Who's Controlling My Device? Multi-User Multi-Device-Aware Access Control System for Shared Smart Home Environment

AMIT KUMAR SIKDER and LEONARDO BABUN, Florida International University, USA
Z. BERKAY CELIK, Purdue University, USA
HIDAYET AKSU, Google, USA
PATRICK MCDANIEL, Pennsylvania State University, USA
ENGIN KIRDA, Northeastern University, USA
A. SELCUK ULUAGAC, Florida International University, USA

Multiple users have access to multiple devices in a smart home system – typically through a dedicated app installed on a mobile device. Traditional access control mechanisms consider one unique, trusted user that controls 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 that cannot be handled by traditional access control techniques. Moreover, smart devices from different platforms/vendors can share the same home environment, making existing access control obsolete for smart home systems. 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 four main components: user interaction module, backend server, policy manager, and policy execution module. Users can easily specify their desired access control settings using the interaction module that are translated into access control policies in the back-end server. The policy manager analyzes these policies, initiates automated negotiation between users to resolve conflicting demands, and generates final policies to enforce in smart home systems. We implemented Kratos+ as a platform-independent solution and evaluated its performance on real smart home deployments featuring multi-user scenarios with a rich set of configurations (337 different policies including 231 demand conflicts and 69 restriction policies). These configurations also 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. We also performed an extensive user study with 72 smart home users to better understand the user's needs before designing the system and a usability study to evaluate the efficacy of Kratos+ in a real-life smart home environment.

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

Authors' addresses: A. K. Sikder, Florida International University, USA, 10555 West Flagler St. EC 3900, Miami, Florida, 33174; email: asikd003@fiu.edu; L. Babun, Florida International University, USA, 10555 West Flagler St. EC 3900, Miami, Florida, 33174; email: lbabu002@fiu.edu; Z. Berkay Celik, Purdue University, USA, 610 Purdue Mall, West Lafayette, Indiana, 47907; email: zcelik@purdue.edu; H. Aksu, Google, USA, 1600 Amphitheatre Parkway, Mountain View, California, 94043; email: hidayetaksu@gmail.com; P. McDaniel, Pennsylvania State University, USA, W303 Westgate Building, University Park, Pennsylvania, 16802; email: mcdaniel@cse.psu.edu; E. Kirda, Northeastern University, USA, 360 Huntington Ave., Room 617, Boston, Massachusetts, 02115; email: ek@ccs.neu.edu; A. Selcuk Uluagac, Florida International University, USA, 10555 West Flagler St. EC 3900, Miami, Florida, 33174; email: suluagac@fiu.edu.

2577-6207/2022/09-ART27 $15.00
https://doi.org/10.1145/3543513

CCS Concepts: • **Security and privacy** → **Access control;**

Additional Key Words and Phrases: Smart home system, access control, internet-of-things, policy language, privilege escalation

ACM Reference format:
Amit Kumar Sikder, Leonardo Babun, Z. Berkay Celik, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A. Selcuk Uluagac. 2022. Who's Controlling My Device? Multi-User Multi-Device-Aware Access Control System for Shared Smart Home Environment. *ACM Trans. Internet Things* 3, 4, Article 27 (September 2022), 39 pages.
https://doi.org/10.1145/3543513

1 INTRODUCTION

Cyberspace is expanding fast with the introduction of new smart home technologies dedicated to making our homes automated and smarter [55]. This trend will only continue, and billions of smart devices will dominate our everyday lives by the end of this decade [44, 49, 51]. **Smart home systems (SHSs)** typically allow multiple devices to be connected to increase the overall efficiency of the home and automate daily tasks, making it more convenient for its occupants. 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 via 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 an SHS, multiple users access the same smart device, typically via an app (e.g., SmartThings App) installed on their smartphones or smartwatches (a controller device), causing 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 users' conflicting demands 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 [5, 9, 26, 35]. For instance, smart home platforms often give automatic full access to every user added to the SHS [40]. With full access, a new user can easily add new unauthorized users or reconfigure the connected devices without the device owner's consent [56] and cause safety issues [10, 52]. For instance, a temporary guest can acquire sensitive information from the homeowner or a rogue user can leave the smart lock open for unauthorized physical access. 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. Once the setup is done, smart home platforms do not allow fine-grained controls or dynamic conditional choices to meet users' complex demands.

In this paper, we introduce Kratos+, a multi-user, multi-device-aware access control system designed for smart home systems. We designed Kratos+ based on an access control user study with 72 real-life smart home users. 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 policy negotiation results, and enforcing the negotiation results over the smart home devices and apps. We designed Kratos+ as a platform-independent access control system and implemented it in a real a multi-user multi-device SHS that included 22 different sensors and actuators from three different smart home platforms (i.e., Samsung SmartThings, Philips Hue, and LIFX). We further evaluated Kratos+ performance on 337 different access control policies, including 231 demand conflicts and 69 restriction policies. We also assessed the performance of

Kratos+ against five different threats. Kratos+ resolves demand conflicts and detects different threats with a 100% success rate in a multi-user smart home system with minimal overhead. Finally, we performed a usability study of 43 smart home users. Overall, Kratos+ achieved an average rating of 4.6 out of 5 based on user-friendliness, demand, deployment, and effectiveness. Note that this work is an extension of our previous work [50]. Kratos+ is more user-centered as we conducted a user study among real-life smart home users to understand and address real-world access control needs. We also significantly improved our prior work to build a platform-independent and cloud-integrated access control system to ensure deployability in a real-life smart home environment. In addition, we implemented and evaluated Kratos+ in different smart home layouts and platforms with new user data. Lastly, we performed a usability study with real-life smart home users and devices to evaluate the usability of our proposed access control system.

Contributions: Kratos+ is the first work implementing a priority-based access-policy negotiation technique based on real users' needs to resolve conflicting user demands in a shared smart home system with multiple users and devices in an automated and configurable fashion. The summary of the main contributions of this work are as follows:

- We conducted a user study of 72 different smart home users to understand the user needs for multi-user multi-device access control mechanisms in their SHS.
- We introduced Kratos+, a platform-independent multi-user and multi-device access control mechanism for SHS. Kratos+ implements flexible policy-based user controls to define user roles and understand users' demands on a smart home platform, 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 smart home system.
- 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 an inadequate access control system. Our evaluation shows that Kratos+ can detect different threats with a 100% success rate.
- Finally, we performed a usability study with 43 different smart home users to understand the effectiveness of Kratos+. Our results showed that Kratos+ achieves an average of 4.6 ratings out of 5 from users on user-friendliness, demand, deployment, and effectiveness.

Organization: The remainder of the paper is organized as follows. In Section 2, we present the main findings of our access control study and discuss the shortcomings of existing access control mechanisms in SHS. In Section 3, we articulate the problem with a use case and explain our threat model. Later, we detail the architecture 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+ in resolving and optimizing diverse user demands and in detecting different threats in an SHS. In Section 7, we discuss the findings extracted from the usability study of Kratos+. The benefits of Kratos+ are presented in Section 8. Finally, Section 9 discusses the related work, and Section 10 concludes the paper.

2 MOTIVATION AND DEFINITIONS

2.1 Access Control Needs of Users in a Smart Home

Access control in multi-user SHSs poses 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 that can lead to conflict scenarios [21, 45]. However, existing smart home platforms mostly offer a binary control mode where a user acquires all the control or has 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 [17]. It is also important to understand smart home users' relationships, social norms, and personal preferences before designing a fine-grained access control system as these dynamics can lead to diverse users' needs in an SHS. For instance, parents may want to restrict smart TV access for the kids, roommates may want privacy for bedroom locks in an apartment that is shared, or owners may want to give AirBnB guests temporary access. Hence, a fine-grained access control system should address users' diverse needs in a multi-user, multi-device smart home ecosystem. Several prior works have focused on understanding user preferences and needs by conducting user studies among smart home users [20, 62, 63]. Although these are useful studies, they were prematurely designed and are limited in their scope. In this work, we consider the access control needs and suggestions of SHS users reported in prior works, along with our findings from our user study, to design a platform-independent, fine-grained access control system for multi-user multi-device SHSs.

2.2 User Study

We first conducted a user study to understand the real needs for multi-device, multi-user access control systems in smart home settings. We then considered these findings to propose a novel access control system. Although the user study is not the primary goal of this work, it is instrumental in understanding the users' needs in a multi-user smart home environment. To conduct our user study, we obtained appropriate approvals from the **Institutional Review Board (IRB)** and provided monetary compensation to each participant. While prior works established the need for access control mechanisms in smart home systems, they do not cover users' expectations in an effective access control system [20]. In our study, we asked the participants a total of 28 questions organized into three different categories, including (1) user characterization, (2) smart home setting preferences, and (3) user preferences in multi-user multi-device scenarios. Additionally, we asked for the users' expectations regarding design features and implementation of an access control system – topics that are missing in the prior works. In the following, we present the survey's results and discuss how Kratos+ addresses the needs of users in each case.

Selecting participants. We selected the study participants by a university-wide open call for participation and flyers distributed through communities of participants from different backgrounds, technical expertise, ethnicity, age ranges, etc. Our main target was to recruit participants either with hands-on experience in using smart home devices or willing to use them in the future. While the prior participant group may provide insights into existing access control needs, the latter would shed light on access control expectations from potential future users.

Block 1- User Characterization. We surveyed a total of 72 smart home users. We recruited the smart home users with an open call for participation on our website to avoid bias in our study. We aimed to fully characterize the group of users, their households, and their smart home system experiences with the questions.

Age: Out of the total, 20 (27.7%) users reported ages in the range of 16–24 years, 41 (56.94%) users were in the range of 25–34 years, and 11 (15.36%) in the range of 35–44 years.

Occupation: 31 out of 72 participants (43.05%) had a university degree, 28 (38.89%) had a post-graduate study, and 12 (16.67%) were full-time job holders.

Smart Device Usage: 61 out of 72 participants (85.7%) stated that they either have used or have some smart home device in their homes. The rest of the users mentioned minimum knowledge about smart home devices and were interested to use smart devices in the future.

Smart Home Device Types: We also asked participants about the device types they had. The most popular devices among the surveyed users were: smart light 45 (62.5%), Smart TV 37 (51.38%), smart thermostat 23 (31.94%), smart camera 16 (22.22%), smart lock 14 (19.44%), and smart switch 8 users (11.11%).

Smart Home Platforms: Out of 10 different smart home platforms included in the survey, the users stated that they were familiar with four smart home platforms: Google Home (63 users - 87.5%), Samsung SmartThings (57 users - 79.16%), Amazon Alexa (48 users - 66.67%), Apple HomeKit (39–54.17%), and OpenHAB (16 users - 22.22%). Further, we asked for details about the specific smart home platform that they actually used (or were willing to use). Similar to the previous results, Google Home, Samsung SmartThings, and Apple HomeKit were the majority of the answers, 38 (52.77%), 20 (27.78%), and 12 (16.67%), respectively.

