Who's In Control? On Security Risks of Disjointed IoT Device Management Channels

Yan Jia[1,3,2,†], Bin Yuan[4,5,2,†,*], Luyi Xing[2,*],
Dongfang Zhao[2], Yifan Zhang[2], XiaoFeng Wang[2], Yijing Liu[1],
Kaimin Zheng[4,5], Peyton Crnjak[2], Yuqing Zhang[7,3,8,*], Deqing Zou[4,5], Hai Jin[6,5]
[1]College of Cyber Science, Nankai University, China, [2]Indiana University Bloomington, [3]School of Cyber Engineering, Xidian University, China, [4]School of Cyber Science and Engineering, Huazhong Univ. of Sci. & Tech., China, [5]{National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Hubei Engineering Research Center on Big Data Security, Cluster and Grid Computing Lab, Huazhong Univ. of Sci. & Tech., China}, [6] School of Computer Science and Technology, Huazhong Univ. of Sci. & Tech., China, [7]National Computer Network Intrusion Protection Center, University of Chinese Academy of Sciences, China, [8]School of Computer Science and Cyberspace Security, Hainan University, China

ABSTRACT

An IoT device today can be managed through different channels, e.g., by its device manufacturer's app, or third-party channels such as Apple's Home app, or a smart speaker. Supporting each channel is a management framework integrated in the device and provided by different parties. For example, a device that integrates Apple HomeKit framework can be managed by Apple Home app. We call the management framework of this kind, including all its device- and cloud-side components, a *device management channel* (DMC). 4 third-party DMCs are widely integrated in today's IoT devices along with the device manufacturer's own DMC: HomeKit, Zigbee/Z-Wave compatible DMC, and smart-speaker Seamless DMC. Each of these DMCs is a standalone system that has full mandate on the device; however, if their security policies and control are not aligned, consequences can be serious, allowing a malicious user to utilize one DMC to bypass the security control imposed by the device owner on another DMC. We call such a problem Chaotic Device Management (`Codema`).

This paper presents the first systematic study on `Codema`, based on a new model-guided approach. We purchased and analyzed 14 top-rated IoT devices and their integration and management of multiple DMCs. We found that `Codema` is both general and fundamental: these DMCs are generally not designed to coordinate with each other for security policies and control. The Codema problems enable the adversary to practically gain unauthorized access to sensitive devices (e.g., locks, garage doors, etc.). We reported our findings to affected parties (e.g., Apple, August, Philips Hue, ismartgate, Abode), which all acknowledged their importance. To mitigate this new threat, we designed and implemented `CGuard`, a new access control framework that device manufacturers can easily integrate into their IoT devices to protect end users. Our evaluation shows that CGuard is highly usable and acceptable to users, easy to adopt by manufacturers, and efficient and effective in security control.

CCS CONCEPTS

• **Security and privacy** → **Embedded systems security**; *Access control*; • **General and reference** → *Empirical studies.*

KEYWORDS

IoT, smart home, attack, device management channel, access control

ACM Reference Format:
Yan Jia, Bin Yuan, Luyi Xing, Dongfang Zhao, XiaoFeng Wang, Yifan Zhang, Yijing Liu, Kaimin Zheng, Peyton Crnjak, Yuqing Zhang, Deqing Zou, Hai Jin. 2021. Who's In Control? On Security Risks of Disjointed IoT Device Management Channels. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3460120.3484592

1 INTRODUCTION

Today's Internet of Things (IoT) are managed by different frameworks that allow users to utilize mobile apps to configure and control devices through local connections (e.g., Bluetooth) or cloud services. Some of these frameworks are provided by device manufacturers: e.g., a house owner can use the August app to lock/unlock her door [12], the Philips Hue app to turn on/off her light bulbs [30], and the ismartgate app to open/close her garage door [23].

In the meantime, the increasing diversity of IoT also gives rise to third-party solutions, which handle different devices regardless of their manufacturers. Prominent examples include Apple's *HomeKit* [11], Zigbee [39] and Z-Wave [37] compatible frameworks. In our research, we call such a framework, including all its device- and cloud-side components, a *device management channel* (*DMC*), or simply a *channel*. Today's IoT devices tend to support multiple DMCs, both the ones from manufacturers and those offered by third parties. For example, the August smart lock can be controlled by HomeKit, SmartThings hub (a Z-Wave compatible DMC), as well as August's own app. Each channel is a standalone system and their

[†]Most work was done when the first two authors were at Indiana University Bloomington.
[*]Corresponding authors: Luyi Xing, Bin Yuan, Yuqing Zhang.

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484592

integration on a device, if not done right, can have serious security implications, exposing the device to unauthorized access. With its importance, this problem, however, has never been studied before.

Risks of disjointed DMCs. Indeed our research shows that *today's DMC integration is fundamentally flawed, leaving different channels on the same device completely disjointed or inadequately coordinated in their security controls*. More specifically, with the heterogeneous architectures of different DMCs, whose security policies and enforcement components scattered across device and cloud ends, they all have a full mandate on the device, capable of independently determining whether a specific functionality (e.g., door opening) should be granted. However, such access decisions are often *uncoordinated*, with one channel operating in a way completely oblivious to other channels. As a result, any security policies configured and enforced through one channel could be circumvented through another channel. We call this risk *chaotic device management* or *Codema*.

Exploiting Codema: analysis and discoveries. To understand the real-world impacts of Codema, we focus on four major third-party DMCs (Apple HomeKit, Zigbee or Z-Wave compatible channel and BLE-based smart speaker channels) and their integration into IoT devices besides device manufacturer DMCs (Section 3). Using a model-guided approach , we systematically analyzed the security management and policy coordination among different DMCs in a set of highly popular devices, which shows that all these channels integrated in popular devices contain Codema vulnerabilities and can be easily exploited.

Particularly, third-party channels are typically disjointed from manufacturer channels and between themselves. So for a device owner who solely relies on her favorite channel, such as HomeKit, to manage her device, other channels supported on the device become unprotected and thus exposed to the malicious actor, a risk unaware to the device owner. For example, an Airbnb homeowner could manage her ismartgate garage controller through Apple HomeKit, which however leaves the manufacturer DMC "dangling": whoever *only temporarily* given the access to her home Wi-Fi (e.g., her Airbnb guest) becomes able to stealthily configure this channel to gain a permanent remote control on the garage door (Section 3.1).

Further, although some manufacturer DMCs include the mechanisms to control third-party DMCs by managing their individual policies and internal states (e.g., allowing the user using the manufacturer app to open/close a third-party channel such as HomeKit or control its network provision), we found that such mechanisms are all flawed in the absence of a proper cross-DMC management protocol enabling policy interoperability. This problem turns out to be fundamental to the design of IoT access control, as discovered in our study. For example, although the Abode smart hub is designed to manage access to its HomeKit channel through the manufacturer channel (by generating HomeKit setup code), any user with temporary access to the hub (e.g., an Airbnb guest) can acquire the setup code to stealthily take over the HomeKit channel, even after the temporary right has been revoked through the manufacturer channel. This allows the user to later disarm the home security system of the hub (Section 3.2).

Since the Codema risk is related to human behaviors (e.g., whether the owner of an IoT device leaves certain DMCs unused/open, whether she tends to temporarily grant the access to the device to the party not fully trusted), we performed a user study to answer such questions and better understand the attack feasibility. We further performed a survey to understand whether the users have been informed of the risks by device vendors in user manuals. Our results indicate that Codema attacks are highly practical against real-world users (Section 4).

Further, we discovered that the Codema risk is pervasive. We looked into 14 high-profile devices, which all turn out to be vulnerable. Examples include the Philips Hue bulb (dubbed "the Best Seller of Smart Bulb in Amazon" [31]) and the August Lock ("the most Advanced Smart Lock in Amazon" [26]). Note that these confirmed or potentially vulnerable devices cover almost all types of IoT devices, including bridges, cameras, garage door controllers, lights, locks, outlets, security systems (Section 3.3). Once such security-, privacy- or safety-sensitive devices are stealthily controlled by an unauthorized party, the consequences can be dire.

These findings provide strong evidence that the access control on today's IoT systems in the presence of multiple DMCs has not been well thought-out. We reported the results of our study to the manufacturers of all the devices we analyzed. They all acknowledged that our findings are real and significant. We have been formally recognized by HomeKit, Philips, August, ismartgate, and Abode, and are helping them fix these problems. Video demos of our attacks and parts of manufacturer responses are posted online [49].

Mitigating Codema. Given the significant impact of Codema, finding effective protections that can work with today's IoT systems is imperative. For this purpose, we introduce *Channel Guard* (CGuard), a new, light-weight access control framework for cross-DMC security management. The core idea is to have a centralized access control framework in the device to oversee and govern the accessibility of all DMCs on a device. Device manufacturers can easily integrate CGuard into their firmware to help ensure that no DMC is left in an unexpected accessibility status, such as dangling or being enabled/accessed stealthily by the attacker.

We implemented CGuard and deployed it on a proof-of-concept smart LED light we built on Raspberry Pi 3b, which supports multiple DMCs. Our evaluation shows that the prototype eliminates the Codema risk, and works properly with mainstream DMCs including Apple HomeKit and Amazon Alexa. To evaluate the usability and practicality of CGuard, we conducted another user study, which shows our approach is highly acceptable by users to enhance the security and privacy of smart homes. We make all the code publicly available on Github [15] and further discuss a clean-slate design of multiple-DMC IoT to fully solve Codema, through joint-effort across multiple DMC providers and device manufacturers (Section 6).

Contribution. The contributions are outlined as follows:

• *New understanding*. Our research reveals a new category of unexpected and security-critical weaknesses in today's IoT designs, which integrate multiple disjointed DMCs on the same device without proper coordination in place to manage their policy configuration and enforcement. We demonstrate that such weaknesses could expose many IoT devices today to realistic security risks with serious consequences. Our study brings attention to this new problem, sheds light on its fundamental causes and offers insights that can lead to its solution.

• *New technical solutions.* Based upon the understanding, we designed a new access-control framework to mitigate the Codema risks. Our approach can be easily adopted by a device manufacturer, without changing third-party DMC designs and thus working well with existing systems such as HomeKit. We implemented our design and demonstrated its efficacy and usability and open-sourced our prototype. This new technique will enhance the security quality of IoT devices, not only those already on the market but also those to be built in the years to come.

2 DEVICE MANAGEMENT CHANNELS

On an IoT device, the user console, the IoT cloud, hub, and the on-device software stack together form the DMC to allow the user to manage the device. In this section, we explain the operations of the popular DMCs integrated into mainstream IoT devices and security policies they support (Section 2.1 and 2.2). Then we summarize an abstracted state-machine model to generally describe a DMC's operations in IoT devices (Section 2.3).

2.1 Manufacture DMCs

Each manufacturer provides its own DMC(s), which we call *m-DMC*, to control its products. *m-DMC* typically has one of the following architectures:

• *Cloud-based architecture.* Many device manufacturers run a backend cloud service to support their mobile apps and enable remote control. In this architecture (see Figure 1), one can issue commands through the manufacturer app, which forwards the commands to the cloud; after proper authentication and authorization, the commands are delivered to the devices connected to the cloud through the Internet. The cloud maintains a set of security policies about users' access rights on devices, which are used to mediate the delivery of commands to the devices.

• *Local-control architecture.* Another common DMC is for local control (also see Figure 1): the device can be paired with the manufacturer app through BLE or home Wi-Fi, to establish a connection for receiving commands from the user.

