Title
Secure and Safe Edge Computing for the Internet-of-Things

Author
Liu, Renju

Publication Date
2020

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Secure and Safe Edge Computing for the Internet-of-Things

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Renju Liu

2020

ABSTRACT OF THE DISSERTATION

Secure and Safe Edge Computing for the Internet-of-Things

by

Renju Liu
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2020
Professor Mani B. Srivastava, Chair

Internet-of-Things (IoTs) have developed for more than one decade since their first appearance. At their early stage, IoT devices are generally developed as sensor extensions, which are not capable of performing complicated computing tasks. Hence, offloading the computational tasks to the cloud servers is the only option to learn the information from the sensory data. However, with the rapid evolution of the technologies in the field of System-on-Chip (SoC), more complex computational tasks are now able to be performed on IoT devices, which is inevitably facing multi-tenancy requirements. For example, a UAV-as-a-Service can be used by different users to perform object detection and traffic monitoring tasks. Unfortunately, current multi-tenancy edge systems face several issues from the security of the sensors and actuators and the safety of their cyber-physical environment.

In this dissertation, we build an edge computing framework to address the security and safety problems of current multi-tenancy edge systems. This framework focuses on a broad category of IoT devices with sensors and actuators, including both rich computing resource devices with powerful edge processors such as ARM Cortex-A series processors and bare-metal IoT devices with only microcontrollers such as ARM Cortex-M series processors. The

first part of this work focuses on securing the sensors and the actuators at the edge. More specifically, we propose PROTC and VirtSense that utilize the hardware-assisted trusted execution environment (TEE) such as ARM TrustZone to isolated the actuation and sensing service from the untrusted execution environment. Furthermore, for those bare-metal IoT devices that the hardware-assisted TEE is not available, we design Aerogel that utilizes WebAssembly (Wasm) initially designed for secure JavaScript execution environment in web browsers, to protect the sensors and actuators. Our results show that the protection of Aerogel introduces 0.14% to 1.04% overhead to our benchmark tasks.

After the sensors have been adequately secured, the second part of this dissertation focuses on the data usage from the sensors by preventing the applications from extracting unexpected information from the sensory data, which could ultimately lead to the leakage of users' privacy. To achieve such a goal, we design a performant and secure runtime named SecDeep that protects the inference libraries' integrity and sensor data's confidentiality by leveraging the TEE to securely host the deep learning code and the drivers of sensors, actuators, and the edge accelerator. Our results indicate that with the assistance of the edge accelerator, SecDeep achieves 16 to 172 times faster performance than without using it while still providing data protection.

Although ensuring the security of the IoT devices can mitigate the safety issues, not all safety issues are caused by security problems such as the user's misconfiguration. We propose RemedIoT in the third part of this dissertation, which tackles the IoT device actuation conflicts by introducing actuation programming abstractions to provide remedial actions of those conflicts. Our evaluations on RemedIoT shows that around 80% of device conflicts can be provided remedial actions.

The dissertation of Renju Liu is approved.

Harry Guoqing Xu

Jens Palsberg

Anthony (Tony) John Nowatzki

Mani B. Srivastava, Committee Chair

University of California, Los Angeles

2020

To dedicate my parents who always love and support me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First of all, I would like to give my sincerest thanks to my Ph.D. advisor, Prof. Mani Srivastava, who gave me priceless supports both financially and mentally towards my Ph.D. study. I appreciate the guidance of Mani's thoughts about research ideas and insights based on his excellent research senses and knowledge. I also thank his encouragement when I got lost exploring the research problems. Without Mani's support, I would never achieve the point of getting conquering the research problems I have done so far.

I want to give special thanks to my previous advisor, Prof. Felix Xiaozhu Lin, who taught me the basics of doing research. Without him, I would have to struggle longer getting adapt to the mindset of being a Ph.D. student from an undergraduate student. I would also like to thank my committee members, Prof. Harry Xu, Prof. Jens Palsberg, and Prof. Tony Nowatzki, for their help and suggestions.

I want to thank all of my collaborators across different universities. Their supports tremendously help me finish the projects during my Ph.D. study. I am very luckily working at the NESL lab at UCLA with my excellent labmates. I want to say thanks to all of my current and previous labmates — Dr. Salma Elmalaki, Prof. Fatima Anwar, Dr. Bo-Jhang Ho, Dr. Moustafa Alzantot, Dr. Chenguang Shen, Dr. Luis Garcia, Dr. Bharathan Balaji, Debbie Tsai, Yue Xin, Zhengxu Xia, Akash Singh, Botong Ou, Brian Wang, Vikranth Jeyakumar, Joseph Noor, Pengrui Quan, Sandeep Sandha, Siyou Pei, Swapnil Saha, Tianwei Xing, and Ziqi Wang. I also want to shout out special thanks to Dr. Luis Garcia for his invaluable and excellent ideas, suggestions, and feedback for my Ph.D. projects.

Without any hesitation, I would like to thank my families, especially my parents, who always give me support, love, and encouragement. As the only child in my family, my parents put all resources they have on me. Without their supports, I can never be who I am today.

Lastly, I wish to thank the funding agencies who provide me financial supports during my Ph.D. study. I want to thank the National Science Foundation (NSF), the CONIX

Research Center, and Alliance for Internet of Battlefield Things (IoBT) Research. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies or the U.S. Government.

VITA

2010-2014	B.S. in Computer Science and B.S. in Electrical Engineering with distinction, Purdue University, West Lafayette, Indiana.
2014-2016	Graduate Research Assistant, Electrical and Computer Engineering Department, Purdue University, West Lafayette, Indiana.
2017	Software Engineer Intern, Facebook Inc., Menlo park, California.
2018	Software Engineer Intern, Pinterest Inc., Sanfrancisco, California.
2018	M.S. in Computer Science, UCLA, Los Angeles, California.
2019	Pinterest Lab Research Intern, Pinterest Inc., Sanfrancisco, California.
2016-2020	Graduate Student Researcher, Electrical and Computer Engineering Department, UCLA, Los Angeles, California.

PUBLICATIONS

Renju Liu, Luis Garcia, Mani Srivastava, "Aerogel: Lightweight Access Control Framework for WebAssembly-Based Bare-Metal IoT Devices". (Under Submission)

Renju Liu, Luis Garcia, Zaoxing Liu, Botong Ou, Mani Srivastava, "SecDeep: Secure and Performant Deep Learning Inference Framework for Edge Devices". (Under Submission)

Renju Liu, Ziqi Wang, Luis Garcia, Mani Srivastava, "RemedIoT: Remedial Actions for

Internet-of-Things Conflicts". In Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '19). Association for Computing Machinery, New York, NY, USA, 101–110.

Renju Liu, Mani Srivastava, "VirtSense: Virtualize Sensing through ARM TrustZone on Internet-of-Things". In Proceedings of the 3rd Workshop on System Software for Trusted Execution (SysTex'18). Association for Computing Machinery, New York, NY, USA, 2–7.

Bo-Jhang Ho*, **Renju Liu***, Hsiao-Yun Tseng, and Mani Srivastava, "MyoBuddy: Detecting Barbell Weight Using Electromyogram Sensors". In Proceedings of the 1st Workshop on Digital Biomarkers (DigitalBiomarkers '17). Association for Computing Machinery, New York, NY, USA, 27–32.

Renju Liu, Mani Srivastava, "PROTC: Protecting Drone's Peripherals through ARM TrustZone". In Proceedings of the 3rd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications (DroNet '17). Association for Computing Machinery, New York, NY, USA, 1–6.

*Both authors are equally contributed.

CHAPTER 1

Introduction

Smartphones became a platform for sensory applications that enable situational understanding of user environments since their appearances as a simple communication tool. Similarly, this evolution trend has now impacted the development of Internet-of-Things (IoTs) devices.

Back to a decade ago, people used IoT devices only as of the extension of sensors, and those early-stage IoT devices are only able to collect sensory data of their environment with no onboard data processing capability. Hence, if users want to learn useful information from the data, such tasks have to be done in the cloud environment. With the technology revolution of both Internet such as 5G and the System-on-Chips (SoCs), IoT devices are equipped with the computational abilities for data processing such as machine learning inference on devices. Moreover, under the considerations of network latency, network availability, security, and data privacy requirements, computing pushing to the edge (i.e., edge computing) has now attracted researchers' attention than ever [SCZ16, Sat17, SD16b, LOD18a, CDO17].

While some computing tasks are shifted to the edge, a single-tenant edge system can never satisfy the emerging workloads when multiple users or applications need to share the edge resources. Hence, a multi-tenant edge system is inevitably needed [SD16a, YSB16, BZP10, Mic19, CGK16, CMG15, TSG18]. However, the current multi-tenant edge system faces both security and safety issues due to the heterogeneity of the IoT environment, such as isolating the execution environment and resolving control conflicts among different tenants. Next, we explain the currently existing challenges of multi-tenant edge systems.

1.1 Challenges

Since IoT devices are mostly equipped with sensors and actuators, enabling multi-tenancy at the edge requires a secure and robust system to protect the usage of sensors and actuators, which could ultimately lead to the concerns of the leakage of user privacy or the safety of the environment. Next, we highlight three significant challenges from the perspectives of the sensors and actuators usage.