Technical Experience: We also surveyed the participants regarding their technical experience with smart home devices and apps. Out of the total participants, 57 (79.17%) reported that they knew how to set up smart devices, 43 (59.72%) stated that they knew how to install Apps, and 37 (51.38%) said that they felt comfortable integrating different smart home devices using a hub or cloud.

Household Characteristics: We concluded the first block of the survey by asking questions about the household characteristics. The participants reported that they lived with different family/roommate sizes. For instance, 9 (12.5%) users lived in a family size of 5 or higher, 27 (37.5%) users stated that they lived in a size of 4, 21 (29.17%) reported living in a size of 3, and 15 (20.83%) users shared their spaces with at least another person. Further, we asked about how many members of these households shared smart devices. Interestingly, out of 36 participants, only 6 (8.33%) reported that they did not share deployed smart devices with anyone else. Then, out of the 72 users remaining, 66 of them disclosed that they shared devices with at least two more household members and up to 7.

We used the answers obtained in this block of questions to characterize the target users of Kratos+. In most cases, the users of smart home devices and apps know the basics of how to configure devices and install apps. Additionally, multi-user smart households represent a positive potential environment for multi-user access control systems like Kratos+. Finally, we note that most of the users reported that they share a smart device with at least two other household members.

Block 2- Smart Home Setting Preferences and Characterization. In this part of the user study, participants answered questions regarding the need for an access control mechanism in the smart home system.

Multi-member Settings: We asked users if they had ever considered a need for defining various controls on the other users while installing or using smart apps. In total, 49 (68.05%) users answered "Yes" to this question.

Multi-member Access: Further, we investigated if the users were ever had given device access to other members. In this case, a higher majority of 59 (81.94%) users answered "Yes".

Conflicts Among Settings: In multi-user scenarios, 64 (88.89%) surveyed participants disclosed having to update or re-evaluate smart device settings after discovering that original settings had been changed/modified by other members of the household that had authorized access to the devices.

Table 1. Summary of User Study and the Need of Access Control Among Smart Home Users

	Category	Features/survey questions	User response
Block 1	User Characterization	Age range	16–44 years
		Prior Smart Device Experience	85.7%
		Common smart devices	smart light, smart TV, smart thermostat
		Common smart home platform	Google Home 87.5%, SmartThings 79.16%, Amazon Alexa 66.67%, Apple HomeKit 54.17%, OpenHab 22.22%
		Technical experience	Device installation 79.17%, App installation 59.72% Hub/cloud integration 51.38%
		Shared Household	91.67% users
Block 2	Smart home settings and preferences	Need for multi-user settings	68.06%
		Shared device access	81.94%
		Conflicting device settings	88.89%
		Access control admin interface	84.72%
		Guest Access	94.44%
Block 3	Multi-user multi-device access features	Built-in multi-user access feature	86.11%
		Third-party multi-platform access control	77.58% (22.41% users willing to pay for the service)
		Access control identifier	Email - 65.27%, Smart home ID - 62.2%, Smart home account - 55.56%
		Automatic policy negotiation	75%
		Integrated policy update	80.56%
		Minimum user interaction in access control	72.22%
		Multiple policies for single device	77.78%
		Centralize admin monitoring	76.39%
		Restricted access for guests	100%

Multi-member Admin Interface: Regarding the multi-member scenarios, 61 out of 72 users (84.72%) agreed that smart home apps should have an interface that can be regularly checked by the device owner to control the access rights of devices.

Guest Access: Lastly, we asked about giving device access to guest members (i.e., visitors, tenants, etc.). A vast majority of users (68–94.44%) answered "Yes" to the option of smart Apps having an automated mechanism to revoke access requests from guest users.

Overall, this set of questions shows the need for access control mechanisms in smart home systems. We found that the device owners frequently had to deal with conflicts due to settings changed by other household members. Additionally, the device owner wants to control who accesses the devices and desires to enforce limited access controls for the users that are not fully trusted.

Block 3- Multi-user/Multi-device Access Control. Finally, we assessed the need for the access control mechanisms and received feedback from the users on designing and implementing such mechanisms.

Integrated Access Control: We asked the users whether they thought an access control mechanism should be provided in smart home scenarios. A majority of 62 (86.11%) users answered that it would be an essential feature, and smart home platforms should provide an integrated access control system.

Separated Access Control: We further asked whether there is a need for a separate app/system to manage the access controls. 58 (80.55%) users answered positively. Out of 58 users, 45 (77.58%) users desired to use an access control application if it were free, and secure while 13 users (22.41%) stated that they might even pay for the service. Additionally, the users agreed to share some specific **personal information (PI)** with the access control app if required for the app design. Out of six different options provided, the users stated that they would allow the app to use their email address (47 users - 65.27%), smart home user ID (52 users - 62.2%), smart home account credentials (40 users - 55.56%), and smart device ID (32 users - 44.4%).

Member Priorities: In a multi-user environment, access control can be provided by assigning priorities to different household members. We asked the users which household should have the highest priority (the most trusted member) and the lowest one (the least trusted member). The user's assigned priority levels in between these boundaries. The "Spouse/Partner", "Father", and "Mother" are among the members with the highest priorities. On the contrary, "Babysitter", "Temporary Guest", "Frequent Visitor", and "Cleaning Personnel" were among the members that were assigned the lowest priority.

Device Priorities: We evaluated what type of devices should be included in the access control mechanism. Out of 18 options, the devices related to security and safety were selected to be the most important devices for access control. This list includes the smart lock, smart thermostat, smart fire alarm, smart monitoring system, presence sensor, and smoke sensor. On the other hand, devices with the least importance to the user were the smart coffee machine, doorbell, and smart light.

Automated System: Out of 72 participants, 54 (75%) users answered positively to the possibility of having an automated negotiation system to solve access control conflicts among members with the same level of priority.

Update Policy Feature: Out of 72 participants, 58 (80.56%) users expressed their interest in having a feature to update/change current access control policies.

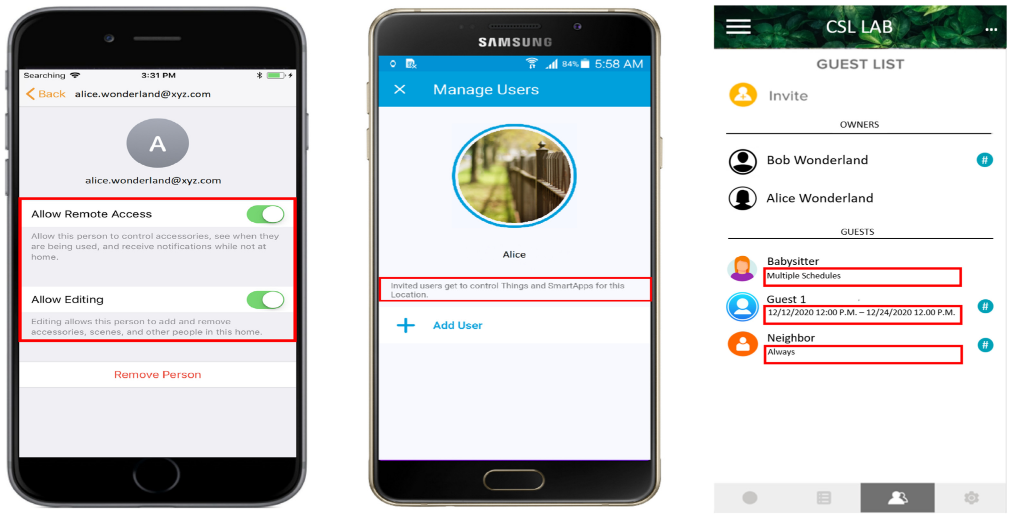
Negotiation Process: We presented four different options to the users about how the policy negotiation process should work among users of the same priority (superusers or admin level). Users' answer shows that users desire to automate this process with minimal interaction (52 users - 72.22%). The users' answers also suggested that the access control system should notify the members affected by the policy conflict.

Multiple Policies: The participants reported that the access control system should allow multiple policies when a conflict occurs. 56 (77.78%) users suggested that a simple notification and an automated approval of the non-conflicting policies are sufficient.

Conflicting Policies: If two policies conflict and one of them is defined by a member with lower priority, 33 (45.83%) users suggested that the lower priority policy should be rejected. However, 25 (34.72%) other users indicated that the member with the higher priority should be asked to assess the possibility of changing its policy to allow the lower priority member to add its settings without conflict. In both cases, they again suggested that a notification system is a critical feature to resolve policy conflicts. Further, we presented a situation where a member with the same priority level as the owner introduces a conflicting policy on behalf of a low-priority member. In this scenario, 39 (54.17%) users suggested that both the owner and the member with similar priorities should be notified to negotiate together on how to solve the conflict. However, 22 (30.56%) users proposed that the high-priority member should be notified about the conflict with the owner, and the new policy should be automatically rejected.

Low-Priority Members: The last two questions were about managing the low-priorities members. The majority of the users (55–76.39%) confirmed that the access control system should have a feature to monitor the actions and settings of low-priority members, while the other 17 (23.61%) suggested that this may be an additional feature to have. We obtained similar results when we specifically asked about access control for guest members. In this case, 51 (70.83%) surveyed users replied that guest members needed to have some restrictions while other 21 (29.17%) users said that this would be an additional feature to include.

Clearly, the participants expressed their interest in having access control mechanisms for smart home scenarios. Also, they suggested that despite a necessary notification system, the access



Allow this person to control accessories, see when they are being used, and receive notifications while not at home.

Editing allows this person to add and remove accessories, scenes, and other people in this home.

Invited users get to control Things and SmartApps for this Location.

(a) Apple HomeKit with coarse-grained access

(b) Samsung SmartThings with binary access

(c) August SmartLock with time-based access

Fig. 1. Existing access control system in smart home platforms.

control mechanism should work effectively and independently with minimal user interaction. Finally, users stated that the access control system should be able to differentiate between users with different levels of priority and negotiate accordingly. Conflicts between users with the highest priorities should be resolved with human intervention when necessary. In contrast, conflicts between members with high and low-level priority should automatically be resolved by rejecting the latter's requests. Additional features were suggested to monitor and restrict the actions of low-level priority members. Table 1 summarizes the outcomes of the user study and desired access control features in a multi-device multi-user smart home environment.

2.3 Existing Access Control Mechanisms in Smart Homes

Existing smart home platforms offer limited features to support access control mechanisms. Current smart platforms often provide access control via an edge device (i.e., hub) that is part of the network and to where smart devices are connected to. Users then may use smartphone applications offered by the smart home vendors to configure basic rules to control and manage their smart devices. For instance, smart home platforms such as Samsung SmartThings and Apple HomeKit allow users to install smart lights and define start and end times using corresponding mobile applications. In some cases (e.g., Samsung SmartThings), the mobile application also enables users to install different smart home apps to enable new functionality on a device or change the way the device is controlled. In a multi-user smart home environment, all authorized users can control and change device tasks as the current access control systems in smart home platforms work in a binary control mode where a user gets all the control or no control. For example, *Samsung SmartThings* provides full access to all the connected devices to a user (Figure 1(b)). Moreover, any user gets the privilege to add new users and install new apps in the system. To protect smart devices from unauthorized app installation and device settings, some smart home platforms (e.g., *Apple HomeKit*) offer two different options: remote access and editing (Figure 1(a)). In remote access, a new user gets an "access-only" privilege to the connected devices. In the editing option, a new user obtains permission to add or remove any app, device, or user in the system. However, this access control system cannot reflect the conflicting demands generated in a multi-user environment. For instance, if two users try to set the smart thermostat in different temperatures simultaneously, current smart home platforms cannot differentiate user conflicts and execute any user command without verifying with other users. We also found that some IoT devices such as August smart lock and RemoteLock offer immature time-based access control mechanisms [31, 38]. However, these solutions are vendor- and device-specific, and thus, are not ready and applicable in a multi-device multi-user smart home system. In summary, *existing access control mechanisms in smart home technologies fail to deliver the diverse and complex user demands in a multi-device multi-user setting.*

2.4 Terminology

We define several terms that we consistently use throughout this work.