• *Hub-based DMC.* Another common architecture is the hub-based *m-DMC*: the device is directly connected to a hub through local communication protocols (e.g., BLE, Z-Wave, Zigbee, etc.); the hub connects to the manufacturer cloud through the Internet and relays the messages between the device and the cloud.

2.2 Third-party DMCs

HomeKit DMC. HomeKit is Apple's framework for configuring and controlling smart-home devices, which has been widely supported by mainstream IoT manufacturers. Through Homekit, users can manage their IoT appliances using Apple's uniform management console [10], i.e., the Home app on iOS, iPadOS, etc.

To support HomeKit, the device manufacturer needs to integrate into its IoT device the `HomeKit Accessory Protocol` (HAP) library [25] (see Figure 1). HomeKit supports Wi-Fi and Bluetooth as the communication channel. The HAP library processes device operation commands received by the Wi-Fi or Bluetooth interfaces, and passes them up to the manufacturer's control program on the *Application Logic Layer* (*ALL*) (see Figure 1). The *ALL* program then calls device drivers in the Hardware Abstract Layer (HAL) to operate the IoT device (e.g., opening the lock).

The IoT device needs to be paired with the user's Apple device (e.g., iPhone with Home app): the user needs to be authenticated by the HAP library with a secret eight-digit setup code (entered in the Home app); once succeeded, she can use the Home app to establish an encrypted connection with the device – a process similar to BLE bonding [1]. The commands (e.g., open the lock) from the Home app are sent through encrypted sessions to the HAP library, which passes the commands to the ALL program.

Zigbee/Z-Wave compatible DMC. Mainstream IoT devices often support a DMC built on top of the Zigbee [40] or Z-Wave [50] communication protocol. Its in-device architecture is similar to HomeKit, as outlined in Figure 1. Specifically, the manufacturer integrates a protocol library in firmware, which we refer to as the *Zigbee/Z-Wave library*, into IoT devices with hardware supports for the protocols at the message transport layer. Similar to HAP, the *Zigbee/Z-Wave library* processes commands received from the transport layer, and passes them up to the ALL program, which then operates the device (e.g., open the lock).

To control a device through this DMC, one should first pair the device with a hub (called *Z-channel hub* in our research). For this purpose, she needs to bind the hub to her account (with the hub vendor) through the hub vendor's app. Note that the hub vendor (e.g., SmartThings) may not be the device's manufacturer (e.g., August). The follow-up device-hub pairing process can have different levels of protection, depending on the protocol version. More specifically, the early versions (e.g., Z-Wave S0 security) do not have authentication in place and the later ones (e.g., Z-Wave S2 security) come with protection similar to that for HomeKit pairing and BLE binding: the user enters a secret code (install code of Zigbee [72] and device specific key of Z-Wave [64]) into the hub app to enable the hub and the device to exchange encryption keys, which are stored and used for later secure communication between them.

Smart-speaker Seamless DMC. Smart speakers (e.g., Amazon Echo [8] and Google Home [17]) offer another DMC, allowing one to use voice or related mobile apps to control the devices. Smart speaker DMCs have two different architectures, the local architecture and cloud-based architecture. The local seamless architecture, introduced to Google Home in 2019 [5, 46], is emerging: the smart speaker connects to the IoT device using Bluetooth and sends the commands directly to the IoT device without going through the cloud. The device needs to integrate a smart speaker DMC library into the device (see Figure 1). In contrast, the traditional cloud-based architecture is more widely used: a user needs to first set up her IoT device, for example, a LIFX bulb, using the LIFX app; then after an access delegation from the LIFX server to the smart speaker's server (e.g., with an OAuth token [44] issuing to the latter), the user can issue commands to the smart speaker, and the commands go through the clouds to reach the LIFX bulb. In this paper we study the local architecture of smart speaker which presents a new DMC (called smart-speaker Seamless DMC or smart-speaker DMC), while with the cloud-based architecture the IoT device in nature uses the *m-DMC* to communicate with the smart speaker.

2.3 State Machine Model of DMCs

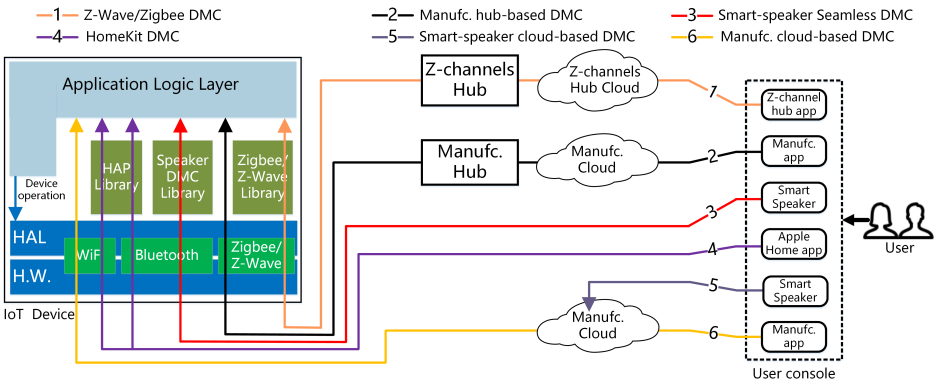

Figure 1: The different IoT control channels (Manufc. is short for Manufacture; H.W. is short for hardware)

Based on the prior models [56, 71] of IoT devices, we abstract a state-machine model to describe any DMC's operations in IoT devices. Generally speaking, a DMC on an IoT device is characterized by four states - *Factory*, *Waiting for Network*, *Waiting for Binding*, and *Running*, as illustrated in Figure 2.

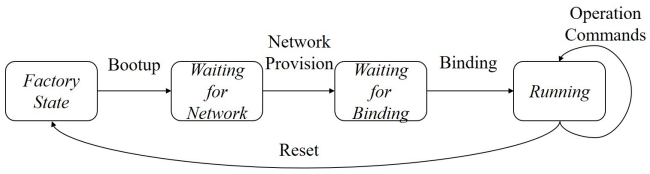

Figure 2: State machine model of a DMC

• *Factory (Fct) State.* This is the initial state of any DMC when the device is at its factory setting. After booting the device, the state will automatically transit to the next one (*WfN*, see below).

• *Waiting for Network (WfN) State.* At this state, a DMC is waiting for joining a network to communicate with the user (user app or other consoles). A *network provision* operation will drive the *WfN* state to the next state (*WfB*, see below). Here, the network provision process depends on the communication techniques utilized by specific DMCs: for example, HomeKit and many *m-DMCs* can leverage Wi-Fi, and thus the network provision involves connecting the device/DMC to the local Wi-Fi network [47]; a Zigbee-compatible DMC connects to a Zigbee hub through the Zigbee pairing process; the smart-speaker DMC leverages BLE for communication, and will involve a BLE pairing process to communicate with a smart speaker.

• *Waiting for Binding (WfB) State.* After network provision, the DMC can communicate with the user app/console and is ready to bind with a specific user (a.k.a., user binding). In general, the first user who binds the device becomes the *owner* of the DMC and has full control over it. Typically other users are not allowed to go through the binding process again. The user binding process of different DMCs can be quite different: for example, HomeKit DMC requires the user to manually enter a setup code (labeled on HomeKit-enabled device [33]) in the app; some *m-DMCs* ask one to pass a physical proximity challenge (e.g., by pushing the physical button on the Philips Hue Bridge [29]); certain *m-DMCs* (e.g, ismartgate [23], LIFX [24]) make the process much simpler: anyone can use the companion app to bind with the DMC automatically when it is not bound.

• *Running (Run) State.* The DMC in this state is ready to receive commands from the device owner and control its hosting device. Through the user console (e.g., mobile app), the owner can issue commands and manage users (e.g., adding a shared/guest user in this DMC). The permission of the shared/guest user is subject to revocation and expiry. In different DMCs, such policies are often enforced by different components of their DMC architecture (e.g., the IoT cloud, the IoT hub or the IoT device).

Under any state other than *Fct*, a reset operation can drive the state back to the *Fct* state. This can be done, for example, by pushing a button on the device (for a few seconds).

3 UNDERSTANDING CODEMA

Overview. Our research shows that those co-located DMCs in the wild are designed to independently manage an IoT device, without communicating with each other their individual policies and coordinating on their enforcement. This exposes a new attack surface, allowing an unauthorized party to leverage one DMC to silently bypass the owner's device control implemented through a different DMC. Although some manufacturer DMCs include the mechanisms to manage third-party DMCs (e.g., allowing the user using the manufacturer app to enable/disable a third-party DMC or control its network provision, see Section 3.2), we found that such mechanisms are all flawed due to the lack of proper protocols for cross-DMC security management/coordination. This problem turns out to be fundamental to the design of IoT access control, as discovered in our study on 4 leading third-party DMCs integrated in mainstream devices.

Threat model. We consider a typical use scenario where the device owner opts for some (typically one) but not all channels through her favorite app(s), such as Apple Home, to manage the device, which we believe is realistic given the hassle that configuring all DMCs incurs (Section 2). This practice has also been confirmed in our user study (Section 4).

Today, IoT devices often need to be shared with babysitters, tenants, Airbnb guests, etc., who are granted temporary access. Such a temporary permission has been considered by recent studies [52, 53, 61, 62, 69], and its real-world demand is evidenced from the descriptions of vacation rental services and related blogs [2–4, 34]. Therefore, in some cases, we assume that a malicious user may

temporarily come in close proximity (e.g., a home) to target IoT device(s): e.g., an Airbnb guest checks into a home equipped with a smart door lock and a garage controller. Through the owner's management app, she may intentionally, temporarily share some of her devices with the malicious user (e.g., the smart lock for door opening during the guest's stay).

The attacker aims to silently acquire persistent, unauthorized control on the owner's devices, including those once shared to him with his access rights later revoked. In the meantime, we consider that all components of a DMC are benign (e.g., the management app, cloud, hub, hardware and software inside the device), and the attacker does *not* physically alter the device (e.g., to disassemble the device or solder wires).

Identification of Codema flaws. To identify the Codema flaws in each device that supports multiple DMCs, we leverage a model-guided approach. Generally, given a device, we first model all its DMCs with finite-state-machines (Figure 2) considering the four states discussed in Section 2.3, particularly identifying any dependency relation between the state transitions of two DMCs: i.e., fully configuring a DMC *A* (from the *Fct* to *Run* state) requires an approval/operation step through DMC *B*. Depending on whether or not the dependency relation exists, the approach to assess the security policy coordination between two DMCs is slightly different:

• *Scenario 1: no dependency between DMC A and DMC B*: (1) our approach considers that the owner fully configures one DMC (e.g., DMC B, leaving DMC A dangling); (2) if the adversary can successfully configure DMC A to gain device control without involvement/awareness of the owner, our approach reports a potential Codema flaw and further confirm it through end-to-end attacks.

• *Scenario 2: DMC A depends on DMC B*: (1) our approach considers that the owner opts for and fully configures DMC B (leaving DMC A dangling) and shares device-access rights with the adversary; (2) leveraging the access-rights on DMC B, the adversary fully configures DMC A to gain device access; (3) later the owner revokes the adversary's access rights using DMC B; (4) at this stage, if the (unauthorized) adversary still has device-control via DMC A, our approach reports a Codema flaw and confirms it through end-to-end attacks.

