Thesis Proposal:

Practical and Secure Splitting of IoT Device Functionalities

Han Zhang

August 2022

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Yuvraj Agarwal, Co-chair
Matt Fredrikson, Co-chair
Vyas Sekar
Alec Wolman, Microsoft

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Abstract

Internet-of-things (IoT) devices have rapidly gained popularity in people's daily lives. While these devices provide many smart functionalities and enable new applications, they raise several security and privacy concerns and practical operational challenges for device users and vendors. With their growing adoption and sheer volumes in deployment, IoT devices have become attractive targets for attackers, and many recent security incidents have broad and serious impacts. Meanwhile, IoT devices can collect a wide range of personal data through sensors and ubiquitous placements. It is an important challenge for device vendors to protect users' privacy and manage the access control properly. In addition, device vendors have to invest heftily in cloud infrastructures to mitigate the limited computation resources on devices. With more and more devices installed in the future, the demand for more computation will also increase.

We attribute these concerns and challenges of future IoT deployment partially to the predominant monolithic design of IoT devices and applications. Device vendors nowadays are responsible for many tasks, including addressing security and privacy concerns and maintaining their infrastructure to facilitate application demands. However, device vendors mainly focus on building compelling applications to attract more users. Therefore, they have to prioritize certain tasks over fulfilling other responsibilities given limited engineering resources. As a result, the current monolithic design leads to many vulnerabilities, security incidents, and inefficiencies.

In this thesis, we propose three new system architectures to split various functionalities current IoT device vendors need to manage themselves and offload them to third-parties. These offloading solutions improve the overall security, privacy, and efficiency in future smart home landscapes. Specifically, we demonstrate the following benefits of functionality splitting through efficient and secure designs for IoT devices. First, we can improve device security by relieving developers from the burden of managing third-party libraries themselves. Second, we can better protect users' privacy by having IoT devices delegate the task of managing ownership and data access control for users' private data. Finally, we can help reduce device vendors' management overhead and operating costs by enabling local computation offloading across different devices in users' homes, while providing integrity and security guarantees for the computation results.

Contents

Chapter 1

Introduction

Internet-of-Things (IoT) devices have rapidly gained popularity in modern homes, buildings, and shared spaces [26, 46, 84]. Equipped with a diverse set of sensors and actuators, IoT devices enable many new applications and smart functions.

With their growing ubiquity in people's daily lives, IoT devices have started to raise more and more concerns to device users. These devices keep breaking out the news of high-profile security incidents [5, 64, 67, 89], plotting a worrisome landscape of future IoT deployment with a large number of devices [2, 84]. Moreover, IoT devices can collect various sensitive data from their sensors (e.g., audio, video, behavior), raising additional privacy concerns for device users.

Meanwhile, many IoT devices are heavily resource-constrained and can not perform heavy-weight computation locally. Therefore, they must utilize device vendors' cloud services to assist with application functionalities. Exposing these devices to the Internet further exacerbates the aforementioned security and privacy concerns. In addition, managing cloud infrastructures for these transient computation demands incurs additional operating costs and extra responsibilities such as protecting user data privacy for device vendors.

These concerns and operational challenges mentioned above can be attributed, at least partially, to the predominant monolithic design of IoT devices. As illustrated in Figure 1.1, device vendors nowadays are responsible for implementing the entire device software and hardware stack, as well as the backend services and applications on the cloud. However, their main business focus is building compelling applications for their smart devices to attract users. With limited engineering resources, device vendors must prioritize certain application and feature developments, falling behind in fulfilling other responsibilities. On the other hand, many common functionalities are critical to the device's operation but are less relevant to extending the device's unique feature sets.

In this thesis, we explore new opportunities to split those functionalities that are "uninteresting" to individual device vendors to third-parties to address many future concerns in IoT deployment. Although code offloading has long been studied and advocated for applications such as mobile systems [16], enabling functionality splitting in IoT devices imposes new challenges.

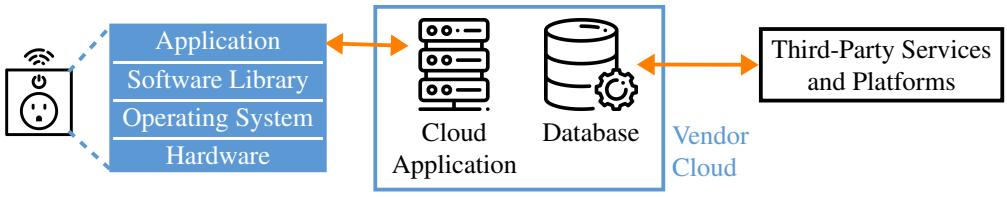


Figure 1.1: The current monolithic design of IoT device firmware and cloud backend applications. Device vendors need to implement and integrate all components in blue. Orange lines indicate communication over the public Internet.

1.1 Overview of the Thesis

The goal of this thesis is to break down the existing monolithic design of IoT devices and services, offload key functionalities, and design novel system architectures to improve the overall security, privacy, and cost efficiency of future IoT deployments. Specifically, this thesis identifies several opportunities for function offloading and proposes practical and secure solutions toward this goal in the following aspects:

1. Improving device security by relieving developers from the burden of managing third-party libraries themselves.
2. Protecting user privacy by enabling IoT devices to delegate ownership management and data access control.
3. Reducing devices' operating costs and enhancing application performance by offloading computation and neural network inferences across local devices.

To alleviate the added overhead of offloading, we identify key performance bottlenecks and incorporate optimization solutions to improve the practicality of our systems. To address security concerns that arise from offloading to external parties outside of device vendors' control, we employ several principled approaches to conduct security analyses of our system designs.

1. Improving device security by relieving developers from the burden of managing third-party libraries themselves (§2). We first investigate the security challenges in IoT device firmware, specifically from their poor management of third-party libraries. Similar to traditional software systems, IoT devices use third-party libraries extensively. Vulnerable libraries, if left unpatched, can affect massive numbers of IoT devices (e.g., CallStranger [88] and Ripple20 [90]). Unfortunately, our analysis shows that existing IoT device vendors do not patch their library vulnerabilities promptly, which underscores a huge security threat from outdated library usage and exposed attack surfaces.

We propose a new architecture to take over the library management responsibility from individual device vendors to a central service in users' local homes. Devices can utilize the latest copy of the library on the trusted central hub without worrying about patching those libraries themselves. We employ many existing security tools to provide an isolated execution environment for each device and preserve the confidentiality and integrity of their communication. With additional optimization techniques, we implemented a practical solution with low overhead ($< 15\%$ latency increases in most cases) and high scalability (supporting hundreds of devices

with a single hub).

2. Protecting user privacy by enabling IoT devices to delegate ownership management and data access control (§3). In addition to securing IoT devices using the previous solution, another important challenge in the future ubiquitous IoT deployment is to protect device users' privacy even under complicated use cases and ownership management scenarios. Specifically, IoT devices installed in shared spaces and temporary residences are often "owned" by building managers but their actual "users" are separated from their owners.

We propose a novel model of IoT device ownership well-suited to address these management difficulties in the emerging IoT deployment beyond typical smart homes and implement a new system architecture to realize this vision. This architecture removes the role of determining device ownership and securely storing users' private data from the individual device's list of responsibilities. Instead, we isolate these functionalities into their own service, enabling data owners to retain control over their data while minimizing the required trusted computing base on various entities. We apply formal security analysis and verify the new protocol design satisfies all of our security goals.

3. Reducing devices' operating costs and enhancing application performance by offloading computation and neural network inferences across local devices (§4). The first two parts of the thesis focus on addressing the security and privacy concerns, in particular from the device users' perspective. Although device vendors may be interested in adopting these approaches to stay attractive to the end-users, in this final part of the thesis, we will explore new opportunities for functionality splitting that can directly benefit the device vendors. Specifically, we will investigate how to leverage existing local resources to enable computation offloading for the emerging machine learning applications in a secure and cost-efficient way.

We propose a new offloading mechanism and an efficient verification algorithm to ensure the integrity of machine learning inferences in IoT devices. With this approach, device vendors can enlist the assistance of other devices co-located in the same user's local home to perform heavyweight machine learning inferences. We design a new verification algorithm aiming to reduce the communication overhead while retaining high confidence in verification correctness compared to related works.

Thesis Statement: By combining formal security analysis and performance optimization techniques, we can enable IoT device vendors to offload functionalities to improve overall IoT deployment security, privacy, and cost efficiency.

The remainder of this proposal is outlined as follows:

- Chapter 2 describes completed work on leveraging network infrastructure to secure common third-party library management.
- Chapter 3 describes completed work on coordinating mobile devices to enable flexible device ownership and data protection at the application level.

- Chapter 4 outlines proposed work on designing efficient computation offloading across local devices while preserving security and privacy.
- Chapter 5 presents my expected timeline in completing the remaining work for the thesis and plan for graduation.

Chapter 2

Securing IoT Devices by Offloading Third-Party Library Management

This section of the thesis illustrates the security benefits for IoT device vendors to offload library management responsibilities to a central trusted service. We first present our study on real-world IoT device firmware and demonstrate the widespread issue of mismanaged third-party libraries and its security implications. We identify the opportunity and potential benefits of consistently applying library security patches in IoT devices.

To achieve our vision and reduce the attack surface of future IoT deployment, we propose a new system architecture, Capture, that centralizes library management tasks across heterogeneous IoT devices. We propose the use of a central hub that maintains the security upkeep of common libraries and enable local devices to share the up-to-date library runtime. Although the security benefits of updating library dependencies are fairly obvious, we need to address several challenges in enforcing isolation, preserving device integrity, and reducing adoption barriers to make our solution practical.

The rest of this section describes Capture at a high level. Additional details on our completed work in this space are presented in a full paper on this topic [92].

2.1 Background

This section provides background on current IoT device software and hardware stacks, related IoT application frameworks and commercial operating systems, and home IoT networks.