Multi-device Multi-platform Smart Home. Modern smart home systems allow users to install multiple devices from different vendors in the same physical environment. We consider a multi-device smart home environment as a physical environment with multiple smart home devices installed from the same vendor. On the contrary, a multi-platform smart home environment refers to a physical environment with multiple smart home devices installed from different vendors.

Vendor-specific Smartphone App. We consider vendor-specific smartphone app as the smart home device controller app published by the device vendors.

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 the 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 of 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 to the system/local 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 demand negotiation processes. In Section 4, we detail the different priority levels considered in this work.

Conflict. For this work, *Conflict* is defined as the dispute process generated from two or more demand policies that interfere or contradict based on the specific requests of the policies. Based on the nature of 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 conditions.

(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 restricted user is disputing the restriction policy for a device.

Permission Level. Permission level specifies the ability of a user to modify current configuration in the smart home system.

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

Common access point. Smart home devices are usually connected to each other via a common access point such as a hub or router. The main purpose of the common access point is to create a network of devices in the home environment and provide a seamless internet connection to the smart devices. In this work, we consider both the smart home hub and router as common access points.

Credential Array. We define the credential array as a structured format to add or delete a user in an access control system with specified priorities. A credential array includes the following information- commanding user ID, new user ID, priority level, permission level, and access period of the user in the system. In Section 4, we detail the credential array and how Kratos+ uses this to assign new users and policies.

3 PROBLEM AND THREAT MODEL

This section introduces the challenges of an access control mechanism in the smart home through an example scenario. Then we articulate the threat model considered in this work.

3.1 Problem Definition and Assumptions

We assume a smart home setting (S) similar to the one depicted in Figure 2. The smart home has several installed devices to create an automated smart environment. In S, four different users - Bob (father), Alice (mother), Kyle (child), and Gary (guest) interact with the devices. We assume Bob and Alice are the smart device owners, and all four users have access to the smart home system through their controller app (installed on their smartphone or tablet). Here, the term *access to the smart home system* refers to the ability to control the devices, configure the system (add/delete devices), and add new users to the system. We assume that the users are performing the following activities which result in conflicting demands- (1) Bob and Alice configure the smart thermostat to different overlapping values at the same time (soft conflict), (2) Alice wants to limit access to smart lock after midnight while Gary wants to have access (hard conflict), (3) Alice tries to restrict Kyle from using the smart coffee machine, but the smart home system does not allow her (restriction conflict), (4) Alice wants Kyle to have access to the smart light only while Kyle is present (location access). Hence, a new access control system is needed and designed to answer the following questions: (1) How can Bob and Alice solve their conflicting demand and use the thermostat simultaneously? (2) How can Alice give exclusive permission to use the smart lock to Gary after midnight? (3) How can Alice restrict a specific device for a specific user? (4) How can Alice give location-based access to Kyle? (5) How can Bob limit the access of Gary to add a new user? To address these questions, 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 in the system.

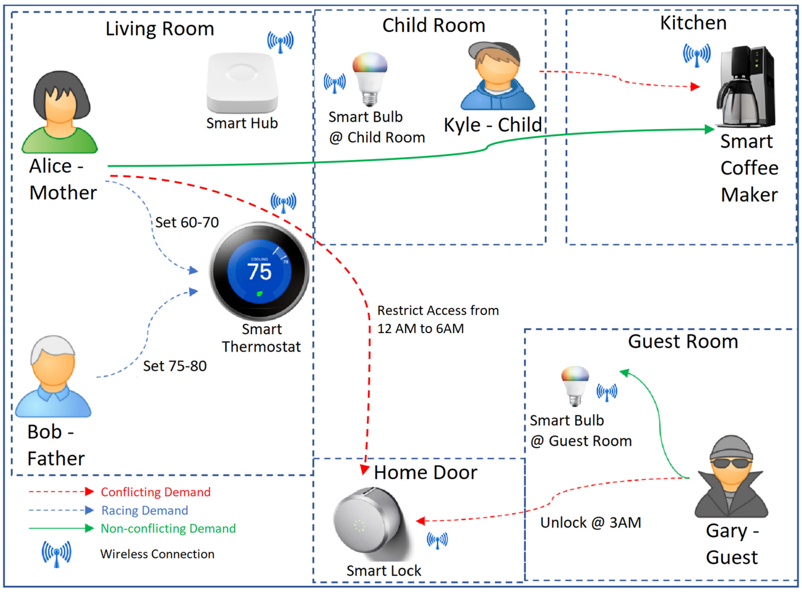


Fig. 2. Sample smart home with multiple users attempting to control multiple devices with conflicting demands.

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), leading to undesired device access. Also, Kratos+ considers legitimate smart home users trying to change the system settings without authorization (e.g., overriding the 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 homes (i.e., transitive privilege). In fact, access to an 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 has been added automatically at the time of system installation as they are different problems from the contributions of Kratos+. In Table 2, we detail five different threats that we use later to evaluate the performance of Kratos+ (Section 6).

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

4 KRATOS+ ARCHITECTURE

In this section, we present the architecture of the Kratos+ and its main components. Kratos+ is a comprehensive access control system for multi-user smart home system where users can express

Table 2. Summary of the Threat Model Considered in Kratos+

Threat	Attack Method	Attack Example
Threat-1	Over privileged controls	A newly added smart home user gets the access to use all the connected devices which can lead to undesired activities in the smart home system.
Threat-2	Privilege abuse	A newly added smart home user can abuse the granted privilege to perform *malicious activities* in the smart home system.
Threat-3	Privilege escalation	A newly added smart home user can use the legitimate permissions to remove devices and apps, change device settings, or make a device unavailable to the owner.
Threat-4	Unauthorized access	A temporarily added smart home user can have an access to the smart home system if the owner forgets to delete the access manually.
Threat-5	Transitive privilege	A newly added user adds a new user in the system who automatically gets the same privilege level as the owner and may utilize this transitive privilege to perpetrate his/her exploits.

their conflicting demands, desires, and restrictions through policies. Kratos+ allows an authorized user to add new users and enforce different policies to connected 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 as well as assign policies in the smart home system. We integrate Kratos+ into the mobile app provided by smart home vendors to provide a single user interface to manage users and assign policies to the connected devices.

Diverse User Roles/Complex Relations. In a smart home, 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 usage conflict among smart home users in a shared smart home environment. The main challenge of an access control system in a smart home is to resolve these conflicts in a justified way. In addition, users in a multi-user smart home environment should agree with the conflict resolution outcome 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. Additionally, Kratos+ notifies the users of the results of the policy negotiation system.

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

Unified Policy Enforcement. All user commands to the smart devices should go through an *access control enforcement* layer to provide fine-grained access control in a smart home

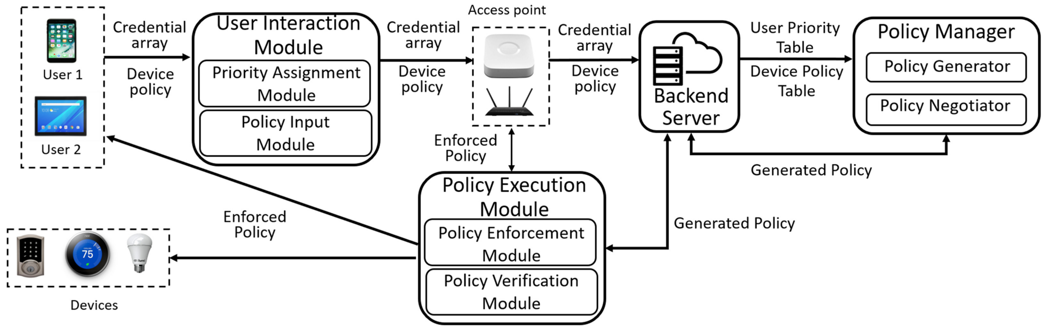


Fig. 3. Architecture of Kratos+ system.

environment. Kratos+ uses an execution module that checks all enforced policies before executing a user command in the smart home environment.

Multi-platform Support. Access control in the smart home environment should support multiple smart home platforms as smart devices from different vendors may share the same physical environment. To support a multi-platform smart environment, Kratos+ offers access control implementation both at the app-level (smart home apps) and access point level (hubs and routers). While app-level implementation supports standalone smart devices, Kratos+ ensures fine-grained access control in the multi-platform smart environment by enforcing policies in a common access point.

Figure 3 details the architecture of the Kratos+ system. Kratos+ includes four main modules: (1) user interaction module, (2) back-end module, (3) policy manager, and (4) policy execution module. 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 common access point (smart home hub or router). The back-end module captures these data and creates a user priority and device policy list for the users. The *policy manager* module then gathers user priorities and device policies from the generated lists and triggers the policy generation and negotiation process. After successfully negotiating policies among smart home users, the policy manager generates final device policies, and the policy execution module implements the policies. The following subsections detail each module in Kratos+ and explain 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. All the authorized users in a smart home environment can use controller devices (smartphones, smart tablets) to access the user interaction module. 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 to the users. 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 time of the new user in the system. The user priority level is used in the policy generation module to negotiate policies among users and create device policies. To add a new user and assign priorities, we consider the following rules to avoid conflicts in the priority assignment.

- Each user has the authority to add new users and assign a priority.
- The Owner of the smart home system will have the highest priority in the system by default.
- 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 a 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 only 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, in the smart home environment in Figure 2, 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+, the 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 the validity time (T) of a user in the system. After the specified validity time, Kratos+ removes the user from the system to automatically prevent unauthorized access from a temporary guest. The collected credentials are forwarded to the back-end through a hub to create the user priority structure.

Policy Input Module and Access Policy Language. Policy input module provides an interface to the users for assigning policies in connected smart home devices. All the authorized users can choose any connected devices and create a device policy using this module. To define the device policies, Kratos+ introduces a formal access control policy language for the smart home environment to express complex user preferences (e.g., smart home 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 to the smart home system. For instance, sample policies for the smart home of four users are shown in Figure 2, 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 4. The criteria defined by the users are used throughout this sub-section to construct their policies.