Notably, all devices we tested are vulnerable affecting mainstream IoT vendors, demonstrating that the problem is general (see Section 3.3). Since exploiting Codema flaws depends on how the device owner configures and uses multiple-DMC IoT devices, we report an attack feasibility study in Section 4, which shows that Codema risks are realistic in the real world with serious practical impact on IoT security. We discuss the limitation of the current approach in Section 6.

Responsible disclosure. We reported our findings to all affected manufacturers, including Apple, August, Abode, Philips, etc., which all acknowledged the significance of the problems. Mitigation has been deployed or is on the way.

3.1 Disjointed DMC Management

The DMCs in the wild are generally designed not to interfere with each other's operations. For example, the specification of Apple HomeKit highlights that the setup process of HomeKit should not depend on any operation in the manufacturer app, for the purpose of ease of use [22]. It turns out that in mainstream devices, *a DMC's state machine is unrelated to those of others*, indicating that the DMCs are meant to work independently without coordinating their security policies. For such devices, as long as the owner leaves one DMC dangling, the adversary has the opportunity to leverage that DMC to control the device.

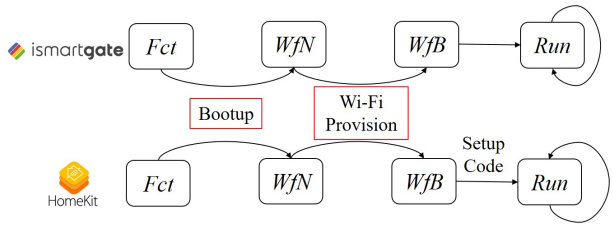

Figure 3: State machines of DMCs in the ismartgate device

Codema Flaw 1: Disjointed HomeKit and *m-DMC*. The ismartgate smart garage door controller [23] enables users to remotely open and close their garage doors through either the ismartgate app (*m-DMC*) or the Apple Home app (HomeKit). Both DMCs leverage Wi-Fi to connect to the Internet. Figure 3 shows the state machines of the two DMCs, whose state machines are fully disjointed.

Consider the scenario where the home owner opts for the Home app to manage the garage door, through HomeKit, whose setup and management (network provision, user binding, etc.) are not contingent on the configuration of the *m-DMC*. This simple treatment, however, brings in a security risk: the *m-DMC* on the device is left open for user binding, while the HomeKit DMC has neither control nor observation on the *m-DMC*, based upon Apple's design. Note that although the two DMCs are not contingent on each other, they share the same HAL and hardware (Figure 1), including network provision that drives the state transition.

As a result, an unauthorized user who is allowed to connect to the home Wi-Fi, such as an Airbnb guest, babysitter, handyman (see the user study in Section 4), can silently use the *m-DMC* to set up the garage controller. This goes through the manufacturer's simple setup process: the malicious user leverages the ismartgate app, which can scan and find the ismartgate controller connected to the same Wi-Fi network automatically, to bind the *m-DMC* with his ismartgate account. Afterwards, the malicious user can remotely control the garage door using the ismartgate app. Such unauthorized control, however, is unaware by the owner, whose Home app shows that the device is under her control, and she is the only one who can access the garage. We successfully conducted a PoC attack with our ismartgate garage controller installed on a real home-garage door. Video recording of the attack is available [49].

Codema Flaw 2: Disjointed Zigbee-based DMC and *m-DMC*. The popular Philips Hue [45] devices include DMCs that can be managed through both Philips Hue Bluetooth app (the *m-DMC*) and alternatively a Zigbee-compatible hub (e.g., Philips Hue Bridge or Samsung SmartThings hub), and managed through the hub vendor's DMC (e.g., using the SmartThings app [48]). However, the *m-DMC* (based on Bluetooth for communication) and Zigbee compatible DMC are found to be completely independent, whose state

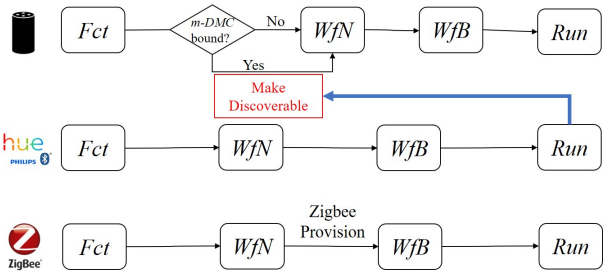

Figure 4: State machines of DMCs in the Philips Hue BLE device

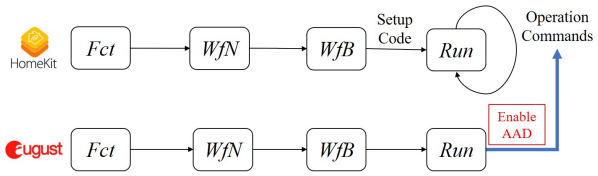

Figure 5: State machines of DMCs in the August lock

machines are completely disjoint (Figure 4). Similar to Flaw 1, security risk arises when the owner uses any single DMC but not both.

Consider that the owner uses the Zigbee-compatible DMC and the SmartThings app to manage her devices, which allows her to control IoT devices from different manufacturers connected to the SmartThings hub, including Philips Hue devices. In this case, the owner does not need to bother installing Philips Hue app or configure the Philips *m-DMC* at all. As a result, the *m-DMC* becomes dangling and stays in its *WfN* state. Any user in the range of Bluetooth (330 feet [14]), including neighbors and even strangers outside a home, can run the Philips Hue Bluetooth app to silently pair with and control the device. More seriously, such malicious behavior is completely oblivious to the SmartThings app running on the Zigbee DMC, which receives no information about the unauthorized access through the *m-DMC*.

We performed end-to-end attacks on our own Philips Hue Bluetooth plug, Philips Hue bridge, and SmartThings hub. The attacker outside the room utilized the dangling DMCs to bind with the plug without the involvement of the "device owner", gaining unauthorized control on the plug (switching it on/off).

3.2 Weak Cross-DMC Management

As mentioned earlier, some manufacturer DMCs include the mechanisms to assert some level of control over third-party DMCs' operations on the same device; e.g., allowing the user in the manufacturer app to open/close a third-party DMC, control its network provision or user binding process. A real example is with the Abode alarm hub: the user must use the Abode app to generate a HomeKit setup code, before she can pair the Abode device with her Home app (through HomeKit). *Such cross-DMC management can be abstracted through the intersection between the state-machines of these DMCs, in which one DMC's state transitions are contingent on the operations/state transitions of another DMC.* For devices with such intersecting state machines, we check whether the two DMCs can coordinate their security policies. In particular, we inspect their user policies: if a user has no/lost permission on the DMC used by the owner, we further look into whether he can control the dangling DMC.

Codema Flaw 3: Insufficient cross-DMC control on *Run* state. August Lock [12] is among the most popular smart locks, which can be managed by the August app (*m-DMC*) and HomeKit DMC. The (iOS) August app allows its user to enable and disable the HomeKit DMC (by simply toggling an 'enable/disable' switch), which by default is disabled.

By inspecting the state machines of the DMCs, we found that such protection is enabled since the August *m-DMC* imposes a control switch on HomeKit DMC's *Run* state, as illustrated in Figure 5. Specifically, we analyzed how the August app enables/disables HomeKit on the lock, by reverse-engineering the app and examining the HomeKit documentations [19], which leads to the discovery that *the gap between Android and iOS platforms limits the capability for August m-DMC to adequately manage HomeKit DMC.* Specifically, to impose control on the HomeKit DMC, the *iOS* August app needs to generate a secret string, called `additional authorization data` or `AAD`, and shares it with the Apple Home app on iOS devices (through an iOS API `updateAuthorizationData` [35]). According to the HomeKit specification [19], the HomeKit library integrated in the device supports verification of the `AAD` for sensitive operations (e.g., operating a lock), as designated by the device manufacturer: that is, to operate on the August lock, the Home app (HomeKit DMC) needs to present an `AAD` in its commands to the device. Since the Home app and the iOS API are not available on Android and we found no guideline from Apple on how HomeKit can be managed on Android, the August Android app cannot work with HomeKit (control it or monitor its status).

Such a limitation imposed by Apple introduces security risks to the lock owner who utilizes the August Android app, since she does not have full observation and control over her lock's accessibility status while an attacker running the August iOS app can have. Specifically, when the owner temporarily allows a (malicious) user to use her lock (e.g., a tenant, Airbnb guest, or employee) through her August Android app, the user can abuse this temporary permission and quietly enable the HomeKit DMC on the lock using his iOS August app. The control acquired on the HomeKit DMC lasts even after the owner fully revokes the user's right (using the Android app).

PoC attack. In our research, we implemented an end-to-end PoC attack on our own August lock (3rd generation). Specifically, the owner first set up her August Android app to control the device and then temporarily invited a malicious user (e.g., a rental tenant) to access her lock. Once given the access right, the user paired his Apple Home app with the lock using the HomeKit setup code on the lock, and then ran his August iOS app to enable HomeKit (simply toggling the switch in the app to share the `AAD` to his Home app). Once this happened, even after the owner later revoked the malicious user's access right through her August Android app (*m-DMC*), the user was still able to control the lock with his Home app (HomeKit DMC). The attack video is online [49].

Discussion. Our further investigation shows that, even the AAD mechanism of HomeKit is not designed for secure *cross-DMC* management against Codema. Based on HomeKit specification [19], it is optional (by default disabled), and designed for finer granularity of authorization: the device manufacturer can generate a set of AADs (as security tokens), and configure the HAP library to restrict specific commands from the HomeKit channel (e.g., *open/close lock*) using particular AADs; the AADs are shared with the Home app (using the aforementioned iOS API `updateAuthorizationData`) of intended users, so users assigned different ADDs assume different permissions in the HomeKit channel to command the device.

Codema Flaw 4: Insufficient cross-DMC control on user binding (*WfB* state). In the absence of a standard, suitable cross-DMC management protocol, besides Flaw 3, mainstream manufacturers' DMCs also attempt to control the third-party DMC's user binding process to better control the device, which however is error-prone.

As a prominent example, Abode Alarm Hub is a top-rated smart home security system [6]. It acts as a Z-Wave/Zigbee hub and can be connected with other Z-Wave/Zigbee compatible devices to enhance their security and usability: e.g., the user can set a rule through the hub to raise an alarm when the window sensor detects a motion. The hub also supports HomeKit. By looking into the state machines of the HomeKit DMC and Abode *m-DMC*, we found that the *m-DMC* introduces a control to manage the HomeKit DMC's user binding process (Figure 6). Specifically, the owner has to first set up the hub through the Abode app, then she can acquire a HomeKit setup code from the app, using which a user can bind the HomeKit DMC and control it. Note that, although it sacrifices some convenience, such a control prevents the adversary in the home Wi-Fi with physical access to the hub from configuring HomeKit, since the adversary does not have the setup code.

However, the protection turns out to be inadequate for securing the temporary permission: the adversary (e.g., an Airbnb/hotel guest) invited by the owner can acquire the setup code from the Abode app to stealthily configure the hub's HomeKit, binding his account to the device, without being observed by the owner; later even after the owner revokes his rights from her Abode app, he can still maintain control over the hub through HomeKit and can even disarm the siren when he breaks into the victim's home. Actually by exploiting this vulnerability, the adversary can also gain access to other home devices like home cameras and sensors [7], since they can all be managed by the Abode hub. Further, similar to August Lock, Abode only manages the HomeKit setup code in its iOS app. So the owner running the Android app may have no idea at all that the HomeKit channel can be activated using the iOS version of the manufacturer app. Even the Abode iOS app, if used by the owner, does not show whether the HomeKit DMC is bound by others.