2.1.1 IoT Device System Stacks

IoT devices use a variety of hardware and software. Their heterogeneity raises the complexity in managing smart devices in homes. Figure 2.1 illustrates three representative IoT devices and their system stack based on teardown blogs [1, 18]. For devices with more capable hardware (such as ARM Cortex-M micro processors), they can utilize Linux operating system and rich set of software libraries (e.g., OpenSSL). On the other hand, inexpensive devices with resource-constrained

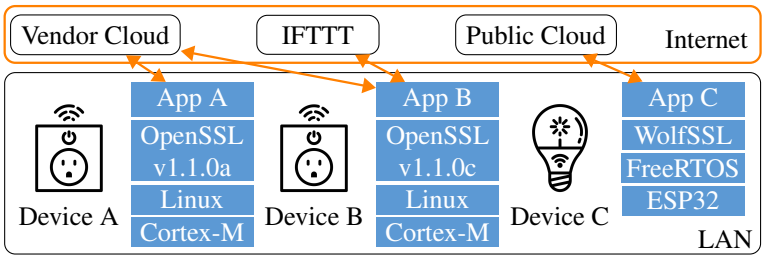


Figure 2.1: Current IoT device software stacks and network communication. Devices have a variety of platforms (ARM Cortex-M, ESP32) but utilize similar third-party libraries.

microcontrollers (e.g., Espressif ESP-32 with 520 KB RAM [24]) can use lightweight real-time operating systems (RTOSes) and libraries (e.g., WolfSSL) for limited functionalities.

Given that IoT device vendors are incentivized to focus their efforts on building compiling high-level applications (Applications A, B, C in the illustration), they sometimes fall short in securing these devices and addressing vulnerabilities in various aspects of the IoT system stack. Several projects have attempted to address these vulnerabilities by modifying the software design. Vigilia [77] requires devices to have public facing *driver* programs to communicate with home automation applications on the cloud. They also introduces capability-based network access control to limit device local communication. Other efforts propose enhancements at the application-layer to improve system security, such as adding operation logging [57], capability-based cloud applets design [82], and verification services for home automation applets [50].

2.1.2 IoT Application Frameworks and OSes

We now describe previous efforts by industry and academia to address IoT software stacks' security challenges by proposing new holistic device operating systems and application development frameworks. HomeOS [19] proposes a unified PC-like platform to manage all local devices. Commercial IoT frameworks emphasize their security offerings and ease of management for third-party developers. Microsoft Azure Sphere [52], Particle OS [61], and AWS Greengrass [3] all provide services to manage device library updates on behalf of developers. These frameworks also include native support for application-level over-the-air upgrades, reducing the barrier for developers to patch bugs. Samsung SmartThings Device SDK [65] reduces the developer burden of managing library updates by directly offering high-level APIs in the SDK (e.g., MQTT services). Developers do not need to worry about patching libraries, as long as they regularly update the SDK runtime.

2.1.3 Home IoT Networking

Previous sections described IoT devices' system stacks and existing approaches to secure them individually. Now we will briefly explain the current smart home networking setup and previous approaches improve security via networking design.

While installing a new device at home, users typically connect the device directly to the Internet by associating them with their home WiFi router, or through a vendor-provided hub

(e.g. Samsung SmartThings Hub and the Philips Hue bridge) which is then cloud-connected. Internet-connected devices can be publicly accessible (via Network Address Translation (NAT) from routers) for functionality reasons, inadvertently becoming reachable by attackers from the Internet as well [13, 87]. Although sometimes devices can be restricted from Internet access, they can still communicate with other devices on the LAN without users' involvement using, for example, the UPnP protocol [43, 48]. This can lead to cross-device exploits and escalation attacks [5, 88].

Figure 2.1 also presents the networking setup of the example smart device deployment. Local devices talk to a variety of external hosts, including vendors' proprietary cloud, third-party automation services (e.g., IFTTT), and possibly generic cloud service providers like AWS and Azure. Considering that every device in this example is publicly reachable, it is concerning that Device A still uses an outdated version of OpenSSL (1.1.0a) compared to newer alternatives (Device B's 1.1.0c). If there is any publicly known vulnerabilities in the older library, Device A may be susceptible to attackers from both local and public networks.

Many prior efforts have looked at the IoT security challenges [84] and proposed mitigation approaches leveraging better network designs. Dreamcatcher [23] uses a network attribution method to prevent link-layer spoofing attacks. Simpson et al. [74], DeadBolt [42], and SecWIR [49] propose adding features and components on network routers to secure unencrypted traffic. HoMonit [94], Bark [35], and HanGuard [17] propose finer-grained network filtering rules and context-rich firewall designs.

2.2 Analysis of Third-Party Libraries in IoT Devices

Although it is common practice for software developers to integrate mature third-party libraries into their applications, we found a lack of concrete evidence on how popular these libraries are in IoT device firmware and how well they are maintained. Therefore, in this section, we set out to address two key questions largely unanswered by previous work. Specifically,

- *How prevalent is third-party library usage among existing IoT devices?*
- *How diligent are device vendors when it comes to releasing firmware updates that patch critical security vulnerabilities?*

Several prior work [12, 56, 95] have studied the prevalence of vulnerabilities in embedded networking equipment, some of which can be attributed to unpatched third-party libraries. A recent study focusing on smart appliances reports similar findings [2]. However, these studies do not address the state of affairs on current IoT devices, and in particular on how frequently libraries are used and updated. To fill this gap in our knowledge, we conducted a measurement study on 122 firmware releases from 26 devices and 5 popular vendors and present the results in this section. We find that third-party library use is prevalent, and even more concerning, that security-essential libraries like OpenSSL often remain unpatched for hundreds of days.

2.2.1 Data Collection

Retrieving Library Information. We collected large number of library usage information from well-known smart device vendors by searching for GPL library disclosure information,

Vendor	Belkin	TP-Link	Ring	Nest	D-Link	Total
Devices	12	3	1	7	3	26
Firmware	12	3	1	74	32	122
Libraries	80	5	53	290	93	441
Library versions	103	5	55	400	114	654

Table 2.1: Summary of devices and vendors included in the measurement. We skip firmware for network equipment since our focus is on smart devices.

as this disclosure is required by the license terms. Compared to alternative approaches such as extracting library information from firmware binary (using tools like BAT [33] and OSS Police [20]), we believe our approach is more efficient, accurate, and easier to scale up to large number of firmware. However, our approach only applies to libraries with GPL licenses, limiting our dataset's coverage. As a result, our analysis will under-represent the true prevelance of library usage, but we believe our insights regarding the third-party library patching frequency and vulnerability management will apply to other libraries as well.

Firmware Selection. Using the previously described approach, we collected 122 firmware releases from 5popular device vendors (Belkin, TP-Link, Ring, Nest, and D-Link). These vendors provide consistent and detailed information about their firmware release history with necessary information about third-party libraries. Table 2.1 summarizes our dataset. Nest and D-Link provide the most comprehensive information about their firmware release history, dating back to 2011. We use these historical releases to analyze longitudinal patching behaviors. Belkin and TP-Link maintain public information for a single firmware version for each device still under support. Ring releases one summary for all open-source libraries used in their devices, which we categorize as a single generic device with a single firmware release.

2.2.2 Results and Findings

We present the findings from analyzing the dataset collected from the previous section. We found concrete evidences to show the extensive usage of common third-party libraries across IoT devices and vendors, as well as the underlying security risks from mismanaged libraries and lack of security patching.

Library Prevalence. As expected, we found extensive usage of common third-party libraries in all of our collected firmware. Figure 2.2 lists the most popular libraries, as more than 50% of the devices in our dataset use them. The most ubiquitous libraries are OpenSSL and BusyBox, which are used by 92.31% and 88.46% of the devices. On the other hand, we also notices a long tail distribution of the diversity of libraries used by different IoT devices. Table 2.1 shows there are a total of 441 libraries used across our dataset, many of which are unique for the specific device or a particular vendor.

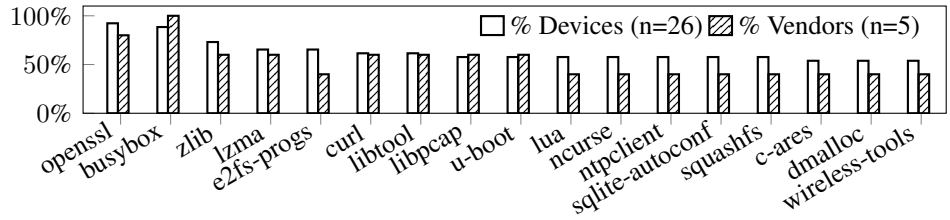


Figure 2.2: List of the most common libraries in all 26 devices across vendors. Among 26 devices, over 50% use these libraries. The most popular ones, OpenSSL and BusyBox, are used by 92.31% and 88.46% of devices. We also show the percentage of vendors who use these libraries on their devices.

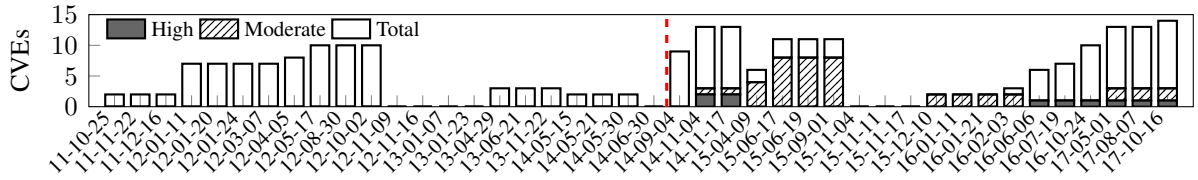


Figure 2.3: Number of publicly known OpenSSL CVEs in firmware releases. X-axis shows the firmware release date. We do not have CVE severity breakdowns for data prior to August 2014 (the red dashed line). For newer libraries, we find many High and Moderate CVEs present in the firmware.

Patching Practices. In this section, we perform a longitudinal analysis of firmware release history to understand the security risk of third-party library usage and poor management. By cross referencing each device's firmware release with the history of the library's security updates, we can analyze vendors' patching behaviors in terms of average delays in responses and the duration for devices to operate with outstanding exploits.

We focus our analysis on the OpenSSL library, one of the most widely used libraries in the dataset. We leverage OpenSSL's well-documented history of security updates and exploit disclosures [58] as a reference for vulnerabilities in each library version. Since the historical data are only available for Nest and D-Link devices (Table 2.1), we pick the 100 firmware releases from these vendors that use the OpenSSL library, spanning a 7-year period.

Overall, we observe that IoT devices do not use the latest available library versions for the vast majority of time. For example, the Nest Learning Thermostat only uses the latest available library 21% of the time ($465/2183$ days). For the remaining time, the device firmware operates with outdated versions. Even worse, the Nest Protect and all of the D-Link devices in the dataset never utilize the latest OpenSSL library over multiple years. They keep operating with legacy versions for as old as over a thousand days behind the patching schedule.

