SPACELORD: Private and Secure Smart Space Sharing

Yechan Bae
Georgia Institute of Technology
Atlanta, USA

Sarbartha Banerjee
University of Texas at Austin
Austin, USA

Sangho Lee
Microsoft Research
Redmond, USA

Marcus Peinado
Microsoft Research
Redmond, USA

ABSTRACT

Space sharing services like vacation rentals are being equipped with smart devices. However, sharing of such devices has privacy and security problems due to no or unclear control transfer between owners and users. In this paper, we propose SPACELORD, a system to time-share smart devices contained in a shared space privately and securely while allowing users to configure them. When a user stays at a space, SPACELORD ensures that the smart devices contained in it run code and configurations the user trusts while removing pre-installed code and configurations. When the user leaves the space, SPACELORD reverts any changes the user has introduced to the smart devices to delete remaining private data and let the owner take back control over the devices. We evaluate SPACELORD for two realistic space-sharing cases—smart home and coworking meeting room—and observe reasonable provisioning delay and runtime overhead.

CCS CONCEPTS

• **Security and privacy** → **Embedded systems security**; **Operating systems security**.

KEYWORDS

bare-metal provisioning, secure boot, smart space, time sharing

ACM Reference Format:
Yechan Bae, Sarbartha Banerjee, Sangho Lee, and Marcus Peinado. 2022. SPACELORD: Private and Secure Smart Space Sharing. In *Annual Computer Security Applications Conference (ACSAC '22), December 5–9, 2022, Austin, TX, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3564625.3564637

1 INTRODUCTION

The spaces around us are becoming more intelligent. Spaces such as residences [48], offices [108], and hospitals [49] are being equipped with smart devices, such as smart door locks, light bulbs, cameras, and speakers, to increase space efficiency, user convenience, and security. Smart homes were a $78 billion industry in 2020, projected to double by 2025 [66]. The smart office market is projected to grow to $57 billion by 2025 [47].

This trend extends to *space sharing*, e.g., vacation rentals [11, 107] and hotel rooms [5, 51], whose control is transferred to short-term users (i.e., tenants or guests). Owners install smart devices in their spaces and allow users to control them to use the spaces conveniently and efficiently. Also, coworking spaces feature hybrid meeting rooms with arrangements of intelligent cameras and microphones for both in-person and remote participants [41, 106, 116].

However, the integration of space sharing and smart devices extensively affects privacy, security, and usability. In a shared space setting, owners choose which devices to install in their spaces and preconfigure them. They typically retain administrative privilege of the devices due to security concerns [27]. This setup introduces security and privacy concerns for users because they have no option but to trust owners and devices they did not choose. Recent studies [27, 65, 119] show that Airbnb users value the convenience of smart devices but are concerned about privacy and security problems. Also, users cannot enjoy the full potential of smart devices with user-specific configuration and personalization due to a lack of permission, privacy concerns, or both. Some mechanisms such as stateless devices [5, 6] or user-driven data control [123] partially mitigate, but do not fully address, these problems. In particular, smart space sharing introduces the following three challenges.

Privacy. Smart devices managed by an owner might capture a wealth of privacy-sensitive information that are beyond user control. For example, apartment rental companies might use smart door locks to surveil tenants [83, 92]. Voice-controlled speakers in vacation rentals can reveal the user's browsing history or preferences which might be used for user-profiling [27, 65, 119].

Security. Lack of control over smart devices threatens both users and owners. For example, adversarial settings of smart devices such as carbon monoxide detectors or garage door openers can affect users. Adversarial access to audio or video streams from corporate meetings could result in secret leakage ranging from product plans to financial statements. Also, malicious users might tamper with smart devices (e.g., install persistent malware).

Configurability. Lack of control over smart devices prevents users from personalizing them. Even if owners transfer control to users, it is unclear whether short-term (e.g., hours or days) users can efficiently configure smart devices. For example, users might want to configure a smart door lock to recognize their faces during their stay and erase them immediately at check-out. However, manually

Authors are alphabetically ordered. Part of this work was done while Yechan Bae and Sarbartha Banerjee were interns at Microsoft Research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ACSAC '22, December 5–9, 2022, Austin, TX, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9759-9/22/12…$15.00
https://doi.org/10.1145/3564625.3564637

configuring individual smart devices is time-consuming and does not scale [110]. Also, any solution to the configurability challenge is complicated by incompatibility between smart devices [30, 43].

In this paper, we present SPACELORD, allowing a user to exclusively time-share smart devices in a shared space while addressing the three challenges. A user can verify whether each shared device in a space is compliant with SPACELORD and runs code they trust with configurations they provide during their stay. Pre-installed code and configurations, which are beyond their control, are removed at check-in. SPACELORD reverts all modifications the user has made during their stay at check-out to remove any private data and return control to the owner. SPACELORD only requires minor hardware and firmware changes (§4), minimizing its Trusted Computing Base (TCB) as well as maximizing its potential adoption.

SPACELORD currently focuses on hub-based smart spaces [15, 21, 30, 43, 57, 61, 124] and it enables a user to control such a smart space in two steps. First, SPACELORD allows the user to take full control of the hub except for firmware. The user can deploy an entire software stack (including an operating system) they trust on the hub while eliminating old ones which might be under the control of the owner or malicious users. The hub only maintains tiny, trusted firmware (i.e., a bootloader) that realizes this provisioning. The trusted bootloader enables *secure boot* of the provisioned software stack and *attests* it to the user. Also, SPACELORD *separates* the device- and user-specific parts from the rest of the software stack and makes them *replaceable* across different hubs and spaces to overcome hardware incompatibility. The user-specific part, which contains private configuration, data, and automation rules, becomes accessible to the hub only if it has been attested.

Next, SPACELORD allows the user to take control of smart devices paired to the hub through the provisioned hub software. The hub and smart devices perform *authenticated binding* to mutually verify their validity based on attestation and secure device identity, and to establish secure channels. Devices with successful bindings are ready to accept commands from the hub over secure channels. Every SPACELORD device has a certified type, so the hub can securely associate it with user-specific configurations and rules.

Our prototype shows the effectiveness of SPACELORD for two example cases: *smart home sharing* and *meeting room sharing*. They function as expected and their configurations (including automation rules) are securely migrated across two different smart rooms and two different meeting rooms with a reasonable provisioning delay of ~81 s. Our performance evaluation of SPACELORD hubs featuring remote encrypted storage in the public cloud shows an average overhead of 12% on the Phoronix Test Suite [82].

In summary, this paper makes the following contributions:

- SPACELORD is the first system that realizes secure and privacy-preserving smart space sharing with configurability. Users can securely use and move across time-shared smart spaces with their preferred configurations.
- SPACELORD lets a user take control of the hub and paired smart devices in a shared space. SPACELORD realizes it using (a) full provisioning of an entire hub software stack which is separated and replaceable, and (b) authenticated binding between the hub software and smart devices.

2 MOTIVATING EXAMPLES

In this section, we describe two space sharing examples suffering from privacy, security, and configurability problems.

Smart home sharing. Space sharing services (e.g., Airbnb, Hilton Connected Room) [5, 11, 51, 91, 107] are featuring smart devices such as smart door locks and smart light bulbs for convenient usage. While worrying about the devices' privacy and security implication, users would like to customize the devices [27, 65]. This customization includes running automation rules (e.g., turn on the light if it is dark, and the user is inside the room) and having personalized services (e.g., recommendation and calendar). However, creating these rules is too tedious and time-consuming for users to manually configure the system every time they arrive at a new hotel room or rental while suffering from a privacy-usability trade-off.

Meeting room sharing. Meeting rooms in coworking spaces (e.g., WeWork [112]) feature multiple devices to support *hybrid meetings*, a mix of an in-person and remote meeting [41, 98, 116]. Such devices include a hub for management, screens for presentation, and intelligent cameras, microphones, and speakers for individualized conferencing. Remote meeting software (e.g., Teams and Zoom) running on the hub controls the devices to record and stream a meeting. Also, users would configure presentation or other office software (e.g., Impress and PowerPoint) on the hub to access their sensitive meeting materials. The security and privacy of a hybrid meeting are important as it deals with business-sensitive information while being managed and monitored by numerous devices.

3 MODEL AND GOALS

In this section, we explain the system and threat model of SPACELORD and its design goals.

3.1 System and Threat Model

We consider time-sharing of a smart space which is modeled as a location managed by a *hub* and *smart devices*. The hub connects to and controls all shared devices contained in the space and runs various applications that use them. The devices are controlled only through the hub. This hub-based model follows recent academic and industry smart space systems [15, 21, 30, 43, 57, 61, 124] with enhanced access control, management, and compatibility.

Three participants govern the security and functionality of smart space sharing: *manufacturer*, *owner*, and *user*.

Manufacturer. A manufacturer develops the hardware and firmware of hubs or devices which are compliant with SPACELORD. It also provides security services such as hardware certification and revocation. In our model, every participant trusts the manufacturer's hardware, firmware, and certification. Any attacks from it (e.g., built-in backdoor) and against its hardware and firmware (e.g., physical attacks) are beyond this paper's scope §9.