1.1.1 Runtime Securities of Sensors and Actuators

The first challenge we are facing is how we can secure the safety and privacy-related sensing and actuation services on IoT devices. For example, suppose a UAV has a flight-control application and an image capturing application on board, where the image capturing application is malicious and can compromise the operating system. In that case, the image capturing application will gain full control of the UAV, which can fly the UAV to anywhere the malicious application wants.

Merely relying on the memory address separation of the Operating System (OS) to isolate potential malicious applications is not sufficient because the design vulnerabilities can compromise traditional OS through rootkits such as buffer overflow attacks (e.g., CVE-2019-17666). Therefore, the whole OS environment should be treated as an untrusted environment, and securing sensing and actuation services under such an untrusted execution environment creates the challenges to be addressed by this dissertation.

Moreover, the memory address separation techniques require hardware assistance such as Memory Management Unit (MMU). However, not all devices such as bare-metal IoT devices equipped with ARM Cortex M series microcontrollers have MMU support. Hence, providing sensors and actuators protection and allowing the users to have fine-grained access control for those bare-metal IoT devices is another open question to be answered.

1.1.2 Sensor Data Privacy

When access to the sensors has been secured, we then encounter the challenges of how the applications can use those sensor data. Recently, deep learning inference at the edge is one of the most ubiquitous functionalities of using the sensor data from the IoT devices. For example, the facial recognition application on a smart camera is programmed using deep learning inference libraries. Although a prior work [LPZ20] shows that current commodity cameras cannot perform complex tasks on-device, several recent works [HPC18, HZ19, OSD19, LSJ19, VFC18] have shown that such inference tasks could be accomplished with the help of edge-available neural network accelerators such as Intel Neural Sticks [Intb].

However, directly feeding the raw data from the sensor will lead to the concerns of user privacy because the application can infer more information than allowed. In the previous example, if the facial recognition application is malicious and raw images are sent to it, this application can also steal the user's other private information, such as what the user is doing every day.

Moreover, if the OS is not trusted, any raw sensor data sent to the applications is risked to be eavesdropped by the OS. Meanwhile, the edge devices are now enabled with edge accelerators, such as ARM Mali GPU and Intel neural sticks, to accelerate edge inference. Hence the question of how the edge systems can securely and performantly make deep learning inference remains open.

Although some prior works have tried to protect the inference data confidentiality through Intel SGX [LLP19, HSS18, TB19, GHZ18, TGS18] on the cloud environment, they failed to provide performant protections on edge devices when OS is trusted.

1.1.3 Actuation Conflicts

Conflicts among actuators can lead to severe consequences in terms of safety and energy consumption. For example, in a smart home, if one smart home application tries to turn on

the air conditioner and the other application tries to turn off the air conditioner, then they will form a racing condition such that the air conditioner jumps between on and off states, which could ultimately cause the damage of the device.

Although some actuator conflicts are due to the system's security issues, such as malicious applications, more cases of these conflicts are caused by user misconfiguration. Prior works [MAH16, MPS17, CTMb, LKL15a] have focused on detecting such conflicts, and none of them are trying to provide runtime resolutions for these conflicts. Therefore, the question of resolving runtime actuation conflicts still needs to be answered.

1.2 Contributions

This thesis contributes to the solutions to address the challenges mentioned earlier from three aspects. Explicitly speaking, we focus on how to ensure the security of sensors and actuators on an untrusted execution environment, how to ensure the sensor data is not maliciously used at the edge, and how to resolve the device actuation conflicts because of user's misconfiguration.

In the beginning, we propose a system utilizing both hardware and software isolation mechanisms to ensure the security of the sensors and actuators under the untrusted execution environment. We first introduce **PROTC** that protects essential actuation services from being maliciously accessed by either untrusted OS or applications. The protection is abstracted through the feature of ARM TrustZone. **PROTC** implements a trusted computing base within ARM TrustZone that enforces a secure access control policy for the UAV's essential protected actuation services. Such an access control policy integrated with cryptographic techniques ensures that the control commands are genuinely from the user but not from untrusted entities. The hardware protection from ARM TrustZone ensures that the trusted computing base of **PROTC** is isolated from the OS. Next, we build a similar system named **VirtSense** to protect the essential sensing services. **VirtSense** is an ARM

TrustZone based virtual sensing system that provides each sensing application a virtual sensor instance, which further enables a safe, flexible, and isolated sensing environment on the IoT devices. The crucial part of **VirtSense** is a virtual sensor management unit that delivers sensor samples to sensing services through different sensing signal reconstruction techniques such that the sensors can be securely hidden from the OS while still serving for the applications.

Although the above two systems can appropriately protect the sensors and actuators through hardware-assisted mechanisms, the bare-metal IoT devices such as ARM Cortex M series lack such hardware mechanisms. To address the protection problems on these devices, we design **Aerogel** that uses lightweight WebAssembly (Wasm) features for execution environment isolation and further enables the fine-grained access control on bare-metal IoT devices. More specifically speaking, **Aerogel** is an access control framework that interfaces Wasm runtime with access control mechanisms for sensors, actuators, processor energy usage, as well as memory usage. In particular, we treat the runtime as a multi-tenant environment and leverage the inherent sandboxing mechanisms of Wasm to enforce the access control policies to sensors and actuators.

The second part of this dissertation contributes to guarantee the sensory data confidentiality of deep learning inference at the edge. We designed a secure runtime system **SecDeep**, a low-power deep learning inference framework demonstrating that both security and performance of deep learning inference on edge devices are well within our reach. More specifically speaking, leveraging TEEs with limited resources, **SecDeep** guarantees full confidentiality for input and intermediate data, as well as the integrity of the deep learning model and framework. By enabling and securing neural accelerators, **SecDeep** is the first of its kind to provide trusted and performant deep learning model inferencing on edge devices. **SecDeep** partitions the deep learning libraries into two to reduce its core trusted computing base size. The partition inside TEE is named as confidential computing base whose data confidentiality is protected by the hardware. This partition requires the com-

puting code that has to use plaintext tensor data such as the raw data from the sensor to finish the computing. The other partition is outside of TEE named nonconfidential computing base. This partition can only host the code that depends on the property of the tensor data such as the size of the data rather than the raw data. Next, **SecDeep** modifies the OS kernel table as read-only to guarantee the integrity of the deep learning code running in an untrusted execution environment such that each time when the kernel table needs to be changed, **SecDeep** can examine whether such change will affect the results from nonconfidential computing base. Moreover, **SecDeep** provides a data manager that uses a format-preserving encryption method to sanitize the data communication between the two partitioned computing bases.

The last part of this dissertation tackles the challenge of designing a framework to provide remedial actions on IoT actuation conflicts. We propose **RemedIoT** a remedial action [1] framework for resolving Internet-of-Things conflicts. The **RemedIoT** framework uses state of the art techniques to detect if a conflict exists in a given set of distributed IoT applications with respect to a set of *policies*, i.e., rules that define the allowable and restricted state-space transitions of devices. For each identified conflict, **RemedIoT** will suggest a set of remedial actions to the user by leveraging **RemedIoT**'s programming abstractions. These programming abstractions enable different realizations of an IoT module while safely providing the same level of utility, e.g., if an air-conditioner application that is used to implement a *cooling module* conflicts with a CO_2 monitor application that requires ventilation at home, a non-conflicting smart fan application will be suggested to the user. However, not all conflicts can be resolved by remedial actions, if in this situation, **RemedIoT** will just block the conflict events.

[1] We draw on the analogy of remedial action schemes for safety-critical, complex industrial control systems such as the electric power grid.

1.3 Organization

This dissertation addresses the security and safety issues of edge computing environments from three perspectives.

- *Part 1: Securing Sensors and Actuators at Runtime.* This part describes the runtime protection for sensors and actuators on IoT devices by **PROTC** in Chapter 2, **VirtSense** in Chapter 3, and **Aerogel** in Chapter 4.
- *Part 2: Securing Inferencing from Sensor Data at the Edge.* This part presents how deep learning inference at the edge can be secured by **SecDeep** in Chapter 5.
- *Part 3: .* This part presents the resolution framework for actuation conflicts by **RemedIoT** in Chapter 6.

We discuss the future research directions and conclude this dissertation in Chapter 7.

Part I

Securing Sensors and Actuators at Runtime

CHAPTER 2

PROTC: Protecting Unmanned Aerial Vehicles' Peripherals through ARM TrustZone

2.1 Introduction

Unmanned Aerial Vehicles (UAVs) are becoming pervasive and are heavily used in various industries in recent years because of their flexibility and ease of use. For example, people use UAVs for search and rescue, to inspect oil pipelines, or to take photographs. FAA has projected that the sales of commercial Unmanned Aerial Vehicles will be roughly 2.7 million by 2020 [FAA]. With the increased usage of UAVs, the security issues of UAVs are becoming urgent. UAVs can cause millions of dollars of loss if crashing with an airplane[gov, Blo].