As a consequence, even new firmware often contain multiple publicly-known vulnerabilities *as soon as* they are released. We take an in-depth analysis using the example of Nest Learning Thermostat. Figure 2.3 presents the number of OpenSSL CVEs in each Nest firmware release. We also include the CVE severity breakdowns for dates after August 2014 (when data are available). It is important to note that all of these vulnerabilities can be avoided if the firmware uses

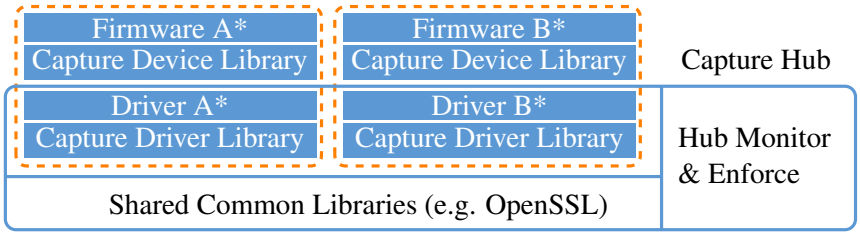


Figure 2.4: Capture system architecture. Every device consists local device firmware and driver on the hub. They form a logical unified entity, Virtual Device Entity (orange dashed box). The Capture Hub maintains a central version of common libraries and has extra monitoring and enforce modules.

the latest available OpenSSL versions at the time of its release.

2.3 Centralized Library Management for Heterogeneous IoT Devices

To mitigate the security threats from using outdated third-party libraries, we propose Capture, a novel system architecture for deploying IoT devices with centralized library management support. This section briefly describe the design of Capture and key technical insights in the design and implementation of Capture's architecture.

2.3.1 Design Overview

We introduce the design of Capture Hub, a central entity in the local home network that manages library security updates and enables other devices to share the library runtime. Figure 2.4 illustrates the system architecture in a home network with multiple devices. Every Capture-enabled device consists of two parts: firmware on the device (Firmware A* and B* in the example) and drivers on the hub (Drivers A* and B*). Developers can leverage the API from Capture libraries to implement both components, or use the default drivers to migrate existing applications. Since the hub maintains the shared libraries, individual devices can only use the libraries in their drivers. The Capture Hub's Monitor and Enforcement module manages all drivers and provides runtime and network isolation to ensure integrity of individual devices.

Threat Model and Security Goals. We assume the central hub (i.e., the Capture Hub) is trusted, and all standard network protocols and Linux security tools we used are up-to-date to address any vulnerabilities. We assume IoT devices need to communicate with arbitrary hosts for functionality purposes, making whitelist-based approaches limiting devices to selected network hosts (e.g., vendors' cloud backend) [35, 42] too restrictive.

We consider an adversary who seeks to compromise IoT devices through publicly-known vulnerabilities in outdated libraries. To achieve this, the adversary can attempt exploits from the public Internet (outside of the local network) or affect more co-located devices in the same home

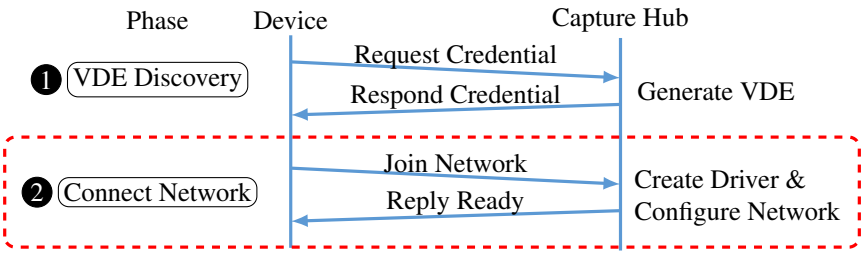


Figure 2.5: Device bootstrap procedure for network isolation. In Step 1, the device connects to the Capture Hub using a shared setup network. Then it joins a VDE-specific VLAN network in Step 2 (dashed box).

deployment through already compromised ones. Attack vectors from zero-day vulnerabilities (i.e., no patches available) and non-library vulnerabilities (e.g., weak passwords) are outside of the scope of this work. In addition, we exclude side-channel attacks rising from the shared hub access for multiple drivers in our threat model, although mitigating side channels in shared execution environment is relatively well studied by prior works [54].

Capture sets out to achieve the following security goals.

- **Prevent library exploits.** We want to centralize the management of third-party libraries so that local devices can enjoy the security benefits of using the latest libraries without the additional burden.
- **Preserve device integrity.** Because of Capture's firmware splitting design, we should protect the communication integrity between the device and the driver running on the hub, and prevent any entity from intercepting or interfering with the communication.
- **Strong isolation.** To prevent compromised devices from affecting other local devices, we must enforce strong isolation between devices and drivers. This goal raises additional challenges as drivers share the same Capture Hub as execution environment.

2.3.2 Device Virtualization and Isolation

Virtual Device Entity. The key insight of Capture is to virtualize existing, holistic IoT device firmware into a *Virtual Device Entity* (VDE) and enforce strong isolation across VDEs. A VDE consists of the Capture-enabled firmware on the device and an associated software driver on the hub. Figure 2.4 illustrates two VDE examples. The Capture Hub ensures the communication integrity and confidentiality within individual VDEs and provides isolation across VDEs with approaches described in later sections.

Communication Isolation. A Capture-enabled device effectively operates in a "local-only" mode since it can only communicate with its driver within the same VDE. To preserve integrity and confidentiality, the Capture Hub isolates VDE's network and blocks cross-device and cross-driver communication. We achieve this design by leveraging virtual network interfaces and subnets.

Figure 2.5 illustrates the process of new device bootstrapping and the initialization of VDE-

specific network. A device first connects to the *setup network* with pre-shared credentials, similar to existing home WiFi (Step ①). The hub creates a fresh VDE credential and generates a new. virtual network setup for the VDE. After receiving the credential, the device disconnects from the setup network and reconnect to the *operation network* (Step ②). The operation network is unique for each VDE so the hub can easily enforce isolation.

The Capture Hub maintains multiple virtual VDE networks by managing two separate WiFi Access Points (APs) simultaneously. The first AP is for the setup network shared by all uninitialized devices. We set it up with WPA2-Personal protocol, the same as traditional home networks. The second AP is set up with WPA2-Enterprise security protocol for the operation networks. The hub generates unique credentials (using RADIUS server) for every VDE and bound each physical device with its own virtual network interface (vNIC). The hub only needs a single WPA2-Enterprise AP to support multiple operation networks in parallel. To limit drivers' communication capability, we enable TOMOYO [75], a Linux security module, to bind the driver to the VDE-specific vNIC, and use `iptables` to block other network communication.

Resource Isolation. Since multiple drivers run on the same Capture Hub, it is important to impose secure and fair resource sharing on the hub. One option is to containerize all drivers to achieve process isolation. However, they are not suitable for Capture since every container will have its own library dependencies and runtime. Instead, Capture centralizes the library management by maintaining a single copy of the up-to-date library and enable drivers to share the library with Capture APIs.

Capture utilizes lightweight Linux security primitives to isolate processes and limit the driver's access to system resources. We assign all processes from the same VDE into their own TOMOYO security domain and enforce security policies to limit their access to the network and filesystem. Each VDE also has its own Linux user account on the hub, so we can utilize all standard Linux filesystem and memory protection mechanisms. We also leverage `cgroups` [34] to enforce fair sharing of computation resources across different VDEs.

2.3.3 Security Analysis

In this section, we analyze Capture's design and how it can prevent internal and external threats from compromising our security goals. We believe our analysis is comprehensive and demonstrates Capture is secure by design, under our threat model (§2.3.1).

External Threats. Capture protects devices from external threats (i.e., Internet attackers) by removing connectivity of local devices and securing the companion drivers on the hub. Since the Capture Hub maintains the third-party libraries, drivers can utilize the latest library versions whose vulnerabilities are patched automatically. As the public-facing components, drivers in every VDE are reachable from the Internet but meet our security goals of being vulnerability-free.

On the other hand, the actual device may still contain outdated libraries in their firmware. Capture's network isolation prevents external entities from reaching local devices and exploiting the vulnerabilities. This security protection is contingent on vendor adoptions and proper

implementation of the driver software.

Internal Threats. We consider internal threats which include compromised devices, drivers, and other devices within the WiFi range. Capture prevents compromised *devices* from attacking other VDEs through network isolation, because each device is bound to its own vNIC and can only communicate with its driver. Similarly, compromised *drivers* are also isolated from other VDEs through the network and resource isolation on the hub. Moreover, malicious devices (including Capture-incompatible ones) cannot eavesdrop VDE credentials from the setup network, which is shared by all uninitialized devices. This is because WPA2 uses link layer encryption with unique keys for individual devices [49, 81]. However, this encryption is insufficient for Capture's VDE isolation because all drivers running on the hub share the same link layer.

An adversary can potentially impersonate as the Capture Hub and perform man-in-the-middle attacks during the device bootstrap process. This threat can be mitigated by using techniques such as certificates and public key infrastructures for identity verification. Alternatively, we can integrate recent works in secure device bootstrapping to alleviate this issue [29, 68].

2.4 Integration Approaches

We envision one of the major adoption challenges for device vendors is to integrate existing devices with Capture. In this section, we describe several integration solutions aimed at reducing developer efforts. Our goal is to provide paths of least resistance to help with device vendors while providing programming flexibility.

OS Library Replacement. The first approach is to provide a Capture-enabled version of the standard OS library. For example, ESP32 microcontrollers use APIs from the `WiFi.h` header file for networking functions. We provide a `CaptureWiFi.h` header that is fully compatible with the default library APIs. Developers just need to replace the header file and import the new Capture runtime library. We also provide a default Capture driver, so developers can directly integrate their devices into Capture. However, this approach is heavily platform-dependent and it will require more engineering efforts to extend to more operating systems.

IoT Framework SDK Extension. Similar to the previous approach, we implement another drop-in replacement library for a common IoT development framework SDK. IoT frameworks (e.g., Azure Sphere [52], Particle OS [61], and Samsung SmartThings Device SDK [65]) provide rich set of functionalities across multiple hardware platforms. They help application development by integrating with their native cloud backend and services.

We develop a replacement library for the Samsung SmartThings Device SDK (ST-SDK) as a proof of concept. We select this framework because its SDK is open-source so we have a reference implementation. In addition to the typical networking functionaility, ST-SDK also facilitates MQTT messages with the existing SmartThings Hub (ST-Hub) as a data broker. We recreated similar functionalities with a Capture-enabled ST-Hub and provide default drivers for ST-SDK devices as well.

Native Driver Development. The previous two approaches provide default drivers on the Capture Hub to aid developer adoption. In addition, we provide a full set of Capture APIs for developers. With the added flexibility, developers can create their own device drivers to achieve better performance and efficiency.