Owner. An owner prepares a smart space by purchasing a hub and smart devices from manufacturers and placing them in the space. The owner lends a user the space and lets them control the hub and devices during their stay. However, the owner does not fully trust them because they might refuse to return control over the hub and devices even after they leave the space (e.g., install malware on the

hub or pair devices with different hubs). The owner aims to revert any modification users have made after they leave.

User. A user exclusively leases a smart space and the hub and smart devices contained within it from an owner for a time. The user does not trust convoluted software and configurations already installed on the hub and devices because they might still be under the control of the owner or other users. Thus, users aim to remove the existing software and configurations and install the ones they trust and that are compatible with SpaceLord.

Out of scope. We do not model the following general threats. First, the owner or other users might place hidden devices, which are not for sharing, in a space. Technologies to detect such devices have been proposed [17, 62, 88, 89, 93], and SpaceLord could rely on them. Second, attackers might be able to compromise the hub or devices by exploiting vulnerabilities in their hardware, firmware, or installed user software. Numerous researchers propose hardening mechanisms for them and SpaceLord can leverage such mechanisms. In addition, we do not consider external attackers as they are weaker than the three participants. Finally, we do not model management problems such as how to agree on the length of stay and cost and how to permit and revoke space entry.

3.2 Design Goals

G1. User privacy and security: SpaceLord protects a user's privacy and security when and after they use a space from the owner and other users.

Requirements. First, the hub only runs code verified by the user or manufacturer (i.e., the user's TCB). Second, every shared smart device in the space is either under the user's control or disabled. Third, the hub and devices will not be able to access private user code and data after the user leaves the space.

G2. Secure configuration migration: SpaceLord enables a user to move across different spaces without configuration and compatibility concerns.

Requirements. First, a user needs an abstract smart space representation to create a universally applicable configuration. Second, the configuration as well as user code and data are securely deployed to the space where the user arrives. Third, during the deployment, some of the user's software stack will be securely replaced to accommodate heterogeneous hardware.

G3. Space recoverability: SpaceLord allows the owner to recover control of a smart space and restore its configuration.

Requirements. Both hub and smart devices have methods to allow the owner to revert any control and configuration changes regardless of prior user actions.

4 DESIGN

In this section, we describe the design of SpaceLord, including the hub, smart devices, and other components (Figure 1).

4.1 Primitives

We outline the primitives SpaceLord relies on.

Secure boot. Secure boot checks the integrity of the multilayered boot [8, 115]. Before the next software layer is executed, the current layer computes the next layer's identity (e.g., a hash over the binary

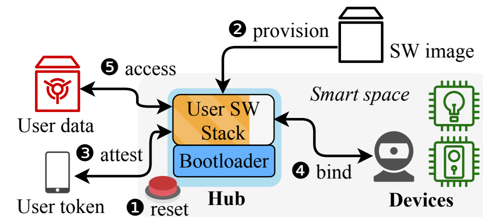

Figure 1: Overall flow of SpaceLord. A user resets the hub ❶ to provision a software stack to it ❷ and attest this provisioning ❸. The hub connects to the smart devices to control them ❹. The user lets the hub access their data after a successful attestation ❺.

code) and decides whether it is authorized to run (e.g., by checking whether the hash is in an allow list and whether it is signed by a trusted key). If this check fails, the code is not executed.

Authenticated boot. Authenticated boot also makes each layer compute the identity of the next software layer [36]. However, unlike secure boot, authenticated boot does not prevent code from running. The computed identities are simply recorded (e.g., in a Trusted Platform Module (TPM) [104]) and are used in attestation and sealed storage to gate access to resources.

Remote attestation. Remote attestation allows one principal (i.e., attestor) to provide signed, verifiable claims (e.g., this computer has booted Linux version X with a measurement value Y) to another principal (i.e., verifier) [72]. The TPM [104] and Device Identifier Composition Engine (DICE) [102] are well known examples of it.

Secure device identity. A hardware device requires an unforgeable unique identity [53] to be attested to a remote principal. This hardware identity does not depend on the software state of the device and is often implemented through attestation mechanisms.

Attested authenticated key exchange. Authenticated Key Exchange (AKE) [29] lets two parties exchange a cryptographic session key such that each party can authenticate the identity of the other. Transport Layer Security (TLS) [87] is a widely deployed AKE protocol. AKE can be augmented with attestation claims [45, 59]. Our attestor generates X.509 certificates with additional fields containing attestation claims which are checked as part of a TLS handshake.

Proximity verification. Proximity verification confirms whether two parties are physically close to each other. This verification ensures that the user interacts with the hub in the same space (and vice versa), mitigating relay attacks like Cuckoo [81]. SpaceLord can use any verification mechanism, such as distance bounding [13, 28, 86] and biometrics-based presence attestation [24, 125].

4.2 User Token

SpaceLord uses attestation, AKE, and proximity verification to prove to the user (i.e., verifier) that the system is in a trustworthy state. It requires the user to have a trusted device (or token) that can participate in these protocols on their behalf. All of them can be implemented as a smartphone app. The token is configured with the public keys of all manufacturers or other entities the user trusts.

4.3 Hub Device

We explain hardware requirements and setup for the SpaceLord hub and essential software components running on it.

Hardware. The hub supports secure boot, attestation, and proximity verification. The manufacturer provisions every hub device

with a unique public-private key pair (pk_h, sk_h). The private key sk_h is protected based on the attestation or secure device identity technology employed (e.g., TPM). The manufacturer certifies the hub public key pk_h with its private key sk_m.

Bootloader. The manufacturer equips the hub with a trusted bootloader akin to existing ones [26, 101]. This bootloader installs and loads the user's software stack with attestation.

As a first step, the bootloader orchestrates the acquisition and installation of this software. It begins with the user (e.g., a newly arrived guest) instructing the hub to install their software stack by pressing a dedicated button on the hub to ensure *local presence*. This button resets the hub and leaves one bit of information for the bootloader to start the software acquisition and installation.

The bootloader adapts the hub to a new user by booting into the hub manager—a small operating system with enough driver support to access the network and local storage. The hub manager first obtains the software stack chosen by the user—typically by copying it from the local cache (for popular public software stacks) or removable storage (e.g., the user token or USB drive) or by downloading it. Then, the hub manager installs the chosen image on the hub's storage device and finally resets the hub. The hub manager is only a networking and storage conduit and *not part of the TCB*.

After the reset, the bootloader performs an authenticated boot of the user's image by measuring the image, extending the attestation chain with it, and transferring control to it.

User software stack. SPACELORD allows the user to run a software stack they select while addressing three key challenges. First, the software stack must support different hubs with diverse hardware configurations. It involves deploying and attesting compatible device drivers. Second, the user's smart space configuration and data (e.g., hub apps, ML models, automation rules) must be secure and portable to the new space. Third, the user must be able to verify the software stack running on the hub. This involves establishing trust in all software components and verifying that they are running on the hub in front of them. §4.4 and §4.5 describe this in detail.

4.4 Software Stack Customization

We explain how SPACELORD securely and efficiently customizes software stacks for different hub hardware while supporting various peripherals and smart devices as well as how it enables users to access their code and data consistently and securely across hubs.

Software stack layers. SPACELORD partitions the user's software stack into four layers: (a) *hardware layer*, (b) *system layer*, (c) *configuration layer*, and (d) *user layer*.

The hardware layer contains hardware-specific files such as device drivers. There is a different hardware layer for each hub type based on its Instruction Set Architecture (ISA) and built-in peripherals (e.g., GPU, Embedded Multi-Media Card (eMMC), and Ethernet). This layer is typically provided by the hub manufacturer.

The system layer contains essential system files (e.g., system utilities and libraries) that are independent of hardware (i.e., peripherals). In the case of Linux, it contains the root filesystem except for device drivers relocated to the hardware layer. SPACELORD uses a common system layer for various hub models with the same ISA. This layer typically comes from an operating system vendor.

The configuration layer specifies peripherals and smart devices that the owner has placed in the smart space in the form of a space manifest §4.6. The hub connects to them via USB, Ethernet, Bluetooth, or other methods. In addition, the configuration layer contains drivers and device-specific software necessary for the hub to interact with these devices. This software generally originates with the manufacturers of the devices and not with the owner.

The user layer contains modifications the user has made on top of the clean operating system installation (i.e., the system layer). It includes the user's programs and files, and hub applications [52, 79] with customization and automation rules which the user wants to apply to the smart spaces they visit.

Layer management. SPACELORD manages the four layers with two different deployment and security policies based on their security and functionality requirements. The system, hardware, and configuration layers make up the *device-specific* part of the software stack. This part is read-only and small (e.g., typically around a gigabyte when compressed). The bootloader fully acquires and installs this part via the hub manager §4.3. SPACELORD individually hashes these three layers and attests them for verification.

In contrast, the user layer (or *user-specific* part) is both readable and writable, and private. It maintains the user's private data and has to be protected from disclosure. Moreover, it can be large (e.g., tens of gigabytes of security camera video streams or slides and demo videos) with only a portion accessed during typical usage. Thus, fully populating this part during hub provisioning could lead to an excessive and unnecessary delay. In addition, any data written in this part while the software stack is running on a hub must be accessible from other hubs the user may use in the future.

Encrypted storage. SPACELORD integrates hub-side storage encryption with remote storage (e.g., third-party cloud storage) or portable storage using a union filesystem to ensure the security and functionality of the user-specific part. The user token provides the hub with the access credentials (e.g., login information, storage decryption keys) only if it has verified the hub's software stack §4.5.