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: $\langle$users$\rangle$: $\langle$devices$\rangle$: $\langle$conditions$\rangle$: $\langle$actions$\rangle$. The first part of the policy is *users*, which defines a user assigned to the policy in the system. This part also includes the information of the targeted user for whom the policy is assigned. The second part, *devices*, specifies the list of the devices included in this statement. Similar to the *users* part, this can be a single device or a list of devices. Kratos+ uses device ID assigned by the smart home system 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 smart devices' capabilities. For instance, a user may define a condition where only a pre-defined range of commands or a certain time-window is matched. The final part of the policy is $\langle$action$\rangle$ which states the clause type, either *demand* or *restrict*. We note that the Kratos+'s policy language

@U_1: Alice	
U_2–	restrict smart thermostat between 60-70 degrees.
U_3–	restrict access to smart coffee maker.
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

Fig. 4. An example demand and restriction requirements of users in Figure 2.

$$
\begin{aligned}
@\mathrm{U}_1\ & \text{restrict} :: : \text{thermostat}_1 : \text{temperature} \notin [60 - 70]\ ; \\
& \text{restrict} :: \mathrm{U}_3 : \text{coffeemaker} : ; \\
& \text{demand} :: \mathrm{U}_4 : \text{bulb}_4 : ; \\
& \text{restrict} :: \mathrm{U}_4 : \text{lock}_1 : \text{time} \notin [6 : 00am - 9 : 00pm]; \\
@\mathrm{U}_2\ & \text{restrict} :: : \text{thermostat}_1 : \text{temperature} \notin [75 - 80]\ ; \\
& \text{demand} :: \mathrm{U}_3 : \text{bulb}_3 : \text{time} \in [7 : 00pm - 7 : 00am]; \\
& \text{demand} :: \mathrm{U}_4 : \text{lock}_1, \text{lock}_4 : ; \\
& \dots \dots
\end{aligned}
$$

Fig. 5. Sample policy clauses to partially implement demands and restrictions shown in Figure 4.

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 5. Here, two users, U_1 and U_2, define different demand and restriction policies in a smart home environment.

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 a user priority list and (2) generating a 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. Here, 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 to the user priority list based on the validity time specified in the credential arrays. The back-end module automatically removes users with expired validity and updates the user priority table. A sample user priority list is shown in Figure 6.

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 for different devices.

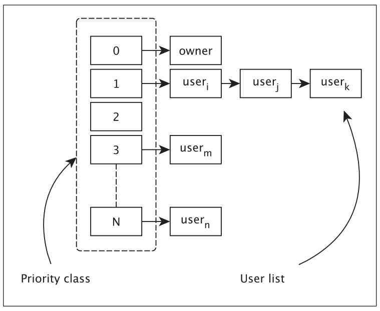


Fig. 6. A sample user priority list generated by Kratos+.

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. 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' relationships, social norms, and personal preferences. To address these diverse needs, Kratos uses an automatic policy negotiation module to resolve conflicts in a 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. Kratos+ considers three types of conflicts in a multi-user smart home environment: hard conflict, soft conflict, and restriction conflict.

A *hard conflict* happens when device policies enforced by two or more users with the same or different priorities conflict with each other over different non-overlapping device conditions. For instance, in Figure 2, Alice and Bob try to set the thermostat temperature to two distinct temperature ranges 60–70 and 75–80, respectively. Kratos+ considers this conflicting demand as a hard conflict and starts policy negotiation. On the other hand, *soft conflicts* occur when device policies assigned by the users with the same or different priorities have an overlapping device condition. For example, in Figure 2, if Alice and Bob try to set the thermostat temperature with overlapping ranges (65–75 and 70–80 respectively), Kratos+ considers this as a soft conflict. *Restriction conflicts* occur when restricted policy disputes with a device policy. As an example, in Figure 2, Kyle (child) wants to access the coffee machine, but Alice restricts the access for Kyle, which results in a restriction conflict. In Kratos+, a policy negotiation algorithm is developed to resolve policy conflicts and trigger the policy generation module to create acceptable policies for all authorized users.

Policy Negotiation Algorithm. 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 processes all the policies and computes the negotiated results by modeling the users' authorities (classes, roles) in a multi-layer list. Figure 6 illustrates this model. 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 in the same priority class share the same priority. Kratos+ considers three types of conflicts between user policies after users are classified into authorities. 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. Finally, $\mathcal{A} \in \{demand, restrict\}$ is the action requested by this statement when the set of conditions is satisfied. Kratos+ implements an algorithm to solve the policy conflicts through a set of equations as follows:

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

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

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

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

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

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

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

$$\begin{aligned} RC(\Psi_i, \psi_j) \leftarrow Restriction\_conflict(\Psi_i, \psi_j) \wedge \Xi(P_i) > \Xi(P_j) \\ \wedge A_i = restrict \end{aligned} \tag{8}$$

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.

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 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 cases. In other words, when policies have no possible way of cooperation or compromising, (e.g., Alice demands 60–70 range while Bob demands of 75–80 range for the same thermostat). In such cases, 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, conflicts are further categorized as *Priority Conflicts* or *Competition Conflicts* based on the priority of policy owners. When the conflict happens between users' policies with different priority classes, Kratos+ defines a *priority conflict.* However, if the users have the same priority, *competition conflicts* happens. Additionally, if any interference is caused by the nature of action requested in two

$$
\begin{aligned}
ABAC(\Psi_i) \coloneqq \{B \mid & \text{action}(B) = \mathrm{A_i} \land \\
& \text{subject}(B) = \mathrm{U_i} \land \\
& \text{resources}(B) = \mathrm{D_i} \land \\
& \text{constraints}(B) = \mathrm{T}_{C_\mathrm{i}}\} \\
\text{where} & \\
T_C \coloneqq \{c \mid & c \text{ satisfies the same} \\
& \text{conditions of C in mapped} \\
& \text{attributes into ABAC policy}\}
\end{aligned}
$$

Fig. 7. An example of mapping a sample policy to ABAC rule through a transformation function. $ABAC(\Psi_i)$ is a translation of action, subject, resources, constraints defined in a policy.

different policies, Kratos+ detects a restriction conflict in the system. Incorporating these with hard, soft, and restriction conflicts, Kratos+ overall implements five distinct conflict types.

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).

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

In the case of a **hard priority conflict (HPC)**, (e.g., mother vs. a 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 negotiation result is a new clause with a common set of conditions. For restriction conflict, both restricted user and policy assigner is notified, and if the policy satisfies conditions in Equation (9), the restriction policy is enforced in the device.

Policy Generation Module. The policy generation module's goal 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 (final policy table). Thereafter, these policies are enforced in smart devices.

Access Control Rule Generation. 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 ABAC-like enforcement for the final generated rules. Here, the policy, B, is the set of {action, subject, resource, constraints} tuples for a negotiated smart home policy. As an example, Figure 7 illustrates a simple example where $ABAC(\Psi_i)$ 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

The policy execution module enforces the final policies generated from the policy negotiation process and verifies any user commands with enforced policies before execution. This module has two sub-modules: the policy enforcement module and the policy verification module.

Policy Enforcement Module. Smart home devices can be controlled in two different ways - (1) by sending user commands using a controller device (smartphone or smart tablet) and (2) by installing customized apps in targeted smart home devices. Hence, any access control mechanism should consider both control options to ensure fine-grained access control.

In a smart home environment, devices are connected via a common access point such as a hub or router. Smart home devices are also connected to the Internet via a router to provide remote access to the users. Any user command coming from the controller device has to go through the hub or router to reach the targeted smart device. The policy enforcement module of Kratos+ considers this property of the smart home environment and enforces the generated policies at the common access point (hub/router). The policy enforcement module creates a user-policy table that comprises the final generated policies in the policy manager module. Based on this user-policy table, Kratos+ builds a customized filter to control user access to the installed smart home devices. For instance, in Figure 4, Alice restricts the smart lock access after 12 A.M. while Gary tries to access the lock at 3 A.M. As Alice has a higher priority than Gary, the final policy generated by Kratos+ restricts Gary to access the smart lock after 12 A.M. The user-policy table generated by policy execution module creates a filter in the hub/router to block any incoming command from Gary via controller device (smartphone/tablet). Here, the policy enforcement module uses the device ID to detect incoming commands from targeted users and allow or drop the commands based on final generated policies.

Smart home devices can also be controlled using vendor-specific controller apps installed on smartphones or tablets. Users can also install customized smart apps to control smart home devices (e.g., Samsung SmartThings). Kratos+ offers policy enforcement at the app level (both controller apps and customized apps) to provide fine-grained access control. The policy enforcement module generates conditional statements that represent the final policies. Kratos+ uses the existing static analysis technique to append these conditional statements in the controller app or customized apps [8] and enforces final policies at the application level. An example of Kratos+-modified app is presented in Appendix A.

Policy Verification Module. The policy verification module matches and verifies each user command with the generated policies before executing them. The matching process ensures fine-grained access control in a smart home environment. The policy verification module uses Kratos+ modified apps or user-policy table enforced in the hub/router to verify users' commands. When a user tries to change the device's state, the app asks the policy verification module to check in the final policy table generated by the policy generator. If an acceptable condition is matched, the policy verification module returns the policy to the app. It 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 verification engine, the user command in a smart home app is executed. For a multi-platform smart home environment, the policy verification module verifies the policies based on the user-policy table enforced in the common access point (hub/router). Here, each incoming user command is cross-referenced with the final policy table and device ID to identify the controller device and targeted smart device. Upon matching with an acceptable condition, the policy verification module generates a filter to allow (for accepted policy) or block (restricted policy) the user command. Kratos+ creates customized filters in the access point for each installed smart device based on the generated policies.

5 KRATOS+ IMPLEMENTATION

We implemented Kratos+ as a platform-independent access control system for smart home platforms and devices. To implement and test the efficacy of Kratos+, we choose several smart home platforms including Samsung SmartThings, Philips Hue, LIFX smart bulb, and Amazon Alexa. We next provide details of our implementation.