2.5 Overview of Evaluation

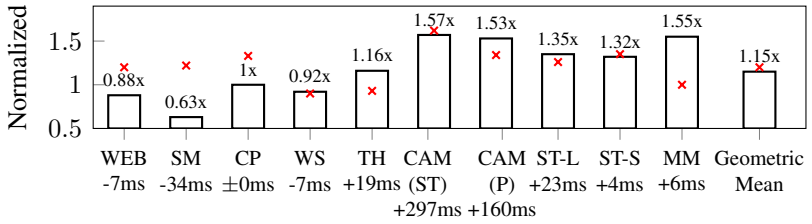
We now briefly summarize the results of our evaluation of the Capture prototype with several smart devices implemented from the open source repositories. A more detailed evaluation is available in our full paper [92].

Implementation. We implement the Capture Hub using a Raspberry Pi 3B+ with Linux in 1874 lines of C++ and we have open-sourced our prototype implementation at `https://github.com/synergylabs/iot-capture`. We integrate 9 prototype IoT devices and applications into the Capture prototype on a variety of hardware platform and operating systems (ESP32, Raspberry Pi, Samsung SmartThings, and Linux).

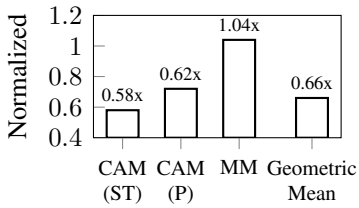
Optimizations. Our baseline applications often use blocking network calls since their programming paradigm directly fetches payload from the network buffer. This assumption no longer holds after we integrate these applications into Capture, since those networking calls will be translated into communication between the device and driver. Keeping these blocking network calls as-is leads to a latency increase up to 9.56x. We implemented several optimization solutions to alleviate this added communication overhead. First of all, we implement read and write buffers on the device to prefetch network packets and buffer outgoing messages. This buffering reduces latency from 9.56x to 1.62x. We further analyzed the network packets with Wireshark and identify the latency bottleneck from sending multiple small packets. As a result, we aggressively merge smaller packets into bigger ones and extend the protocol header fields to support multiple packets. With these optimizations, we reduce the total communication overhead for each operation to around 10 ms; as a reference, it takes the IoT device (ESP32 board) 5-6 ms to send a single network packet.

Performance Overhead. We measure Capture's overhead running the 9 prototype apps and compare them with the original versions. Figure 2.6 shows the relative overhead normalized with the original baselines. Most apps experience a latency increase less than 23 ms. Sometimes the latency improves because they can offload computationally expensive operations to a more capable platform (ESP32 to Raspberry Pi). The camera app experience a significant latency increase ($+297$ ms) and throughput reduction (-42%). However, the relative increase is on par with other apps (e.g., MagicMirror) so we conclude that even the original version is not suited for real-time monitoring use cases.

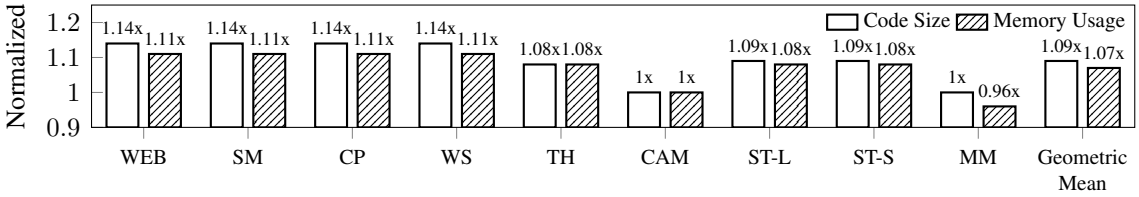
In addition to microbenchmarking with prototype apps, we integrate Capture-enabled devices with a home automation platform (IFTTT [38]) to measure its real-world performance impacts. We create three aotumation applets involving the device and measures the end-to-end latency



(a) Average latency and numerical differences. Red crosses show *median* latency changes.



(b) Average throughput.



(c) Application code size and memory usage on-device.

Figure 2.6: Performance overhead for all prototype apps. Data are normalized to results from the original apps. Based on geometric means, Capture-enabled devices experience 34% latency increase, 34% throughput reduction, and 10% more on-device resource utilization. Abbreviation for these apps are Web Server (WS), Servo Motor (SM), Color Picker (CP), Weather Station (WS), Temperature and Humidity sensor (TH), Camera (CAM) in streaming and picture-taking modes, SmartThings Lamp (ST-L), SmartThings Switch (ST-S), and MagicMirror (MM).

on different Capture devices (ESP32 and Raspberry Pi). Overall, we do not observe significant differences and attribute the majority overhead to the automation platform's cloud backend, since their latencies are usually several seconds [51].

Scalability. The main scalability bottleneck is the computation resources on the Capture Hub, since all local devices require a driver to be executed on the hub. We identify the hub's memory capacity as the key limiting factor compared to other resources such as CPU, IP addresses, and network interfaces. Drivers' memeory consumption ranges from 3.7 MB to 42 MB, depending on their implementation. Therefore, we emulate a deployment of 40 devices using the smallest drivers and 10 devices using the largest drivers. This emulation consumes 664 MB memory on a Raspberry Pi 3B+ based Capture Hub (1 GB RAM, quad-core) and the CPU load average never exceeds 0.8 (max of 4.0). While RAM capacity is a limiting factor, alternative hardware platforms (e.g., Raspberry Pi 4 with 8 GB RAM) for the Capture Hub can improve the scalability to hundreds of devices.

2.6 Conclusion

This section of the proposal illustrates the benefits of offloading library management responsibilities for IoT device vendors to a trusted third-party. Based on historical data, we can identify the necessity of better library management and timely security patching for future IoT devices.

To achieve our goal, we propose the design of Capture, a new system architecture for IoT device deployment. We demonstrate that Capture is a practical solution to mitigate the security challenges rising from the prevalent use of outdated libraries while maintaining the security and performance requirement of individual IoT device vendors.

Chapter 3

Protecting IoT Device Users by Offloading Ownership Management and Access Control

The previous section showed the opportunity to improve the security protection of individual IoT devices through offloading the library management tasks to a central service. In addition to security concerns, many IoT device users worry about the potential privacy implications with the ubiquitous IoT device deployment and the breadth of the sensitive data they can collect [9, 22, 31].

Although IoT device vendors already have basic access controls in their existing applications, those systems are not sufficient to handle complex use cases in future IoT deployments. Specifically, many prior works have looked into how to improve IoT device's authorization and delegation systems' expressiveness [8, 25, 39, 66], decentralized storage models [7, 21, 62, 69], and cryptographically enforced access policies [37, 44, 47, 59, 70, 71, 80]. One common pitfall of these approaches is that they primarily focus on addressing privacy concerns of the device owners and fail to take into account of concerns from other stakeholders. For those whose data may be captured by the IoT device, they have to trust the device owners or the small group of device administrators.

In this section of the thesis, we describe our work on the design of a new IoT device ownership model (TEO — Io*T* *E*phemeral *O*wnership) giving direct controls to the device stakeholders — who may be impacted by the presence of the device — and the implementation of a new system architecture enabling such a model. This architecture splits the role of maintaining secure data storage and managing access controls of users' private data from the list of responsibilities of current IoT device vendors into a third-party service. More importantly, we limit the trusted computing base and avoid any trusted requirement for the cloud and storage service providers. At the same time, the owners of the data retain controls and make the ultimate decisions on who can access their data. To provide strong security guarantee, we apply formal security analysis and verify the proposed protocol design satisfies all of our security goals.

The rest of this section describes TEO at a high level. Additional details on our completed work in this space are presented in a full paper on this topic [93].

3.1 Background

This section provides background on the challenge of protecting stakeholders' privacy in future IoT deployments and the limitations of many existing access control systems for current IoT devices.

Stakeholder Privacy. Recent works have identified the emerging challenges due to the discrepancy between decision makes and device users in complex IoT deployment scenarios [15, 31, 91], motivating the need for stakeholder privacy protection. Some of the prior work examine privacy issues from the perspective of bystanders [9, 83] and incidental users [14]. Moreover, research on preventing intimate partnet violence (IPV) for smart devices [27, 28, 30, 60, 78] also echoes the needs for stakeholder privacy. All of these works touch upon the of an effective IoT device access control mechanism capable of protecting all stakeholders of the device — anyone using the device or may be impacted by the device because they are in the vicinity of the device — and motivate our thesis's contribution.

IoT Device Access Control. Motivated by real-world incidents and research findings [85], researchers have proposed many improvements on existing IoT access control systems, as surveyed by He et al. [32]. A few improvements include expanding the policy language with more expressiveness [8] and contextual information in smart homes [25, 39, 66] and extending the access control to support multiple users and devices in the same home [73]. Moreover, many new frameworks leverage decentralized solutions to distribute the trusts in enforcing access controls [4, 7, 21, 62, 69]. Finally, the use of novel cryptographic constructions to facilitate access control and delegation has been presented by several recent works [37, 44, 47, 59, 70, 71].

3.2 Enabling IoT Ephemeral Ownership

To address stakeholders' privacy and security concerns, we envision a new model of device ownership that protects the interests of both the device *users* and its *administrators*. We propose the design of TEO — Io*T* *E*phemeral *O*wnership — that grants users, as ephemeral device owners, full control over the device's operation. Historical data collected by the device will always belong to their ephemeral owners at the time of collection. When someone wants to access the data, they must receive permissions from all of the data owners (stakeholders when the data are collected). Meanwhile, device administrators can decide who is capable of becoming an ephemeral owners, but they can not interfere with the device's operation or access private data captured by the device without owners' approval.

3.2.1 Target Use Cases and Design Goals

TEO aims to provide ephemeral owners full control over who can access their data and when, agnostic of the data type. Therefore, TEO is well suited for applications and IoT devices that store operational data potentially containing sensitive information about their users, such as recordings from cameras and speakers or sensor readings. For data access requests, we primarily focus on

use cases where someone wants to request historical data for analysis purposes, such as training machine learning models or review past events.

Since TEO devices maintain their current list of owners, we can extend TEO to enable real-time access control of the device as well. For example, a TEO-enabled smart door lock should reject commands issued by previous owners after a new user claims exclusive ownership. To achieve this, the device can require all incoming commands include an authorization certificate signed by the owner. In addition, we believe it would be a promising future work to extend TEO with proof-carrying authorizations [6, 8, 10, 45] for more expressive policy specification languages.