Storage caching. Remote user storage for the SPACELORD hub might be slow due to high network latency, low bandwidth, or both. SPACELORD can optionally use a portion of the local storage as a transparent cache. SPACELORD places this cache between the remote encrypted storage and the in-memory decryption logic, so the local storage does not store any plaintext blocks.

4.5 Hub Attestation

Since the user layer contains private data, SPACELORD allows the hub to access it only if a valid device-specific part is installed, booted, and running on the hub. SPACELORD uses attestation in combination with AKE to do so §4.1. The token verifies the manufacturer certificate to decide whether to trust the hub. It also decides whether to trust the software running on the hub, identified by the attested hashes of the hardware, system, and configuration layers.

The user token must know the expected hashes against which to compare the hashes from the attestation statement. The hardware layer is typically authored or endorsed by the manufacturer who can securely publish the signed hash (e.g., via a trusted Content Delivery Network (CDN)) such that the token can find and download it. The system layer is authored by either an operating system vendor or the hub manufacturer, and the token can obtain its hash similarly.

Obtaining the expected hash of the configuration layer is challenging because it consists of different components from various

sources. The owner, who composes this layer, publishes a specification of its components that the token will verify §4.6. The specification lists all binaries and their sources (i.e., the device or software vendors). The token will download the hash or signature for each binary from its source, and canonically combine these component hashes into the expected hash.

On the hub side, a small *agent* application that is part of the system layer establishes a connection to the user token (e.g., over Wi-Fi or Bluetooth) and implements the AKE protocol (e.g., TLS) including obtaining attestation claims. A similar program implements the AKE protocol on the token side. In parallel, they can do proximity verification §4.1 to mitigate Cuckoo attacks [81].

If all checks succeed, the agent and the token establish a secure channel, and the token sends the credentials for the user layer to the agent. At this point, the user has assurance that their full software stack is running on the local hub. The next step is to allow the hub to control the devices throughout the smart space.

4.6 Peripherals and Connected Devices

Space manifest and abstraction. Spacelord requires the owner to specify a *space manifest* consisting of the cryptographic hardware identifier of the hub and the information of every device placed in the smart space (i.e., an extended listing description [3]). For each device d_i, the manifest lists its information including certified public key pk_{d_i}, type, and location in the smart space and how to discover or address it. The latter consists of the bus or protocol through which the device can be reached (e.g., Wi-Fi, Bluetooth, Z-Wave) and a protocol-specific way to identify the device such as an auto-discovery mechanism [21, 71, 78] or a special address.

The type and location of each device in a space manifest must follow an abstraction model [23, 30, 43, 57, 61, 79] specified by Spacelord. This allows the hub to associate devices with corresponding user rules through the space manifest to control or disable them (e.g., *LivingRoomLight* instead of *Light#2* or *Device#5*).

Hardware and firmware. Spacelord requires attestation and secure boot support for all smart devices. Many low-cost microcontrollers [18, 94, 95] already provide this or functionally similar features. Each device has a local presence button (or other physical control such as power control [117]) to configure it.

The manufacturer provisions each device d_i with a unique public-private key pair (pk_{d_i}, sk_{d_i}) during manufacturing. The manufacturer certifies pk_{d_i} with its private key sk_m, thereby asserting that the device is legitimate. The manufacturer includes the principal device properties in this certificate, such as device model and type (e.g., camera, door lock), device manufacturer, and URLs to obtain any required device-specific software (e.g., drivers).

In contrast to the hub, the devices are fixed-function sensors or actuators. They do not run arbitrary user code, but adhere to their specifications. Secure boot ensures that the device firmware can only be updated by the manufacturer.

Device pairing. The device communicates with and accepts commands only from the *single* local hub, identified by its cryptographic hardware identifier. This requires a pairing step in which the hub's public key pk_h is stored in the device. The challenge is to perform the pairing without trusting the owner or burdening the user.

Every Spacelord device exposes a command `SetPairingKey` which allows the hub to provision the device with pk_h. This command is available only if the device's local presence button is pressed like Wi-Fi Protected Setup (WPS) [114]. It allows the owner to manually pair the devices in the space with the hub by instructing the hub to send the `SetPairingKey` command while pressing the local presence button on each of the devices. Typically, the owner would complete the pairing for all devices *only once*, when the smart space is originally set up, obviating the need for any further manual pairing steps. Although the owner is not trusted and may pair a device with a different hub, the user can detect it as described below.

Authenticated binding and reset. The hub can request a device to reset and establish a new binding. It typically happens when a user arrives at the space or leaves it. After installing user software, the hub reconnects to every device listed in the space manifest. The hub calls the device's `ResetAndBind` command, which resets the device to effectively erase the device state including the binding, conducts secure boot, and goes through an attested AKE protocol with the hub, making use of the key pairs and attestation capabilities on either side. The device accepts a binding only if the originator's public key matches pk_h stored in it during the pairing. Similarly, the hub accepts the binding only if the device's public key and attestation information is listed in the space manifest. In a simple case, both hub and device could use X.509 certificates generated by the TPM's `TPM2_CertifyX509` command to establish a TLS session.

This mechanism provides confidentiality and integrity for all subsequent communications between the hub and the device. All further device commands and their associated communications are protected by this cryptographic channel.

Device inventory and usage control. The hub inventories available devices and shares the list with the user, which consists of successfully bound devices and their properties extracted from the corresponding device certificates. The hub alerts the user of any mismatch between device properties in the space manifest and device certificates (likely a corrupt space manifest) and any failed bindings (likely compromised or turned-off devices).

A smart space and its manifest might contain types of devices that a user has never used before or does not trust (e.g., device types or manufacturers which are suspicious or unknown). To deal with such devices, the hub disallows communication with all devices by default even if it has established bindings to them. The hub permits communication with devices only if it finds corresponding rules from the user layer or corresponding public keys from the user token, or the user explicitly authorizes them (and writes new rules or downloads certificates for them). In addition, users can terminate bindings to some devices if they decide not to use them.

4.7 Space Reset

Spacelord wipes all user information from a smart space if a user requests it (i.e., resets the hub) typically when they leave the space. On every hub power cycle or reboot, the Spacelord bootloader sanitizes the whole RAM. This step is not required for storage devices which are encrypted. All external smart devices are subsequently reset when the restarted hub initiates the authenticated binding (§4.6). Owners also use the same reset mechanism to take back control over the smart space. In addition to the reset button, the hub can be

Component	ROCKPro64	iMX8MQ
SoC	RK3399	i.MX 8M
CPU	2 × Cortex A72@1.8 GHz 4 × Cortex A53@1.4 GHz	4 × Cortex A53@1.5 GHz
RAM	4 GiB	3 GiB
Storage	PCIe Gen 2 SSD 250 GB eMMC 32 GB	USB 3.0 SSD 250 GB eMMC 16 GB

Table 1: Hardware characteristics of hub devices.

configured with an authenticated timer [103, 118], enabling remote reset which is not suppressible by a malicious software stack.

5 IMPLEMENTATION

In this section, we explain a reference implementation of Spacelord. We introduce two different hub configurations along with their bootloader and software stack modifications. Then, we prototype several smart devices satisfying Spacelord's security requirements. Also, we explain implementations of the external components. Finally, we describe complete smart home and meeting room instantiations.

5.1 Hub Hardware

We implement Spacelord hubs on two Single-Board Computers (SBCs): PINE64 ROCKPro64 (ROCKPro64) and NXP iMX8MQ-EVK (iMX8MQ) with different performance and security characteristics. Both SBCs support an open-source bootloader, U-Boot [26], that our bootloader is based on §5.2.

ROCKPro64. Our ROCKPro64's characteristics are shown in Table 1. We use eMMC power-on write protection to write-protect blocks containing the bootloader and the hub manager, respectively. This protection stays in place until the next device reset.

We attach a TPM [54] to the ROCKPro64 for attestation. During hub provisioning, we use the TPM's own procedure to create an Attestation Identity Key (AIK) pair [104] using the manufacturer as a Certificate Authority (CA), and treat its public and private parts as pk_h and sk_h. The TPM does not reveal sk_h to the outside and all operations with sk_h (e.g., signing) are performed inside it.

iMX8MQ. Our iMX8MQ's characteristics are shown in Table 1. Instead of requiring additional hardware (i.e., TPM), we implement DICE [102] using secure storage that the iMX8MQ already provides via its Cryptographic Acceleration and Assurance Module (CAAM) [77]. During provisioning, we encapsulate sk_h in an encrypted and integrity-protected blob and store it in a write-protected eMMC block. Since DICE delegates signing to higher-level software, we augment DICE with a monotonic counter which is maintained by the bootloader and checked by the devices.

5.2 Hub Bootloader

Our bootloader is based on U-Boot [26]. In total, we add 995 and 753 lines of `C` code to the ROCKPro64 (with TPM) and iMX8MQ (with CAAM) bootloaders, respectively, as measured by SLOCCount [113]. We extend U-Boot's MMC driver to activate eMMC power-on write protection and include the RIoTCrypt library [69] in it for cryptographic operations. We use SHA-256 for measurement and the Elliptic Curve Digital Signature Algorithm (ECDSA) with the NIST P-256 curve and SHA-256 for signing and validation. The measurement is computed over the GUID Partition Table (GPT) of the storage and the boot partition.