PoC attack. We demonstrated that the risk is realistic by performing a PoC attack on our Abode Smart Security Kit. The adversary was able to disable the security siren on the hub from Apple Home app by setting it to "stand by" mode even after the owner revoked his rights through the Abode app (*m-DMC*).

Codema Flaw 5: Insufficient cross-DMC control on network provision (*WfN* state). Also, a manufacturer's *m-DMC* in the wild may impose inadequate control on any state of the third-party

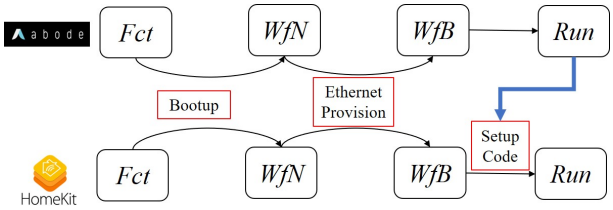

Figure 6: State machines of DMCs in the Abode alarm hub

DMC, in the absence of a standard, suitable protocol for multiple-DMC management. As an example, Philips Hue devices can be managed by smart speakers' local DMC (Section 2), as well as by the Hue Bluetooth app (*m-DMC*) — both DMCs rely on BLE for communication. By looking into their state machines (Figure 4), we found that although the *m-DMC* imposes control on the smart speaker DMC's network provision process, the protection is only effective when the *m-DMC* is first bound to the user, before the other channel; otherwise, there is no protection for the smart-speaker channel and the device becomes exposed to a practical Codema attack through the *m-DMC*.

Specifically, if the device is paired with the Hue app (the *m-DMC* through BLE) first, it will deny a new Bluetooth pairing request. So, if the owner using the Bluetooth *m-DMC* wants to control her device with a smart speaker, she needs to make the device discoverable again for pairing with the speaker, through her Hue app. However, such a cross-DMC control is conditional, based upon the aforementioned assumption: it bestows a high privilege to the *m-DMC* (through its app), yet still allows an alternative control path without going through the *m-DMC*. Particularly, when the owner first opts for the smart speaker to control the device, the *m-DMC* of the device is left open and an unauthorized party in the range of BLE (330 feet [14]) can use the manufacturer app to silently pair with the device and gain control.

PoC attack. We performed a successful PoC attack using our own Google Home Mini [18], Amazon Echo plus, and Philips Hue Bluetooth plug [32]. In our experiment, we first set up a speaker to switch on/off the plug; then a "malicious user" ran the Philips Hue Bluetooth app [28] to silently pair with the plug, and was thus able to switch on/off the plug.

3.3 Measurement

To better understand the prevalence and impact of Codema problems in the wild, we searched devices on popular American and Asian online stores (i.e., Amazon, Best Buy, Walmart, and Taobao) using smart-home and IoT related keywords (e.g., "smart light bulb", "smart garage opener/controller", "smart lock/camera/plug") and manually browsed 200 most popular devices sorted by customer ratings or sale numbers. Through manually reading their product descriptions online, we observed a total of four third-party DMCs, i.e., HomeKit DMC, Zigbee/Z-Wave compatible DMC, and smart-speaker Seamless DMC. We further identified the devices that support multiple DMCs, from which we randomly selected and purchased 14 highly rated and popular devices (Table 1) covering multiple device types including cameras, lights, locks, plugs, hubs, etc.

We analyzed all the 14 devices with end-to-end exploit experiments and confirmed that they were all vulnerable to Codema attacks, potentially affecting at least millions of users (estimated by the download numbers of their mobile apps on Google Play). Table 1 shows the results including 25 successful exploits (each row without "N/A" indicates a successful end-to-end exploit). Each device can have multiple Codema flaws and can be subject to multiple exploits depending on which DMCs the victim/attacker uses. In general, the DMCs if dangling, are exploitable by the adversary. We observe that the DMCs on 11 of the 14 devices (except Abode, August, and Yeelight) are completely disjointed (Flaw 1 and 2), which demonstrates that the DMCs generally work independently without coordination on most devices.

Table 1: Summary of Measurement Results

Device	Victim's DMC	Exploitable DMC	Flaw Type	Google Play App Installs
Abode Alarm Hub	M	H	Flaw 4	5k+
Aqara Camera G2H	M	H	Flaw 1	10k+
	H	N/A		
August Smart Lock Pro Gen3	M	H	Flaw 3	500k+
		Z	Flaw 5	
	M and H	Z	Flaw 5	
	M and Z	H	Flaw 3	
iHome plug	M	H	Flaw 1	100k+
	H	N/A		
ismartgate garage controller	M	H	Flaw 1	5k+
	H	M	Flaw 1	
Koogeek plug	M	H	Flaw 1	10k+
	H	N/A		
LIFX bulb	M	H	Flaw 1	500k+
	H	M	Flaw 1	
Meross Smart Plug Mini	M	H	Flaw 1	500k+
	H	N/A		
MiHome Lamp	M	H	Flaw 1	10,000k+
		Yeelight	Flaw 4	
	H	M	Flaw 1	
		M and Yeelight	Flaw 1	
Philips Hue BLE bulb Philips Hue BLE plug	M(BLE)	Z	Flaw 2	500k+
	Z	M(BLE)	Flaw 2	
	S	Z	Flaw 2	
		M	Flaw 5	
Philips Hue bridge	M	H	Flaw 1	5,000k+
	H	M	Flaw 1	
Refoss Smart Wi-Fi Garage Door Opener	M	H	Flaw 1	50k+
	H	N/A		
Yeelight Lightstrip Plus	MiHome	Yeelight	Flaw 4	1,000k+

M stands for *m-DMC*, H for HomeKit, Z for Z-channel, S for smart speaker.

Note that for the five devices with "N/A" fields in the table, once the owner/victim uses HomeKit, the dangling m-DMC becomes not even usable by legitimate users since the device manufacturer's mobile apps can no longer discover or bind with the device. This might be due to functionality bugs rather than a security design, since for the same devices, if the owner uses m-DMC, our exploits using HomeKit succeeded as shown in the table. Note that the dangling m-DMC on such devices may still be exploited with additional engineering efforts of the adversary. For example, although MiHome Lamp's m-DMC no longer advertises its service/existence to the local area network after its HomeKit DMC is used (so the MiHome mobile app cannot automatically discover the device), the m-DMC is not closed and we were able to use a crafted program (by modifying the MiHome mobile app's binding process) to manually connect with the device on its m-DMC, configure and control it.

Discussion. Device events (i.e., push notifications shown on mobile phones when an IoT device is operated) are not a solution to rely on to address Codema threats. Typically the events are not available (with mainstream vendors such as August, ismartgate, LIFX, Philips, and Google-Home) or do not provide timely support for mitigating Codema threats. Above all, all Codema flaws enable attackers to gain device-control unaware by the owner. When the control is exercised (e.g., turning off a plug/siren/door), harms could have immediately been done to the owner even if events are issued (e.g., turning off the alarm/lock during a midnight break-in).

Moreover, for most exploits (18/25) we performed, under the apps' default setting, no notification events were observed. For other six exploits, the owner/victim uses HomeKit-DMC that pushes notifications for device operations, and might notice exploits have occurred. However, to receive HomeKit notifications remotely, the owner must configure a separate Apple-TV/iPad as a HomeKit-hub [42] in her house. Such a hardware/configuration requirement raises the bar for using notification as Codema warning. Also, in our study, the only m-DMC app that pushes notifications for device-operations is Refoss app when the Refoss garage door has a status change. Again, the notification is after-exploit (after the garage door is already opened by the adversary) and the miscreants could have entered the building leveraging the Codema exploit.

4 ATTACK FEASIBILITY STUDY

The Codema risk is related to human behaviors: e.g., whether the owner of an IoT device leaves open unused DMCs, whether she tends to grant the access to the device to the party not fully trusted and whether she is informed of the risk. In this section, we present a study that seeks the answers to these questions, from both the device owner's perspective (the way she configures and shares the device) and the manufacturer's perspective (the way its customers have been instructed).

4.1 The User Perspective

We performed an on-site user study to investigate two key issues for understanding the significance of Codema: (1) how a typical user configures and uses her IoT devices with multiple DMCs, and (2) how likely our pre-conditions for each attack in Section 3 can be satisfied in practice. Such conditions are summarized as follows:

- *C1 (All Flaws): The owner leaves her unused DMC at its factory setting.* That is, the device owner opts for some but not all DMCs to manage a device.
- *C2 (Flaw 1, 3, 4): The adversary can access the target device's Wi-Fi network.*
- *C3 (Flaw 3, 4): The owner grants the adversary a temporary access to the target device.*

Recruitment. Under an IRB approval, we recruited 24 participants from our organizations, based on their IoT experience, education background, etc.[1] Particularly, most participants (18/24) have IoT experience. All of them have a technical or related education background (e.g., pursuing a Bachelor or graduate degree in Computer Science). Their ages range from 18 to 40 (<20: 4 people; 20-30: 19

[1]Due to COVID-19, we carefully arranged our study to follow the CDC guidelines, keeping social distancing, limiting the number of individuals in a room, etc.

people; 30-40: one person). 8/24 are female and 16/24 are male. These individuals were selected since we believe that ones with IoT experience and proper technical background are more likely to securely configure and use devices than an average user. Note that this selection bias tends to make our findings more conservative when estimating the impact of the Codema risk.

Procedure of the user study. Our user study took each participant 20-30 minutes to finish the assigned task, with a compensation of 15 US dollars. During the study, each participant was asked to freely configure and use an IoT device with user manuals provided (chosen by the participant from a MiHome lamp pro, LIFX bulb, and Aqara camera). These devices all support both HomeKit and *m-DMC* and the experiment was conducted in a home-like environment. After they finished setting up the device, we recorded the DMC(s) that was/were configured and the time it took to finish the configuration.

Then we asked the participants to complete a questionnaire, which covers the following key issues (among others): (1) why they did or did not configure/use both DMCs; (2) under what circumstances they would share Wi-Fi and IoT devices with others. All survey questions are detailed in Appendix A.

Results. This user study shows that the preconditions for Codema attacks can be easily met in practice, which poses a realistic threat to a big portion of IoT users. We elaborate on the findings below:

C1: 20 participants (83.3%) configured only one DMC. The average time for configuring a DMC is nine minutes for all participants, and thus, most of them had enough time to set up another DMC should they intend to do so. Interestingly, when we asked how much time they expect to spend in configuring a device before use, 54.2% of the participants answered "less than 5 minutes", with 37.5% of them indicating "5 to 10 minutes". This suggests that an ordinary IoT user might not want to take additional time to configure another DMC.

We also explicitly asked the participants whether they want to use both DMCs (both apps) to control a device. 95.8% of them preferred not, with comments such as "it is enough with one app". This is expected, since many mainstream vendors such as Apple have long been advocating low-hassles even "zero configuration" device setup [9, 22]. Hence, we believe that when the vendors support multiple DMCs, they aim to provide their users flexibility, not the burden to configure and use all channels.

Also interestingly, 12 participants (50%) stated that they never noticed that the device could be controlled by multiple apps during the setup and device use (although the user manuals do advertise this feature). The above results indicate that in a real-world scenario a substantial portion of IoT users would not configure or manage all DMCs, leaving them dangling.