TEO targets a variety of IoT deployment scenarios beyond personal homes, such as rental homes and shared offices. They have different design requirements nd considertaions. In rental homes (e.g., Airbnb), the host sets up smart devices and lets guests use them. Guests have lower churn rate (each stays a few days) and smaller group sizes. Sometimes, a single owner would suffice as group members implicitly trust each other if they stay together. On the other hand, smart devices in shared offices and conference rooms have more frequent user changes and the group of users often expect to have an equal say in decision makings. Although these devices are managed by the building managers, TEO aims to protect the actual device users with their private data.

Design Goals. We want to achieve the following design goals with the new ownership model:

1. **Flexible Association of Devices and Users.** We expect frequent occupancy changes in the physical spaces with smart devices. A conference room may see ownership change happening on an hourly basis. Moreover, multiple users can share the same space and hence be collectively impacted by the device. Ideally, all stakeholders should have a say in controlling the device and managing the access decisions of the data.
2. **Preserving Data Ownership.** Data collected by smart devices should always belong to the group of users present at the time of capture. Anyone trying to access the data should request the data owners' permissions. More importantly, dynamically changing ownership of the device's users and administrators should not affect historical data.
3. **Decentralized Trust.** Users should be able to manage access requests themselves, without relying on any external trusted third-parties. Centralized access control systems require users to fully trust the system maintainers (e.g., company and building owners) and in their capability of protect user data and enforcing correct privacy policies. We want to empower users to decide who can access their data without compromising the benefits of high efficiency and availability currently provided by centralized services.
4. **Verified Security.** Our goal is to provide strong security guarantees for our proposed TEO system. To do so, we leverage formal methods and verification tools to reason about our design and potential security vulnerabilities. Since the main component of TEO is a set of complex communication protocols designed to enable the new ownership model, we turn to the protocol verification tools to provide assurances of TEO's security and correctness.

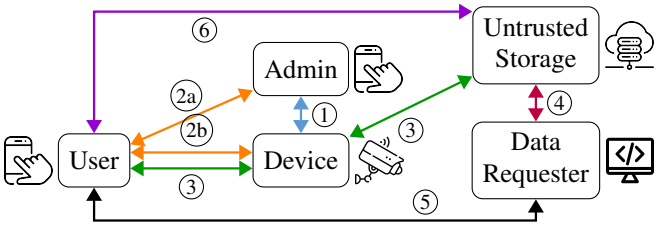


Figure 3.1: Overall TEO workflow. An admin initializes the device (①). Next, the user claim device ownership with the admin's pre-approval (2a and 2b). During normal operation (③), the device encrypts users' data and uploads it to storage. A requester can download the data (④), but needs the owner's approval to decrypt it (⑤). To revoke access, the user can directly issue a request to the storage provider (⑥).

3.2.2 Threat Model

We design TEO under the assumption of a power attack who can monitor all network communications and attempts to undermine TEO's design goals by either (1) controlling the device without active consent of the ephemeral owners, (2) accessing the data generated by the device without owners' permissions, or (3) impersonating one of the parties in the TEO architecture. Concretely, such an attacker might correspond to someone within the vicinity of the device, a malicious admin who wishes to violate the privacy of an ephemeral owner, or a previous device owner who aims to extend their control of the device and its data past the agreed-upon terms.

We assume that local devices are trusted to correctly execute the protocol and will not maliciously leake user data, encryption keys, or bypass authorization checks. We assume that third-party storage providers may be passively malicious (i.e., semi-trusted, honest-but-curious): they might attempt to extract private information from the data they receive but will faithfully execute the TEO protocol as specified. This is consistent with using reputable cloud service providers, with whom users may not trust storing cleartext data but for whom the reputation risk stemming from actively-malicious behavior is too great.

3.3 TEO Workflow

In this section, we provide a high-level overview of TEO's workflow and its key technical contributions. For more in-depth information, please refer to our full paper [93].

Figure 3.1 illustrates the workflow of multiple TEO entities. The admin of the device (e.g. the Airbnb host) first installs the device and initializes it (Step ①). Afterward, the device is ready to be claimed by the new owners only if they are authorized by the admin. The potential users (e.g., Airbnb guests) all have a user agent program running on their phones. After booking their reservation, they need to ask the host to issue "pre-auth tokens" with everyone's public keys (Step 2a). Pre-auth tokens prevent unauthorized people such as malicious neighbors from access the device. When users arrive at the rental home, the user agent on their phones initiate the ownership claiming process (Step 2b). As the group membership changes (users join and leave), the device dynamically adjusts the set of current owners. The device preserves users' data ownership with a series of encryption operations and distributes individual keys to each owner (Step ③) and

uploads the encrypted data to third-party cloud providers for storage. When someone wants to access the data (Step ④), they need to seek permissions from the original owners to decrypt them (Step ⑤). Later on, if the user wants to revoke the access, they can contact the storage provide directly and issue re-key tokens to invalidate previously distributed decryption keys to the data requester (Step ⑥).

3.3.1 Data Storage and Access

To preserve users' ownership of the data generated by the device, one approach could be to transmit data directly to the user and let them store the data locally. This would be too demanding for average users in terms of technical knowledge and computation resources. Instead, we leverage third-party cloud storage providers to facilitate the storage challenge. Since we do not trust the storage provider with users' raw data, the device must encrypt the content before uploading to ensure confidentiality. This poses several challenges, including how to enable revocable access controls and how to support dynamically changing user groups.

Revocable Access Control. To manage access to encrypted user data, data owners can decide who to share the decryption keys with. However, standard symmetric key encryption can not address the need for revoking third-party accesses. An inefficient approach to do so would be to download all ciphertext to the user's device, decrypt and re-encrypt with new keys, then upload them again. Instead, we leverage key-homomorphic encryption algorithms proposed in prior works [80] to implement an efficient, revocable access control mechanism.

In addition to encrypting users' data with fresh session keys (with symmetric key encryption), the device performs another round of encryption to protect the value of the session key using key-homomorphic encryption to support efficient revocation. When delegating access, data owners can share the key used for the homomorphic encryption so the data requesters can eventually decrypt the original data. To revoke access, data owners can create a fixed size rekey token and send it to the storage provider. The storage provider can apply this token in-place on the ciphertext to change the decryption key, without being able to access the underlying plaintext.

Group Ownership. To extend TEO to support group ownership, we leverage threshold encryption and Shamir Secret Sharing [72] to split the single session key into multiple key shares. We create one share for each owner in the current group and send it to the corresponding owner. Owners manage their key shared independently and make access delegation decisions individually. To access the shared data, the requester needs to seek permissions from each owner and obtain enough key shares to reconstruct the session data key.

We conclude by noting that threshold encryption can support several data access policies by adjusting the threshold value necessary to reconstruct the original content. Currently, TEO requires accessors to have every group member's approval because we want to give everyone the right to veto the request. It is straightforward to extend TEO with alternative policies such as majority approval.

3.3.2 Formal Security Analysis and Protocol Verification

We want to verify and ensure that TEO's design is secure and does not contain any vulnerabilities compromising the intended security and privacy protections. To obtain a definitive answer on our design's security and correctness, we leverage protocol verification tools to formally analyze TEO. We choose a *symbolic* protocol verifier (ProVerif [11]) among other alternatives (e.g., computational verifiers) because it requires lower human guidance and is better suited for automated analysis given we want to identify design-level bugs in TEO's protocol. Hence, we implicitly assume the cryptographic primitives are secure and exclude their computation details from the formal security analysis.

Formalize Security Goals. We set out to achieve the following security goals for TEO.

- **Secrecy.** A user's private data should not be accessible by anyone without explicit authorization of the user. For group ownership, the policy requires that only entities with the consent of all owners are able to access the data.
- **Mutual Authentication.** After the device is initialized and claimed by owners, all parties must mutually authenticate and agree on each other's roles.
- **Resilience to Data Spoofing.** Attackers should not be able to spoof data, potentially overwhelming users' local storage space with keys for non-exist data sessions. If the user concludes a data store operation, then the device must have indeed stored the user's private data for the corresponding session.
- **Effective Revocation.** If the owner revokes someone's access, the they should no longer be able to decrypt the data if they download the ciphertext again from the storage provider. Conversely, revocation should only happen when the owner requests, and the new key should be able to used to decrypt the data in the future. Note that TEO does not preclude a requester from storing the decrypted data offline in perpetuity.

We encode every security goal with concrete ProVerif's reachability and correspondence queries. We translate these security goals in the format of *"if A happens, B must have already happened"* and ask the tool to search for property violations. In addition, we add secrecy queries to ensure user's private data stay unknown to the attacker.

Expand to Group Ownership Models. To support group ownership, we utilize Shamir Secret Sharing as explained earlier. However, ProVerif currently does not have native support of threshold encryption [55], particularly for encoding variable-sized sets of co-owners. To address this limitation, we encode the size of the owners statically in the model. Moreover, all parameters used for cryptographic operation must also be set statically, such as the number of users and decryption threshold. We have to create unique processes for every user in the group to manage their internal states and key shares. As the group size grows, it is very challenging to manually encode these parameters and processes, especially since we often change the designs in the iterative protocol revision process. Therefore, we develop a custom template language that automates the generation of static models and implement a preprocessor to compile the protocol template into concrete ProVerif models with configurable group sizes.

Operation	User App Battery (μAh)	Average Latency $\pm$ Standard Deviation (*ms*)	
		RPi 4	RPi Zero
Initialize Device	20.18	44 ± 9	65 ± 35
Acquire Pre-Auth Token + Claim Device	34.43	187 ± 52	258 ± 128
Claim Device	21.19	67 ± 10	94 ± 31
Store Data, 1MB	43.03	308 ± 57	684 ± 155
Access Data, 1MB	22.25	170 ± 54	
Revocation and Re-encrypt	25.07	62 ± 15	

Table 3.1: Average latency (in *ms*) for TEO operations, with a performance comparison of different IoT device hardware. We also measure battery usage for the TEO phone app (in μAh). Data access and revocation operations do not involve devices' participation.

Data Size	Data Encryption	Data Upload	Total Time (vs. Upload Time)
10KB	< 1	19	101 (5x)
100KB	2	29	116 (4x)
1MB	25	127	308 (2.42x)
10MB	168	1429	1791 (1.25x)
100MB	1577	15256	16293 (1.07x)

Table 3.2: Data store operation mean latency breakdown for Raspberry Pi 4, in *ms*.

3.4 Overview of Evaluation

This section provides highlights of TEO's evaluation on our prototype implementation. Our evaluation results demonstrate that TEO introduces nominal communication and power consumption overhead. For one-time operations like device setup and ownership transfer, TEO adds additional latency of up to 187 ms. When the device continuously storing user data during operation, TEO adds 7-25% extra latency compared to directly uploading the content to the cloud storage for larger file sizes and 101-308 ms for smaller sizes.