ROCKPro64. We modify U-Boot (v2020.01) customized for this board [105] to support the TPM.

iMX8MQ. We modify U-Boot (v2020.04) customized for this board [76] to enable the NXP-specific `caam` command for secret storage. After every reset, the bootloader loads the capsulated blob containing sk_h from eMMC, decapsulates it, and sets the `PRIBLOB` bits to disable further access until the next reset. The bootloader uses sk_h to certify a public-private key pair it generates for later software.

On-demand hashing. Spacelord uses a runtime integrity protection mechanism (e.g., dm-verity [99]) for the software stack image (i.e., the device-specific part). The bootloader only hashes a small part of the image that is needed to enforce the runtime protection.

5.3 Hub Manager

We use initramfs-tools [25] to build the hub manager. We write hooks to specify programs (e.g., `wget` and `lz4`) and drivers (e.g., Ethernet) to include in the hub manager.

5.4 Software Stack Generation

We build a common system layer for our SBCs and their respective hardware layers. The system layer includes all files in the RK3399 Debian 9 Desktop image [84] except for the kernel binary, modules, and drivers. We also install programs for Spacelord operations to the system layer, i.e., OpenHAB [79] and WireGuard [32]. For the hardware layers, we build Linux kernels, modules, and drivers for the SBCs (versions 5.6.0 and 5.4.47, respectively) [76, 105]. We run `veritysetup` on the system and hardware layers, respectively, to compute dm-verity root hashes and hash devices [99]. Each of our images has (a) a boot partition containing the kernel, initramfs, Device Tree Blob (DTB), and root hash, (b) a system partition containing the system layer, (c) a device partition containing the respective hardware layer, and (d) additional partitions containing hash devices. We measure the GPT and boot partition of each image and sign the resulting hash with sk_m.

5.5 Storage Server and Device Counterpart

We prototype a remote storage server running Internet Small Computer Systems Interface (iSCSI) via WireGuard [32]. We create a Linux Unified Key Setup (LUKS) [42] image with a passphrase, format it with `ext4`, and upload it to the server. We also generate a WireGuard key pair and store it with the LUKS passphrase in the user token while uploading the WireGuard public key to the server.

We create initramfs to allow the user software stack to access a LUKS image in the storage server via iSCSI with WireGuard. The initramfs obtains the WireGuard private key and LUKS passphrase from the user token via the agent only if the hub attestation is successful. It uses dm-cache [100] to configure a portion of the local storage as a cache for LUKS-encrypted iSCSI blocks. It also uses overlayfs [14] to overlay the above storage on top of any other partitions to capture and store any new file writing with encryption.

If a user uses portable storage instead of remote storage, they only need to ensure a LUKS passphrase is stored in the user token.

Device	Sensor/Actuator	REST APIs
IP camera	Camera [38]	`/api/camera`
Door lock	Solenoid lock [19]	`/api/lock`
Light bulb	LED [38]	`/api/led`
Sensor	Motion [85], light [37]	`/api/motion`, `/api/light`

Table 2: Our smart device implementation with ESP32.

5.6 User Token and Agent

We develop the user token and agent as Linux applications. The user token connects to the agent running on the hub (i.e., the hub manager or a user software stack) via TLS. They consist of 421 and 376 lines of `Rust` code, respectively.

5.7 Peripherals and Connected Devices

We prototype smart devices with ESP32 boards with built-in camera and light sensor [37, 38] and with external motion sensor and actuator [19, 85] using the Espressif IoT Development Framework (ESP-IDF) [39]. The four devices we prototype are listed in Table 2.

Each smart device exposes REST APIs to control the sensor or actuator or to retrieve the sensor data. For example, a hub can make the IP camera take a picture and download the captured image by sending a `GET` request to `/api/camera` and waiting for a response.

We revise the HTTPS server component and Mbed TLS [40] to support the authenticated binding §4.6. The HTTPS server running on a smart device accepts a TLS handshake only if a client (i.e., hub) certificate is attested. The device restarts itself before binding to wipe out previous user data.

5.8 Smart Home Instantiation

We equip two rooms representing two separate vacation rental apartments or hotel rooms each with a hub and a small collection of smart devices. They enable smart entry and lighting as two compelling functions. Our prototype is based on OpenHAB [79] to abstract and manage the smart home and contained devices.

Hub software customization. We modify OpenHAB's HTTP binding [56] for the authenticated binding §4.6. As a client, it establishes two-way TLS sessions with the HTTPS servers running on the smart devices §5.7. We configure the HTTP binding's `HttpClient` [33, 35] with `SSLContextFactory.Client` [34] and specify a `KeyStore` for the private key derived by TPM or DICE and `TrustStores` for the certificates of smart devices contained in the configuration layer §4.4. This binding controls the smart devices via their REST APIs §5.7.

Device deployment. We deploy an IP camera and a smart door lock to support smart entry and a smart light bulb and a motion and a light sensor for smart lighting. To demonstrate migration across different smart homes, we implement two versions of each of the device types §5.7 and deploy them to the two rooms. The two devices of the same type expose slightly different interfaces (i.e., different REST API endpoints, such as `/api/camera` versus `/api/v2/photo`) to test migration. In room A, we deploy the ROCKPro64 hub and version A of the four devices. In room B, we deploy version B of the devices and the iMX8MQ hub.

Configuration. The owner prepares configuration layers (or space manifests) for the two rooms, specifying which devices exist and how to interact with them. A configuration layer based on OpenHAB contains two different specification files: `.things` and `.items` files. A `.things` file specifies each device by its name, address (e.g., IP address), location (e.g., bathroom or living room), and OpenHAB binding (e.g., HTTP and MQTT). A `.items` file abstracts devices [23, 30, 43, 57, 61, 79] to associate them with user rules. For example, we expose a living room light irrespective of its model as an OpenHAB item called *LivingRoomLight*. This abstraction allows users to travel with their universal automation rules (`.rules` files).

User state. We prepare user state to demonstrate face-recognition entry and smart lighting. In the former, the hub coordinates the IP camera and smart door lock with face-recognition software to unlock the door if the user is in front of the camera. In the latter case, the user's automation rules running on the hub control the smart light bulb based on inputs from the light and/or motion sensor. In particular, we configure the Face service of Azure Cognitive Services [70] with user images (i.e., one of the authors). We use a Python script to connect to the service, upload an image from the camera, and retrieve the recognition result. We add an automation rule to OpenHAB to unlock the door if the user's face is recognized. Also, we write automation rules to turn the light on and off depending on the reading on the light sensor and the time of day. All these rules, scripts, and credentials are included in the user state.

5.9 Meeting Room Instantiation

We instantiate two meeting rooms managed by the two hub devices explained in §5.1. Unlike the smart home example, this instance uses Commercial Off-The-Shelf (COTS) peripherals and software for video conference and presentation.

Device deployment. We prepare two USB webcams with built-in microphones [63, 109], monitors, and speakers, and attach them to the two hubs. We also attach USB keyboards and mice to the two hubs to let users control video conferences and presentations. These two sets of devices make up meeting rooms A and B, respectively.

Configuration. We create configuration layers for the two different meeting rooms including device drivers and the secure identifiers of individual smart devices.

User state. We prepare a common user state image for meeting rooms. We install the Firefox browser into the user state image to use the Teams and Zoom web applications. We also install LibreOffice's Impress as a presentation program. In addition, we prepare `.odp` files in the user state image to open them using Impress.

6 EVALUATION

We evaluate SPACELORD by answering the following questions:

- **RQ1. Case satisfaction:** Does SPACELORD satisfy the example cases mentioned in §2? (§6.2)
- **RQ2. Provisioning latency:** How long does a user have to wait for a SPACELORD smart space to be ready? (§6.3)
- **RQ3. Performance overhead:** How much performance overhead does SPACELORD introduce on the hub? (§6.4, §6.5)

6.1 Environment

Server. We deploy the storage server in Microsoft Azure. We use a Standard D2s v3 instance consisting of two vCPUs running at 2.6 GHz, 8 GiB of RAM, 30 GiB of Premium SSD storage for the

operating system (Ubuntu Server 18.04). The storage server additionally has 128 GiB of Standard SSD storage to store LUKS images and runs an iSCSI server with a block size of 4 KiB via WireGuard.

Hub and network. We evaluate SPACELORD with the two hub devices §5.1 and external devices §5.7. We measure the network bandwidth between the hub devices and the Azure Virtual Machine (VM) using `iperf3` [73]. The average download and upload bandwidths are 49.5 MiB/s and 48.8 MiB/s.

Device. We use ESP32 boards §5.7 to implement smart light bulb, camera, door lock, and sensor. We connect them to Wi-Fi while assigning static private IP addresses.

User token. We run the token application on a Linux laptop. The token has a negligible effect on provisioning latency and does not affect hub performance.

Software image and LUKS image. We use the RK3399 Debian 9 Desktop image mentioned in §5.4 to generate software stacks for evaluation. The sizes of the `lz4`-compressed software stack images we generate for the ROCKPro64 and the iMX8MQ are 1.4 GiB and 1.6 GiB (2.9 GiB and 3.4 GiB before compression). We configure them to have a 100 MiB boot partition, sufficient to store the kernel, initramfs, and other files. We format each partition storing files (boot, system, and device) with `ext4`. These compressed images are cached and write-protected on the hub's eMMC device.