C2: Our survey results show that most participants are willing to share their Wi-Fi with others when necessary, including Airbnb guests (58.3%), tenants (62.5%), babysitters (62.5%), temporary workers (33.3%) (e.g., a plumber), neighbors (29.2%), and even strangers who seek for help (8.3%). Even after sharing the Wi-Fi, most participants (79.2%) expect that other users should not be able to access their IoT devices in the same Wi-Fi network, unless they explicitly grant them the access. Note that, this expectation is in line with the mainstream DMCs' security policies assumed by device vendors, but Codema attacks invalidate such a security property.

Worse still, our user study also shows that most people do not change Wi-Fi passwords for a long time (for months: 16.7% of users; for years: 33.3%; never: 45.8%), which gives the attacker a large attack window after he gets the Wi-Fi access. Interestingly, one participant with significant IoT experiences stated that "I almost never change the Wi-Fi password because I have to re-configure all the IoT devices again after a reset of the Wi-Fi password."

C3: Based on the recent study [61], IoT users are willing to share smart home devices with other people, e.g., Airbnb guests, babysitters, visitors. One of our attacks (Flaw 3) require the adversary to get the temporary permission for editing devices, which is found to be completely realistic: the participants we interviewed were willing to grant such an access right to Airbnb guests (25%), tenants (29.2%), babysitters (29.2%), and workers(12.5%). This indicates that the Codema risks are indeed high, for example, considering the large number of Airbnb hosts and tenants in the wild.

4.2 The Vendor Perspective

To understand whether the vendors are aware of the Codema risks and help their users avoid the risks, we inspected both the specifications and apps of all 14 devices we studied. Results show that vendors provide limited guidance (if any) to users.

Specifications. After inspecting the manuals of the IoT devices that support multiple DMCs, we found that they only provide instructions for configuring individual device DMCs without requiring the users to set up all channels. This clearly shows that the vendors are not aware of the Codema risks, let alone to instruct their customers to avoid the risks.

Worse still, some manuals even miss DMCs their devices support, possibly due to the complexity of the IoT supply chain. For example, the manual of MiHome Lamp (branded and sold by Xiaomi) states that the device can be managed by HomeKit and MiHome; however, we found that it also includes the Yeelight DMC, which can be abused to acquire unauthorized device access. It turns out that Yeelight is one of Xiaomi's original equipment manufacturers and MiHome Lamp reuses its firmware, thereby inheriting its DMC, even though the manual fails to mention this hidden channel. Also interesting is the observation that device manuals could become out-dated after the firmware updates. We found that some vendors (Abode and Yeelight) use firmware updates to add additional DMCs to their devices, but users likely will fail to get informed about such changes. For example, Abode upgrades the firmware of the hub to support HomeKit, which can leave the device owner (especially who uses an Android phone) vulnerable to Flaw 4 (Section 3).

As mentioned earlier, Codema attacks on HomeKit require its setup code. Such codes are on the printed labels of all devices (except Abode and Yeelight Lightstrip) we studied, as suggested by Apple [19], yet none of the device vendors inform their users of the importance of the codes and ask them to keep the codes confidential. Instead, the only instruction we found from the manufacturers is: "Please do not lose the code that is at the bottom of the device. Adding the device back will need the setup code after factory reset."

Manufacturer app. Also by inspecting all IoT appliances' companion apps, we discovered the significant differences between a

device's Android app and iOS app, which exposes it to Codema risks. Specifically, none of the Android apps provide interfaces for managing HomeKit, rendering the channel vulnerable to the exploits unnoticed to Android users. Also interesting is the observation that Koogeek's iOS app only works with HomeKit, not even its own *m-DMC*. As a result, the iOS user inevitably leaves Koogeek's *m-DMC* dangling.

Further, we found that Philips Hue provides two official apps, the Philips Hue [27] app (old) and the Philips Hue Bluetooth [28] app (new). The new app is meant to control its new Bluetooth-enabled devices through the BLE DMC, a functionality missing in the old app. Therefore, those who use the old app to control their Bluetooth-enabled Philips devices are not aware of the presence of the BLE channel, and therefore vulnerable to attacks (Section 3.1).

5 MITIGATING CODEMA

Given the serious impact of Codema on the IoT ecosystem, finding effective mitigation becomes imperative. In this section, we describe the design (Section 5.2) and implementation (Section 5.3) of a new protection mechanism called CGuard that enables the first systematic and also practical cross-DMC access control. We thoroughly evaluated CGuard's usability, performance overhead, effectiveness and feasibility for real-world deployment (Section 5.4).

5.1 Goals and Challenges of Protection

Root causes of Codema risks. Our user study shows that a user typically does not bother configuring all supported DMCs on her device and tends to leave some DMCs open. As a result, she cannot have full control and visibility over her device. Actually no technique today empowers her to conveniently and effectively manage unused DMCs, nor is she offered any assurance that the DMCs that she does not choose pose no threat to her device control.

Ideal solution and challenges. Considering both security and usability, an ideal design of multiple-DMC IoT is expected to have the following properties: (1) given a device (under its factory setting), the user can choose any of its supported DMCs; (2) any DMC she opts for helps her fully control the device by coordinating security policies across all DMCs. We discuss such a clean-slate design in Section 6. In practice, however, achieving both goals can be hard in the short term since different stakeholders — including providers of third-party DMCs (e.g., Apple, Amazon, Google, Zigbee Alliance, Z-Wave Alliance) and mainstream device manufacturers — need to adopt a standardized cross-DMC management protocol. For this purpose, they are expected to modify their current DMCs to enable full interoperability of security policies between their respective DMC protocols, which can be hard. Also making the task challenging are their heterogeneous architecture (their respective security policies/enforcements span clouds, hubs, and devices) and implicit security assumptions made by different vendors [69].

Practical mitigation goals. Before an ideal long-term solution can be agreed upon, fully developed and deployed, we propose to build practical, light-weight, and effective mitigation that can be easily adopted by device manufacturers to mitigate Codema attacks without requiring a change to the current third-party DMCs. We summarize two design goals for such a mitigation:

• *Control Goal (C-Goal):* Give the user the option and tool to fully control her devices (managing all DMCs' accessibility status). Allowing any DMC of the user's choice to manage other DMCs is hard, which requires a universal cross-DMC management protocol adopted by many stakeholders. Instead, we seek an access control mechanism that can be easily adopted by the device manufacturers, and enable users to control (enable/disable/monitor) dangling DMCs. This is possible by looking at the in-device technical stack in Figure 1, where the manufacturer's ALL logic is in the position to oversee all DMCs and can be enhanced with a centralized access control framework (see the design in Section 5.2).

• *Usability goal (U-Goal):* Despite the C-Goal, we envision that users may not have to leverage the above capabilities. Hence, a mitigation design should also offer the assurance that unused DMCs pose no risks to a user's device, at no additional cost to the user: that is, the mitigation should have minimum impacts on usability. Note that, a key usability benefit of the current multiple-DMC IoT paradigm is that given a device under its factory setting, the users can choose any DMC it supports [21, 46], for example, when she has already used Apple Home to manage all her other devices. A mitigation design should preserve such usability, as confirmed in our study (see Section 5.4).

5.2 CGuard: Design of Codema Mitigation

This section presents *Channel Guard* (CGuard), a new, light-weight access control framework for cross-DMC security management. Device manufacturers can easily integrate CGuard into their firmware to enable the *m-DMC* to achieve the above mitigation goals without depending on all third-party DMC providers to fix their problems.

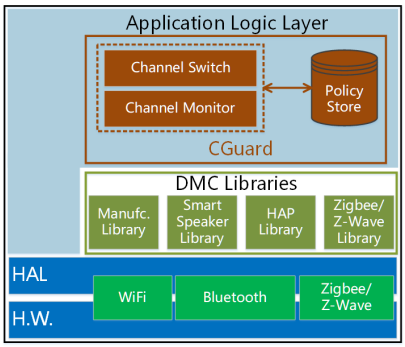

Figure 7: Architecture of CGuard.

Architecture. To achieve the *C-Goal*, our core idea is to have a centralized access control framework to oversee all DMCs and govern the accessibility of all DMCs on a device. The insight is that unlike the *m-DMC*s interfering with third-party DMCs' state machines and attempting to manage their individual policies and internal states, which we show is error-prone (Flaw 3 to Flaw 5) in the absence of a universal cross-DMC management protocol enabling policy interoperability, we introduce an access control framework (CGuard) that utilizes the device manufacturer's Application Logic Layer (ALL) in the device to control each DMC's accessibility status (enabled/disabled). CGuard helps ensure that no DMC is left in an unexpected accessibility status, such as dangling or being enabled/accessed stealthily by the attacker.

At a lower level, CGuard wraps the underlying DMC libraries on behalf of device manufacturers, exposing three universal, DMC-agnostic APIs (detailed below) for the ALL program to enable, disable, and monitor individual third-party DMCs. For this purpose, we built into CGuard three key components:

• *Policy Store*. It stores the centralized policy that specifies each DMC's accessibility, either *on* or *off*.

• *Channel Switch*. It provides two APIs `enableChannel(name)` and `disableChannel(name)` for the manufacturer's ALL program to enable/disable a third-party DMC by specifying a channel name. The two APIs update the policy to the *Policy Store* and internally invoke corresponding APIs in the DMC libraries to start/stop a DMC's operations (see implementation details in Section 5.3).

• *Channel Monitor*. It provides the `getChannelStatus(name)` API for manufacturer's ALL program to monitor/check the accessibility status (*on/off*) of each third-party DMC. The API reads the policy from the *Policy Store*.

With CGuard, when a user wants to enable/disable a DMC (the *C-Goal*), she needs to use the *m-DMC* (through the manufacturer app) to set up and bind with the device. In the manufacturer app, she can toggle on/off for individual third-party DMCs, and such a command will be sent through the *m-DMC* to the ALL program, which calls corresponding CGuard APIs for DMC control.

The principle of fail-safe default. For an owner who opts for *m-DMC*, CGuard automatically closes third-party DMCs, until she re-enables them using the manufacturer app. However, the above channel control cannot be achieved if the user first opts for a third-party DMC (requiring a universal cross-DMC management protocol, as discussed earlier). If a user chooses HomeKit, for example, to set up and use her devices, CGuard automatically closes all other DMCs (including the *m-DMC*), since HomeKit cannot control/oversee other DMCs. CGuard ensures that those DMCs remain silently closed until the user factory-resets the device. This ensures no DMC is dangling with minimum efforts from the users, in particular for many who prefer not to manage DMCs at all. Our design is highly acceptable to IoT users, as shown in our user study (Section 5.4).

5.3 Implementation of CGuard

Overall, we implemented CGuard in 1381 line of C code (released online [49]). To implement the *Channel Switch* APIs, we manually inspected the third-party DMC libraries' documentations, API lists, and/or source code, including the HomeKit open-source ADK [20], the Ti Z-stack [38] (Zigbee DMC), and the Amazon Alexa Gadgets Toolkit [41] (smart speaker DMC). We identified their respective APIs (Table 2) to start/stop their DMC's execution (a typical manufacturer also leverages these APIs to run individual DMCs in the devices). The *Policy Store* is very light-weight as well, recording each DMC's on/off designation in a few bytes.