Implementation. We implement our TEO prototype on multiple platforms and release an open-source repository at `https://github.com/synergylabs/TEO-release`. We develop an Android app as the mobile agent program for end-users and device administrators. We also implement test clients for different roles on x86 Linux desktops and popular single-board computers with ARM SoCs (Raspberry Pi 4 and Zero W). Moreover, we develop a storage provider daemon as a key-value store for encrypted data contents with support for revocation.

Performance Overhead. Table 3.1 presents TEO's battery and latency overhead in microbenchmark results. Several operations such as device initialization and revocation are lightweight, while the initial claiming of a device has higher latency. Although operations related to data storage and access requests seem to have very high latency, we need to compare them to their

baseline of data uploading and downloading to the cloud storage providers for a fair comparison. Table 3.2 shows a breakdown between data upload time and encryption time in such operations. For larger data sizes, the relative encryption overhead is much smaller (7-25%).

Integration Efforts. We integrate three real-world smart IoT applications into TEO-enabled devices. We found them by searching for popular open-source IoT applications on Github and tutorial websites. In all three cases, we extend their original implementation with new functionalities using TEO for secure data store and real-time ownership management. In general, the integration process requires minimal codebase changes (¡121 lines) because we provide TEO device driver as a standalone program with REST APIs exposed to other programs running on the localhost. Overall, the performance overhead in these applicaitons matches our microbenchmark results.

3.5 Conclusion

This section of the proposal illustrates the added benefits for IoT device vendors to offload the role of ownership management and access control enforcement to third-party services. Specifically, device users can enjoy better privacy and security protections and directly control how their data can be accessed. By proposing the design of TEO, we materialize this vision with a set of formally verified operation protocols to ensure their correctness and security. We demonstrate TEO's practicality in terms of low performance overhead and ease of device vendor adoptions with our evaluation results from the prototype implementation.

Chapter 4

Proposed Work: Efficient Computation Offloading for IoT Devices with Neural Network Applications

Thus far, this proposal has been focusing on addressing the security and privacy concerns, mostly from the device users' perspective, and assumes device vendors would be interested in adopting these approaches to stay attractive to end-users. In the final part of this thesis, we will investigate another opportunity for functionality splitting that is directly relevant to the device vendors' incentives and provides stronger motivation for device vendor adoption.

There is a mismatch in the computation demands and the total available computation resources in a deployment of heterogeneous IoT devices. Currently, device vendors have to rely on their individual cloud backend to perform heavyweight computations because of limited resources on the device. Meanwhile, many other IoT devices and general purpose computers are sitting idly in the same user's home.

We propose a new architecture, VeriSplit, that enables local IoT devices to share computation resources, with a specific focus on the emerging machine learning applications. By offloading computation to other devices, vendors of the *offloading devices* can reduce spending in their cloud backend for those transient computation demands. On the other hand, the *worker devices* can receive incentives for leasing out their idle resources.

To enable computation offloading across devices from *different vendors*, we must address several interesting security and efficiency challenges. First, offloading devices must be able to verify the correctness of the computation results generated by worker devices. Workers may generate wrong results because there is no explicit trusts between the two parties. Second, the local offloading overhead must be reasonable when compared to the existing cloud offloading solutions. This is especially challenging with the additional communication needed for verification. Finally, offloading devices may want to keep their machine learning models private from the worker devices. However, workers need the model to perform computations. Therefore, it is important to provide model privacy in the offloading mechanism.

The rest of this section describes the preliminary design of the VeriSplit system architecture and our proposed solutions for addressing the challenges for verifiable offloading of machine learning applications in IoT devices. We will implement these solutions and perform a compre-

hensive evaluation of the system prototype to finish the remaining work of the thesis.

4.1 Background and Motivation

Computation Offloading. Nowadays, smart home users often have multiple IoT devices installed and these devices have a wide range of hardware capabilities. For example, a cheap smart camera may only have the bare-minimal hardware to stream videos to the vendor's cloud, where many machine learning applications are executed in order to present interesting events and features to the device user. Meanwhile, more expensive IoT devices (e.g., $1000 robot vacuums [63]), computers, and gaming consoles are equipped with powerful hardware (potentially with accelerators for machine learning) are sitting idle in the user's home most of the time. It is a great opportunity to pair these devices together to satisfy the computation demands.

Many prior work have proposed frameworks to enable local computation offloading. One of the earliest system is HomeOS [19], which aims to aggregate all local sensors and actuators into one central server to process. More recently, many approaches use a trusted local hub to execute the offloaded functions [3, 41, 86]. All of these approaches require trusts between the offloading and the worker devices, creating a very wide trusted computing base.

Verifiable Neural Network Inferences. With the rapidly growing popularity of deep learning applications, many recent works have looked at the challenges of verifying the inference results when the computation is offloaded to third-parties. However, most of them focus on facilitating verifiable offloading between cloud servers and hold assumptions unsuitable for the IoT use cases. First, many approaches still require a trusted computing base in the worker devices, usually using secure enclaves [36, 76, 79]. Second, they often incur significant communication overhead in order to maintain very high accuracy in verifying a single inference result [53, 76]. These overheads might be acceptable in the datacenter networking condition, but are not suitable for many IoT devices connected over WiFi.

4.2 Preliminary Design

Figure 4.1 illustrates the high-level workflow of offloading an inference in VeriSplit. The verifier first sends the input to the worker, and wait for the results along with the inference commit. The commit is a hash value constructed from all of the neural network intermediate results in this inference. After receiving the inference result, the verifier request part of the intermediate values from the worker for verification. The verifier repeat the computation to generate the selected values. If the results match and the verifier can recreate the original inference commit hash value with these values, the verifier accepts this inference as correct.

The key insight in VeriSplit is that all of the intermediate values of a particular inference is encoded into the inference commit succinctly and shared with the verifier (offloading device) along with the inference result. This commit allows verification to be performed *after* the inference is complete. Therefore, the verifier can dynamically decide which past inferences to check

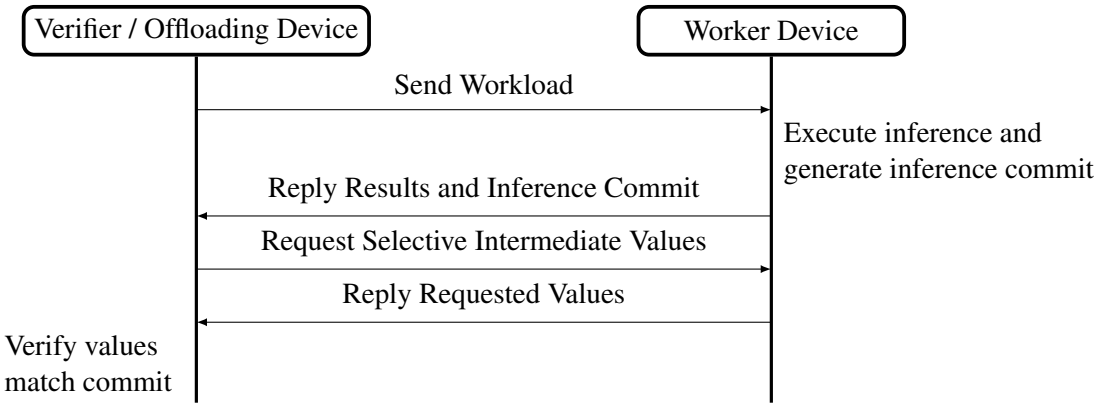


Figure 4.1: Workflow of VeriSplit in executing and verifying an inference.

and what part of the intermediate values to verify whenever it is idle and has free computation cycles.

4.3 Performance Overhead and Selective Verification

The goal of VeriSplit is to consistently offload over a long period of time, ensuring the worker generates correct results in the long run with high probability. Prior works focusing on verifying single inferences incur a high overhead because they have to check all of the intermediate results. Since there are multiple inference offloading between the same pair of devices, VeriSplit verifiers can randomly select part of the intermediate values in every inference to verify, reducing the communication overhead while retaining high confidence of the correctness. If there are large numbers of offloading during a fixed time period, the verifier only needs to check a small portion of the results. On the other hand, if offloading happens rarely, the verifier can check one inference in its entirety.

To enable selective verification of partial results, we have to break down the outputs of neural network inference into smaller auditing units, and construct a merkle tree hash of all of these units. Selecting candidates to check on a layer-by-layer basis is too coarse and will yield inconsistent verification overhead, because the output size of each layer varies significantly. Instead, we choose to further splitting the intermediate activations into smaller sub-matrices to enable granular verification selection.

4.4 Numerical Errors from Architectural Differences

Generating the correct inference commits for verification requires both the worker and the verifier have the exact same values for the intermediate results. This should not come as surprises because we want to rely on hash function's collusion resistances to ensure correctness. Unfortunately, it turns out that devices might compute slightly different results using the same model parameters due to the randomness in parallelism on different hardware platforms.

One concrete example is that we will get different results running GPU inferences versus CPU inferences. Although the amount of differences (i.e., δ) is insignificant, this yields completely different hash commits and leads to high false positive rates of verification. On the other hand, if we simply accept any differences below a threshold to eliminate false positives, malicious workers can easily create incorrect inference results without getting detected (i.e., high false negative rates). Therefore, we empirically evaluates several δ-mitigation strategies and found the optimal ones with low false positives and false negatives.

4.5 Model Privacy

So far, we have been focusing on designing VeriSplit for public networks and transfer learning applications in IoT devices. In transfer learning, developers create their own model by extending pre-trained public networks with custom classification layers in the end. The vast majority of the computation still happens in the public networks for purposes like feature extractions. Therefore, to preserve model privacy of the offloading devices, they can only choose to offload the public part of their network models and keep running the final layers on the original devices. This solution will be useful to alleviate the demand for transient computation resources to certain degrees.

In addition, we will look into alternative solutions to further protect model privacy for offloading devices to reduce the adoption concerns of device vendors. One promising technique is Fully Homomorphic Encryption (FHE) and its application within neural networks. Although network models integrated with FHE still exhibits significant overhead than their baselines, we believe it is a promising direction given its recent, orders of magnitudes of speedups from optimized algorithms and hardware accelerations [40]. To provide model privacy, device vendors can encrypt the model weights with secret keys before sharing with the worker devices. During inference, the worker can only compute over encrypted model parameters and generate a final results in encrypted ciphertext. Since only the offloading device can decrypt the ciphertext, device vendors do not need to worry about the leaked model parameters because they are only useful for their own devices with the decryption keys.