5.1 Implementation

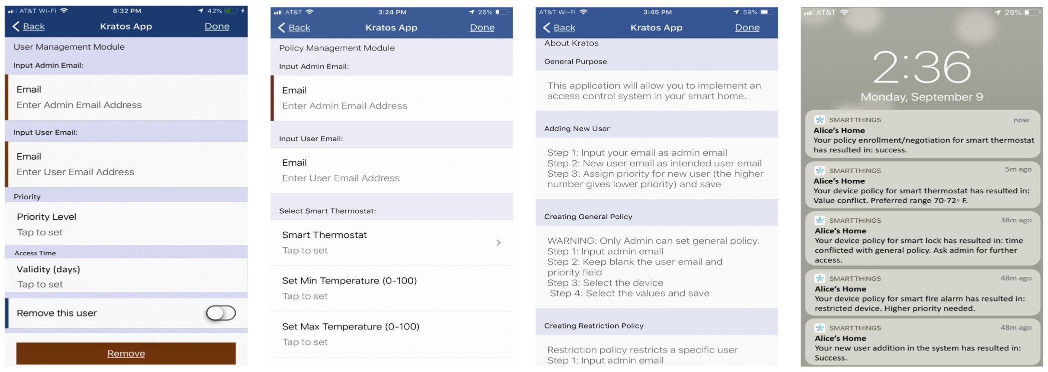
The implementation of Kratos+ has three basic integration phases: user interface, cloud integration, and policy enforcement. Kratos+ provides a user interface (user interaction module) to assign policies and priorities to the smart home users. The cloud integration includes the backend module and policy negotiation module, which capture the user priorities and policies and generate final policies. Finally, Kratos+ enforces the generated policies in smart home systems via appending conditional statements in installed apps or creating filters in the access point (hub/router) to supervise users' commands. The details of Kratos+'s implementation are given below:

User Interface. We built a customized smart app (Kratos+ app) that represents the user interaction module described in Section 4. The Kratos+ app has two main modules - user management and policy management. The user management module allows users to add new users and assign priorities. Additionally, users can configure Kratos+ as a role-based access control system by assigning roles to the users. For the implementation purposes, we define five different roles and priority levels in Kratos+ (i.e., father/owner - priority 0, mother/owner - priority 0, adult - priority 1, child - priority 3, and a guest - priority 4). These roles and priorities can be assigned by the smart homeowner or by authorized users with the same or higher priority than the one being assigned. Upon creating a new role/priority, the information is sent and stored in the cloud-integrated part of Kratos+. In the policy management module, users select installed smart devices and assign new policies to control them. As mentioned in Section 4, Kratos+ provides options to add three types of policies - demand policy, restriction policy, and location-based policies. Users can assign different device conditions (time-based, value-based, etc.) in the policies. As most of the smart home devices only allow time-based and value-based conditions, we classified the policies into 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 the following device policy array:

$$Device\ Policy,\ P = \{U, D, C_1, C_2, R, L\}. \tag{10}$$

The elements of the policy array are explained below.

- *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):* To identify the intended device for the policies, we use the device ID assigned by the smart home system. Here, we use the unique device ID assigned by the hub-connected smart home system (e.g., Samsung SmartThings) or MAC ID of the smart device as the device ID in the policies.
- *Time conditions (C_1):* In a device policy, users could assign a specific time to control device actions. Here, users can assign a time range or a specific time for instantaneous action. For example, a smart light can be accessed from sunset to sunrise only by assigning a time range. Similarly, a smart camera can be programmed to take pictures at a specific time.



User Management Module
Input Admin Email:
Email
Enter Admin Email Address
Input User Email:
Email
Enter User Email Address
Priority
Priority Level
Tap to set
Access Time
Validity (days)
Tap to set
Remove this user

Policy Management Module
Input Admin Email:
Email
Enter Admin Email Address
Input User Email:
Email
Enter User Email Address
Select Smart Thermostat:
Smart Thermostat
Tap to set
Set Min Temperature (0-100)
Tap to set
Set Max Temperature (0-100)
Tap to set

About Kratos
General Purpose
This application will allow you to implement an access control system in your smart home.
Adding New User
Step 1: Input your email as admin email
Step 2: New user email as intended user email
Step 3: Assign priority for new user (the higher number gives lower priority) and save
Creating General Policy
WARNING: Only Admin can set general policy.
Step 1: Input admin email
Step 2: Keep blank the user email and priority field
Step 3: Select the device
Step 4: Select the values and save
Creating Restriction Policy
Restriction policy restricts a specific user
Step 1: Input admin email

SMARTTHINGS now
Alice's Home
Your policy enrollment/negotiation for smart thermostat has resulted in: success.

SMARTTHINGS 5m ago
Alice's Home
Your device policy for smart thermostat has resulted in: Value conflict. Preferred range 70-72◦ F.

SMARTTHINGS 38m ago
Alice's Home
Your device policy for smart lock has resulted in: time conflicted with general policy. Ask admin for further access.

SMARTTHINGS 48m ago
Alice's Home
Your device policy for smart fire alarm has resulted in: restricted device. Higher priority needed.

SMARTTHINGS 48m ago
Alice's Home
Your new user addition in the system has resulted in: Success.

(a) User Management (b) Policy Management (c) Instruction Set (d) Notification System

Fig. 8. User interfaces of Kratos+.

- *Value conditions (C_2):* As smart home devices offer value-specific operations, 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. This is an optional parameter for assigning policies in Kratos+.
- *Location (L):* Users could define location-specific operations for smart home devices by assigning "yes" or "no". Kratos+ interprets "yes" and "no" as binary numbers ("1" and "0", respectively) for policy negotiation and generation.

Figure 8 shows the user interface of Kratos+ implemented in the Samsung SmartThings platform. The information on new users and device policies is forwarded to the cloud-integrated backend server and policy generation module. After performing policy negotiation and generation, the policy generator's outcome is sent back to the user interface in the form of push notifications in the controller device (smartphone or tablet). The notifications inform the user when a policy is successfully generated or why the policy generator failed to create a new policy. Figure 8(d) shows the notification system of Kratos+ integrated into the Samsung SmartThings platform.

Cloud Integration. The majority of the smart home devices are resource-constrained devices and do not have the capability to run customized apps within the device [43]. Some smart home platforms, such as Samsung SmartThings, offer customized app installation. However, these apps run in the smart home hub or cloud to get the necessary processing capacity. Hence, we implemented the backend and policy manager module of Kratos+ as a cloud-integrated solution to minimize the overhead in resource-constrained smart home devices. As Kratos+ supports multiple smart home platforms in the same physical environment, the backend server collects and merges user-defined policies from different platforms and builds a device policy list. Additionally, the backend server builds the user priority list by collecting user priorities assigned via the Kratos+ app. The policy manager uses the user priority and device policy list to identify conflicting policies and starts the policy negotiation process between targeted users. Upon successful negotiation, the policy manager generates the final policies for installed smart devices and forwards the policies to the policy enforcement module.

Policy Enforcement. The final step during implementation is to enforce the generated policies by Kratos+ in the smart home system. As mentioned earlier, Kratos+ can enforce policies both

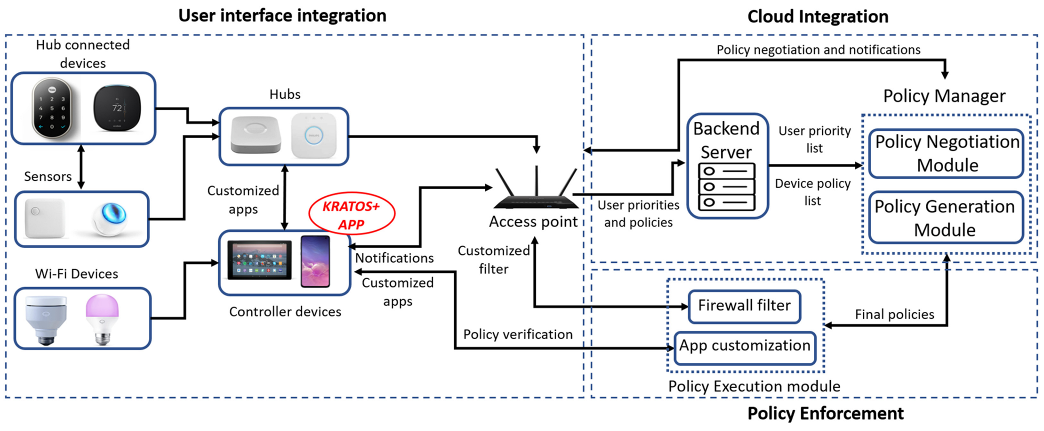


Fig. 9. Kratos+ implementation in a multi-platform smart home environment.

at the app level and access point level. Here, we explain how app-level and access point-level implementation work.

App-level Implementation. We implemented a tool similar to the available static analysis tool to modify the apps to connect with the backend server and capture the generated policies from the policy generator [7, 8]. The static tool also appends the policies in the form of conditional statements to execute the policies. A sample modified app is given in Appendix A to illustrate the steps to enforce policies in smart home apps. While sending commands from an app, Kratos+ verifies the conditional statements appended in the app before executing the command.

Access Point Level Implementation. To enforce policies in the access point (router/hub), Kratos+ builds a user-policy table from the generated policies and creates a firewall filter to control incoming commands to the smart home devices. The user policy table consists of: (1) user ID, (2) controlling device MAC ID, (3) targeted device MAC ID, (4) generated policy, and (5) decision (accept/reject). Any incoming commands from the controlling app are checked and matched with the user policy table at the access point. Here, Kratos+ scans the incoming traffic to extract source MAC ID, destination MAC ID, and time to validate the command. The customized firewall filter checks the MAC ID to detect incoming commands from the targeted users and it allows or drops the commands based on final generated policies. As the incoming traffic either from local device or cloud includes destination MAC ID which can be extracted even if encrypted [1, 58, 64], access-point level implementation can successfully allow or drop user commands. However, due to encrypted payload, access point level implementation cannot identify the value-based operation of smart home devices (details in Section 8.2). Figure 9 shows the implementation of Kratos+ in a multi-platform SHS.

5.2 Data Collection

To test the effectiveness of Kratos+ in a real-life smart home system, we implemented a smart home environment where users can interact with the smart home devices and assigned desired device policies via Kratos+ app. We considered both a single platform and multi-platform smart home environments for testing Kratos+.

Single platform SHS: A single-platform SHS offers a hub-connected multi-device environment where all the installed smart devices are connected via a hub. We chose the Samsung SmartThings platform to design the single platform smart home environment because of its large app market

Table 3. Devices and Sensors Used in Our Smart Home Setup to Evaluate Kratos+

Device Type	Model	Quantity
Smart Home Hub	Samsung SmartThings Hub	1
Smart Light	Philips Hue Light Bulb	4
	LIFX Smart Bulb	2
Smart Lock	Yale B1L Lock with Z-Wave Push Button Deadbolt	1
Smart Camera	Arlo by NETGEAR Security System	2
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

and compatibility with other smart devices [18]. We implemented the Kratos+ app as a third-party app to collect user priorities and device policies assigned by the users. For policy enforcement, we selected 10 different official Samsung SmartThings apps that control 17 different devices and installed them in the system. We modified the apps to append generated device policies and test the efficacy of Kratos+ in a multi-user multi-device environment.

Multi-platform SHS: For multi-platform SHS, we considered three different smart home platforms and devices in a single physical environment. We chose Samsung SmartThings, Philips HUE, and LIFX smart bulbs as smart home platforms. We specifically chose these platforms for third-party app integration and app customization capabilities [22, 28, 41]. Here, the multi-platform SHS has both hub connected (Samsung SmartThings) and independent entity (Philips HUE and LIFX Bulb). All the installed smart home devices from different platforms share the same network access point to perform different tasks. We customized and implemented Kratos+ app in each of these smart home platforms to collect user priorities and device policies assigned by the users. For Kratos+ app customization, we used SmartThings API for Samsung SmartThings, LIFX API for LIFX smart lights, and HUE API for Philips Hue. All the customized versions of Kratos+ app forward the device policies assigned by the users to the cloud-integrated backend server and policy manager to start policy generation and generate final device policies. In the multi-platform SHS, we implemented a customized firewall filter in the network access point (Wi-Fi router) to enforce the generated policies by Kratos+.