To install and run the Phoronix Test Suite [82], we create a LUKS image (40 GiB), format it with `ext4`, and upload it to the storage server while storing the mobile user state mentioned in §5.8 and §5.9. We use AES-XTS mode with a 256-bit key for the LUKS image. All benchmarking software §6.5 is installed in the LUKS image. On each hub device, we reserve a 40 GiB area of the local storage to cache the LUKS image blocks accessed via iSCSI. In general, SPACELORD does not require such a large amount of storage space. For example, if a user only needs a few OpenHAB plugins and configurations, the user storage size can be smaller than 10s of MiB.

6.2 Case Satisfaction

Smart home sharing. We demonstrate how SPACELORD satisfies the smart home sharing case §2. In room A, we initialize the ROCKPro64 hub with the user's software stack including OpenHAB and the modified HTTP binding §5.8. We use OpenHAB's web interface to verify that all devices are recognized and configured correctly. We test our automation rules for the smart light and confirm they work correctly. We test smart entry by having the user step in front of the camera and observing that the lock opens. Finally, we reset the hub to erase all user data. We repeat the experiment in room B and observe everything to work correctly.

Meeting room sharing. We evaluate whether SPACELORD satisfies the meeting room sharing case mentioned in §2. We confirm that the two meeting rooms managed by the ROCKPro64 and iMX8MQ hubs support both video conferencing (i.e., Teams and Zoom via Firefox) and presentation (i.e., Impress). We also validate hub migration by joining the same Zoom meeting and opening the same Impress slides from the storage server in the two meeting rooms in order without overlap (i.e., reset and reprovision hubs).

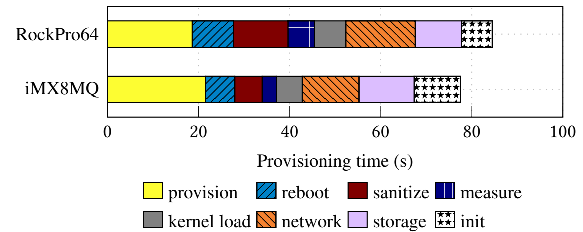

Figure 2: Hub provisioning time (81 s on average).

6.3 Provisioning Latency

We measure (a) how long it takes to provision a user software stack to the hub and attest it; and (b) how long it takes to create an authenticated binding between the hub and each smart device. We repeat these measurements 10 times for each smart space configuration. The standard deviation is less than 10% in all cases.

Hub provisioning. The average provisioning time for a user software stack is 84.5 s for ROCKPro64 and 77.5 s for iMX8MQ (Figure 2). We break down this time: (a) copy and decompress an image; (b) reboot and initialize the device; (c) sanitize memory; (d) measure the boot partition; (e) load the kernel and initramfs; (f) initialize the network and do attestation; (g) configure the network storage; and (h) initialize the software stack's remaining part. The most time-consuming stage is (a), 18.6 s on ROCKPro64 and 21.5 s on iMX8MQ, which can be skipped if a user simply reboots the hub while still using the space. Memory sanitization on iMX8MQ is faster than on ROCKPro64 because it has less RAM (3 GiB versus 4 GiB) and its memory write throughput is higher than that of ROCKPro64 (512 MiBs versus 341.3 MiBs) on U-Boot.

A provisioning time of <1.5 min is reasonable for both smart home and meeting room. The provisioning time is negligible for smart home users who typically rent the space for at least a day. It is also far shorter than the time that would be required to manually configure smart devices or to set up a machine and external devices or prepare meeting materials in a meeting room.

Authenticated binding. We measure the elapsed time to establish an authenticated binding between a hub and a smart device. This binding can be conducted in parallel with network storage configuration (g) and software stack initialization (h). In total, it takes ~9.4 s consisting of device reset (~3 s) and two-way TLS handshakes before and after the device reset (~3.2 s each). The hub can initiate multiple bindings in parallel. Thus, the total delay is largely independent of the number of smart devices it manages. It adds up to the OpenHAB startup time that takes ~14.9 s (mostly due to Java VM initialization). The differences in device binding and OpenHAB startup times between ROCKPro64 and iMX8MQ are negligible.

6.4 Microbenchmark

A SPACELORD hub uses encrypted storage, which is backed by remote storage, to securely store any data generated or collected when a user stays at a smart space. This encryption and network overhead affects the hub's overall storage performance. We use a microbenchmark program, fio 2.16 [12], to evaluate storage performance in terms of sequential read/write and random read/write throughput. For comparison, we selectively enable SPACELORD's

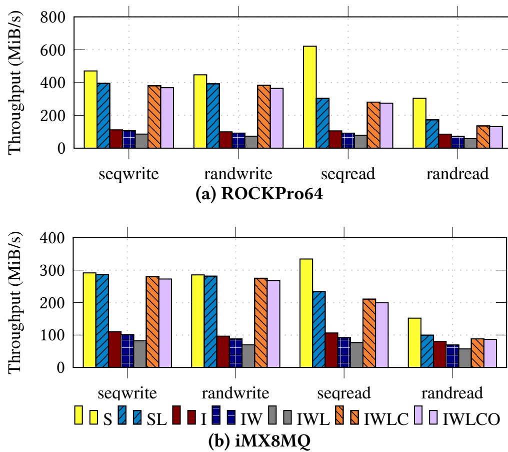

Figure 3: fio benchmark results on our two hub devices (S: SSD, L: LUKS, I: iSCSI, W: WireGuard, C: dm-cache, O: overlayfs).

security and performance-enhancement features, including storage encryption, network encryption, and local cache.

The seven configurations are SSD (S); SSD with LUKS (SL); iSCSI (I); iSCSI with WireGuard (IW); iSCSI with WireGuard and LUKS (IWL); iSCSI with WireGuard, LUKS, and dm-cache (IWLC); and iSCSI with WireGuard, LUKS, dm-cache, and overlayfs (IWLCO, SPACELORD's configuration). Configuration (SL) resembles the best case for SPACELORD: all encrypted data are cached locally. In contrast, configuration (IWL) resembles the worst case for SPACELORD: all data have to be fetched from remote storage via an encrypted channel. We configure fio to read and write a 2 GiB file with asynchronous I/O depth 32 and access block size 1 MiB. We iterate fio 10 times to obtain the average values. The standard deviation is less than 6% in all cases.

SPACELORD outperforms bare iSCSI and is comparable to SSD with LUKS (Figure 3). On the ROCKPro64, SPACELORD's storage throughputs are 368.5 MiB/s (sequential write), 364.2 MiB/s (random write), 273.8 MiB/s (sequential read), and 131.4 MiB/s (random read). They outperform iSCSI by 3.3×, 3.7×, 2.6×, and 1.5×, and underperform SSD with LUKS by 6.6%, 7.1%, 9.7%, and 23.9% (Figure 3a). On the iMX8MQ, the throughputs are 272.7 MiB/s (sequential write), 268.2 MiB/s (random write), 199.7 MiB/s (sequential read), and 86.6 MiB/s (random read). They outperform iSCSI by 2.5×, 2.8×, 1.9×, and 1.1×, and underperform SSD with LUKS by 4.9%, 4.7%, 14.8%, and 12.9% (Figure 3b). The iMX8MQ's lower storage performance hides SPACELORD's overhead.

Consequently, SPACELORD introduces negligible overhead (4.7%–7.1%) to storage write performance and low overhead (9.7%–23.9%) to storage read performance. Thus, it does not affect data generation and collection. In contrast, it slows down read operations (e.g., reading configuration files or meeting materials), but such files are typically small and would be cached in memory in most cases.

6.5 Runtime Overhead

SPACELORD's state separation and remote encrypted storage affect hub performance. We carefully select 20 tests from the Phoronix Test Suite v10.2.0 (which has more than 450 tests) [82] related to the hub's core functionality or users' general expectations, including browser, data compression, cryptography, database, machine learning, and multimedia processing. First, the hub can run various web-based applications (e.g., OpenHAB, Zoom). Second, the hub needs to compress data collected by smart devices to effectively store them. Third, the hub runs cryptographic algorithms to secure communications between itself and smart devices as well as the external services. Fourth, the hub can run database programs to store the structured configuration and data of smart devices. Fifth, the hub can run machine learning algorithms for object recognition or face detection. Finally, the hub can run multimedia processing programs to handle audio, image, and video collected by devices.

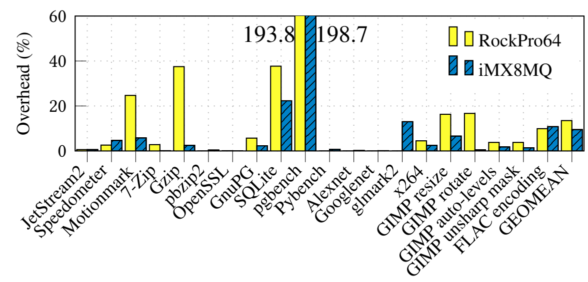

Figure 4: Phoronix benchmark results. The average runtime performance overhead of SPACELORD is moderate (12% on average).