To extend VeriSplit with FHE, we will evaluate the applicability of VeriSplit's selective verification algorithm with the intermediate results of FHE networks. A number of foreseeable challenges include:

- FHE's usage of finite field elements and how it affects the way VeriSplit splits intermediate values with fine granularity.
- The lack of mature FHE-enabled machine learning frameworks and efficient, stable implementations on GPUs.
- Additional optimizations introduced in prior works, such as offline preprocessing, and how to incorporate these results into the online inference process.

Chapter 5

Timeline

Time	Task
Aug. 2022	Thesis proposal
Sept. 2022	Submit VeriSplit to NSDI
Oct. 2022 – Jan. 2023	Extend VeriSplit to additional machine learning inference systems, including support for model privacy and verifying encrypted networks (e.g., Fully Homomorphic Encryption)
Dec. 2022 – April 2023	Job search
Feb. 2023 – April 2023	Thesis writing
May 2022	Thesis defense

Table 5.1: Proposed timeline for completing the thesis.

My goal is to complete the dissertation by May 2023. Table 5.1 summarizes the timeline to completing the remainder of the thesis.

Bibliography

[1] Alasdair Allan. The problem with throwing away a smart device. https://www.hackster.io/news/the-problem-with-throwing-away-a-smart-device-75c8b35ee3c7, 2020. 2.1.1

[2] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. SoK: Security evaluation of home-based IoT deployments. In *2019 IEEE Symposium on Security and Privacy*, 2019. 1, 2.2

[3] Amazon Web Services. AWS IoT Greengrass. https://aws.amazon.com/greengrass/, 2020. 2.1.2, 4.1

[4] Michael P Andersen, Sam Kumar, Moustafa AbdelBaky, Gabe Fierro, John Kolb, Hyung-Sin Kim, David E Culler, and Raluca Ada Popa. WAVE: A decentralized authorization framework with transitive delegation. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1375–1392, 2019. 3.1

[5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium*, 2017. 1, 2.1.3

[6] Andrew W Appel and Edward W Felten. Proof-carrying authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 52–62, 1999. 3.2.1

[7] Gbadebo Ayoade, Vishal Karande, Latifur Khan, and Kevin Hamlen. Decentralized IoT data management using blockchain and trusted execution environment. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 15–22. IEEE, 2018. 3, 3.1

[8] Lujo Bauer, Scott Garriss, Jonathan M McCune, Michael K Reiter, Jason Rouse, and Peter Rutenbar. Device-enabled authorization in the grey system. In *International Conference on Information Security*, pages 431–445. Springer, 2005. 3, 3.1, 3.2.1

[9] Julia Bernd, Ruba Abu-Salma, and Alisa Frik. Bystanders' privacy: The perspectives of nannies on smart home surveillance. In *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*, 2020. 3, 3.1

[10] Arnar Birgisson, Joe Gibbs Politz, Úlfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. Macaroons: Cookies with contextual caveats for decentralized authorization in

the cloud. In *Network and Distributed System Security Symposium*, 2014. 3.2.1

[11] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Found. Trends Priv. Secur.*, 2016. 3.3.2

[12] Daming D. Chen, Maverick Woo, David Brumley, and Manuel Egele. Towards automated dynamic analysis for Linux-based embedded firmware. In *NDSS*, 2016. 2.2

[13] CNET. Search engine shodan knows where your toaster lives. `https://www.cnet.com/how-to/shodan-dyn-botnet-searches-all-your-iot-devices/`, 2015. 2.1.3

[14] Camille Cobb, Sruti Bhagavatula, Kalil Anderson Garrett, Alison Hoffman, Varun Rao, and Lujo Bauer. "i would have to evaluate their objections": Privacy tensions between smart home device owners and incidental users. *Proceedings on Privacy Enhancing Technologies*, 4:54–75, 2021. 3.1

[15] Jessica Colnago, Yuanyuan Feng, Tharangini Palanivel, Sarah Pearman, Megan Ung, Alessandro Acquisti, Lorrie Faith Cranor, and Norman Sadeh. Informing the design of a personalized privacy assistant for the internet of things. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020. 3.1

[16] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62, 2010. 1

[17] Soteris Demetriou, Nan Zhang, Yeonjoon Lee, XiaoFeng Wang, Carl A. Gunter, Xiaoyong Zhou, and Michael Grace. HanGuard: SDN-driven protection of smart home WiFi devices from malicious mobile apps. In *10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec, 2017. 2.1.3

[18] Brian Dipert. Teardown: WeMo switch is highly integrated. `https://www.edn.com/teardown-wemo-switch-is-highly-integrated/`, 2020. 2.1.1

[19] Colin Dixon, Ratul Mahajan, Sharad Agarwal, A.J. Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. An operating system for the home. In *9th USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2012. 2.1.2, 4.1

[20] Ruian Duan, Ashish Bijlani, Meng Xu, Taesoo Kim, and Wenke Lee. Identifying open-source license violation and 1-day security risk at large scale. In *2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS, 2017. 2.2.1

[21] Chethana Dukkipati, Yunpeng Zhang, and Liang Chieh Cheng. Decentralized, blockchain based access control framework for the heterogeneous internet of things. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*, pages 61–69, 2018. 3, 3.1

[22] Pardis Emami-Naeini, Janarth Dheenadhayalan, Yuvraj Agarwal, and Lorrie Faith Cranor. Which privacy and security attributes most impact consumers' risk perception and willingness to purchase IoT devices? In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1937–1954, 2021. 3

[23] Jeremy Erickson, Qi Alfred Chen, Xiaochen Yu, Erinjen Lin, Robert Levy, and Z Morley

Mao. No one in the middle: Enabling network access control via transparent attribution. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 651–658, 2018. 2.1.3

[24] Espressif. ESP32 series datasheet. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf, 2020. 2.1.1

[25] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. Flowfence: Practical data protection for emerging IoT application frameworks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 531–548, Austin, TX, August 2016. USENIX Association. ISBN 978-1-931971-32-4. URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/fernandes. 3, 3.1

[26] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, and Rajesh K Gupta. ACES: Automatic configuration of energy harvesting sensors with reinforcement learning. *ACM Transactions on Sensor Networks (TOSN)*, 16(4):1–31, 2020. 1

[27] Diana Freed, Jackeline Palmer, Diana Elizabeth Minchala, Karen Levy, Thomas Ristenpart, and Nicola Dell. Digital technologies and intimate partner violence: A qualitative analysis with multiple stakeholders. *Proceedings of the ACM on Human-Computer Interaction*, 1 (CSCW):1–22, 2017. 3.1

[28] Diana Freed, Sam Havron, Emily Tseng, Andrea Gallardo, Rahul Chatterjee, Thomas Ristenpart, and Nicola Dell. " is my phone hacked?" analyzing clinical computer security interventions with survivors of intimate partner violence. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–24, 2019. 3.1

[29] Jun Han, Albert Jin Chung, Manal Kumar Sinha, Madhumitha Harishankar, Shijia Pan, Hae Young Noh, Pei Zhang, and Patrick Tague. Do you feel what I hear? enabling autonomous IoT device pairing using different sensor types. In *2018 IEEE Symposium on Security and Privacy*, 2018. 2.3.3

[30] Sam Havron, Diana Freed, Rahul Chatterjee, Damon McCoy, Nicola Dell, and Thomas Ristenpart. Clinical computer security for victims of intimate partner violence. In *28th USENIX Security Symposium (USENIX Security 19)*, 2019. 3.1

[31] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. Rethinking access control and authentication for the home internet of things (iot). In *27th USENIX Security Symposium (USENIX Security 18)*, 2018. 3, 3.1

[32] Weijia He, Valerie Zhao, Olivia Morkved, Sabeeka Siddiqui, Earlence Fernandes, Josiah D. Hester, and Blase Ur. SoK: Context sensing for access control in the adversarial home iot. In *Proceedings of the 6th IEEE European Symposium on Security and Privacy*, 2021. 3.1

[33] Armijn Hemel, Karl Trygve Kalleberg, Rob Vermaas, and Eelco Dolstra. Finding software license violations through binary code clone detection. In *8th Working Conference on Mining Software Repositories*, MSR, 2011. 2.2.1

[34] Tejun Heo. Control group v2. https://www.kernel.org/doc/Documentation/cgroup-v2.txt, 2015. 2.3.2

[35] James Hong, Amit Levy, Laurynas Riliskis, and Philip Levis. Don't talk unless i say so! securing the Internet of things with default-off networking. In *3rd ACM/IEEE International Conference on Internet-of-Things Design and Implementation*, IoTDI, 2018. 2.1.3, 2.3.1

[36] Weizhe Hua, Muhammad Umar, Zhiru Zhang, and G Edward Suh. Guardnn: Secure dnn accelerator for privacy-preserving deep learning. *arXiv preprint arXiv:2008.11632*, 2020. 4.1

[37] Qinlong Huang, Yixian Yang, and Licheng Wang. Secure data access control with ciphertext update and computation outsourcing in fog computing for internet of things. *IEEE Access*, 5:12941–12950, 2017. doi: 10.1109/ACCESS.2017.2727054. 3, 3.1

[38] ifttt. IFTTT. `https://www.ifttt.com`, 2020. 2.5

[39] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Z. Morley Mao, and Atul Prakash. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms. In *21st Network and Distributed Security Symposium*, Feb 2017. 3, 3.1

[40] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 114–148, 2021. 4.5

[41] Dohyun Kim, Prasoon Patidar, Han Zhang, Abhijith Anilkumar, and Yuvraj Agarwal. Self-serviced iot: Practical and private iot computation offloading with full user control. *arXiv preprint arXiv:2205.04405*, 2022. 4.1

[42] Ronny Ko and James Mickens. DeadBolt: Securing IoT deployments. In *Applied Networking Research Workshop*, ANRW, 2018. 2.1.3, 2.3.1

[43] Deepak Kumar, Kelly Shen, Benton Case, Deepali Garg, Galina Alperovich, Dmitry Kuznetsov, Rajarshi Gupta, and Zakir Durumeric. All things considered: An analysis of IoT devices on home networks. In *28th USENIX Security Symposium*, 2019. 2.1.3

[44] Sam Kumar, Yuncong Hu, Michael P Andersen, Raluca Ada Popa, and David E. Culler. JEDI: Many-to-many end-to-end encryption and key delegation for iot. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1519–1536, Santa Clara, CA, August 2019. USENIX Association. ISBN 978-1-939133-06-9. URL `https://www.usenix.org/conference/usenixsecurity19/presentation/kumar-sam`. 3, 3.1

[45] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems (TOCS)*, 10(4):265–310, 1992. 3.2.1