An end-to-end deployment. We deployed CGuard to a proof-of-concept smart LED light we built on Raspberry Pi 3b (running Raspbian GNU/Linux 10), which is popularly used for IoT device development. The smart light integrates a PoC *m-DMC* we developed and multiple third-party DMCs including HomeKit (Wi-Fi), Zigbee compatible DMC, and a smart-speaker DMC. For the manufacturer DMC, we developed a prototype manufacturer app on Android (developing an iOS app is similar and trivial). The communication between our Android app and the light leverages a typical cloud-based architecture (see Section 2): our app sends commands to an IoT server (using MQTT protocol[43]) we deployed on AWS IoT core [13], which forwards the commands to the LED light. To let our light support third-party DMCs, we integrated their libraries to the firmware, i.e. HomeKit open-source ADK, Z-Stack™, and the Amazon Alexa Gadgets Toolkit.

Our light can be turned on/off using our Android app, Apple Home app (on an iPhone), and other DMCs' console (such as Alexa for the smart speaker DMC). To use CGuard, our Android app includes switches to allow the owner to turn on/off individual third-party DMCs, whose commands go through CGuard in the device and achieved the channel control. The source code of our PoC smart light and mobile app (in two versions with and without CGuard) is released online [49], along with a video demo showing how the mobile app manages the light.

Table 2: Implementation of Channel Switch in CGuard

DMC Name	DMC Library APIs Used (to enable/disable)
HomeKit	`HAPAccessoryServerCreate()` `HAPAccessoryServerStop()`
Smart Speaker (Alexa)	`AlexaGadget.set_discoverable()` `AlexaGadget._bluetooth.unpair()`
Zigbee DMC	`ZDOInitDevice()` `ZDO_ProcessMgmtLeaveReq()`

5.4 Evaluation

This section reports our evaluation of CGuard on its usability (for end users), performance, the level of effort for the manufacturer to integrate, and effectiveness in secure channel control.

5.4.1 Usability. To ensure that CGuard is easy to use for normal IoT users (see *U-Goal* in Section 5.1), introducing minimum or no impacts on usability, we performed a user study with 72 users in North America and Asia including many with non-technical backgrounds. The result demonstrates that our protection is well received by users. We elaborate the study as follows.

Recruitment. Under the IRB approval of our universities, we recruited 72 participants from three universities (in North America and Asia) or from local communities. Their ages span a wide range (18-20: 18 people; 21-30: 41 people; 31-40: 5 people; 40-50: 1 person; 50+: 7 people); 33/72 are female; 29/72 do not have an IT-related education background; 57/72 had experience of using IoT devices.

Procedure of the user study. We interviewed 30 participants on Zoom, interviewed 17 participants face-to-face and asked the rest (25) to finish an online survey while practicing social distancing following CDC guidelines due to COVID-19. During the interview, each participant was asked to watch a short video that introduces the Codema risks. The video includes less technical details to ensure that non-technical participants can easily understand the Codema risk (i.e., unused IoT channels can enable an attacker to control the device). We have released the video online [49], which can also be used to educate the general public. We then asked each participant to finish a questionnaire, taking about 10-20 minutes with 15 USD of Amazon gift card as compensation.

The questionnaire includes the following key questions to better understand the usability of CGuard for general users: (Q1) whether the users believe that the Codema risks should be addressed; (Q2) whether it is easy to use CGuard to enable/disable a DMC; (Q3) whether the users favor CGuard's automatic/default closing of unused DMCs; (Q4) whether it is important to freely choose any DMC supported on a device. Further, in the questionnaire, we require the participants to briefly explain each choice they made. The full questions are presented in Appendix B.

Results. In general, the users agreed that it is imperative to address the Codema risks, and favored the usability of CGuard:

Q1: All participants consider that the Codema risk is serious and needs a serious fix. They all agree that there must be a way to securely control the channels they do not use.

Q2: Almost all (69/72) participants agree that CGuard provides an easy way (e.g., toggling a button in the manufacturer app) to enable/disable a channel. The other 3 participants think the manufacturer should provide a fully automatic protection, rather than relying on the users to perform any operation.

Q3: Almost all (70/72) participants favor CGuard's automatic, default "off" status for unused channels (including both *m-DMC* and third-party DMCs). In their explanations (required in the questionnaire), more than 52 participants proactively, explicitly commented that automatic turning off needs the least user efforts and is preferred. In the meantime, 18 participants explicitly expressed that they favor the capabilities and options CGuard offers to manually control DMCs through the manufacturer app.

Q4: Most participants (58/72) indicate that it is very important to freely choose any DMC supported on a device to set up and use a new device. Such usability benefit is preserved by CGuard.

5.4.2 Performance overhead. Based on our end-to-end deployment – the PoC smart light supporting multiple DMCs (see Section 5.3), we evaluated the performance overhead of CGuard. We recorded the run-time memory and binary file size of the implementation with and without CGuard. Results show that it only uses 0.88% (4KB/452KB) and 0.72% (4KB/556KB) of run-time memory and storage, which is negligible. To evaluate the operation delay, we recorded the time from a command is issued from our Android app to it is executed by the device. We repeated the experiment 20 times: on average, it took 2094ms and 2088ms with and without CGuard respectively for the command to reach the device. The delay introduced by CGuard is negligible compared to the delay of network and server processing.

5.4.3 The level of efforts for manufacturer adoption. Based on our end-to-end deployment, we compared the amount of source code of the manufacturer DMC with and without CGuard integrated to our smart light. Results show that it requires little effort to integrate CGuard.

The device firmware. To integrate CGuard, we added about 182 lines of C code to the ALL program (see Figure 7). The added code is used to (1) process and interpret the commands (to enable/disable/monitor a DMC) received from the cloud, and (2) invoke the CGuard APIs to enable/disable/monitor the DMC.

The mobile app. To support CGuard, we added 448 lines of Java code to our Android app. The added code is used to (1) add UI switches to toggle on/off for third-party DMCs such as HomeKit, (2) the code logic to respond to user actions on the switches; (3) upon a user action on the switch, send commands to AWS IoT Core by invoking APIs provided by AWS SDK.

5.4.4 Effectiveness. After the owner binds any channel, CGuard proactively closes unused ones, leaving no channels dangling. Specifically, if the owner uses *m-DMC*, he can use our Android app to enable/disable any DMC on the device and check the status of all DMCs (see the video demo online [49]).

Note that an attacker who is able to temporarily access the device may try to factory-reset the device as an attempt to bypass CGuard. However, the owner can easily observe such anomalies since factory-resetting under CGuard resets all DMCs and causes the owner to immediately lose control of the device (requiring a fresh setup to re-gain control). Interestingly, some vendors (e.g., August) only allow the owner who sets up the m-DMC to factory-reset the device, which can also mitigate factory-resetting attacks.

6 DISCUSSION

Lessons learnt. The most important lesson learnt from our research is the caution one should take when integrating multiple DMCs into a single device. In the absence of a standardized, fully coordinated cross-DMC management, there is no guarantee that such DMC integration would not inadvertently bring in new security flaws, exposing the device to unauthorized access. More specifically, without proper mediation on the different DMCs, which all have a full and independent mandate on the device, there is a risk that security policies configured and enforced through one DMC could be violated by the access through another DMC, when they are utilized by different users.

Actually, not only the DMCs from different parties failed to coordinate, but even those from the same manufacturer are not well synchronized for security policies. We further found that HomeKit internally has two channels, a local channel to directly connect an iPhone (with the Home app) to the IoT device and a cloud-based channel where commands from the Home app go through HomeKit cloud to reach IoT devices. When the two channels have conflicting security policies, there is no proper protocol to resolve the issue, leading to a privilege escalation attack (see Flaw 6 in Appendix C).

Clean slate design. To fundamentally address the Codema risk, we envision a clean-slate design with a cross-DMC management standard that allows different DMCs to work together for consistent, fine-grained device control. We propose two principles for designing and implementing such a standard:

• *Multi-layered and coordinated authorization.* A fundamental cause of Codema is the design of today's DMCs, which work independently and have the same control rights on a device. The problem can be solved by multi-level security, with some DMCs (such as *m-DMC*) given a higher privilege than others. Such a DMC can help coordinate policy configurations from different channels and resolve conflicts to provide coherent, comprehensive protection, under the command of the authorized party (typically the device owner).

• *Standardized middleware and interfaces.* The heterogeneity of DMCs in terms of their architectures (e.g., with or without cloud)

and the technical stacks (e.g., HAP, Zigbee, Z-Wave) has made it difficult to configure and enforce security policies across DMCs. Therefore it is imperative to design standardized middleware to make different DMCs compatible and interoperable. Equally important is to define standardized software interfaces and protocols to be followed by DMC developers, which allow different DMCs to exchange user information, access control policy, etc.

Other mitigation strategies observed. After we reported Codema vulnerabilities to affected vendors, some of them have deployed mitigation. Without a systematic approach like CGuard, their mitigation appears to be ad-doc, less usable and secure. For example, after meeting with us, August closes all third-party DMCs by default and device owners must configure/use the m-DMC (August app) to enable/close third-party DMCs. In addition, when the owner uses the August app to remove other/guest users, a popup warning is shown to communicate the Codema risks: `"The removed user may still have access via HomeKit"`. Compared to CGuard, (1) it does not satisfy the *U-Goal* since it forces the owner to use m-DMC first (see Section 5.1); (2) its cross-DMC control (enabling/disabling HomeKit) is only available on August's iOS app (subject to *Codema Flaw 3*). As another example, the *m-DMC* of MiHome lamp will be automatically closed once the user fully configures the lamp with HomeKit DMC, which only partially exercises the *fail-safe default* principle (see Section 5.2). The problem is, if the owner opts for the *m-DMC*, the HomeKit DMC is still left open silently and can be bound and controlled by the adversary. Moreover, after our Codema report, LIFX has enhanced their protection by updating the LIFX iOS app to force its users to take control of both HomeKit and LIFX's *m-DMC* during the setup, which would protect those running the LIFX iOS app. However, the iOS users who choose to use the Apple Home app and all Android users are still at risk.

Codema discovery at scale. Our current approach to detect Codema flaws (Section 3) has limitations in identifying the dependency relations between the state transitions of two DMCs since we relied on manual efforts to set up individual DMCs and confirm whether fully configuring a DMC relies on any operation/approval in another DMC (e.g., configuring HomeKit DMC in the Abode Alarm Hub requires a setup code generated in the m-DMC's app, see Figure 6). Fully automating the step can be challenging since configuring a DMC often requires manual steps such as scanning a setup code or device-specific physical operations. To further scale our work and enable a large-scale assessment of Codema flaws involving more devices, an improved approach can be built on our current model-guided approach by automating the model building process. In particular, we may extract the model of each DMC on a device by exploring semantic based UI analysis of the DMC mobile apps, in particular identifying whether the setup process of a DMC involves a step in another DMC.

7 RELATED WORK

IoT platform security. Many efforts have been made to analyze the security problems of the IoT platforms, such as [51, 54–59, 65, 68, 70, 71]. [51] provided a methodology to analyze security properties for home-based IoT devices from the perspectives of attack techniques, mitigations, and stakeholders. [54, 55, 57, 58, 65, 68, 70] presented methods to detect inter-rule vulnerabilities, mishaving devices, malicious apps and security policy violations. [71] and [56] found problematic device management of IoT clouds based on state machine model of a single device. [69] studied the IoT cross-cloud delegation process (the cross-cloud delegation goes from the manufacturer-DMC's backend server to another cloud). In contrast, our work attempts to understand and reveal the security risks of co-exiting of multiple DMCs in the same device instead of identifying the flaws in a single DMC or platform.