The complete list of devices in our smart home environment is provided in Table 3. 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 smart home settings [53]. We also used three different types of sensors: motion, temperature, and contact sensors to provide autonomous control.

In our implementation phase, we collected data from 43 different smart home users. We obtained the necessary Institutional Review Board (IRB) approval to collect access control data from real-life smart home users. We selected current smart home users who live in a shared home environment to participate in our study by circulating university-wide open calls and community outreach via emails and flyers. We grouped our participants into 14 different groups and asked them to choose different roles in a smart home system. 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 policies that include at least two policies from multiple users), including 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 in the SmartThings marketplace (light control with motion sensor, door sensor, and luminance level, respectively). We 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 on different devices using one single app. We collected 32 sets of policies in this scenario, including 12 hard, 15 soft, and 3 restriction conflicts.

Scenario 5: Multiple smart platforms in a same physical environment. We considered smart home devices from different platforms in the same smart home environment. We chose 17 devices from three different smart home platforms (Samsung SmartThings, Philips HUE, LIFX) installed in the same physical environment. We asked users to enforce device policies on different devices using the corresponding Kratos+ app. We collected 28 sets of policies in this scenario, including 10 hard, 8 soft, and 4 restriction conflicts.

Scenario 6: Temporary users in the system. We considered a temporary user is added to 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 7: 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 restrictions 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.

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? (Section 6.1)

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

Table 4. Different Usage Scenarios and Outcomes of Kratos+

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 restricts 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 two days. After two 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 restricts 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+ disregards 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.

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 the policy negotiation module to solve the conflicts. For instance, let us assume Alice and Bob have the same priority level: 2, and assign temperature ranges 60–70 and 75–80, respectively. Kratos+ considers this as a hard competition conflict and starts the negotiation process with an average range of 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) to the system for two days. After the validity period (two days), Gary tries to access the smart home devices. However, Kratos+ automatically detects any expired validity of the users in the system and restricts the temporary users from accessing the system. Table 4 summarizes the outcome of Kratos+ in different usage scenarios. Additionally, Table 5 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. The 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. For the multi-platform smart home environment in Scenario-5, Kratos+ evaluated 28 sets of policies in three different smart home platforms. Here, Kratos+ resolves 10 hard conflicts and 8 soft conflicts in an average policy generation time of 0.53 seconds. In Scenario-6, Kratos+ successfully manages 30 sets of policies and automatically detects unauthorized access for expired temporary access. For location-based access in Scenario-7, Kratos+ successfully manages

Table 5. Kratos+'s Performance in Different Scenarios

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	28	10	8	4	6	0.53	100%
Scenario-6	30	6	9	6	9	0.2	100%
Scenario-7	30	10	8	8	4	0.32	100%

Table 6. Performance of Kratos+ Against Different Threats

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

30 sets of policies and provides location-based access 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 50 malicious occurrences in total to evaluate Kratos+ against these threats. Table 6 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 an average notification time of 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.

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 7)
RQ4. What is the impact of Kratos+ in executing a user command in the SHS via the smart home apps? (Table 8)
RQ5. How does the impact of Kratos+ change with different parameters in the SHS? (Figure 10)

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 an SHS and the smart apps to analyze given policies and solve conflicts. Table 7 illustrates the delay introduced by Kratos+ while handling 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

Table 7. Overhead of Kratos+ in Handling Policy Negotiations

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 8. Overhead of Kratos+ in Policy Executions

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%

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.

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 8 depicts the impact of Kratos+ on executing user commands based on generated policy. Here, we 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). In contrast, 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 [46–48].

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 the delay imposed by Kratos+ in the SHS's normal operation. In Figure 10, the delay introduced by Kratos+ is shown based on the number of policies, conflicts, users, and devices. One can notice from Figure 10(a), the delay introduced by Kratos+ increases with the number of

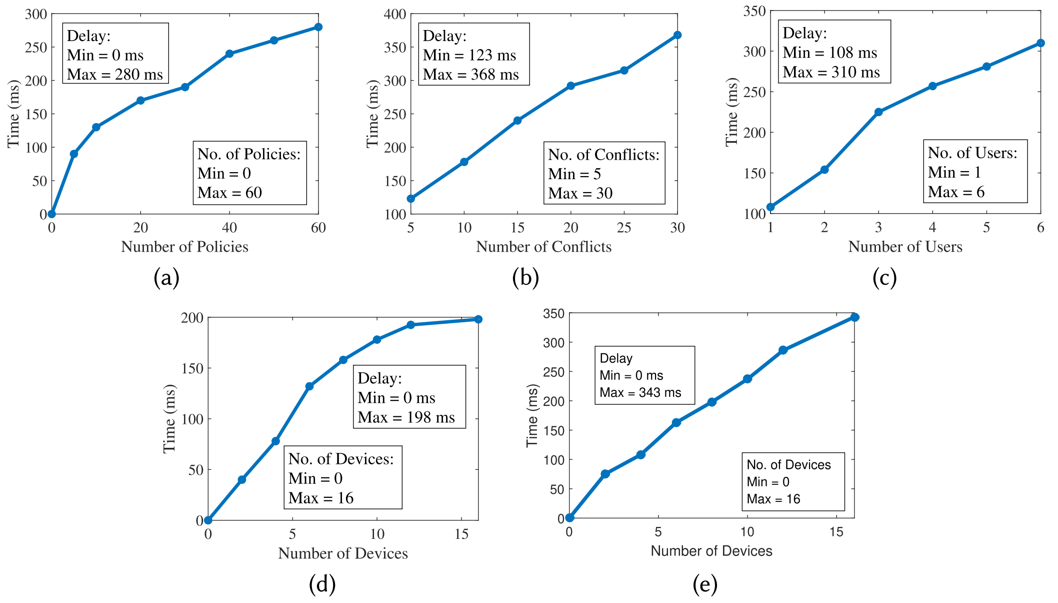


Fig. 10. Impact of different evaluation parameters on Kratos+'s performance: (a) number of policies, (b) number of conflicts, (c) number of users, (d) number of devices in single platform SHS, and (e) number of devices in multi-platform SHS.

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 (Figures 10(b) and 10(c)). 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 present 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 in a single platform SHS, the delay introduced by Kratos+ becomes steady after adding 12 different devices in the SHS (Figure 10(d)). For multi-platform SHS, the delay becomes higher as Kratos+ takes a slightly higher time to execute and verify user policies in a multi-platform environment. However, the delay is still low (343 ms for SHS with three different platforms and 16 devices), which indicates efficiency in multi-platform SHSs.

7 USABILITY STUDY

To understand the usability of Kratos+ among users, we also performed a usability study with 43 smart home users. Although it is not the primary goal of this work, it is important to understand the users' perspectives on the usability effectiveness of Kratos+. Again, we obtained Institutional Review Board (IRB) approval, and we gave monetary compensation to the users to test our proposed access control system. For selecting participants, we focused on recruiting existing smart home users with - (1) basic understanding of app installation from online marketplaces, (2) minimum knowledge of controlling smart home devices via controller apps/smartphones, and (3) aware of access control needs in the smart home system. We selected users from our user study participants (Section 2.2) experience of using and sharing smart home devices with multiple users in their households. In this study, users experienced the proposed access control system in a real-life smart home environment supported by Samsung SmartThings. We created Kratos+ app for

Table 9. Summary of the Usability Study of Kratos+

Category	Rating	Category	Rating
User interface	★★★★★	Processing time	★★★★☆
Tutorial	★★★★★	Policy generation	★★★★★
Installation process	★★★★☆	Conflict resolution	★★★★★
Notification system	★★★★★	User restriction	★★★★★
Availability	★★★★★	User-friendly	★★★★⯪
Ease	★★★★⯪	Effectiveness	★★★★★

Samsung SmartThings and made it available to the users to install and use it to add new users, add demand policies, add restriction policies for specific users, and experience policy conflict resolution provided by Kratos+. For testing conflict resolution, we have selected seven different multi-user scenarios explained in Table 4. Users were asked to test each of the scenarios and provide feedback regarding the usability of Kratos+. For selected scenarios (hard and soft competition conflicts), we asked the users to test Kratos+ in pairs to evaluate the usability of Kratos+ in a multi-user smart home environment. More details of the questions included in the usability survey are provided in Appendix C. The questions included in the usability study were divided into three different categories:

- *Installation and tutorial:* In this part, users were asked to install the Kratos+ app in the system and learn how to use Kratos+ in the smart home systems.
- *Policy enforcement and notification system:* In the second part of the usability test, users were asked to create different types of policies (demand and restrict policy) using Kratos+ and experience the notification system implemented in Kratos+.
- *Policy conflict and implementation:* In the last part, users experience the conflict resolution of Kratos+ and observe the implemented policies in the system.

In the following, we summarize the findings of the usability study and discuss how users took Kratos+ in a smart home system. A summary of the study is given in Table 9.

Installation and tutorial. 95.3% of the users installed the app successfully using the instructions provided in the app, and 97.7% of the users thought the provided tutorial was adequate to operate the app and perform different functions successfully. In terms of device availability for policy enforcement, Kratos+ scored 5 on a scale of 5.

Policy enforcement and notification system. In terms of priority assignment, 93% of users understood and correctly added new users to the system. For assigning demand policies, 100% of the users successfully enforced and understood the notifications correctly. 97.7% users understood the notification messages clearly.

Policy conflict and implementation. In this part, users experience how Kratos+ implemented the generated policies in the system and resolve conflicts between different user demands. Finally, 97.7% of users were satisfied with the demand policy decisions generated by Kratos+ while 100% of the users were satisfied with the restriction policy decisions.

8 BENEFITS AND LIMITATIONS OF KRATOS+

8.1 Benefits of KRATOS+

Consider a user, Bob, who defines himself as a technology-savvy person and owns a smart home. The home is set with devices such as a smart lock, thermostat, fire alarm, and smart coffeemaker. Bob is the head of a family of three members, including his wife Alice, and his teenage son Matt. Finally, Bob is an enthusiastic entrepreneur that 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 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 considers 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 a cooler temperature). Kratos+ offers an access control mechanism for the SHS that allows Bob to provide access control based on the users' needs and priorities.

Multi-users/Multi-devices. As mentioned before, Bob's setup comprises several different devices with different usability levels based on their impact on the quality of life of users and their contribution to the general protection and security of the household. Additionally, different users may have different access levels based on Bob's and the household's best interests. Bob expects a smart home access control system capable of managing multi-user and multi-device environments based on these scenarios. 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 that consider the importance of the various devices and the needs of the users based on the 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 system increases complexity due to new conflicts between users and 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 ecosystem. First, security-related devices (e.g., smart locks) have the highest priority. Second, he desires to have strict and unique control over these devices so that no other user can change their settings or expected behavior. Finally, users with the lowest priority (e.g., 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.

Intuitive and Easy User Interaction. Finally, Bob desires a user-friendly tool, especially because some users with little technical knowledge may need to interact with the new access control system. Kratos+ addresses all the steps from gathering users' declared demands and policies to access control enforcement with minimal user interaction. For this, our framework learns from the different priorities of users and devices to create efficient and fully automated policy conflict resolution mechanisms.