The software development for UAV, including UAV apps and UAV operating systems, is still at an early stage. In the current UAV market at the time of writing (Mar 2017), the most popular UAV piloting systems are ArduPilot [Ard] and PX4 [PX4]. ArduPilot and PX4 support either bare-metal real-time OS or real-time Linux based OS. Some UAVs [Sol] choose embedded OS, such as NuttX, as their UAV OS. The biggest drawback of the bare-metal systems is that the control application and other applications share the same address space. Hence, a malicious application can easily take control of the UAV by overwriting the control application code on memory. Another drawback of bare-metal systems is that it is hard for the user to install a third-party application. For example, if the UAV user decides to install a data collector, he or she needs to rebuild the whole system stack. More and more commercial UAV companies [DJI] and UAV SoC vendors [NAV, Beb, Rob] adopt

real-time Linux based OS as the UAV OS. The advantage of RT Linux based OS is that control applications and other applications have their own virtual memory space, such that a malicious application cannot directly overwrite the code of control application on memory. Another advantage of using RT Linux based OS is that the third-party programs such as UAV image processing program [PIX] and data collecting programs [Sen] can be easily installed during runtime without recompiling the whole system stack. However, RT Linux based OS exposes vulnerabilities that could be maliciously used to control the UAV and steal important private data by giving the kernel space privilege to malicious user space program [ret09]. Malicious programs with rootkits, such as motochopper [mot], Vroot [vro], are examples of where a malicious user space program can compromise the kernel and hence control the UAV.

Our proposed mechanism, PROTC, can be used against memory attacks that escalate user space applications' privilege. PROTC has two major goals: 1) it ensures the UAV's safety and important data integrity even when the UAV's OS is compromised; 2) it allows for installing third-party applications easily (i.e., high flexibility). PROTC utilizes ARM TrustZone technology to achieve these two goals. ARM TrustZone divides the instructions on the ARM processor into two privileged blocks. Lower privileged instructions are executed in *normal world*, and higher privileged instructions are executed in *secure world*. The secure world programs can restrict the memory access from the normal world programs. In PROTC, we regard the RT Linux based UAV OS as the untrusted OS that resides in normal world, and we design a trusted computing block resides in a secure world to ensure that the user permits access from an untrusted OS.

PROTC has three important components: *critical applications, normal world applications* and the *trusted computing block*. Critical applications need to access the protected peripherals and are installed inside the UAV's controller denoted as the ground control station in our design. Normal world applications are installed in the untrusted environment and do not have privileges to access any protected peripherals. The trusted computing block

inside the secure world is composed of a decision maker and a command executer. The decision maker enforces the protected peripherals' access control policy so that only authorized applications can access the protected peripherals. The command executor executes the command sent from critical applications to pass the policy checking algorithm inside the decision maker.

We propose four micro-benchmarks to test our PROTC prototype on Raspberry Pi 3 used by the commercial UAV SoC NAVIO2 [NAV] with Linaro RT Linux OS in the normal world and OP-TEE [opt] OS in the secure world. These four micro-benchmarks stand for the authorized and unauthorized access to protected peripherals with a base micro-benchmark that directly accesses peripherals without the protections of PROTC. Our results show that the average overhead introduced by PROTC is 143ms and that PROTC successfully prevents unauthorized access to protected peripherals.

Our contribution is: We design a system-level protection mechanism, PROTC, to ensure that the user grants access to protected peripherals and that all the unauthorized access to protected peripherals is rejected.

The rest of this chapter is organized as follows: we first discuss the background and related work for UAVs and ARM TrustZone in section 2.2 and section 2.3. We elaborate on the existing vulnerability in Linux based OS and our motivations to this work in section 2.4. In section 2.5 we introduce our PORTC model. We present preliminary results in section 2.6. We conclude our work in section 2.7 and discuss the future direction of this work.

2.2 Background

2.2.1 ARM TrustZone

PROTC uses ARM TrustZone [ARMg] protection features to create an extra secure policing zone. ARM TrustZone contains two different privilege blocks, as shown in Figure 2.1.

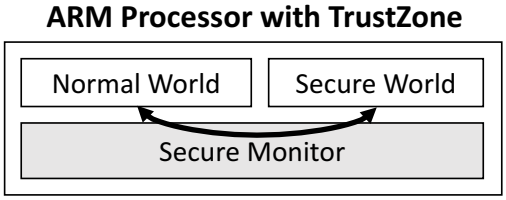


Figure 2.1: ARM TrustZone overview

Secure World is a privileged computing block that only runs privileged instructions. The secure world instruction is triggered when the NS bit in the SCR register is not set. Once a program runs in the secure world, it exposes the full access to the memories, including those memory-mapped peripheral registers. The secure world can define the memory regions that can only be accessed by privileged instructions.

Normal World runs all untrusted components, such as an untrusted operating system and its applications. The instructions inside normal world are unprivileged. Normal world instructions can only access the memory region if it is not marked as the privileged memory region by secure world.

Secure Monitor is used to bridge secure world and normal world. The normal world programs cannot directly access the memory protected by the secure world. However, when a normal world program needs to get access from the memory region protected by the secure world, it will trigger a secure monitor call (SMC) so that the context in normal world will be switched to secure world.

2.2.2 Open Portable Trusted Execution Environment (OP-TEE)

OP-TEE [opt] is an open-source project supported by several independent contributors and the Linaro group. OP-TEE is aimed to provide the system support that runs on ARM TrustZone enabled SoCs.

OP-TEE contains a normal world OS and a secure world OS. Normal world OS is con-

sidered as untrusted OS, while secure world OS is the privileged trusted OS. Normal world OS has a kernel driver that can use SMC to context switch to secure world OS. In PROTC, we use OP-TEE APIs to make the context switch between normal world and secure world.

2.3 Related Work

Arm TrustZone Prior research utilizing ARM TrustZone mainly focuses on the smartphone domain. ARM TrustZone is used to monitor normal world kernel so that malicious code cannot be injected to kernel binaries [ANS14], or to virtualize an OS apart from RTOS [SHT10]. Brasser *et al.* [BKL16a] uses TrustZone to enforce smart devices to comply with privacy regulations. ARM TrustZone is also used to build trusted runtime on mobile devices [SRS14, SSW15b].

ARM TrustZone is also used to ensure data integrity from smartphone sensors [YH, CZG15, LSW12a]. In our work, we are not only targeting at protecting the integrity of data, but also providing a mechanism to prevent malicious access to UAV's essential peripherals.

UAV Security Most previous UAV research focuses on deploying or localization aspects, such as reactive control for the UAV pilot system [BCC16], task cooperations [DKW11], and capturing cinema scenes [FGT16]. Few prior work in UAV main focuses on the security issue. ARDrones described the vulnerability over the network communication because of the authentications on Telnet and FTP [SFH12, PBC14]. Son *et al.* compromised UAV sensors by generating resonance signals [SSK15]. Sel *et al.* proposes a new framework to protect important data on the UAV delivery service path by enhancing the white-box cryptography [SWB16]. However, none of the above work considers the vulnerabilities existing on UAV's operating system. PROTC proposes thorough protection for UAV peripherals in the system level.

2.4 Motivations

UAV OS needs to ensure UAV's control integrity against the memory attacks of malicious applications installed on UAV and allows the user to install third-party applications easily. When considering both system security and easiness for users to install third-party applications, embedded UAV OS is not a right choice because the OS and the applications share the same address space, and a malicious application can easily control the UAV by directly writing to the memory-mapped registers of protected peripherals such as actuators. Embedded OS is also hard to install third-party applications because it requires the whole system stack to be recompiled each time.

The optimal choice for UAV OS is to use real-time Linux based OS. Linux based OS is more secure because of the separation of address space, and it is also easy to install any third-party applications. UAV OS needs to carry both UAV's pilot program [Ard, PX4], and other programs such as image processing programs [PIX, eve], and data collection programs [Sen]. However, to use such an OS architecture, the security of UAV OS might be reduced because the piloting program and other programs share the same OS.

Attacks: A malicious application can utilize the rootkit [vro, tow, mot] to compromise the kernel even when the OS is not rooted. Vroot [vro] can change the kernel function pointer's destination to malicious user space code while maintaining its kernel privilege. Towelroot [tow] and motochopper [mot] can trick the kernel to change the kernel data in memory to escalate a user level process. Long *et al.* [II16] shows that malicious programs can efficiently escalate the user space privilege by wildcard injection, physical address attacks, etc in Linux. Moreover, previous research demonstrates that the user program can compromise the kernel through the return-to-libc attack by overwriting the return address of a function stack [ret09]. These attacks allow user processes to obtain kernel privilege so that they can have unrestricted access to all the peripherals.

