrusion detection method for smart devices with sensors. (Sept. 17 2019). US Patent 10,417,413.

[32] Amit Kumar Sikder, Leonardo Babun, Hidayet Aksu, and A Selcuk Uluagac. 2019. Aegis: a context-aware security framework for smart home systems. In *Proceedings of the 35th Annual Computer Security Applications Conference*. 28–41.

[33] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, and A Selcuk Uluagac. 2018. A survey on sensor-based threats to internet-of-things (iot) devices and applications. *arXiv preprint arXiv:1802.02041* (2018).

[34] Statista. 2017. Ownership of smart home technology products in the United States in 2017 (in million households/units in use), by category. (2017). https://www.statista.com/statistics/757684/smart-home-technology-product-ownership-in-the-us-by-category/

[35] Trefis Team. 2017. Why Smart Home Devices Are A Strong Growth Opportunity For Best Buy. (2017). https://www.forbes.com/sites/greatspeculations/2017/07/05/why-smart-home-devices-are-a-strong-growth-opportunity-for-best-buy/2bbe77114984

[36] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, Xianzheng Guo, and Patrick Tague. 2017. SmartAuth: User-Centered Authorization for the Internet of Things. In *26th USENIX Security Symposium*. Vancouver, BC.

[37] Alpana Tyagi, Anna Squicciarini, Sarah Rajtmajer, and Christopher Griffin. 2016. An in-depth study of peer influence on collective decision making for multi-party access control. In *17th International Conference on Information Reuse and Integration (IRI)*. IEEE, 305–314.

[38] Moosa Yahyazadeh, Proyash Podder, Endadul Hoque, and Omar Chowdhury. 2019. Expat: Expectation-based Policy Analysis and Enforcement for Appified Smart-Home Platforms. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*. 61–72.

[39] Eric Zeng, Shrirang Mare, and Franziska Roesner. 2017. End User Security and Privacy Concerns with Smart Homes. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. Santa Clara, CA.

[40] Eric Zeng, Shrirang Mare, and Franziska Roesner. 2017. End User Security and Privacy Concerns with Smart Homes. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. Santa Clara, CA.

[41] Eric Zeng and Franziska Roesner. 2019. Understanding and Improving Security and Privacy in Multi-User Smart Homes: A Design Exploration and In-Home User Study. In *28th {USENIX} Security Symposium*.

A POLICY NEGOTIATION ALGORITHM

During policy negotiation, each policy clause is compiled into a quintuple, $\Psi = \{P, U, D, C, A\}$, where P is the policy assigner (that shows who states this clause), U is the assignee (about whom this statement is), D is the targeted smart device, C is a set of conditions over D and U, and configurable environmental attributes, and finally $\mathcal{A} \in \{demand, restrict\}$ is the action requested by this statement when the set of conditions are satisfied. Kratos implements an algorithm to solve the policy conflicts represented as follows:

$$interfere(\Psi_i, \Psi_j) \leftarrow U_i = U_j \wedge D_i = D_j \tag{1}$$

$$\begin{aligned} hard\_conflict(\Psi_i, \Psi_j) \leftarrow interfere(\Psi_i, \Psi_j) \wedge (\\ (A_i \neq A_j \ \wedge \forall c \in C_i \cap C_j : \Theta(\mathcal{V}(c, C_i), \mathcal{V}(c, C_j))) \\ \vee (A_i = A_j \ \wedge \exists c \in C_i \cap C_j : \neg\Theta(\mathcal{V}(c, C_i), \mathcal{V}(c, C_j)))) \end{aligned} \tag{2}$$

$$\begin{aligned} soft\_conflict(\Psi_i, \Psi_j) \leftarrow interfere(\Psi_i, \Psi_j) \wedge (\\ (A_i = A_j \ \wedge \forall c \in C_i \cap C_j : \Theta(\mathcal{V}(c, C_i), \mathcal{V}(c, C_j))) \\ \vee (A_i \neq A_j \ \wedge \exists c \in C_i \cap C_j : \mathcal{V}(c, C_i) \neq \mathcal{V}(c, C_j))) \end{aligned} \tag{3}$$

$$HPC(\Psi_i, \Psi_j) \leftarrow hard\_conflict(\Psi_i, \Psi_j) \wedge \Xi(P_i) \neq \Xi(P_j) \quad (4)$$