IoT access control. Access control on today's IoT ecosystem is not only distributed but also heterogeneous and ad-hoc. To cope with the new challenges arising in today's IoT access +control, [53] presented WAVE, which fulfills the requirements of today's complicated IoT delegation by offering fully *decentralized trust*, which supports decentralized verification, transitive delegation and revocation. [60] also introduced *Decentralized Action Integrity* to prevent an untrusted IoT platform from misusing OAuth tokens. [67] presented a user-centered authorization mechanism to protect users from overprivileged apps in Samsung SmartThings, while [63] proposed a fine-grained context-based access control system for Samsung SmartThings. [66] presented "environmental situational oracles", which encapsulate the implementation of how a situation is sensed, inferred or actuated, to avoid over-privilege, redundancy, inconsistency, and inflexibility in today's IoT situational access control. By contrast, we attempt to fix the access control problem caused by Codema, which is new and unexplored, by developing CGuard, a new technique that can be unilaterally implemented by the device manufacturer to fully mediate third-party DMCs without any change to their designs.

8 CONCLUSION

This paper reports the first systematic study on the security risks introduced by the presence of multiple DMCs on IoT devices. Lack of coordination across these DMCs exposes a new attack surface, allowing the unauthorized party to circumvent the protection enforced by one channel through another channel. Our study shows that this security weakness is pervasive and fundamental. Our user study and measurement analysis further demonstrate that the security risk is realistic and significant. To mitigate the risk, we introduced a new access control framework that enables the IoT manufacturer to uni-laterally control third-party channels' accessibility status. Our evaluation provides further evidence for the effectiveness, usability, and feasibility of the design.

ACKNOWLEDGMENTS

We would like to thank our shepherd Prof. Z. Berkay Celik and the anonymous reviewers for their insightful comments. Special thanks to Haoran Lu's discussion and Yiyu Yang's help for testing. Yan Jia is funded by China Postdoctoral Science Foundation (No. 2021M691673) and in part by China Scholarship Council. The authors of Huazhong University of Science and Technology are supported by the National Natural Science Foundation of China (No. 61902138). Yuqing Zhang is supported by the National Natural Science Foundation of China(U1836210), and the Key Research and Development Science and Technology of Hainan Province (ZDYF202012). IU authors are supported in part by NSF CCF-2124225 and Indiana University FRSP-SF and REF.

REFERENCES

[1] 2016. BLE Bonding.
[2] 2017. The best smart home devices for your rental property. https://cozy.co/blog/the-best-smart-home-devices-for-your-rental-property/.
[3] 2019. https://www.airbnb.com/partner.
[4] 2019. https://smartrent.com/product/access-control/.
[5] 2019. Local Technologies for the Smart Home (Google I/O'19). https://www.youtube.com/watch?v=Y6Ue5hQ9meM&t=2044s
[6] 2020. abode: the top-rated smart home security system in USA. https://goabode.com.
[7] 2020. Abode's HomeKit certification. https://help.goabode.com/hc/en-us/articles/360038823531-Homekit-FAQ.
[8] 2020. Amazon Alexa. https://developer.amazon.com/en-US/alexa.
[9] 2020. Apple Bonjour. https://developer.apple.com/bonjour.
[10] 2020. Apple Home app. https://support.apple.com/en-us/HT204893.
[11] 2020. Apple HomeKit. https://www.apple.com/ios/home/.
[12] 2020. August Smart Lock. https://august.com.
[13] 2020. AWS IoT device SDK - embedded C. https://github.com/aws/aws-iot-device-sdk-embedded-C.
[14] 2020. Bluetooth Low Energy. https://en.wikipedia.org/wiki/Bluetooth_Low_Energy.
[15] 2020. Channel Guard Github. https://github.com/ChannelGuard/CGuard.
[16] 2020. Frida. https://frida.re.
[17] 2020. Google Home. https://store.google.com/us/product/google_home.
[18] 2020. Google Home mini. https://store.google.com/product/google_nest_mini?gclid=CjwKCAjw-YT1BRAFEiwAd2WRtn7q4mSlQ-S-OBd_0kjZBbP6jnN5IvrrDWKKenIT4CPjHCqannddwhoChZUQAvD_BwE&gclsrc=aw.ds.
[19] 2020. HomeKit Accessory Protocol Specification. https://developer.apple.com/support/homekit-accessory-protocol.
[20] 2020. HomeKit ADK. https://github.com/apple/HomeKitADK.
[21] 2020. Hue works with the Google Assistant for easy voice control. https://www2.meethue.com/en-us/works-with/google-home-products.
[22] 2020. Human Interface Guidelines - Setup. https://developer.apple.com/design/human-interface-guidelines/homekit/overview/setup/.
[23] 2020. ismartgate Garage/Gate Controller. https://ismartgate.com.
[24] 2020. LIFX. https://www.lifx.com.
[25] 2020. MFi Program. https://developer.apple.com/programs/mfi.
[26] 2020. Most Advanced Smart Locks in Amazon: August Lock. https://www.amazon.com/s?k=smart+lock&ref=nb_sb_noss_2.
[27] 2020. Philips Hue app. https://play.google.com/store/apps/details?id=com.philips.lighting.hue2.
[28] 2020. Philips Hue Bluetooth app. https://play.google.com/store/apps/details?id=com.signify.hue.blue.
[29] 2020. Philips Hue bridge. https://www2.meethue.com/en-us/p/hue-bridge/046677458478.
[30] 2020. Philips Hue Bulb. https://www2.meethue.com/en-us/p/hue-white-1-pack-e26/046677555689.
[31] 2020. Philips Hue bulb ranks # 4 in Best Seller in LED Bulbs and # 1 in Smart Bulbs on Amazon. https://www.amazon.com/gp/bestsellers/hpc/2314207011/ref=pd_zg_hrsr_hpc.
[32] 2020. Philips Hue Smart Plug. https://www2.meethue.com/en-us/p/hue-smart-plug/046677552343.
[33] 2020. Set up HomeKit accessory devices. https://support.apple.com/en-us/HT204893.
[34] 2020. Smart Home Gadgets: 6 Additions That Will Revolutionize Your Vacation Rental. https://www.lodgify.com/blog/smart-home-gadgets-vacation-rental.
[35] 2020. updateAuthorizationData:completionHandler. https://developer.apple.com/documentation/homekit/hmcharacteristic/1624193-updateauthorizationdata?language=objc.
[36] 2020. Yale iM1 (HomeKit) Network Module and Secure App FAQs. https://www.yalehome.com/en/support/yale-im1-homekit-network-module-secure-app-faqs.
[37] 2020. Z-Wave Alliance. https://z-wavealliance.org.
[38] 2020. Zigbee ADK. https://www.ti.com/tool/Z-STACK.
[39] 2020. Zigbee Alliance. https://zigbeealliance.org.
[40] 2020. Zigbee Protocol. https://en.wikipedia.org/wiki/Zigbee.
[41] 2021. Alexa Gadgets Raspberry Pi Samples. https://github.com/alexa-samples/Alexa-Gadgets-Raspberry-Pi-Samples.
[42] 2021. Control your home remotely with iPhone. https://support.apple.com/guide/iphone/control-your-home-remotely-iph1d10f7f2b/ios.
[43] 2021. MQTT: The Standard for IoT Messaging. https://mqtt.org/
[44] 2021. OAuth. https://oauth.net/2
[45] 2021. Philips Hue. https://www2.meethue.com.
[46] 2021. Set up and control Seamless setup smart devices. https://support.google.com/googlenest/answer/9367121?hl=en
[47] 2021. SmartConfig. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_smartconfig.html#:~:text=The%20SmartConfig%20TM%20is%20a%20provisioning%20technology%20developed,or%20a%20tablet%2C%20to%20an%20un-provisioned%20Wi-Fi%20device.
[48] 2021. SmartThings App. https://play.google.com/store/apps/details?id=com.samsung.android.oneconnect
[49] 2021. Who's In Control? On Security Risks of Disjointed IoT DeviceManagement Channels. https://sites.google.com/view/cguard.
[50] 2021. Z-Wave Protocol. https://en.wikipedia.org/wiki/Z-Wave.
[51] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. SoK: Security Evaluation of Home-Based IoT Deployments. In Proceedings of 2019 IEEE Symposium on Security and Privacy. 1362–1380.
[52] Safwa Ameer, James Benson, and Ravi Sandhu. 2020. The EGRBAC Model for Smart Home IoT. In 2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI). IEEE, 457–462.
[53] Michael P. Andersen, Sam Kumar, Moustafa AbdelBaky, Gabe Fierro, John Kolb, Hyung-Sin Kim, David E. Culler, and Raluca Ada Popa. 2019. WAVE: A Decentralized Authorization Framework with Transitive Delegation. In 28th USENIX Security Symposium. 1375–1392.
[54] Z. Berkay Celik, Patrick D. McDaniel, and Gang Tan. 2018. Soteria: Automated IoT Safety and Security Analysis. In Procedings of 2018 USENIX Annual Technical Conference. 147–158.
[55] Z. Berkay Celik, Gang Tan, and Patrick D. McDaniel. 2019. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT. In Proceedings of the 26th Annual Network and Distributed System Security Symposium.
[56] Jiongyi Chen, Chaoshun Zuo, Wenrui Diao, Shuaike Dong, Qingchuan Zhao, Menghan Sun, Zhiqiang Lin, Yinqian Zhang, and Kehuan Zhang. 2019. Your IoTs Are (Not) Mine: On the Remote Binding Between IoT Devices and Users. In 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. 222–233.
[57] Jiwon Choi, Hayoung Jeoung, Jihun Kim, Youngjoo Ko, Wonup Jung, Hanjun Kim, and Jong Kim. 2018. Detecting and Identifying Faulty IoT Devices in Smart Home with Context Extraction. In Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. 610–621.
[58] Wenbo Ding and Hongxin Hu. 2018. On the Safety of IoT Device Physical Interaction Control. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 832–846.
[59] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2016. Security Analysis of Emerging Smart Home Applications. In 37th IEEE Symposium on Security and Privacy. 636–654.
[60] Earlence Fernandes, Amir Rahmati, Jaeyeon Jung, and Atul Prakash. 2018. Decentralized action integrity for trigger-action IoT platforms. In Proceedings of 2018 Network and Distributed System Security Symposium.
[61] 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 ({USENIX} Security 18). 255–272.
[62] Y. Jia, L. Xing, Y. Mao, D. Zhao, X. Wang, S. Zhao, and Y. Zhang. 2020. Burglarsâ€™ IoT Paradise: Understanding and Mitigating Security Risks of General Messaging Protocols on IoT Clouds. In 2020 IEEE Symposium on Security and Privacy (SP). IEEE Computer Society, Los Alamitos, CA, USA, 465–481. https://doi.org/10.1109/SP40000.2020.00051
[63] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Zhuoqing Morley Mao, and Atul Prakash. 2017. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms. In Proceedings of the 24th Annual Network and Distributed System Security Symposium.
[64] Silicon Labs. 2017. Introduction to Z-Wave SmartStart.
[65] Dang Tu Nguyen, Chengyu Song, Zhiyun Qian, Srikanth V. Krishnamurthy, Edward J. M. Colbert, and Patrick D. McDaniel. 2018. IotSan: fortifying the safety of IoT systems. In Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies. 191–203.
[66] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. 2018. Situational Access Control in the Internet of Things. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 1056–1073.
[67] Yuan Tian, Nan Zhang, Yue-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. 361–378.
[68] Qi Wang, Pubali Datta, Wei Yang, Si Liu, Adam Bates, and Carl A Gunter. 2019. Charting the Attack Surface of Trigger-Action IoT Platforms. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. ACM, 1439–1453.
[69] Bin Yuan, Yan Jia, Luyi Xing, Dongfang Zhao, XiaoFeng Wang, and Yuqing Zhang. 2020. Shattered Chain of Trust: Understanding Security Risks in Cross-Cloud IoT Access Delegation. In 29th USENIX Security Symposium (USENIX Security 20). USENIX Association, 1183–1200. https://www.usenix.org/conference/usenixsecurity20/presentation/yuan
[70] 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.
[71] Wei Zhou, Yan Jia, Yao Yao, Lipeng Zhu, Le Guan, Yuhang Mao, Peng Liu, and Yuqing Zhang. 2019. Discovering and Understanding the Security Hazards in the Interactions between IoT Devices, Mobile Apps, and Clouds on Smart Home Platforms. In 28th USENIX Security Symposium. 1133–1150.
[72] Zigbee. 2016. Base Device Behavior Specification Version 1.0 [Section 10.1].