Platform-independent Implementation. As a technology enthusiast, Bob intends to try smart home devices from different vendors. However, smart home devices come with different vendor-specific associated smart apps to control the devices. Hence, Bob cannot find a common access control system that allows centralized monitoring for all the installed smart devices regardless of their vendors and platforms. Kratos+ offers a platform-independent access control system that enables users to control installed smart devices from different vendors using a single app (Kratos+ app). Additionally, Kratos+ allows automatic conflict resolution and executes user-defined policies in different devices in a smart home environment regardless of device vendors and platforms.

Encrypted Sensitive Data. Kratos+ accumulates user preferences, device usage, and connected users' credentials which can be considered as sensitive data. These data should be encrypted to ensure the privacy of the users. Kratos+ is implemented in Samsung SmartThings which uses encrypted communication channels between smart home devices and the controller devices (smartphones, tablets, etc.). Furthermore, we used an encrypted cloud space (Google Cloud) to store the user priority list and generated device policies to ensure user privacy in Kratos+.

Cost vs. Users' Needs. The initial design goals of Kratos+ are derived from the user study conducted in Section 2.2. While the user study indicates the demand for fine-grained access control among users, this raises a question of cost vs. users' needs on the vendor side. As users always want to have more flexibilities and options in the access control system, integrating a fine-grained access control can introduce additional vendors' costs such as operational, maintenance, and upgrade costs. The current version of Kratos+ is built as a smart home app that users can easily download and install on their controlling devices (e.g., smartphone, tablet) without introducing any additional cost from the vendors. For policy execution and implementation, Kratos+ uses static analysis technique [7] to append the generated policies in smart home apps. In our future work, we plan to extend Kratos+ as an online web interface (similar to our prior work [23]) where users can easily upload and automatically modify a smart home app to enable Kratos+. The only additional cost for the vendors is integrating and maintaining the cloud backend of Kratos+ which can be low due to the existing cloud-based architecture of smart home systems. One possible future research direction is investigating cost vs. users' needs study from both users' and vendors' perspectives to improve the operation of Kratos+.

8.2 Limitations

Closed-source Vendor Apps. Modern smart home devices offer users to control devices either by customized or vendor-specific apps. While the customized smart home apps (e.g., apps from the Samsung SmartThings app market) can be modified to enable Kratos+, vendor-specific smartphone apps (e.g., Philips Hue) are closed-source and cannot be amended [6]. In this, a rationale way to implement Kratos+ with vendor-specific apps is to execute the policies in the access point. Currently, vendors are offering open-source APIs [22, 28] to allow users to customize and build their own rules which can be modified by Kratos+ (similar to Samsung SmartThings app in Appendix A). Another solution can be the use of an open-source smart home hub (e.g., OpenHab, Home Assistant, etc.) to control each user command coming from a vendor-specific app [6].

Dependencies on MAC ID. Kratos+ implements the generated policies at the app-level or access point-level. While the app-level implementation does not raise any additional dependency in SHS, implementing policies at the access point can introduce dependencies on MAC ID. Kratos+ uses MAC ID to identify the source and destination of a user command and control access based on implemented policies. However, prior works demonstrated MAC ID spoofing on smart home devices can redirect policy-enforced commands to a fake device [29]. One solution is to use encrypted traffic for each user [33] to ensure operational integrity. However, this will increase overhead in resource-limited smart home devices. Again, encrypting traffic reduces the visibility of incoming user commands at the access point level, affecting the fine-grain access control feature of Kratos+. For example, with access-point policy execution, Kratos+ can match time and targeted device ID with generated policies limiting access control to time-based operation only. These open problems can be good candidates for future research.

9 RELATED WORK

Rather than providing fine-grained user access control, most of the prior works emphasize limiting malicious activities via controlling app access [12, 13, 15]. Moreover, several works focus on device access control and authentication on an IoT network for single-user scenarios [2, 11, 24, 34, 37, 39].

In 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 [20]. Zeng et al. discuss their findings related to security and privacy concerns among smart home users [62]. In both works, smart home users clearly raise their concerns regarding the need for an SHS access control mechanism. In addition, these studies also summarize several design specifications to reflect users' needs in an access control mechanism. Matthews et al. also point out relevant issues with smart home users that share the same devices and accounts [32]. In a recent user study, Garg et al. explored the constraints of sharing smart home devices among users with different social relations and listed future design requirements for smart home access control [16]. Dutta et al. proposed a context-aware access control mechanism *PALS* that uses an ABAC to understand the context of a device's policies and enforce policies to the devices [14]. However, no explicit solution for multi-user access and conflicting user needs 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 [30]. Ni et al. presented *DiffUser*, a user access control model for the Android environment based on access privileges [36], which is only effective for a single device. Tyagi et al. discussed several design specifications needed for multi-party access control in a shared environment [57]. 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 [19]. However, this system is not flexible enough to express the real needs of the users. Jang et al. presented a set of design specifications for access control mechanisms based on different use scenarios of multi-user SHS [25]. Schuster et al. proposed a situation-based access control in the smart home system, which considers different environmental parameters [42]. Here, the authors considered the state of the device and the users' location 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 [61]. However, *Expat* does not consider a multi-user multi-device smart home environment in the design and can not solve conflicting user demands at run-time. As an extension to this work, researchers proposed *PatrIoT*, which monitors app behavior at run-time to detect policy violations in smart home devices [60]. Alrumayh et al. proposed an audio-based access control system, *CANVAS*, which uses audio signals to identify the context of a user command and provides access to smart home devices based on preset rules [3, 4]. Although *CANVAS* addresses the basic need for access control in multi-user SHS, the dependency on voice assistants and voice training data minimizes the usability of the proposed solution in multi-platform SHS and standalone smart home devices. Lee et al. proposed an authorization scheme that ensures secure device access delegation for smart home users and devices [27]. Tabassum et al. conducted a survey among smart home users to understand the need for guest access policies in smart home devices and proposed a community-based access control scheme to ensure secure guest access in smart home environments [54]. However, both [27] and [54] only considered secure guest policies. Xue et al. presented a block-chain based overlooking the need for fine-grained access control within a multi-user smart home environment. Xue et al. presented a blockchain-based access control to ensure data security from undesired access or insider threats [59]. In recent work, Zeng et al. built an access control prototype with different access control options for smart home users [63]. Here, the authors considered four different access control mechanisms and assessed a very limited user study among seven households to understand the users' needs and improve the design. However, that work does not have the capability to handle real-life systems, is inflexible, not configurable, and did not consider user conflicts in a multi-user smart environment.

Table 10. Comparison Between Kratos+ and Other Access Control Mechanisms for Multi User Environment

Prior Work	Domain	Multi-user Multi-device environment	Multi-platform environment	User interface	User conflict resolution	Overhead analysis	Access control language	Usability study	User study
xShare [30]	Smartphone	○	○	●	○	●	○	●	●
DiffUser [36]	Smartphone	○	○	●	○	●	○	○	○
Capability-based access control [19]	IoT network	●	○	●	○	○	○	○	○
Situation-based access control [42]	Smart home	●	○	○	○	○	●	○	○
Expat [61]	Smart home	●	○	○	○	●	●	○	○
PALS [14]	Smart home	●	○	○	○	○	○	○	○
PBAC [59]	Smart home	●	○	○	○	●	○	○	○
PatrIoT [60]	Smart home	●	○	○	○	●	○	○	○
Zeng et al. [63]	Smart home	●	○	●	○	○	○	●	●
CANVAS [4]	Smart home	●	○	●	○	●	○	○	○
Kratos+	Smart home	●	●	●	●	●	●	●	●

Differences from existing works. Traditional access control systems such as **Attribute-Based Access Control (ABAC)**, **Role-Based Access Control (RBAC)**, or even classic Linux environment with multiple users sharing multiple I/O devices cannot address the complex and conflicting demands of users with multiple devices in the burgeoning smart home systems. Moreover, smart home devices come with limited computational resources (e.g., RAM, CPU); they are dynamic, moving without static parts and configurations. As different smart devices have different attributes, modes of operations, and dependencies, it is nontrivial to simply adapt ABAC/RBAC/Linux in a smart environment. Again, users in a smart environment can have diverse roles, time dependencies, and complex social relations and demands which complicates the efficient adaptation of earlier traditional mechanisms in a multi-user smart environment. Kratos+ was built upon considering the user study conducted among real-life smart home users and results from prior user studies [20, 63] to address the access control needs and shortcomings of existing smart home platforms. In summary, Kratos+ presents an access control system that differs from the existing works in the following ways: (1) Kratos+ considers multi-device multi-user smart home environment and supports multiple smart home platforms; (2) Kratos+ considers the conflicting desires of the users and provides a fine-grained mechanism to express complex demands from the users and devices; (3) Kratos+ offers easy new user addition with priority levels and a policy language to assign device policies to the smart home devices via users' smartphone/tablet; (4) Instead of relying on vendor-specified guest policies, Kratos+ offers an automated policy control mechanism that updates user policies and remove/revoke undesired/expired guest access policies from the smart home systems; (5) Kratos+ introduces an automatic policy negotiation engine that detects and resolves policy conflicts among users; (6) Kratos+ considers a realistic threat model arising from over-privileged users and system implementation flaws; (7) We implemented Kratos+ in a real-life smart home environment with different smart home devices and platforms and tested with smart home users. Our evaluation shows that Kratos+ can effectively address diverse access control needs with minimum overhead. Moreover, our usability study indicates efficacy in real-life deployment. Table 10 summarizes the differences of Kratos+ from other existing solutions.

10 CONCLUSION

In a smart home system (SHS), multiple users have access to multiple devices simultaneously. In these settings, multiple users may want to control and configure the devices with different preferences, giving rise to complex and conflicting demands. In this paper, we explored the need for a fine-grained access control mechanism in smart home systems based on real users' demands and developed Kratos+, an access control system that addresses the diverse and conflicting demands of different users in a shared multi-user smart home system. Kratos+ is the first work implementing a priority-based access-policy negotiation technique based on real-users' needs to resolve conflicting user demands in a shared smart home system with multiple users and devices in an automated and configurable fashion. We implemented Kratos+ in real-life settings and evaluated its performance through real devices in a multi-user setting. Kratos+ successfully covers the users' needs, and our extensive evaluations showed that Kratos+ effectively resolves the conflicting requests and enforces the policies without significant overhead. Also, we tested Kratos+ against five different threats and found that Kratos+ effectively identifies the threats with high accuracy.