We compare SPACELORD to a **baseline** of Debian 9 with local storage but without LUKS encryption with results shown in Figure 4. Phoronix ensures that the standard deviation is at most 3.5% by increasing the number of runs as necessary [60]. On average (geometric mean), SPACELORD's runtime overhead is 13.5% (ROCKPro64) and 9.5% (iMX8MQ). The higher overhead on the ROCKPro64 is due to a faster **baseline** with native storage connected over PCIe as compared to the USB 3.0 connection of the iMX8MQ. We think that these overheads are moderate (considering that SPACELORD uses encrypted storage LUKS over iSCSI and WireGuard) and would be acceptable to most users. A PostgreSQL benchmarking tool, pgbench (with 50 clients), introduces the highest overhead (~200%) because it is sensitive to I/O latency and throughput. However, SPACELORD targets a smart space, and a database service simultaneously serving 50 clients is not a likely workload for it. The average overhead of SPACELORD without pgbench is 6%.

7 SECURITY ANALYSIS

We explain how SPACELORD satisfies its security goals of §3.2.

7.1 User Privacy and Security (G1)

SPACELORD protects the privacy and security of users from the owner and other users by (a) deploying a full software stack that users selected to the hub, (b) allowing users to verify whether the software they chose is loaded on the hub, (c) ensuring the smart devices only interact with the software securely running on the hub, and (d) sanitizing the hub memory. All this relies on the hub bootloader and device firmware written by trusted vendors and secured by trusted hardware.

Bare-metal provisioning. SPACELORD allows users to provision a full software stack they choose to the hub except for the bootloader and the hub manager, which are write protected. This provisioning effectively removes any unwanted code and data—potentially under the control of the owner or other users—from the hub. The untrusted

hub manager cannot violate user privacy and security because it does not run concurrently with the software stack and cannot secretly modify the software stack due to authenticated boot.

Authenticated boot. The hub bootloader in conjunction with the hub hardware measures and attests the software stack. The user token compares this measurement value to an expected value to verify whether the user's software stack is running on the hub. This procedure protects users from malicious owners and other users who may try to install arbitrary software via the hub manager.

Authenticated binding. Spacelord devices only interact with the software stack currently running on the hub. A binding request contains the secure device identifier of the hub and is certified by the hub private key whose public key has been placed in the device during provisioning. A Spacelord device will accept a binding request only if the originator's identifier corresponds to the hub public key. The device resets itself before accepting a valid binding request, destroying any binding and data of a previous user.

Memory sanitization. The hub bootloader sanitizes the entire memory after every reset and some portions of it before loading the hub manager or a user software stack. The former wipes out any sensitive user data kept in memory. The latter deletes secret data which is critical to Spacelord operations, such as private keys.

7.2 Secure Configuration Migration (G2)

Spacelord uses encrypted remote or portable storage to securely maintain configuration and data and deploy them to different spaces. A hub can access the encrypted storage only if the user verifies that the hub is running the software stack they have chosen via their token. Also, Spacelord securely customizes the configuration and the software stack's device-specific part to accommodate heterogeneous hardware by checking a canonical combination of component hashes.

7.3 Space Recoverability (G3)

Spacelord ensures the owner can take back control over a smart space. The hub bootloader and the hub manager are write protected while the user software stack is running. Thus, the latter cannot delete or corrupt them. The owner can force a reset into a software stack of their choice by asserting local presence (i.e., by pressing the hub's reset button). If desired, remote administration can be enabled by equipping the hub hardware with a remote reset mechanism [103, 118]. Lastly, the hub manager conducts authenticated bindings with all devices in the space to take control over them. It will inform the owner of any binding failures to resolve their problems.

8 DISCUSSION AND LIMITATION

We discuss alternative designs and limitations of Spacelord.

8.1 Alternative Design

Provisioning. Instead of bare-metal provisioning, Spacelord could use virtualization or container technologies to reduce hub provisioning time [2, 31, 64]. However, these designs have two disadvantages. First, their TCB is significantly larger than a bootloader. Second, peripheral and external device support is restricted. Using Trusted Execution Environments (TEEs) [9, 10, 20, 58] can avoid the TCB size problem, but it also has limited device support.

Decentralization. Spacelord's centralized design might suffer from a single point of failure problem and be incompatible with standalone smart devices. In general, decentralized smart devices are powerful enough [7, 123], so Spacelord could use bare-metal provisioning and configuration migration for every smart device to realize decentralized control as it provisions each hub.

8.2 Limitations

Devices under the physical control of an adversary are subject to physical attacks, and Spacelord is no exception. One way to mitigate this threat is using tamper-proof or tamper-evident hardware [16, 68, 75, 96, 111], as has been done in other systems such as video game consoles and Blu-ray players. Further, in contrast to remote attacks and physical attacks against devices in a public space (e.g., a kiosk), physical attackers in our two examples (i.e., the owner or previous guests) are not anonymous. Thus, investigators can (eventually) identify and penalize them legally or financially.

9 RELATED WORK

Smart home systems. Researchers propose multi-user smart home systems [4, 7, 44, 50, 67, 90, 120–122]. They consider how to assign different access control rights to different people according to their roles (e.g., owner, guest). Unlike them, Spacelord focuses on exclusive time-sharing of spaces [27, 65, 119] with configurability. Some commercial systems [5, 6, 46] pledge exclusive control of smart devices contained in hotel rooms or vacation rentals while preventing owners from accessing and controlling them. However, in these systems, smart devices are stateless and both users and owners are not allowed to customize the devices. A concurrent study, TEO [123], focuses on cryptographic exclusive control of data collected and generated when a user stays at a smart space, but it fully trusts devices and does not consider configurability.

Bare-metal provisioning. Existing bare-metal provisioning [22, 55, 74, 80] considers cloud servers rather than end-user devices. Bolted [74] relies on a bootloader to provision user images with attestation. However, given the homogeneity of cloud servers governed by the same provider and lack of diverse peripherals, Bolted does not consider critical challenges Spacelord overcomes, such as supporting heterogeneous peripherals and smart devices, verifying smart devices, and detecting user presence. In addition, RO-IoT [97] considers bare-metal provisioning for IoT devices. However, it does not support heterogeneous devices, control external peripherals, or provide state continuity.

10 CONCLUSION

Spacelord allows a user to time-share spaces and their contained smart devices in a privacy-preserving and secure manner. Users can freely and securely install a software stack of their choice via bare-metal provisioning. Spacelord gives users complete control of the smart space while customizing it with the user's mobile state. Spacelord supports two realistic example cases, smart home and meeting room sharing, with low performance overhead.

REFERENCES

[1] 2019. *Proceedings of the 28th USENIX Security Symposium (Security)*. Santa Clara, CA.

[2] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Santa Clara, CA.

[3] Airbnb Help Center. 2019. Use of cameras and recording devices. https://www.airbnb.com/help/article/3061/use-of-cameras-and-recording-devices.

[4] Mohammed Al-Shaboti, Gang Chen, and Ian Welch. 2020. Achieving IoT Devices Secure Sharing in Multi-User Smart Space. In *IEEE 45th Conference on Local Computer Networks (LCN)*. 88–99.

[5] Amazon. 2020. Alexa for Hospitality. https://developer.amazon.com/en-US/alexa/alexa-for-hospitality.

[6] Amazon. 2020. Alexa Smart Properties End-User FAQs. https://m.media-amazon.com/images/G/01/asp-marketing/Alexa_for_Hospitality_FAQs.pdf.

[7] 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, See [1].

[8] W.A. Arbaugh, D.J. Farber, and J.M. Smith. 1997. A Secure and Reliable Bootstrap Architecture. In *Proceedings of the 18th IEEE Symposium on Security and Privacy (Oakland)*. Oakland, CA.

[9] Arm. 2021. Arm Confidential Compute Architecture. https://www.arm.com/why-arm/architecture/security-features/arm-confidential-compute-architecture.

[10] Arm Developer. 2008. TrustZone for Cortex-A. https://www.arm.com/technologies/trustzone-for-cortex-a.

[11] August Home. 2017. August Smart Locks for Airbnb Hosts. https://august.com/pages/airbnb.

[12] Jens Axboe. 2017. Welcome to FIO's documentation! https://fio.readthedocs.io/en/latest/.

[13] Stefan Brands and David Chaum. 1993. Distance-bounding protocols. In *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 344–359.

[14] Neil Brown. 2014. Overlay Filesystem. https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt.

[15] Z. Berkay Celik, Gang Tan, and Patrick McDaniel. 2019. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT. In *Proceedings of the 2019 Annual Network and Distributed System Security Symposium (NDSS)*. San Diego, CA.

[16] Tony Chen. 2019. Guarding Against Physical Attacks: The Xbox One Story. Platform Security Summit.

[17] Yushi Cheng, Xiaoyu Ji, Tianyang Lu, and Wenyuan Xu. 2018. DeWiCam: Detecting Hidden Wireless Cameras via Smartphones. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security (ASIACCS)*. 1–13.

[18] Saurin Choksi, Paul Kimelman, and Durgesh Pattamatta. 2020. Extend MCU Security Capabilities Beyond Trusted Execution with Hardware Crypto Acceleration and Protection.

[19] Ashish Choudhary. 2020. ESP32-CAM Face Recognition Door Lock Systems. https://circuitdigest.com/microcontroller-projects/esp32-cam-face-recognition-door-lock-system.