$$SPC(\Psi_i, \Psi_j) \leftarrow soft\_conflict(\Psi_i, \Psi_j) \wedge \Xi(P_i) \neq \Xi(P_j) \quad (5)$$

$$HCC(\Psi_i, \Psi_j) \leftarrow hard\_conflict(\Psi_i, \Psi_j) \wedge \Xi(P_i) = \Xi(P_j) \quad (6)$$

$$SCC(\Psi_i, \Psi_j) \leftarrow soft\_conflict(\Psi_i, \Psi_j) \wedge \Xi(P_i) = \Xi(P_j) \quad (7)$$

$$RC(\Psi_i, \psi_j) \leftarrow Restriction\_conflict(\Psi_i, \psi_j) \wedge \Xi(P_i) > \Xi(P_j) \wedge A_i = restrict \quad (8)$$

where Ψ_i, Ψ_j is the evaluated pair of policies, and $\mathcal{V}(c, C)$ is the value function that returns the value of conditional c in the set C, $\Theta(x, y)$ checks the overlap between the provided (x, y) tuple and $\Xi(u)$ returns the priority of user u as the value of user's assigned priority class.

B POLICY NEGOTIATION PROCESS

The negotiation $\mathcal{N}$ between two given policy clauses (Ψ_i, Ψ_j) can be formally expressed and computed by Equation 9.

$$\mathcal{N}(\Psi_i, \Psi_j) = \begin{cases} \begin{cases} \Psi_i & \text{if } \Xi(P_i) > \Xi(P_j) \\ \Psi_j & \text{otherwise} \end{cases}, & \text{if } HPC(\Psi_i, \Psi_j) \\ \begin{cases} \{P_i \cup P_j, U_i, D_i, C_i \cup C_j, A_i\} & \text{if } A_i = A_j \\ \{P_i \cup P_j, U_i, D_i, C_i \cup \neg C_j, A_i\} & \text{otherwise} \end{cases}, & \text{if } SPC(\Psi_i, \Psi_j) \\ \begin{cases} majority\_vote(\Psi_i, \Psi_j) & \text{if } binary(D_i) \\ arbitrate(\Psi_i, \Psi_j) & \text{otherwise} \end{cases}, & \text{if } HCC(\Psi_i, \Psi_j) \\ \begin{cases} \{P_i \cup P_j, U_i, D_i, C_i \cup C_j, A_i\} & \text{if } A_i = A_j \\ \{P_i \cup P_j, U_i, D_i, C_i \cup \neg C_j, A_i\} & \text{otherwise} \end{cases}, & \text{if } SCC(\Psi_i, \Psi_j) \end{cases} \quad (9)$$

In the case of a hard priority conflict (HPC), (e.g., mother vs. child with contradicting clauses) Kratos prioritizes the clause of the user with the higher priority (e.g., mother). For hard competition conflict (HCC), both users with overlapping conditions are notified and Kratos offers a common operating condition to both. This common condition is enforced as a policy to the device upon users' agreement. On the other hand, in the case of both soft priority (SPC) and soft competition conflicts (SCC), the result of the negotiation is a new clause with common set of conditions. For restriction conflict, both restricted user and policy assigner are notified and if the policy satisfies conditions in Equation 9, the restriction policy is enforced in the device.

C DEVICES USED DURING EVALUATION

We present a detailed list of devices used during the implementation and evaluation of Kratos in Table 1.

Device Type	Model	Quantity
Smart Home Hub	Samsung SamrtThings Hub	1
Smart Light	Philips Hue Light Bulb	4
Smart Lock	Yale B1L Lock with Z-Wave Push Button Deadbolt	1
Smart Camera	Arlo by NETGEAR Security System	1
Smart Thermostat	Ecobee 4 Smart Thermostat	1
Motion Sensor	Fibaro FGMS-001 ZW5 Motion Sensor with Z-Wave Plus Multisensor	6
Temperature Sensor	Fibaro FGMS-001 ZW5 Motion Sensor with Z-Wave Plus Multisensor	1
Door Sensor	Samsung Multipurpose Sensor	2

Table 1: Devices and sensors used in our smart home setup to evaluate Kratos.

D KRATOS-ENABLED SMARTTHINGS APP

We provide an example of Kratos-enabled SmartThings App.