A SURVEY QUESTIONS OF THE USER STUDY ON THE ATTACK FEASIBILITY

Part A: face-to-face study follow-up questionnaire

(1) Did you learn from the specifications that the device can be configured to be controlled by BOTH the device vendor's app and the Apple Home app?
(2) Did you configure both the apps? Why did or did not you configure both of the apps?
(3) If you have multiple smart home devices, do you prefer to control all your devices with a single mobile app, instead of using different apps for each device?
Yes, No, Do not care.
(4) Do you have any security concerns based on your setting up experience just now? If yes, what are they?
(5) How long would you expect to set up an IoT device before you are ready to use it?
Less than 5 minutes, 5 to 10 minutes, 10 to 20 minutes, Do not care.

Part B: Imagine your house/apartment has many smart home devices, including but not limited to smart lock, thermometer, lighting system, garage door controller, alarm system, Hub, etc. You have already set all of them up. These devices are working together at your home network.

(1) Would you share your home Wi-Fi with the following people if he/she is asking for accessing your home Wi-Fi temporarily? (Please assume you were an Airbnb host and were renting your apartment/house out.) [multiple choices]
Airbnb guest, Tenant, Babysitter, Roommate, Friends, Spouse, Girlfriend/Boyfriend, Visting family, Neighbor, Temporary worker (who helps you to fix/install appliances in your house), Strangers who seek for help, None of the above.
(2) Why are you not willing to share your Wi-Fi?
(3) Have you ever shared your home Wi-Fi with [multiple choices]
Airbnb guest, Tenant, Babysitter, Roommate, Friends, Spouse, Girlfriend/Boyfriend, Visting family, Neighbor, Temporary worker (who helps you to fix/install appliances in your house), Strangers who seek for help, None of the above.
(4) How often did you change your Home Wi-Fi password?
Never, Years, Months, Weeks, After sharing with others.
(5) Do you expect people who has access to your home Wi-Fi could control/monitor your smart home devices without your approval?
Yes, she/he can control your devices only when connected to your home Wi-Fi.
Yes, she/he can remotely control your devices (e.g,. control your devices even when she/he is not connceted to your home Wi-Fi and not in your house).
No.
(6) Would you grant "admin" permission, which can be canceled at any time, to other people (if they ask for it)? ("admin permission" means the capabilities of editing/configuring devices, adding/removing other users, etc.) [multiple choices]
Airbnb guest, Tenant, Babysitter, Roommate, Friends, Spouse, Girlfriend/Boyfriend, Visting family, Neighbor, Temporary worker (who helps you to fix/install appliances in your house), Strangers who seek for help, None of the above.
(7) What basic security features do you expect for smart home devices?

B SURVEY QUESTIONS OF THE USER STUDY ON THE USABILITY OF CGUARD

Background Introduction

Please watch the background introduction video [49] in the following link before you answer.

Post-Video Questions

Based on the above background, it is clear that the coexistence of vendor applications (e.g. Ring app) and third-party applications (e.g. HomeKit app) poses a risk to the security of IoT devices, enabling attackers to illegally control target devices in different scenarios. For this reason, we have designed a multi-channel control platform for IoT devices, which actively shuts down channels that are not used by users to prevent unauthorized access by others. You are invited to evaluate the usability of our solution design based on your personal experience and preference.

Users can still choose any channel configuration and use the device after the purchase of the device. For users who prefer different channels, we have added the following security policy to the original features.

(1) The unused channel on your IoT device can enable an attacker to gain access and control over your device. Should these risks be addressed?
A. Yes, this is a serious risk. If my front door lock (a smart lock) has such a risk, I want the manufacturer to seriously fix it.
B. No, I don't care about the risk even though the attacker might control my door lock without my consents and awareness.
C. This is a risk, but I have other thoughts.
D. No, I don't care about this risk on my lock.
(2) If you answered C or D for question 1, please specify your answer and give a brief description of why you answered in that way.
(3) Do you want a way to control/close channels that you don't use for your device (e.g., a smart lock)? For example, if you use the manufacturer app, you can simply toggle to switch on/off the HomeKit channel (by default it is off) on the device.
A. Yes, I want to be able to control channels I am not using. I think this solution is acceptable and easy to use.
B. Yes, I want to be able to control channels that I am not using. I think this solution is acceptable, although a bit annoying to use.

C. Yes, I want to be able to control channels that I am not using. But I think this solution is unacceptable with too many efforts.
D. No, I do not want to be able to control channels that I am not using, although this can leave my unused channels open for the attackers.
E.Other.

(4) Please briefly explain your answer for Question 3.
(5) The scenario: suppose most of your home devices are currently managed using Apple Home app (the HomeKit channel). Now you get a brand new smart lock for your front door, and you configure/connect it with your Apple Home. In the meantime, do you prefer to...
A. Once you use the Apple app to control the device, the manufacturer channel is automatically closed to prevent others from using it to control your device without your consent (for security, the manufacturer channel will silently remain closed unless you factory-reset the device).
B. Manually open/close the manufacturer channel by downloading/using the manufacturer app.
C. The above options are unacceptable to me. I prefer to take the risk that my lock is controlled by others without my consent.
D. The above options are unacceptable to me. I prefer something else.
(6) Please briefly explain your answer for Question 5.
(7) What other suggestions do you have for the above multi-channel control security scheme for devices? (Optional)
(8) On a scale of 1-5, how important is it that you are able to freely choose which channel (manufacturer channel or HomeKit) you initiate your device on (1 as least important and 5 as most important)?
(9) Please briefly tell us how much experience you had for using IoT device(s).
A. I own and use at lease one IoT device
B. My household owns and uses at least one IoT device
C. Both myself and my household owns and uses at least one IoT device
D. Other:

C INCONSISTENT POLICY BETWEEN DMCS FROM THE SAME VENDOR

A surprising finding made in our research is that not only have the DMCs from different parties failed to work together, but even those from the same manufacturer are often not well synchronized in terms of security protection.

Flaw 6: HomeKit in-fight. HomeKit enables the user to manage devices locally and remotely, which are supported by the local HomeKit DMC (just HAP) and the cloud-based HomeKit DMC (through iCloud and a HomeKit hub), respectively. These two DMCs maintain their own access control lists (ACLs) and enforce their policies independently. In the local DMC, the user who first pairs with the device is the owner, and only the owner can add other users (identified by unique public keys) onto the device's local ACL. By contrast, in the cloud-based DMC, the user and the device are managed at the "*Home*" granularity: one who creates the *Home* (the Homekit structure for organizing all devices in a given location such as a house) is its owner, and is allowed to control, add and remove devices, invite other users (identified by their Apple ID, usually an e-mail address) to the *Home* and give them the permission to add and remove devices. In the absence of guidance, even the DMCs from the same party could become inconsistent in their security policy configuration and enforcement, which can lead to unauthorized access.

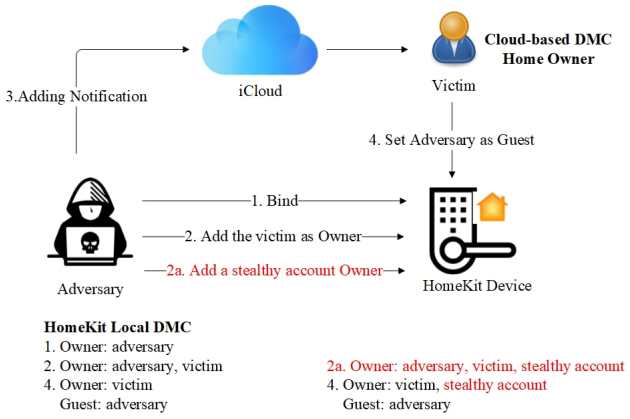

Figure 8: Abusing HomeKit Adding Accessory Workflow

Specifically, in a normal situation, the owner of a *Home* (the cloud-based DMC) should also be the owner in the ACLs (for the local DMCs) of all devices under the *Home*, while other users are the guests. This expectation, however, is fallen short of when a guest adds accessories to the *Home*: we found from both emulated HomeKit devices and the workflow of the "adding accessory" procedure that the guest actually becomes the owner of the new device through the local DMC, since she is the party who first pairs the device with the Home app, as shown in Figure 8. Although HomeKit later will automatically downgrade her to guest and make the *Home* owner the owner of the device, there is a short window before the change of ownership that allows the guest to stealthily add another account (public key) to the device ACL (2a in Figure 8). This addition will not be communicated to the owner and enable the guest to send commands authenticated with the key to the device, even after she has been removed from the owner's *Home*. Note that this trick can be used to allow a malicious guest with the edit permission (e.g, a tenant who may have the need to add his own device) to gain stealthy control on a device already in the *Home*, by simply removing the device and adding it back to place a new key on its ACL. This flaw has been acknowledged by Apple and assigned CVE-2020-9978.

PoC attack. To exploit the flaw above, we conducted an end-to-end PoC attack on our Yale lock (with iM1 network module) [36]. First, using the Apple Home app we created a *Home* through an iPad and added the Yale lock to the *Home*. Then the owner invited another user (the adversary, e.g., a tenant) to the *Home*, and gave him the right to edit accessories. So, to stealthily control the lock, the adversary first removed it from the *Home* and then added it back, during which no notification was sent to the owner's Home

app. Specifically in our experiment, we captured the window before the ownership transfer using Frida [16] to hook the processes *apsd* and *homed* on a jailbroken iPhone to add a new public key (the secret account) we generated to the lock's ACL. As a result, later even after the owner removed the adversary through the Home app, he was still able to open the lock through the key.

Discussion. Without guidance, the coordination among multiple DMCs on one device is found to be error-prone even when they are managed by the same manufacturer. Particularly, our research shows that even though the cloud-based DMC of HomeKit synchronizes the user list to the ACL of a local DMC on a device, this coordination does not go the other way around: any update on the local ACL has never got to the cloud side, allowing a new account added by the adversary to be unnoticed to the *Home* owner. The problem could come from the different ways for these channels to identify users. The Home app (cloud-based DMC) utilizes the user's Apple ID for access control, while the HAP library of the device (local DMC) relies on the ed25519 public key to authenticate a user. Since the key cannot be easily mapped to an Apple ID, updating the ACL changes to the cloud becomes hard.