All the attack methods and tools above demonstrate the vulnerabilities in current com-

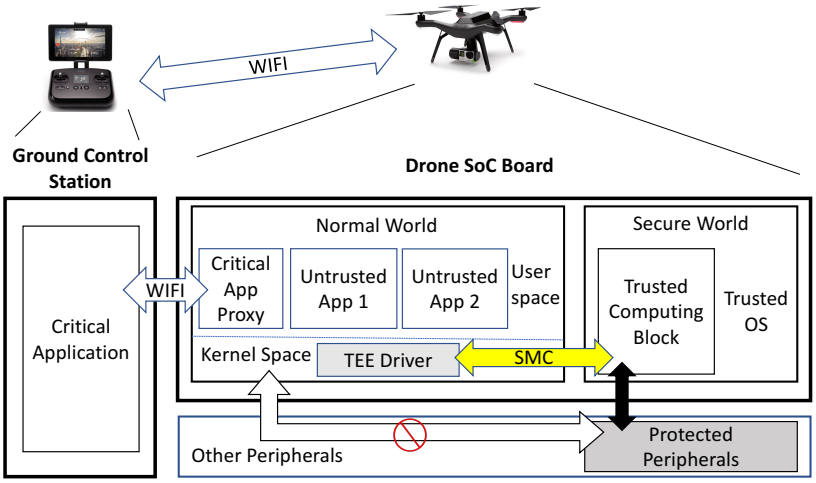


Figure 2.2: Overview of PROTC system design

modity general-purpose OS such as Linux based OS. After the malicious applications have compromised the kernel, they can quickly take over full control of the UAV. Our PROTC provides a protection mechanism to prevent the UAV from being hijacked even when UAV OS is compromised.

2.5 PROTC Design

PROTC mechanism aims to provide a secure UAV OS environment that prevents the UAV from being maliciously controlled. The overview of PROTC design is shown in Figure 2.2. PROTC contains three essential components: *Critical Applications*; *Normal World Applications*; *Trusted Computing Block*. The critical applications are the applications that need to get certain access to protected peripherals, for example, a navigation system. Normal world applications, also named untrusted applications, might be proxy of critical application or independent application from third-party installed in normal world. The trusted computing block is the fundamental protecting block that enforces the security policy for access to protected peripherals. The trusted computing block cannot be compromised because of

TrustZone's hardware protection.

2.5.1 Threat Model

PROTC framework trusts the applications on the ground control station in figure 2.2. Moreover, PROTC also trusts any code running inside the secure world, including all protected peripheral drivers, and the software stack of PROTC. However, the applications installed, the operating system (OS), and the peripherals running in the normal world on the UAV SoC board are malicious and not trusted. We assume the ground control station is secure and cannot be compromised. The Denial-of-Service (DoS) attacks, side-channel attacks towards TEE, sensor attacks, and any other attacks that can maliciously modify or drop the communication messages between the ground control station and the secure world are out of the scope of this paper.

2.5.2 Trusted Computing Block

The trusted computing block is a security guard program that resides in the secure world. It has two main features: 1) It enforces the access control policy described in section 2.5.6 so that only user-approved access to protected peripherals can be permitted (*Decision Maker*). 2) When an access-authorized command is received from the critical applications, it needs to execute the command and return the critical applications(*Command Executer*). As described in section 2.4, the OS in the normal world might be compromised. Hence, access to protected peripherals for normal world applications is revoked by trusted computing block, and these protected peripherals can only be accessed from the trusted computing block due to ARM TrustZone protection.

The memory types of trusted computing block are both shared memory that can be read or written by normal world applications and non-shared secure memory that is only visible to trusted computing block. For all the sensitive data, such as the private key of the trusted

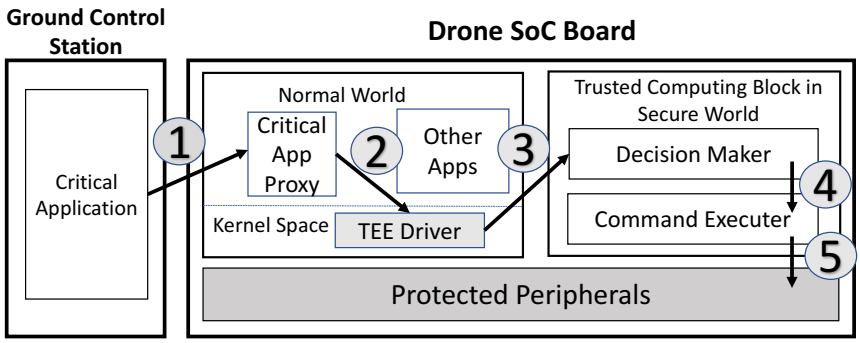


Figure 2.3: Procedure when critical application needs to access protected peripherals. Step①: Critical application sends the encrypted access request to its proxy in normal world. Step②: Critical app proxy passes the access request to TEE driver in normal world. Step③: TEE driver passes the access request through SMC to decision maker in trusted computing block. Step④: Decision maker decides whether to execute the command. Step⑤: Command executer executes the command if decision maker decides to execute.

computing block used to sign protected sensor data sent to the critical application, it is stored in secure non-sharable memory that only the trusted computing block can access.

2.5.3 Critical Applications

Critical applications need a part or full access to protected peripherals. The steps for the critical applications to access protected peripherals are shown in Figure 2.3. One critical application is the UAV control application that needs to access all the sensors and actuators to fly the UAV. In our model, critical applications could be malicious. For example, a sensor data collecting application is only given access to sensors, but it contains a malicious rootkit and tries to change actuators' value to fly the UAV.

Critical applications contain two parts. The first part is installed on the ground control station that could be a UAV controller or a computer on the ground, and its main task is to send the access command to UAV's trusted computing block. The example of UAV's

pilot application sends the control command to the UAV according to the user's input. The second part of the critical application is the proxy installed on the UAV SoC board. Critical application proxy only transmits the information between the ground control station and the trusted computing block. However, some malicious critical applications could have a malicious proxy that compromises UAV normal world OS by rootkit described in section 2.4.

2.5.4 Normal World Applications

Applications in the normal world are referred to as untrusted applications. These applications could be independent applications or the proxy of critical applications. Untrusted applications are used to do tasks that do not require access to protected peripherals. For example, PX4D [PIX] can be installed as an untrusted application if the camera is not protected. However, untrusted applications might be malicious. For example, if an image processing application contains a rootkit that can escalate its privilege, the image processing application can directly write data to actuator memory-mapped registers to control the UAV without the implementation of PROTC.

2.5.5 Secure Communication Channel Establishment

When critical applications on the ground control station need to send control commands to the UAV, it will first establish a secure channel. The purpose is to assign the critical application an access number that will be further used to get authentication from the trusted computing block to access the protected peripherals. The access code assigning algorithm includes the following two steps: 1) Critical Application $\rightarrow$ Trusted Computing Block: Pri_{CRI}(Access Request) (i.e. Critical application signs access request to trusted computing block); 2) Trusted Computing Block $\rightarrow$ Critical Application: Pri_{TCB} (Pub_{CRI} (Access Code)). Then, the critical application verifies the trusted computing block's signature and decrypts the message to obtain access code. We assume the trusted computing block pro-

gram has the public key of the critical application, and the critical application has the public key of the program inside the trusted computing block. We do not use the public key distribution process in PROTC. However, a good and high efficient key exchange protocol could be implemented as SSL/TLS. Once the critical application obtains the access code, it uses the access code as part of the message while sending flight control command to the UAV to reduce the overhead trusted computing block needed when executing the command.

2.5.6 Access Control Policy

The access control policy is enforced by the PROTC decision maker to ensure the integrity of access to protected peripherals. The UAV user determines the access permission to the protected peripherals. When an application on the ground control station (i.e., critical application) wants to access protected peripherals, it will send a request through its proxy to the trusted computing block. If it is the first time sending the access request, the trusted computing block will issue an access request inquiry message back to the access request daemon on the ground control station, and the user will decide whether to give the access. Once the trusted computing block receives permission from the user, it will store the information securely inside a hash table within the secure world. To reduce the computation complexity, the trusted computing block generates a unique random access code for each application, where the random access code is hash-mapped to the application's public key and the protected peripheral access permission information. Once a critical application has access code, it only needs to send the information along with the access code to the trusted computing block as ($Pri_{CRI}(Pub_{TCB}$(request)), Pub_{TCB}(access code)). An overview of the access control decision process is shown in figure 2.4.

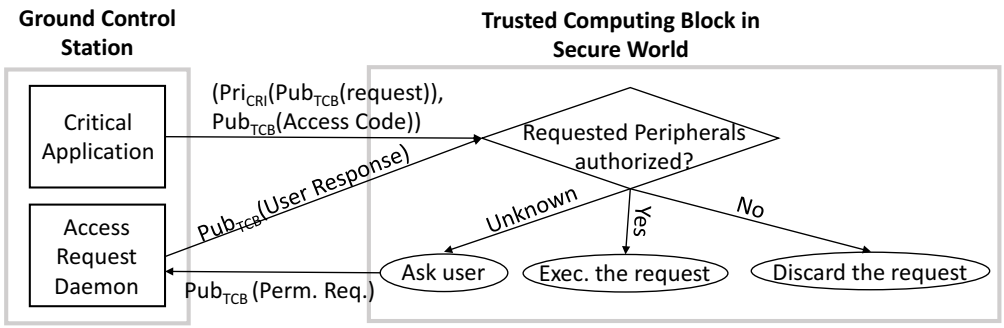


Figure 2.4: Access Control Decision Block Diagram. When critical application sends access request to trusted computing block, trusted computing block checks whether the access has already been authorized: if yes, execute the command; if no, discard the command; otherwise, send the access request back to user for approval.