[20] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086. http://eprint.iacr.org/2016/086.pdf.

[21] CSA-IOT. 2022. Build With Matter | Smart Home Device Solution. https://csa-iot.org/all-solutions/matter/.

[22] Cumulus Networks. 2022. Open Network Install Environment. https://opencomputeproject.github.io/onie/.

[23] Stephen Dawson-Haggerty, Andrew Krioukov, Jay Taneja, Sagar Karandikar, Gabe Fierro, Nikita Kitaev, and David Culler. 2013. BOSS: Building Operating System Services. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Lombard, IL.

[24] Ivan De Oliveira Nunes, Xuhua Ding, and Gene Tsudik. 2021. On the Root of Trust Identification Problem. In *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (IPSN)*.

[25] Debian Wiki. 2019. initramfs-tools. https://wiki.debian.org/initramfs-tools.

[26] DENX. 2007. Das U-Boot – the Universal Boot Loader. https://denx.de/wiki/U-Boot.

[27] Rajib Dey, Sayma Sultana, Afsaneh Razi, and Pamela J Wisniewski. 2020. Exploring Smart Home Device Use by Airbnb Hosts. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–8.

[28] Aritra Dhar, Ivan Puddu, Kari Kostiainen, and Srdjan Capkun. 2020. ProximiTEE: Hardened SGX Attestation by Proximity Verification. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy (CODASPY)*.

[29] Whitfield Diffie, Paul C Van Oorschot, and Michael J Wiener. 1992. Authentication and Authenticated Key Exchanges. *Designs, Codes and cryptography* 2, 2 (1992), 107–125.

[30] Colin Dixon, Ratul Mahajan, Sharad Agarwal, AJ Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. 2012. An Operating System for the Home. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. San Jose, CA.

[31] Docker. 2013. Empowering App Development for Developers. https://docker.com.

[32] Jason A. Donenfeld. 2017. WireGuard: Next Generation Kernel Network Tunnel. In *Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS)*. San Diego, CA.

[33] Eclipse foundation. 2016. Class Httpclient. https://www.eclipse.org/jetty/javadoc/jetty-9/org/eclipse/jetty/client/HttpClient.html.

[34] Eclipse foundation. 2016. Class SSLContextFactory.Client. https://www.eclipse.org/jetty/javadoc/jetty-9/org/eclipse/jetty/util/ssl/SslContextFactory.Client.html.

[35] Eclipse foundation. 2016. The Eclipse jetty project. https://www.eclipse.org/jetty/.

[36] Paul England, Butler Lampson, John Manferdelli, Marcus Peinado, and Bryan Willman. 2003. A Trusted Open Platform. *IEEE Computer* 36, 7 (July 2003), 55–62.

[37] ESP-IoT-Solution documentation. 2022. ESP32-MeshKit-Sense Hardware Design Guidelines. https://docs.espressif.com/projects/espressif-esp-iot-solution/en/latest/hw-reference/ESP32-MeshKit-Sense_guide.html.

[38] Espressif Systems. 2020. ESP-EYE AI Board. https://www.espressif.com/en/products/devkits/esp-eye/overview.

[39] Espressif Systems. 2022. ESP-IDF Programming Guide. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html.

[40] Espressif Systems. 2022. ESP-TLS. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp_tls.html.

[41] Bob Frisch and Gary Greene. 2021. What It Takes to Run a Great Hybrid Meeting. https://hbr.org/2021/06/what-it-takes-to-run-a-great-hybrid-meeting.

[42] Clemens Fruhwirth. 2004. Linux Unified Key Setup (LUKS). https://gitlab.com/cryptsetup/cryptsetup/.

[43] Silvery Fu and Sylvia Ratnasamy. 2021. dSpace: Composable Abstractions for Smart Spaces. In *Proceedings of the 28th ACM Symposium on Operating Systems Principles (SOSP)*. Virtual.

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

[45] Kenneth Goldman, Ronald Perez, and Reiner Sailer. 2006. Linking remote attestation to secure tunnel endpoints. In *Proceedings of the first ACM workshop on Scalable trusted computing (STC)*. 21–24.

[46] Google. 2021. Get the best of Google for your hotel. https://assistant.google.com/hospitality/.

[47] Grand View Research. 2018. Smart Office Market Size & Share | Global Industry Report, 2018-2025. https://www.grandviewresearch.com/industry-analysis/smart-office-market.

[48] Richard Harper. 2006. *Inside the Smart Home*. Springer Science & Business Media.

[49] John Hatcher. 2022. Healthcare facilities: extraordinary care needs exceptional resilience. https://smartbuildingsmagazine.com/features/healthcare-facilities-extraordinary-care-needs-exceptional-resilience.

[50] 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 *Proceedings of the 27th USENIX Security Symposium (Security)*. Baltimore, MD.

[51] Hilton. 2017. Hilton Connected Rooms. https://www.hiltonownersportal.com/resource/1567103925000/ConnectedRoomBrochure.

[52] Home Assistant. 2013. Home Assistant. https://www.home-assistant.io.

[53] IEEE. 2018. IEEE 802.1AR-2018 - IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity.

[54] Infineon. 2018. OPTIGA TPM SLB 9670 TPM2.0 Data Sheet.

[55] Michael Isard. 2007. Autopilot: Automatic Data Center Management. *ACM SIGOPS Operating Systems Review* 41, 2 (2007), 60–67.

[56] Jan N. Klug. 2010. HTTP Binding. https://www.openhab.org/addons/bindings/http/.

[57] Haojian Jin, Gregory William Joseph Liu, David Ethan Hwang, Swarun Kumar, Yuvraj Agarwal, and Jason Hong. 2022. Peekaboo: A Hub-Based Approach to Enable Transparency in Data Processing within Smart Homes. In *Proceedings of the 43rd IEEE Symposium on Security and Privacy (Oakland)*. San Francisco, CA.

[58] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD Memory Encryption. *White paper* (2016).

[59] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. 2018. Integrating Intel SGX Remote Attestation with Transport Layer Security. *arXiv preprint arXiv:1801.05863* (2018).

[60] Michael Larabel. 2019. When will the Phoronix Test Suite pts/fio run more samples? https://www.phoronix.com/forums/forum/phoronix/phoronix-test-suite/1126013-when-will-the-phoronix-test- suite-pts-fio-run-more-samples.

[61] E. Lear, R. Droms, and D. Romascanu. 2019. Manufacturer Usage Description Specification. (2019).