APPENDICES

A KRATOS+ ENFORCED IN A SAMSUNG SMARTTHINGS APP

We evaluated the effectiveness of Kratos+ by enforcing policies in Samsung SmartThings Apps. We selected eight Apps to enforce Kratos+ generated user policies and appended the policies inside the App. Here, we give an example of Kratos+-enabled official SmartThings App.

```
import groovy.time.*
definition(
    name: "Big Turn ON modified",
    namespace: "smartthings",
    author: "Anonymous",
    description: "Turn your lights on when the SmartApp is tapped or activated.",
    category: "Convenience",
    iconUrl: "https://s3.amazonaws.com/smartapp-icons/Meta/light_outlet.png",
    iconX2Url: "https://s3.amazonaws.com/smartapp-icons/Meta/light_outlet@2x.png"
)
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: $evt"
        switches?.on()
}
def getSmartLightJsonData(){
    def listTimes = []
    def listAdmins = []
    def listUsers = []
    def listIDs = []
    def listTimeStarts = []
    def listTimeEnds = []
    def params = [uri: "https://mywebserver/xxxyyyzzz/2/public/values?alt=json",]
    try {
        httpGet(params) { resp ->
            //log.debug "${resp.data}"

            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)
            /*for (listtime in atomicState.SmartLightTimes){
                log.debug "${listtime}"
            }*/
        }
    } 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
 }
```

Listing 1. Policy enforced at install time

B USER STUDY EXAMPLE QUESTIONS

We presented a representative set of questions from all the categories. We will make the full set of questions publicly available at https://github.com/Amitksik/KRATOS-Access-control-for-smart-home.

B.1 User Characterization

(1) While using/ installing a smart home device, did you have to consider suitable settings for other users in your home?
( ) Yes
( ) No

(2) While using a smart home device, do you have to change device settings each time other users change the setting?
( ) Yes
( ) No

B.2 Smart Home Setting Preferences

(1) Do you think current smart home platforms should provide an integrated access control system?
( ) Yes
( ) No
( ) Maybe

(2) In order to provide access control, would you be willing to install and use a separate app in addition to the traditional app controlling the devices?
( ) Yes and I am willing to use and pay for the service, if there is a fee
( ) Yes if the app is free and trusted
( ) Maybe
( ) No, because it is too much hassle
( ) No, because I don't need access control

B.3 Multi-user Multi-device Scenarios

(1) In your smart home, you and your parent/partner/roommate has the same level of priorities. Both of you want to change the setting of a device in different ways. Do you think an automatic negotiation system would help you to solve this?
( ) Yes
( ) No

(2) You already have a policy for a device in the access control system. You want to modify the previous policy and enforce a new policy. Do you think an access control system should have this function?
( ) Yes
( ) No
( ) Maybe

C USABILITY INSTRUMENT

We present a representative set of questions from all the categories. The full set of questions can be found at https://github.com/Amitksik/KRATOS-Access-control-for-smart-home.

C.1 Installation and Tutorial

(1) Does the app provide an organized and easy-to-follow user interface?
( ) Yes
( ) No

(2) On a scale of 1 to 5 (1 being too hard to follow), how easy is it to understand the information provided via the notification(s) in Kratos+?

( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

C.2 Policy Enforcement and Notifications

(1) Does Kratos+ detect any invalid user in the system with no assigned priority and provide feedback via notifications?
( ) Yes
( ) No

(2) On a scale of 1 to 5 (1 being too slow), how quick does Kratos+ notify users upon successful transaction?

( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

C.3 Policy Conflict and Implementation

(1) In a smart home system, an admin might need to restrict the use of a specific device for some users. Does Kratos+ provide this option in policy enforcement?
( ) Yes
( ) No

(2) On a scale of 1 to 5 (1 being really hard to use and 5 being user-friendly), how easy is it to install and use Kratos+?

( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5

REFERENCES

[1] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. 2020. Peek-a-boo: I see your smart home activities, even encrypted!. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 207–218.

[2] 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.

[3] Abrar S. Alrumayh, Sarah M. Lehman, and Chiu C. Tan. 2019. ABACUS: Audio based access control utility for smarthomes. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. 395–400.

[4] Abrar S. Alrumayh, Sarah M. Lehman, and Chiu C. Tan. 2020. Context aware access control for home voice assistant in multi-occupant homes. *Pervasive and Mobile Computing* 67 (2020), 101196.

[5] Leonardo Babun, Z. Berkay Celik, Patrick McDaniel, and A. Selcuk Uluagac. 2021. Real-time analysis of privacy-(un) aware IoT applications. In *Privacy Enhancing Technologies Symposium (PETS)*.

[6] Leonardo Babun, Kyle Denney, Z. Berkay Celik, Patrick McDaniel, and A. Selcuk Uluagac. 2021. A survey on IoT platforms: Communication, security, and privacy perspectives. *Computer Networks* 192 (2021), 108040.

[7] Leonardo Babun, Amit Kumar Sikder, Abbas Acar, and A. Selcuk Uluagac. 2022. The truth shall set thee free: Enabling practical forensic capabilities in smart environments. In *Proceedings of the 29th Network and Distributed System Security (NDSS) Symposium*.

[8] 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.

[9] Z. Berkay Celik, Patrick McDaniel, and Gang Tan. 2018. Soteria: Automated IoT safety and security analysis. In *USENIX Annual Technical Conference (USENIX ATC)*.

[10] 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.

[11] 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.

[12] Adrien Cosson, Amit Kumar Sikder, Leonardo Babun, Z. Berkay Celik, Patrick McDaniel, and A. Selcuk Uluagac. 2021. Sentinel: A robust intrusion detection system for IoT networks using kernel-level system information. In *Proceedings of the International Conference on Internet-of-Things Design and Implementation*. 53–66.

[13] 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*.

[14] Sofia Dutta, Sai Sree Laya Chukkapalli, Madhura Sulgekar, Swathi Krithivasan, Prajit Kumar Das, and Anupam Joshi. 2020. Context sensitive access control in smart home environments. In *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. 35–41.

[15] 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*.

[16] Radhika Garg and Christopher Moreno. 2019. Understanding motivators, constraints, and practices of sharing Internet of Things. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 2 (2019), 1–21.

[17] 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.

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

[19] 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.

[20] 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.

[21] Weijia He, Valerie Zhao, Olivia Morkved, Sabeeka Siddiqui, Earlence Fernandes, Josiah Hester, and Blase Ur. 2021. SoK: Context sensing for access control in the adversarial home IoT. In *IEEE European Symposium on Security and Privacy (EuroS&P)*. 37–53.

[22] Philips Hue. 2018. *How to Develop for Hue: Hue API*. https://developers.meethue.com/develop/hue-api/.

[23] IoTDots. 2021. *IoTDots Modifier*. https://iotdots-modifier.appspot.com/.

[24] 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.

[25] 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.

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

[27] Tam Le and Matt W. Mutka. 2019. Access control with delegation for smart home applications. In *Proceedings of the International Conference on Internet of Things Design and Implementation*. 142–147.

[28] LIFX. 2018. *LIFX http API*. https://api.developer.lifx.com/.

[29] Zhen Ling, Junzhou Luo, Yiling Xu, Chao Gao, Kui Wu, and Xinwen Fu. 2017. Security vulnerabilities of Internet of Things: A case study of the smart plug system. *IEEE Internet of Things Journal* 4, 6 (2017), 1899–1909.

[30] 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.

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

[32] 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.

[33] Abhishek Mishra and Abhishek Dixit. 2018. Resolving threats in IoT: Id spoofing to DDOS. In *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 1–7.

[34] A. K. M. Iqtidar Newaz, Amit Kumar Sikder, Leonardo Babun, and A. Selcuk Uluagac. 2020. HEKA: A novel intrusion detection system for attacks to personal medical devices. In *IEEE Conference on Communications and Network Security (CNS)*. 1–9.

[35] A. K. M. Iqtidar Newaz, Amit Kumar Sikder, Mohammad Ashiqur Rahman, and A. Selcuk Uluagac. 2021. A survey on security and privacy issues in modern healthcare systems: Attacks and defenses. *ACM Transactions on Computing for Healthcare* 2, 3 (2021), 1–44.

[36] 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.

[37] 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.

[38] RemoteLock. 2018. *Smart Locks by RemoteLock*. https://www.remotelock.com/smart-locks.

[39] 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.

[40] Samsung. 2018. *How do I share my location and manage users in SmartThings Classic?* https://tinyurl.com/y86unolb

[41] Samsung. 2018. *Samsung SmartTings Development Guide*. https://developers.smartthings.com/.

[42] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. [n.d.]. Situational access control in the Internet of Things. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1056–1073.

[43] Anuj Sehgal, Vladislav Perelman, Siarhei Kuryla, and Jurgen Schonwalder. 2012. Management of resource constrained devices in the Internet of Things. *IEEE Communications Magazine* 50, 12 (2012), 144–149.

[44] 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. [Online; accessed 9-November-2017].

[45] Amit Kumar Sikder. 2020. *A Comprehensive Security Framework for Securing Sensors in Smart Devices and Applications*. Ph.D. Dissertation. Miami, FL, USA.

[46] 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.

[47] 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).

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

[49] 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.

[50] Amit Kumar Sikder, Leonardo Babun, Z. Berkay Celik, Abbas Acar, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A. Selcuk Uluagac. 2020. Kratos: Multi-user multi-device-aware access control system for the smart home. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 1–12.

[51] Amit Kumar Sikder, Leonardo Babun, and A. Selcuk Uluagac. 2021. Aegis+ a context-aware platform-independent security framework for smart home systems. *Digital Threats: Research and Practice* 2, 1 (2021), 1–33.

[52] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, and A. Selcuk Uluagac. 2021. A survey on sensor-based threats and attacks to smart devices and applications. *IEEE Communications Surveys & Tutorials* 23, 2 (2021), 1125–1159.

[53] Statista. 2017. *Ownership of Smart Home Technology Products in the United States in 2017 (in Million Households/Units in Use), by Category*. https://www.statista.com/statistics/757684/smart-home-technology-product-ownership-in-the-us-by-category/.

[54] Madiha Tabassum, Jess Kropczynski, Pamela Wisniewski, and Heather Richter Lipford. 2020. Smart home beyond the home: A case for community-based access control. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–12.

[55] Trefis Team. [n.d.]. Why Smart Home Devices are a Strong Growth Opportunity for Best Buy. https://www.forbes.com/sites/greatspeculations/2017/07/05/why-smart-home-devices-are-a-strong-growth-opportunity-for-best-buy/#2bbe77114984. [Online: accessed 9-November-2017].

[56] 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.

[57] 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.

[58] Yinxin Wan, Kuai Xu, Guoliang Xue, and Feng Wang. 2020. IoTArgos: A multi-layer security monitoring system for Internet-of-Things in smart homes. In *IEEE International Conference on Computer Communications*. 874–883.

[59] Jingting Xue, Chunxiang Xu, and Yuan Zhang. 2018. Private blockchain-based secure access control for smart home systems. *KSII Transactions on Internet and Information Systems (TIIS)* 12, 12 (2018), 6057–6078.

[60] Moosa Yahyazadeh, Syed Rafiul Hussain, Endadul Hoque, and Omar Chowdhury. 2020. PatrIoT: Policy assisted resilient programmable IoT system. In *International Conference on Runtime Verification*. Springer, 151–171.

[61] 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.

[62] 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*. Santa Clara, CA.

[63] 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*.

[64] Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. 2018. HoMonit: Monitoring smart home apps from encrypted traffic. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1074–1088.

Received March 2021; revised September 2021; accepted May 2022