2.5.6.1 Sensors and actuators hierarchy

If the access request from an application to protected peripherals is denied, the application's remaining execution might be problematic. For example, if an application needs to access IMUs to calculate the corresponding values sent to actuators, but the user denies this access mistakenly, the application might send a dangerous command to actuators and crash the UAV using the problematic results from IMUs. We adopted a hierarchical access design to ensure guaranteed access to lower level protected peripherals if access to a higher level is permitted.

In PROTC, the protected peripherals are divided into two hierarchical groups. All actuators that might affect UAV's safety directly are grouped together as the top level. All other protected peripherals such as barometer and accelerometer are at a lower level. Moreover, the access privilege for actuators is one-for-all that if an application is allowed to access one actuator, it is automatically allowed to access all the other actuators and protected sensors. However, the access privilege for other protected peripherals is one-for-one that if an application has access to cameras, it does not guarantee access to other sensors or actuators

Table 2.1: Summary of micro-benchmarks used to evaluate PROTC design.

Name	Description
AUTH-FULL	The access to all protected peripherals is authorized.
AUTH-PART	The access to part of protected peripherals is authorized.
UNAUTH	The access to all protected peripherals is NOT authorized.
DI-ACCESS	Directly access to peripherals.

automatically.

2.6 Evaluation

We build a prototype of PROTC to test and evaluate our design, and we propose four micro-benchmarks to evaluate PROTC design.

Hardware Setup In our implementation, we choose the Raspberry Pi 3 Model B board that is one of the cheapest ARM TrustZone supported development boards as our prototype testbed platform. Raspberry Pi 3 Model B is also the motherboard of NAVIO2 [NAV] that is one of the most popular UAV SoC board in the current market.

Software Setup In PROTC, we follow the same NAVIO2 OS stack setups for our normal world. More specifically speaking, we choose Linaro that uses a real-time Linux kernel in normal world to ensure real-time constraints. A trusted computing block is installed in the kernel space of OP-TEE[1] in secure world so that the time budget will be guaranteed. We also simulate the sensor data and actuator signals.

[1]OP-TEE on Raspberry Pi 3 is not secure due to the closed source of the hardware design. It is for educational and prototyping purposes only.

Micro-benchmarks We propose four micro-benchmarks that are summarized in table 2.1. The first micro-benchmark is AUTH-FULL that has authorized access permission to access all the protected peripherals. The second micro-benchmark is AUTH-PART that has authorized access permission to access some of the protected sensors only. The third micro-benchmark is UNAUTH that does not access permission to access any of the protected peripherals. The last micro-benchmark is DI-ACCESS, which is served as our baseline benchmark that directly accesses the peripherals from normal world OS through recompiling the whole system stack and disabling the PROTC mechanism. These four micro-benchmarks represent all the possible situations in our system if the normal world OS is compromised. Through evaluating the micro-benchmarks, we prove that PROTC is against the memory attack that tries to control the UAV by compromising the UAV OS.

Security Analysis AUTH-FULL and UNAUTH show that the protected peripherals can be accessed only when the user authorizes the access request. In PROTC, the access code is mapped to both critical application's public key and the permission to protected peripherals, so even if UNAUTH has AUTH-FULL's access code, UNAUTH cannot access the protected peripherals because the trusted computing block will use the public key the access code mapped (i.e. AUTH-FULL's public key) to verify the signature of the message. AUTH-FULL and AUTH-PART demonstrate that the permission given to higher-level protected peripherals can be automatically propagated to lower level peripherals but not vice versa. In one of our experiments, we give AUTH-FULL the access privilege to actuators but not sensors, and due to the design of PROTC, AUTH-FULL still has the access privilege to protected sensors, and this avoids the potential wrong values sent to actuators that might cause the crash of the UAV. AUTH-PART is only given the access privilege for one protected sensor, so when it tries to access other protected peripherals, the decision maker denies its access request. Our results show that even if the UAV OS is compromised, the malicious applications still cannot control the UAV because it does not have permission to access protected

Table 2.2: Statistics of overhead after running 100 times for each micro-benchmark(excluding WiFi latency)

Micro-ben.	**Average (ms)**	**Std. Dev. (ms)**	**Range (ms)**
AUTH-FULL	143.15	0.16	0.64
UNAUTH	143.13	0.10	0.50
DI-ACCESS	0.34	0.02	0.07

peripherals. This result also indicates that PROTC can be against memory-related attacks such as buffer overflow attack and return-oriented programming (ROP) attacks performed by those malicious applications.

Overhead We run the benchmark DI-ACCESS, UNAUTH, AUTH-FULL 100 times to examine our PROTC design's overall overhead, and the results are summarized in table 2.2. The average overhead introduced by PROTC in AUTH-FULL is 143ms. A further breakdown for AUTH-FULL overhead shows that the secure world session connection time from normal world applications occupies 21.49% of the time. The cryptographic algorithm (i.e., pub/pri key algorithm) of PROTC takes 61.17% of the time, and the rest, including sending and executing the commands from normal world to secure world takes 17.34%. The results of AUTH-FULL and UNAUTH tell that the significant overhead of PROTC in AUTH-FULL is the cryptographic algorithms. The overhead of the command executer inside the trusted computing block is only about 200 μs. Because the PROTC mechanism satisfies real-time constraints by adopting RT-Linux kernel in normal world and using kernel space in secure world, after we run the experiments 100 times, it shows that the range of AUTH-FULL is 0.501ms and the deadline while executing command will be guaranteed within 144ms.

2.7 Conclusion Remark

PROTC successfully demonstrates that a new UAV system design based on ARM TrustZone technology can protect UAV from being maliciously controlled. PROTC also shows that it can be against memory attacks on UAV OS. A future direction of this work is to utilize the trusted computing block in PROTC to determine whether to execute a command based on the sensors' values.

CHAPTER 3

VirtSense: Virtualize Sensing through ARM TrustZone on Internet-of-Things

3.1 Introduction

The number of Internet-of-Things (IoTs) has dramatically increased in the past few years. Unlike the early stage IoT devices that are mainly designed as sensor extensions, the current IoT devices are no longer only for data collecting but are capable of learning, or in other words, smart sensing [KLL15]. The software system's key transition from "sensor extensions" to "smart sensing" is the changeover from a single-tenancy sensing system to a multi-tenancy sensing system. We refer to the single-tenancy sensing system as that only one application can access the physical sensors and the multi-tenancy sensing system as that multiple applications can access the same sensors simultaneously without interfering with each other.

The current sensing system on IoT devices inherits the concepts from mobile systems, Linux-based systems, or embedded systems. Such migration shows an inefficiency while workloads have been changed [LL16, LJJ15] from data collecting to sensing. The sensing services on Linux or embedded systems are conceptually analogous to the single-tenancy sensing system because when an application needs to use sensor resources, it sets up a mutex lock so that other applications cannot modify it if a different sampling rate is sought. These sensing mechanisms fail to meet a multi-tenancy sensing system's fundamental requirement because simultaneous sensor access from multiple applications with varying sensing rates is

prohibited. On the mobile system such as Android, the sensing service is performed through a *max-for-all* mechanism. The OS only provides the data with the maximum sampling rate from all applications, although most applications do not need such a sensing granularity. On iOS, sensor management discards the excessive data when doing downsampling for different apps. Moreover, every application only allows having one sensing rate on iOS, so all threads in a multiple-threaded application are only allowed to sense at one rate. Although the sensing service from the mobile system is more advanced than the embedded system's sensing service, all of the burdens of dealing with excessive sensor data fall on the applications themselves. Nevertheless, the sensing system should take over the burdens to achieve the sensing isolation among and within applications on the multi-tenancy system. Moreover, mobile devices' application workloads are different from those on IoT devices, where IoT devices are mostly used for sensing or actuating, but mobile devices lean more as user-interactive workloads. Hence, optimizing the sensing system on mobile devices is not an easy implementation.

Another challenge faced by the multi-tenancy sensing system is the enforcement of sensor access control policy. The traditional access control mechanism on IoT devices inherited from mobile systems usually seeks for the user's permission at the time when an application launches. Under this access control mechanism, if the state of the device changes while using the application, and the new state no longer permits access to certain sensors, the access control policy will not be able to execute such permission changes at runtime. Some research work [EGC10] tracks the data flow to enforce access control policy at runtime. These access control mechanisms rely on the OS boundary of user space and kernel space. However, the OS boundary is not always reliable. For example, the vulnerability in Wifi-module found in 2019 (CVE-2019-17666) on Linux can escalate a userspace application with kernel privilege. If the OS gets compromised, all these access control mechanisms can be bypassed.