[62] Tian Liu, Ziyu Liu, Jun Huang, Rui Tan, and Zhen Tan. 2018. Detecting Wireless Spy Cameras Via Stimulating and Probing. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 243–255.
[63] Logitech. 2020. Logitech BRIO Webcam with 4K Ultra HD Video & HDR. https://www.logitech.com/en-us/products/webcams/brio-4k-hdr-webcam.960-001105.html.
[64] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. 2017. My VM is Lighter (and Safer) than you Container. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*. Shanghai, China.
[65] Shrirang Mare, Franziska Roesner, and Tadayoshi Kohno. 2020. Smart Devices in Airbnbs: Considering Privacy and Security for both Guests and Hosts. *Proceedings on Privacy Enhancing Technologies* 2020, 2 (2020), 436–458.
[66] MarketsandMarkets. 2021. Smart home market. https://www.marketsandmarkets.com/Market-Reports/smart-homes-and-assisted-living-advanced-technologie-and-global-market-121.html.
[67] Tara Matthews, Kerwell Liao, Anna Turner, Marianne Berkovich, Robert Reeder, and Sunny Consolvo. 2016. "She'll just grab any device that's closer": A Study of Everyday Device & Account Sharing in Households. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 5921–5932.
[68] Amanda McGregor. 2021. NXP Introduces its First Cloud-Secured, Microsoft Azure Sphere-Certified Processor Family. https://www.nxp.com/company/blog/nxp-introduces-its-first-cloud-secured-microsoft-azure-sphere-certified-processor-family:BL-I-MX-8ULP-CS.
[69] Microsoft. 2020. RIoTCrypt. https://github.com/microsoft/RIoT/tree/master/Reference/RIoT/RIoTCrypt.
[70] Microsoft Azure. 2019. Cognitive Services—APIs for AI Developers. https://azure.microsoft.com/en-us/services/cognitive-services/.
[71] Brent A Miller, Toby Nixon, Charlie Tai, and Mark D Wood. 2001. Home networking with Universal Plug and Play. *IEEE Communications Magazine* 39, 12 (2001), 104–109.
[72] Chris Mitchell. 2005. *Trusted Computing*. The Institution of Electrical Engineers (IEE).
[73] Mathijs Mortimer. 2018. iperf3 Documentation.
[74] Amin Mosayyebzadeh, Apoorve Mohan, Sahil Tikale, Mania Abdi, Nabil Schear, Charles Munson, Trammell Hudson, Larry Rudolph, Gene Cooperman, Peter Desnoyers, and Orran Krieger. 2019. Supporting Security Sensitive Tenants in a Bare-Metal Cloud. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*. Renton, WA.
[75] Bryon Moyer. 2020. What Makes A Chip Tamper-Proof? https://semiengineering.com/what-makes-a-chip-tamper-proof/.
[76] NXP. 2020. i.MX - Code Aurora. https://www.codeaurora.org/projects/i-mx.
[77] NXP Semiconductors. 2018. Encrypted Boot on HABv4 and CAAM Enabled Devices.
[78] ONVIF. 2020. ONVIF Core Specification.
[79] OpenHAB. 2010. OpenHAB. https://www.openhab.org.
[80] OpenStack. 2021. Welcome to Ironic's documentation! http://docs.openstack.org/ironic/latest/.
[81] Bryan Parno. 2008. Bootstrapping Trust in a" Trusted" Platform. In *Proceedings of the 3rd USENIX Workshop on Hot Topics in Security (HotSec)*.
[82] Phoronix Media. 2008. Phoronix Test Suite - Linux Testing & Benchmarking Platform, Automated Testing, Open-Source Benchmarking. https://www.phoronix-test-suite.com.
[83] Jon Porter. 2019. NYC tenants successfully argue for right to a physical key over a smart lock. https://www.theverge.com/2019/5/10/18564322/new-york-city-apartment-smart-lock-physical-key-judge-lawsuit-latch.
[84] Radxa. 2019. ROCK Pi 4 - Radxa Wiki. https://wiki.radxa.com/Rockpi4.
[85] Random Nerd Tutorials. 2019. ESP32-CAM PIR Motion Detector with Photo Capture (saves to microSD card). https://randomnerdtutorials.com/esp32-cam-pir-motion-detector-photo-capture/.
[86] Kasper Bonne Rasmussen and Srdjan Capkun. 2010. Realization of RF Distance Bounding. In *Proceedings of the 19th USENIX Security Symposium (Security)*. Washington, DC.
[87] Eric Rescorla and Tim Dierks. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. (2018).
[88] Sriram Sami, Sean Rui Xiang Tan, Bangjie Sun, and Jun Han. 2021. LAPD: Hidden Spy Camera Detection Using Smartphone Time-of-Flight Sensors. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems (SenSys)*.
[89] Rahul Anand Sharma, Elahe Soltanaghaei, Anthony Rowe, and Vyas Sekar. 2022. Lumos: Identifying and Localizing Diverse Hidden IoT Devices in an Unfamiliar Environment. In *Proceedings of the 31st USENIX Security Symposium (Security)*. Boston, MA.
[90] Amit Kumar Sikder, Leonardo Babun, Z Berkay Celik, Abbas Acar, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A Selcuk Uluagac. 2020. Kratos: Multi-User Multi-Device-Aware Access Control System for the Smart Home. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*.
[91] SmartRent. 2022. Smart Home Solutions. https://smartrent.com.
[92] Rich Smith. 2019. The Latest Trend in Apartment Living Hits Seattle: Forced Installation of "Smart" Devices. https://www.thestranger.com/slog/2019/09/09/41335556/the-latest-trend-in-apartment-living-hits-seattle-forced-installation-of-smart-devices.
[93] Yunpeng Song, Yun Huang, Zhongmin Cai, and Jason I Hong. 2020. I'm All Eyes and Ears: Exploring Effective Locators for Privacy Awareness in IoT Scenarios. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.
[94] Kedar Sovani. 2018. Understanding ESP32's Security Features. https://medium.com/the-esp-journal/understanding-esp32s-security-features-14483e465724.
[95] STMicroelectronics. 2020. Overview of Secure Boot and Secure Firmware Update solution on Arm TrustZone STM32L5 Series microcontrollers.
[96] G Edward Suh, Dwaine Clarke, Blaise Gassend, Marten Van Dijk, and Srinivas Devadas. 2003. AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing. In *ACM International Conference on Supercomputing*.
[97] Kuniyasu Suzaki, Akira Tsukamoto, Andy Green, and Mohammad Mannan. 2020. Reboot-Oriented IoT: Life Cycle Management in Trusted Execution Environment for Disposable IoT devices. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*.
[98] Neal Taparia. 2021. How One Startup Is Reimagining Office Sharing. https://www.forbes.com/sites/nealtaparia/2021/03/12/how-one-startup-is-reimagining-office-sharing/?sh=6413352cf279.
[99] The kernel development community. 2011. dm-verity. https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/verity.html.
[100] The kernel development community. 2013. Cache. https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/cache.html.
[101] TianoCore. 2022. What is TianoCore? https://www.tianocore.org.
[102] Trusted Computing Group. 2018. Hardware Requirements for a Device Identifier Composition Engine.
[103] Trusted Computing Group. 2019. TPM 2.0 Authenticated Countdown Timer (ACT) Command.
[104] Trusted Computing Group. 2019. Trusted Platform Module Library - Part 1: Architecture.
[105] Kamil Trzciński. 2022. Rock64 and RockPro64 ayufan's packages. http://rock64.ayufan.eu.
[106] tuesday coworking. 2021. Member Benefit: Zoom Room. https://www.tuesdaycoworking.com/rent-zoom-room/.
[107] Monica Uprety. 2020. Smart Home Gadgets: 6 Additions That Will Revolutionize Your Vacation Rental. https://www.lodgify.com/blog/smart-home-gadgets-vacation-rental/.
[108] Vantage Circle. 2022. 7 Features Of A Smart Office That You Can't Overlook. https://blog.vantagecircle.com/features-of-smart-office/.
[109] VITADE. 2020. Webcam 1080P 60fps with Microphone for Streaming, Vitade 682H Pro HD USB Computer Web Camera Video Cam for Gaming Conferencing Mac Windows Desktop PC Laptop. https://www.amazon.com/Microphone-Streaming-Vitade-682H-Conferencing/dp/B086QT9T13.
[110] Neil Walker. 2021. How Long Does It Take To Install Smart Home Technology? Is Home Automation Worth It? https://beonhome.com/how-long-does-it-take-to-install-smart-home-technology/.
[111] David Weston. 2020. Meet the Microsoft Pluton processor – The security chip designed for the future of Windows PCs.
[112] WeWork. 2008. https://www.wework.com/.
[113] David A. Wheeler. 2001. SLOCCount. https://dwheeler.com/sloccount/.
[114] Wi-Fi Alliance. 2006. How does Wi-Fi Protected Setup work? https://www.wi-fi.org/knowledge-center/faq/how-does-wi-fi-protected-setup-work.
[115] Richard Wilkins and Brian Richardson. 2013. UEFI Secure Boot in Modern Computer Security Solutions.
[116] Annabelle Williams. 2021. Take a look at what Microsoft thinks the future of hybrid meetings will look like. https://www.businessinsider.com/microsoft-teams-meetings-hybrid-remote-work-2021-5.
[117] Megan Wollerton. 2020. 3 ways to install a smart LED bulb. https://www.cnet.com/home/energy-and-utilities/how-to-install-smart-led-light-bulbs/.
[118] Meng Xu, Manuel Huber, Zhichuang Sun, Paul England, Marcus Peinado, Sangho Lee, Andrey Marochko, Dennis Matoon, Rob Spiger, and Stefan Thom. 2019. Dominance as a New Trusted Computing Primitive for the Internet of Things. In *Proceedings of the 40th IEEE Symposium on Security and Privacy (Oakland)*.
[119] Yaxing Yao, Justin Reed Basdeo, Oriana Rosata Mcdonough, and Yang Wang. 2019. Privacy perceptions and designs of bystanders in smart homes. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–24.
[120] 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 *Proceedings of the 29th USENIX Security Symposium (Security)*. Boston, MA.
[121] Eric Zeng, Shrirang Mare, and Franziska Roesner. 2017. End User Security and Privacy Concerns with Smart Homes. In *thirteenth symposium on usable privacy and security (SOUPS)*. 65–80.

[122] Eric Zeng and Franziska Roesner. 2019. Understanding and Improving Security and Privacy in Multi-User Smart Homes: A Design Exploration and In-Home User Study, See [1].
[123] Han Zhang, Yuvraj Agarwal, and Matt Fredrikson. 2022. TEO: Ephemeral Ownership for IoT Devices to Provide Granular Data Control. In *Proceedings of the 20th ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*.
[124] Han Zhang, Abhijith Anilkumar, Matt Fredrikson, and Yuvraj Agarwal. 2021. Capture: Centralized Library Management for Heterogeneous IoT Devices. In *Proceedings of the 30th USENIX Security Symposium (Security)*. Virtual.
[125] Zhangkai Zhang, Xuhua Ding, Gene Tsudik, Jinhua Cui, and Zhoujun Li. 2017. Presence Attestation: The Missing Link in Dynamic Trust Bootstrapping. In *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*. Dallas, TX.

11 APPENDIX

11.1 Key Pair Terminology

Term	Meaning
pk_m, sk_m	Public-private key pair of a device manufacturer
pk_h, sk_h	Public-private key pair of a hub (bootloader)
pk_{d_i}, sk_{d_i}	Public-private key pair of an external device d_i

Table 3: Key pairs used in SPACELORD.

11.2 Compatibility with Existing Ecosystem

SPACELORD assumes that a smart space's hub and devices are compliant with the SPACELORD specification. A SPACELORD hub can potentially work with non-SPACELORD devices, but this interaction should require user authorization. For example, SPACELORD presents a user with a list of non-SPACELORD devices in a room to let them decide which devices they use or turn off. Unlike existing systems, SPACELORD prevents multiple hubs or external apps from managing devices. However, they can interact with the main hub to indirectly control the devices, so its usability concern is marginal.