Listing 1: Policy enforced at install-time

```
definition(
    name: "Big Turn ON modified",
    namespace: "smartthings",
    author: "Anonymous",
    description: "Turn your lights on when the SmartApp is tapped.",
    category: "Convenience",
    iconUrl: "https://s3.amazonaws.com/smartapp-icons/Meta/light_outlet.png",
    iconX2Url: "https://s3.amazonaws.com/smartappicons/Meta/light@2x.png"
)
import groovy.time.*
preferences {
        section("When I touch the app, turn on...") {
        input "switches", "capability.switch", multiple: false
        input name: "email", type: "email", title: "Email", description: "Enter
                Email Address", required: true, displayDuringSetup: true}}
def installed()
{ atomicState.SmartLightTimes = [:]
        atomicState.SmartLightAdmins = [:]
        atomicState.SmartLightUsers = [:]
        atomicState.SmartLightDevID = [:]
        atomicState.SmartLightTimeStart = [:]
        atomicState.SmartLightTimeEnd = [:]

    log.debug "${new Date()}"
        getSmartLightJsonData()

    def item = atomicState.SmartLightUsers.indexOf(email)
    if (item>=0){
        int index = atomicState.SmartLightUsers.indexOf(email)
        def between = timeBetween (atomicState.SmartLightTimeStart[index],
                atomicState.SmartLightTimeEnd[index])
        if (between == true){
            subscribe(location, changedLocationMode)
            subscribe(app, appTouch)
            log.info app.getAccountId()}}
}
def updated()
{ atomicState.SmartLightTimes = [:]
        atomicState.SmartLightAdmins = [:]
        atomicState.SmartLightUsers = [:]
        atomicState.SmartLightDevID = [:]
        atomicState.SmartLightTimeStart = [:]
        atomicState.SmartLightTimeEnd = [:]
    getSmartLightJsonData()

    def item = atomicState.SmartLightUsers.indexOf(email)
    if (item>=0){
        int index = atomicState.SmartLightUsers.indexOf(email)
        def between = timeBetween (atomicState.SmartLightTimeStart[index],
                atomicState.SmartLightTimeEnd[index])
        if (between == true){
            unsubscribe()
            subscribe(location, changedLocationMode)
            subscribe(app, appTouch)}}
}
def changedLocationMode(evt) {
        log.debug "changedLocationMode: $evt"
        switches?.on()}
def appTouch(evt) {
        log.debug "appTouch: $evt"
        switches?.on()}
def getSmartLightJsonData(){
    def listTimes = []
    def listAdmins = []
    def listUsers = []
    def listIDs = []
    def listTimeStarts = []
    def listTimeEnds = []
    def params = [uri: "https://mywebserver/xxxyyyzzz/2/public/values?alt=json
        ",]
    try {
        httpGet(params) { resp ->
            for (object in resp.data.feed.entry){
                                listTimes.add (object.gsx$time.$t)
                listAdmins.add (object.gsx$adminemail.$t)
                listUsers.add (object.gsx$restricteduseremail.$t)
                listIDs.add (object.gsx$deviceid.$t)
                listTimeStarts.add (object.gsx$timerangestart.$t)
                listTimeEnds.add (object.gsx$timerangeend.$t)
            }
            atomicState.SmartLightTimes = (listTimes)
            atomicState.SmartLightAdmins = (listAdmins)
            atomicState.SmartLightUsers = (listUsers)
            atomicState.SmartLightDevID = (listIDs)
            atomicState.SmartLightTimeStart = (listTimeStarts)
            atomicState.SmartLightTimeEnd = (listTimeEnds)}
    } catch (e) {
        log.error "something went wrong: $e"}
}

def timeBetween(String start, String end){
    long timeDiff
    def now = new Date()
    def timeStart = Date.parse("yyy-MM-dd'T'HH:mm:ss","${start}".replace("
        .000-0400",""))
    def timeEnd = Date.parse("yyy-MM-dd'T'HH:mm:ss","${end}".replace("
        .000-0400",""))
    long unxNow = now.getTime()
    long unxEnd = timeEnd.getTime()
    long unxStart = timeStart.getTime()
    if (unxNow >= unxStart && unxNow <= unxEnd)
        return true
    else
        return false
 }
```