To remedy the shortcomings of current sensor systems for IoT devices, we propose *VirtSense*, as illustrated in figure 3.1, an enclave and virtualization-based framework that meets the real-time demands of the multi-tenancy sensing system. Such a system aims to achieve

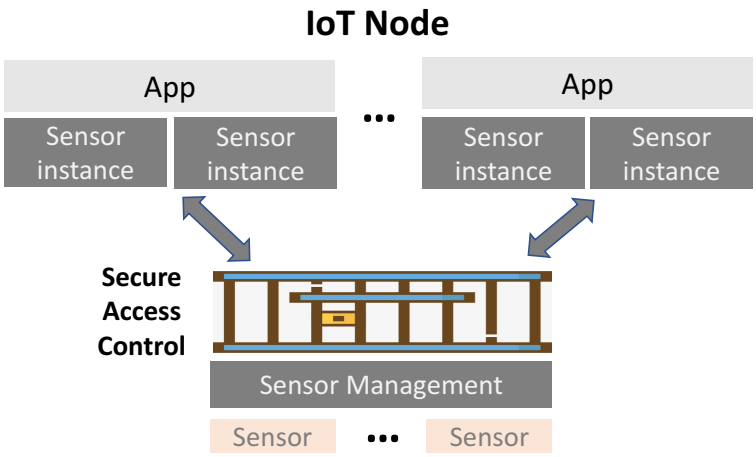


Figure 3.1: VirtSense Architecture

sensing isolation among different sensing applications and enforce the access rules in an untrusted environment. These goals enable simultaneous and legit access to sensors from different applications without affecting each other.

The concept of virtualization helps the sensing system multiplex the limited physical sensor resources to different sensing applications through sensing resampling techniques. While software-based virtualization introduces extra overheads, we balance the trade-offs between sensing accuracy and data delivery speeds. We expose such trade-offs as options for developers to choose so that they can control the granularity of sensor accuracy and data delivery speed.

Enclave technology furnishes *VirtSense* with a trusted computing base (TCB), which further guarantees the enforcement of access control policy. The enclave TCB will maintain its integrity when the OS gets compromised because of the hardware protection. One of TCB's design considerations is to keep its minimal size due to the extra execution overhead. Hence the balance of how often in enforcing permission rules should be carefully chosen. If the access permission is checked too frequently, the overhead is too large; if the access permission is checked too rarely, it will not provide access protection. In *VirtSense*, we

allow the developers to choose how often they want their access rules to be enforced for the applications' access requests.

Our implementation of *VirtSense* uses ARM TrustZone based devices and prototypes the architecture on Raspberry Pi. Our preliminary results show that *VirtSense* satisfies multi-tenancy sensing needs by — 1) Allowing each sensing application control over its specific set of sensing requirements; 2) the access control policy is enforced under simulated compromised OS.

3.2 Background

3.2.1 Sensing System

Most embedded systems and Linux-based OSes only allow one application to sense at a given rate, while all other applications have to wait until the one releases the resource if they have different sensing needs. On Android, the sensor manager manages the sensing requests. All the applications send the sensing requests to the sensor manager through a Binder message, at which the sensor manager will then obtain the sensor data and distribute it to the applications. The sensing rate is not guaranteed because the sensor manager chooses the largest sensing rate from all the applications and broadcast the data to all sensing applications.

3.2.2 ARM TrustZone

ARM TrustZone is the trusted execution environment technology initially developed by the ARM company. It is implemented on all A-series processes and two m-series (m23 and m33). ARM TrustZone separates the execution environment into two - normal world and secure world. The normal world is the untrusted environment running an untrusted OS. The secure world is the trusted computing base of TrustZone. The context switch between the normal

world and secure world is done through Secure Monitor Call (SMC). The code that runs in the secure world has higher privilege than the code in the normal world, and the secure world can define the memory region that can only be accessed by privileged code. These regions could be regular memory or memory-mapped registers of peripherals. If the code running in the normal world tries to access the protected memory regions, TrustZone throws a hardware exception.

3.3 Design of VirtSense

Our crucial motivation for the *VirtSense* sensing system on IoT devices is that the sensing system should take care of the sensing needs from different sensing applications. These sensing needs are the sensing requests for periodic sensing at different sampling rates or event-driven sensing requests. Under the current sensing framework such as Android or iOS, the burden of taking care of the sensing difference is on the applications' side. In other words, the current sensing framework lacks the understanding of applications' sensing needs. Moreover, the current sensor access control mechanisms purely rely on the isolation and protection from the OS. When an OS gets compromised by a malicious application containing a rootkit, the access control mechanisms will no longer protect the device.

We provide a sensing framework *VirtSense*, as illustrated in figure 3.1 that supports the discrepancy of multiple application sensing workloads with the emphasis on sensing flexibility, accuracy, and security. In this section, we will discuss the design of the framework.

3.3.1 Design Goals

To overcome the shortcomings of the current sensing framework while not diminishing the functionalities of the existing sensing framework, we set up the following goals to design our new sensing framework.

- **Sensing simplicity.** Current sensing framework pushes the sensing burdens to each application. If one application changes its sampling rate, it might affect all other applications using the same sensing service. Nevertheless, while dealing with the different sensing requests from different applications, the sensing framework needs to manage the sampling requests' difference but not the applications themselves. Due to the hardware resource limitation that each physical sensor can only have one sampling rate, the sensing framework should multiplex the physical sensor values to different sensing events. The sensing system should provide a high-level abstract for multi-tenancy sensing applications but not pass this burden to the applications.

- **Options for balancing sensing accuracy and sensor data delivery speed.** Downsampling or upsampling is needed when the sensing framework multiplexes the physical sensor values to each sensing request at their requested sensing rate. Downsampling or upsampling requires data reconstruction, and different techniques will affect the data accuracy and the delivering speed because higher accuracy requires more data to reconstruct the sensing signal, which lowers the data delivering speed. The abstraction of choosing such downsampling or upsampling sensing techniques should be exposed to the application developers to properly determine the most beneficial downsampling or upsampling techniques for their applications.

- **Sensor access security.** All the existing sensor access control mechanisms, such as asking users for permissions when launching an app and using data-flow to track the access control, rely on user space and kernel space boundary security. While designing the new sensing framework, we aim at the access security of sensors even if the OS is compromised (i.e., a broken of user and kernel space). The access control policy needs to be independent of applications but only controls the behaviors of how an application accesses the sensory data. Furthermore, the access control should have the capability of dynamically adjusting sensor access permission while using the device rather than only asking for permission at the application launching time. For example,

an access control rule can reject the microphone sensing request if the device is in a secret conference room while approving it if the device is outside a conference room.

- **Scalability.** The sensing framework needs to have a proper level of scalability. When new software or hardware sensors are added to the system, the new sensing framework is able to integrate them into the existing system with the sensors' driver modules without overhauling the whole sensing system.

3.3.2 Design Principles

To achieve the goals explained in section 3.3.1, we present *VirtSense* sensing framework that virtualizes physical sensors based on enclave technology. While designing *VirtSense*, we set forth two essential principles *VirtSense* needs to follow.

3.3.2.1 Principle of Various Sensing Workloads

Every sensing application has its requirements for sensing work. The examples of the sensing workloads can be different sensing rate for periodical sensing or different event-driven sensing requests. The various sensing workloads also indicate that different applications need to share the limited physical sensing resources. *VirtSense* needs to understand the discrepancies among all the sampling requests and satisfy these needs.

Embracing sensor virtualization in *VirtSense*. Virtualization overrides the hardware resource limitations by allowing software management to multiplex the hardware resources for each virtual instance, allowing applications to request one or more instances for the resources. In contrast, each instance is independent and does not interfere with each other. Specifically speaking, the virtual sensor instance in *VirtSense* refers to an instance for a physical sensor or a combination of multiple physical sensors (e.g., software sensors). Each application no longer needs to be "penalized" while other applications require different sens-

ing requests. For example, when application A requests a periodic sampling with a sampling rate of 60 Hz while application B requests a periodic sampling with 100 Hz and application C requests an event-driven sampling, the application A, B and C themselves do not need to comprehend the difference of other applications since *VirtSense* is responsible for multiplexing the physical sensor resources. Meanwhile, each application can also choose the sensing accuracy for its instances. By virtualizing the physical sensors, *VirtSense* is committed to provide applications `sensing simplicity`, `scalability` and the options for `sensing accuracy`.

3.3.2.2 Principle of Untrusted OS Execution Runtime

Due to the large size and various functionalities of the OS, OS can get compromised through malicious kernel extensions or rootkits. Hence, the compromised OS have the abilities to bypass all the security protection mechanism. Under *VirtSense*, it must ensure its code's confidentiality and integrity, especially access control code, which implies that the code is executed under a possible compromised OS environment. Enclave technology provides a protection infrastructure to allow *VirtSense* to run securely under an untrusted environment.