[46] Gierad Laput, Yang Zhang, and Chris Harrison. Synthetic sensors: Towards general-purpose sensing. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 3986–3999, 2017. 1

[47] Tam Le and Matt W Mutka. Access control with delegation for smart home applications. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, pages 142–147, 2019. 3, 3.1

[48] The Security Ledger. Devices' UPnP service emerges as key threat to home IoT networks.

https://securityledger.com/2019/03/devices-upnp-service-emerges-as-key-threat-to-home-iot-networks/, 2020. 2.1.3

[49] Xinyu Lei, Guan-Hua Tu, Chi-Yu Li, Tian Xie, and Mi Zhang. SecWIR: Securing smart home IoT communications via wi-fi routers with embedded intelligence. In *18th International Conference on Mobile Systems, Applications, and Services*, MobiSys, 2020. 2.1.3, 2.3.3

[50] Chieh-Jan Mike Liang, Börje F. Karlsson, Nicholas D. Lane, Feng Zhao, Junbei Zhang, Zheyi Pan, Zhao Li, and Yong Yu. SIFT: Building an internet of safe things. In *14th International Conference on Information Processing in Sensor Networks*, IPSN, 2015. 2.1.1

[51] Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. An empirical characterization of IFTTT: ecosystem, usage, and performance. In *2017 Internet Measurement Conference*, IMC, 2017. 2.5

[52] Microsoft Azure. Azure Sphere. https://azure.microsoft.com/en-us/services/azure-sphere/, 2020. 2.1.2, 2.4

[53] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2505–2522, 2020. 4.1

[54] Soo-Jin Moon, Vyas Sekar, and Michael K Reiter. Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. In *Proceedings of the 22nd acm sigsac conference on computer and communications security*, pages 1595–1606, 2015. 2.3.1

[55] Murat Moran and Dan S Wallach. Verification of star-vote and evaluation of fdr and proverif. In *International Conference on Integrated Formal Methods*, pages 422–436. Springer, 2017. 3.3.2

[56] Asuka Nakajima, Takuya Watanabe, Eitaro Shioji, Mitsuaki Akiyama, and Maverick Woo. A pilot study on consumer IoT device vulnerability disclosure and patch release in japan and the united states. In *2019 ACM Asia Conference on Computer and Communications Security*, AsiaCCS, 2019. 2.2

[57] Hung Nguyen, Radoslav Ivanov, Linh T.X. Phan, Oleg Sokolsky, James Weimer, and Insup Lee. LogSafe: Secure and scalable data logger for IoT devices. In *3rd ACM/IEEE International Conference on Internet-of-Things Design and Implementation*, IoTDI, 2018. 2.1.1

[58] openssl-changelog. OpenSSL changelog. https://www.openssl.org/news/changelog.html, 2020. 2.2.2

[59] Nouha Oualha and Kim Thuat Nguyen. Lightweight attribute-based encryption for the internet of things. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6, 2016. doi: 10.1109/ICCCN.2016.7568538. 3, 3.1

[60] Simon Parkin, Trupti Patel, Isabel Lopez-Neira, and Leonie Tanczer. Usability analysis of shared device ecosystem security: Informing support for survivors of iot-facilitated tech-abuse. In *Proceedings of the New Security Paradigms Workshop*, 2019. 3.1

[61] particlel-device-os. Particle device OS. https://www.particle.io/device-o

s/, 2020. 2.1.2, 2.4

[62] Otto Julio Ahlert Pinno, Andre Ricardo Abed Gregio, and Luis C. E. De Bona. Controlchain: Blockchain as a central enabler for access control authorizations in the iot. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–6, 2017. doi: 10.1109/GLOCOM.2017.8254521. 3, 3.1

[63] Roomba. Roomba® s9+ self-emptying robot vacuum. https://www.irobot.com/en_US/roomba-vacuuming/robot-vacuum-irobot-roomba-s9-plus/S955020.html, 2022. 4.1

[64] RTInsights. IoT devices still exposed, vast majority of traffic unencrypted. https://www.rtinsights.com/iot-security-remains-lacklustre/, 2020. 1

[65] Samsung SmartThings. Direct-connected device SDK. https://smartthings.developer.samsung.com/docs/devices/direct-connected-devices/overview.html, 2020. 2.1.2, 2.4

[66] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Situational access control in the internet of things. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1056–1073, 2018. 3, 3.1

[67] Senrio. 400,000 publicly available IoT devices vulnerable to single flaw. https://blog.senr.io/blog/400000-publicly-available-iot-devices-vulnerable-to-single-flaw, 2016. 1

[68] Mohit Sethi, Elena Oat, Mario Di Francesco, and Tuomas Aura. Secure bootstrapping of cloud-managed ubiquitous displays. In *2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp, 2014. 2.3.3

[69] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. Towards blockchain-based auditable storage and sharing of IoT data. In *Proceedings of the 2017 on Cloud Computing Security Workshop*, pages 45–50, 2017. 3, 3.1

[70] Hossein Shafagh, Anwar Hithnawi, Lukas Burkhalter, Pascal Fischli, and Simon Duquennoy. Secure sharing of partially homomorphic encrypted IoT data. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, pages 1–14, 2017. 3, 3.1

[71] Hossein Shafagh, Lukas Burkhalter, Sylvia Ratnasamy, and Anwar Hithnawi. Droplet: Decentralized authorization and access control for encrypted data streams. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 2469–2486, 2020. 3, 3.1

[72] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979. 3.3.1

[73] Amit Kumar Sikder, Leonardo Babun, Z Berkay Celik, Abbas Acar, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A Selcuk Uluagac. Kratos: multi-user multi-device-aware access control system for the smart home. In *Prelavantroceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020. 3.1

[74] Anna Kornfeld Simpson, Franziska Roesner, and Tadayoshi Kohno. Securing vulnerable home IoT devices with an in-hub security manager. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops*, PerCom, 2017. 2.1.3

[75] tomoyo. TOMOYO linux. https://tomoyo.osdn.jp/index.html.en, 2020. 2.3.2

[76] Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*, 2018. 4.1

[77] Rahmadi Trimananda, Ali Younis, Bojun Wang, Bin Xu, Brian Demsky, and Guoqing Xu. Vigilia: Securing smart home edge computing. In *2018 IEEE/ACM Symposium on Edge Computing*, SEC, 2018. 2.1.1

[78] Emily Tseng, Rosanna Bellini, Nora McDonald, Matan Danos, Rachel Greenstadt, Damon McCoy, Nicola Dell, and Thomas Ristenpart. The tools and tactics used in intimate partner surveillance: An analysis of online infidelity forums. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1893–1909. USENIX Association, August 2020. ISBN 978-1-939133-17-5. 3.1

[79] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. Graviton: Trusted execution environments on {GPUs}. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 681–696, 2018. 4.1

[80] Frank Wang, James Mickens, Nickolai Zeldovich, and Vinod Vaikuntanathan. Sieve: Cryptographically enforced access control for user data in untrusted clouds. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 611–626, 2016. 3, 3.3.1

[81] Wikipedia. IEEE 802.11i-2004. https://en.wikipedia.org/wiki/IEEE_802.11i-2004, 2020. 2.3.3

[82] Rich Wolski, Chandra Krintz, Fatih Bakir, Gareth George, and Wei-Tsung Lin. CSPOT: Portable, multi-scale functions-as-a-service for IoT. In *4th ACM/IEEE Symposium on Edge Computing*, SEC, 2019. 2.1.1

[83] Yaxing Yao, Justin Reed Basdeo, Oriana Rosata Mcdonough, and Yang Wang. Privacy perceptions and designs of bystanders in smart homes. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–24, 2019. 3.1

[84] Tianlong Yu, Vyas Sekar, Srinivasan Seshan, Yuvraj Agarwal, and Chenren Xu. Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things. In *14th ACM Workshop on Hot Topics in Networks*, HotNets, 2015. 1, 2.1.3

[85] Bin Yuan, Yan Jia, Luyi Xing, Dongfang Zhao, XiaoFeng Wang, and Yuqing Zhang. Shattered chain of trust: Understanding security risks in cross-cloud IoT access delegation. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1183–1200, 2020. 3.1

[86] Gina Yuan, David Mazières, and Matei Zaharia. Extricating iot devices from vendor infrastructure with karl. *arXiv preprint arXiv:2204.13737*, 2022. 4.1

[87] ZDNet. Shodan: The IoT search engine for watching sleeping kids and bedroom antics. https://www.zdnet.com/article/shodan-the-iot-search-engine-which-shows-us-sleeping-kids-and-how-we-throw-away-our-privacy/, 2016. 2.1.3

[88] ZDNet. CallStranger vulnerability lets attacks bypass security systems and scan LANs.

https://www.zdnet.com/article/callstranger-vulnerability-lets-attacks-bypass-security-systems-and-scan-lans/, 2020. 1.1, 2.1.3

[89] ZDNet. Hacker leaks passwords for more than 500,000 servers, routers, and IoT devices. https://www.zdnet.com/article/hacker-leaks-passwords-for-more-than-500000-servers-routers-and-iot-devices/, 2020. 1

[90] ZDNet. Ripple20 vulnerabilities will haunt the IoT landscape for years to come. https://www.zdnet.com/article/ripple20-vulnerabilities-will-haunt-the-iot-landscape-for-years-to-come/, 2020. 1.1

[91] Eric Zeng and Franziska Roesner. Understanding and improving security and privacy in multi-user smart homes: a design exploration and in-home user study. In *28th USENIX Security Symposium (USENIX Security 19)*, 2019. 3.1

[92] Han Zhang, Abhijith Anilkumar, Matt Fredrikson, and Yuvraj Agarwal. Capture: Centralized library management for heterogeneous IoT devices. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 4187–4204, 2021. 2, 2.5

[93] Han Zhang, Yuvraj Agarwal, and Matt Fredrikson. TEO: ephemeral ownership for iot devices to provide granular data control. In Nirupama Bulusu, Ehsan Aryafar, Aruna Balasubramanian, and Junehwa Song, editors, *MobiSys '22: The 20th Annual International Conference on Mobile Systems, Applications and Services, Portland, Oregon, 27 June 2022 - 1 July 2022*, pages 302–315. ACM, 2022. 3, 3.3

[94] Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. HoMonit: Monitoring smart home apps from encrypted traffic. In *2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS, 2018. 2.1.3

[95] Yaowen Zheng, Ali Davanian, Heng Yin, Chengyu Song, Hongsong Zhu, and Limin Sun. FIRM-AFL: high-throughput greybox fuzzing of IoT firmware via augmented process emulation. In *28th USENIX Security Symposium*, 2019. 2.2