Embracing enclave in *VirtSense*. Enclave provides hardware-level protection for the confidentiality of essential memory-mapped peripherals. Using enclave protection can guarantee the required safety policy to be enforced regardless of the health status of the OS. The enclave will throw a hardware exception if the malicious OS tries to access the protected memory regions, including those memory-mapped registers for peripherals, without passing through the access control policy. Differing from the traditional OS protection boundary for the access control of all sensors, *VirtSense* adopted enclave for the access for the sake of enhanced security, which provides the system an improved `sensor access security`. The enclave in *VirtSense* provides a minimal yet sufficient trusted computing base. The access for sensors from the OS outside the enclave is prohibited, and the only method to use the

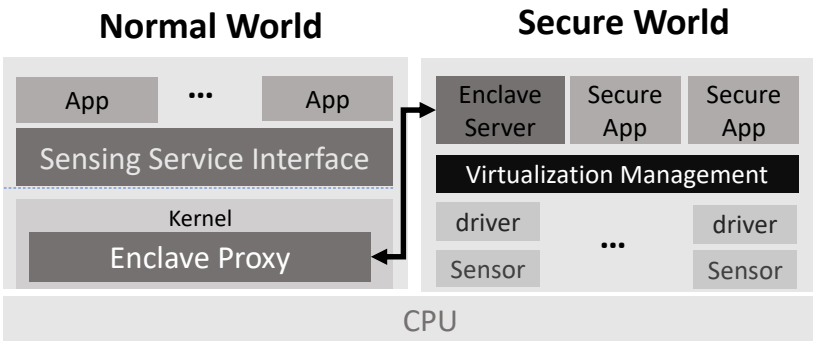


Figure 3.2: VirtSense Overview

sensor resources from the OS is to send an access petition to TCB in the enclave. This protective layer makes sure the sensor access policy inside TCB is enforced regardless of the security status of the OS.

3.4 *VirtSense* Prototype

We prototype *VirtSense* with four key components that are sensing service interface, enclave proxy, secure apps, and virtualization management, as shown in figure 3.2. These four parts provide a secure interface and sensing service for the applications in the normal world.

3.4.1 Threat Model

Under the *VirtSense* framework, we assume that the applications in the normal world are not trusted. These applications could be intentionally malicious and compromise the whole OS. They can be downloaded and installed from any third-party manufacturer while the applications and execution runtime inside the secure world are secure and trusted. The secure world applications need to be conducted extra scrutinizing by an authorized party. The side-channel attacks against TEE and Denial-of-Service (DoS) attacks are beyond this paper's scope. Moreover, *VirtSense* assumes the applications do not maliciously use the

sensor data after the access is granted.

Sensing service interface. The sensing service interface provides the interface APIs to applications for sensing instances creation and the sensing rate and accuracy manipulation. To keep a proper high-level abstraction, the sensing service interface is the only exposure point between the applications and *VirtSense* service. It also plays the role of load balancer of *VirtSense*, which determines how to deliver the sensor values to each application with minimal overhead. For example, if all applications sample at 60 Hz, the sensing service interface will only create one instance for all the applications, even if every application requires a virtual sensing instance.

Enclave Proxy. Enclave Proxy communicates with the secure world inside the enclave. It batches and sends the sensing requests from the normal world applications to the enclave server in the secure world through the SMC driver. Moreover, the secure world's sensor data are also received by the enclave proxy that further delivers to sensing service interface and applications.

Enclave Server. The enclave server is the communication midpoint between the normal world and the secure world, residing in the secure world. The enclave server processes the sensing request batches from the normal world applications and sends them to virtualization management. It also packs up the sensor data updated from the virtualization management and delivers them back to the normal world's applications. Enclave proxy and enclave server are the two communication endpoints that bridge the sensing requests sent from the normal world and the sensor data delivered back from the secure world.

Secure Apps. Secure apps, which are the critical components of the access control mechanism, are pre-installed. They are used to control access from normal world applications to sensors. All the sensor access requests need to be sent to secure apps through virtualization

management for access authenticity checks. Secure apps can contain rules defined by the users to restrict the access of sensors. For example, a rule can be depicted as when the device is within a certain room. The speaker cannot sense the environment by the applications. If the decision is more complicated to make than using rules, a pre-trained neural network can be integrated with the secure apps. Moreover, because secure apps have no restrictions on accessing the protected sensors, the pre-trained neural network can be further trained while using the device. The sensor access request from normal world applications can only be granted if all secure apps approve such access.

Virtualization management. Virtualization management enforces the access policies from secure apps and dynamically chooses the sensing sampling rate set for the real physical sensors to satisfy the sensing requests from the normal world applications. The virtualization management receives the sensing requests through the enclave server, and it checks whether it has violated the access rules from either of the secure apps. If the access request is permitted, the virtualization management will change the sensing sampling rate if necessary and send sensor data back to the enclave server when they are updated. The details of the virtual sensing algorithms and APIs provided to the developers will be introduced in section 3.4.2.

3.4.2 Sensing Virtualization Algorithm

The algorithm of *VirtSense* virtualization management provides each sensing instance a resampled sensor value calculated based on the physical sensor value. As we discussed in the previous section, *VirtSense* allows both event-driven sensing and periodic sensing. In *VirtSense*, each application is permitted to create one or more sensing instance. For different sensing instance, the application developers can specify a resampling method to satisfy the sensing needs. *VirtSense* sets the maximum sensing rate among the sampling requests by all the applications as the physical sensor sampling rate. Figure 3.3 shows a demonstration of different sensing resampling techniques. The sensor's firmware pre-processes all the raw

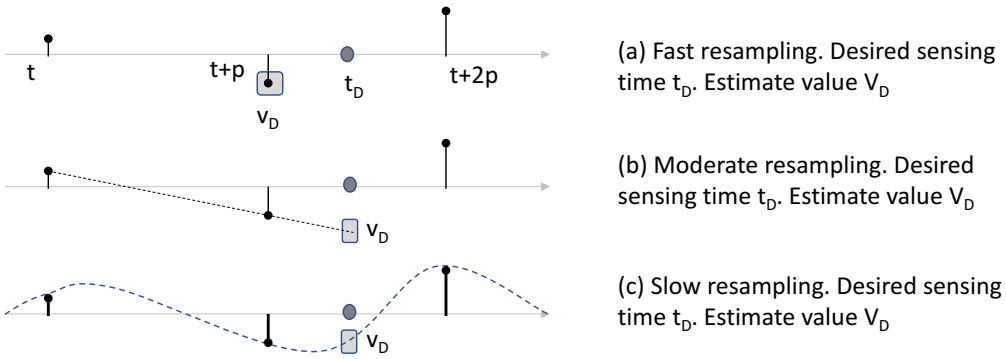


Figure 3.3: Example of resampling methods. Physical sensor value updated at t, t+p and t+2p, where p is the physical sensor sampling rate. t_D is the desired sensing time for a virtual sampling request, and V_D is the returned value marked in a grey box from *VirtSense*.

data for the resampling.

Fast resampling. Fast resampling method returns the closest available sensor value at the requested sampling time. This method provides sensor resampling values with low accuracy but high speed. When the application developer chooses this method, *VirtSense* will deliver the sensing value at the highest priority regarding the other two resampling methods.

Moderate resampling. Moderate resampling uses a linear approximation strategy to resample the sensor values. This method will provide a more accurate sensor resampling value than the fast resampling method, while the calculation and delivering speed will be slower than that. *VirtSense* assigns moderate resampling a medium priority while providing the sampling values to the applications.

Slow resampling. Slow resampling is supposed to provide the most accurate resampled data. These sample data recovering techniques are high-order approximation with least square estimation or [GGG04, AG01, GS03]. Using these methods will be the lowest priority,

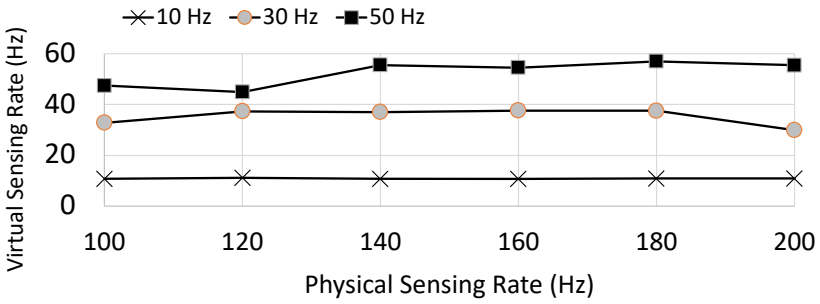


Figure 3.4: Running application with desired sampling rate 10Hz, 30Hz and 50Hz simultaneously (fast resampling) with physical sensor value from 100Hz to 200Hz with 20Hz increment in *VirtSense*.

hence the lowest delivery speed while delivering to the applications.

3.4.3 Sensor Security and Access Control

The secure world restricts all access to the sensors. The secure world prohibits direct access to the sensors from the normal world OS. But rather, the normal world applications need to send the access request to TCB residing in the secure world. In *VirtSense*, we implement secure apps as the safety rule checkers to check the access request's validity. When the enclave server receives the batched access requests from the enclave proxy, it passes the requests to secure apps through virtualization management, and then the secure apps will determine whether the access is allowed. Because the context switch between the normal world and the secure world is expensive, secure apps cannot track the information flow of how sensor data objects are used, such as TaintDroid [EGC10].

3.5 Evaluation

3.5.1 Experiment Setup

Our experiment uses Raspberry Pi 3 as the platform. It is one of the cheapest ARM A-series development board. We use Open-Portable Trusted Execution Environment (OP-TEE)[1] for the software setup. OP-TEE is an ARM TrustZone enabled prototype operating system. OP-TEE uses Linaro Linux in the normal world and Rich OS in the secure world based on GlobalPlatform TEE Client API. We also generate simulated sensor data at different sensing rate. Because the secure world runtime of OP-TEE does not have inter-process communication based on the message queue, we use a secure-world application shared storage file to emulate the inter-process communication.

To evaluate our preliminary implementation, we launch three periodic sampling applications with sampling rate at 10Hz, 30Hz, and 50Hz, and several other malicious applications with various sampling rates. We also provide a secure app that contains the rule only to allow sensor access if they are sampled at 10Hz, 30Hz, or 50Hz. We measure the average sampling received for each application, as shown in figure 3.4.

3.5.2 Preliminary Results

Overhead. Our results show that the overhead introduced by *VirtSense* is 16ms on average. We use two different clocks to calculate the overhead. The first clock starts before calling into the secure world and ends up exiting from the secure world. The second clock measures the overhead introduced by the access control applications inside the secure world. By further breaking down the overhead, roughly 0.18 ms comes from the context switch between the normal world and the secure world, and the rest comes from the enforcement of

[1] `https://www.op-tee.org/`. Current OP-TEE on Raspberry Pi 3 is not secure. It is for educational and prototyping purposes only.

security rules by security apps. The current security rule enforcement communicates to sensor virtualization management through a shared secure file, and it can be further optimized through a more efficient inter-process communication such as message queues.

Multi-tenancy sensing virtualization. To test how the physical sensing rate could affect the virtual sensing rate, we run three sensing applications requiring 10Hz, 30Hz, and 50Hz sensing rate simultaneously with different physical sensor rates, as shown in figure 3.4. We let the applications run for about ten seconds and calculate the average sampling rate. The average sampling rate demonstrates that the more accurate it is when the desired sampling rate is low.

Security analysis. We install a secure app that restricts the access of all the sensing requests except with sensing rate 10Hz, 30Hz or 50Hz. The secure app successfully blocked the sensing requests not specified by the rule. *VirtSense* can prevent memory attacks such as buffer overflow attacks from malicious applications to read sensor data without the user's authorization. Moreover, it also prevents the sensing request conflicts when two different sensing applications require different sensing frequencies. Our implementation of *VirtSense* does not protect against DoS attack because secure apps must scrutinize each request. However, the DoS attacks from malicious applications can be mitigated if the secure app can restrict a certain number of access requests in a unit time.

3.6 Related Work

Enclave technology has been widely deployed in all kinds of different systems in recent years. For example, Intel SGX is for secure machine learning [OSF16a, LJL17, PMS16], Docker container [ATG16], distributed system [ZDB17, HZX16] and and database system [MCS17, BSP16].

Sensor virtualization has been explored in the domain of IoT networks [EM09b, BGA15]. Senaas [ACN10] uses event-driven sensor virtualization technology to provide an interface for IoT clouds. SenseWrap [EM09a] and Kim el. [KK15] designs a network middleware to provide cloud-based IoT virtualization. Furthermore, Islam el. [ILH13] designed a virtualization approach for wireless sensor networks. Ko el. [KLL15] proposed the virtualization of sensor network management on mobile devices. Some other work designed sensor or actuator virtualization framework on specific areas, such as smart camera [JNG17, SIS11]. However, unlike *VirtSense*, none of the above work focuses on the sensor virtualization on a single node. PROTC [LS17] uses ARM TrustZone to protect the drone's safety. SeCloak [LSD18a] uses ARM TrustZone to ensure an actuation control on the mobile device. Unlike the prior work, *VirtSense* uses enclave to achieve a secure virtualization environment for sensors.

3.7 Conclusion

We proposed a new sensing architecture *VirtSense* based on virtualization and enclave to satisfy the needs of multi-tenancy sensing applications. Through our preliminary results, we show that *VirtSense* provides both a secure and flexing sensing system.

CHAPTER 4

Aerogel: Lightweight Access Control Framework for WebAssembly-Based Bare-Metal IoT Devices

4.1 Introduction

The scope of leveraging mobile and internet-of-things (IoT) devices for sensing physical spaces has generalized beyond human activity recognition. Distributed and deployed IoT systems leverage the ubiquitous sensors for a myriad of applications such as smart healthcare, smart city lighting, and transportation [HQS17]. In the 5G context, on-device edge computation enables nascent market applications such as augmented reality, mass IoT, and drone services [STL20]. Consequently, the latency requirements, privacy, and security concerns for these safety-critical applications have naturally shifted computation from centralized cloud resources to decentralized edge IoT and mobile devices [STB16].

The heterogeneity of the underlying device hardware and software ecosystems poses complex challenges for application developers. The dynamicity and heterogeneity of these devices necessitate support for dynamically instantiated, portable workloads stemming from more than one source while maintaining security and performance for applications. More critically, the isolation mechanisms to secure these platforms assume some form of memory management unit (MMU) [ARMi]. Several resource- and energy-constrained IoT hardware platforms do not support MMUs [ARMd, ARMe]. For instance, in 2016, experts estimated ARM to have shipped 22 billion units of the MMU-less Cortex-M based devices [Ens16]. Although Cortex-M processors are enabled with *Memory Protection Units (MPU)* that can provide

memory isolation, MPUs can only support a finite number of memory regions. Moreover, applications need to be rewritten under different bare-metal OSes that use MPUs because they require different OS abstractions.

The requirements of security, performance, portability, and dynamic instantiation for heterogeneous computation platforms are not unique to the 5G IoT edge. In response to the increasing demands of performance and security for web application deployment in fragmented and heterogeneous environments, engineers from the top four major browser vendors collaboratively developed WebAssembly (Wasm) [Com]–a portable low-level bytecode that is platform-independent. Subsequently, industry giants such as Intel and Redhat formed an alliance, known as the Bytecode Alliance [Cla20], to develop a micro-runtime for Wasm that is supported by *bare-metal* IoT devices, i.e., resource-constrained, MMU-less devices where all software models share the same memory space. The Wasm Micro-runtime (WAMR) [Byt] enables applications that can run the target binary at native speeds independent of the device and can achieve sandboxing without a memory-management unit.

Although WAMR is a strong candidate to support secure, performant, and *multi-tenant* computation on edge, the scope of IoT applications is not limited to computation services. The computation abstractions will run alongside sensing and actuation services provided by the IoT device that interacts with the physical world. WAMR currently lacks the abstractions necessary to provide access control to sensors and actuators for IoT devices while maintaining performance and security. Steps have been made towards providing limited access control (e.g., only for certain memory regions or pieces of sensitive information) for multi-tenant IoT devices using hypervisors [KB18, AMT18, GPH11], using the compiler at the compilation time [KSK19, CAS17, CAB18], or using secure runtime memory views based on offline static analysis [YZ18]. However, the proposed architectures are device-dependent, requiring the recoding and recompilation of the software stack for different IoT device architectures. The shifting workloads of the dynamic and heterogeneous IoT edge will require over-the-air (OTA) updates at runtime while supporting other *tenants*. Thus, in this paper, we aim to tackle the

following **challenge**: how can we extend the security capabilities of Wasm on IoT to include access control for multi-tenant IoT device peripherals while maintaining performance and low resource overhead?

In this paper, we design AEROGEL, a runtime framework that utilizes the protection mechanisms of Wasm bytecode sandboxing to provide access protection for IoT device peripherals – even when the applications and the OS are sharing the same address space. AEROGEL builds upon the Wasm runtime to provide micro-management for each tenant (application). Tenant applications are compiled into Wasm bytecode such that applications can be platform-independent. The Wasm runtime isolates application bytecode from any platform-dependent native code that needs to interact with the application. AEROGEL instruments Wasm runtime to provide a fine-grained access control mechanism such that users can easily define the processor energy consumption, memory usage, as well as access to sensor and actuator peripherals for each application.

We evaluate AEROGEL on a low-power, resource-constrained MCU dev board (nRF52840) and benchmark a representative set of safety-critical IoT applications. AEROGEL's runtime overhead ranges from 0.19% to 1.04% extra execution time and from 18.8% to 45.9% extra energy on our proposed benchmarks. Our results show that the fine-grained access control mechanism provides minimal overhead for MCU energy, and peripheral access energy while having a minimal overhead on application execution relative to related works.

Contributions. We summarize our contributions as follows.

- We propose AEROGEL, a Wasm-based access control mechanism for bare-metal IoT devices. Wasm enables platform-independent application execution necessary for heterogeneous IoT networks.
- AEROGEL leverages the sandboxing capabilities of Wasm to isolate tenant applications from each other as well as from platform-dependent native code. AEROGEL enables

secure sandboxing for multi-tenant applications for resource-constrained (less than 1 MB of memory), low-power devices.

- We evaluate AEROGEL on a real low-power, resource-constrained MCU and show results of minimal 0.39μAh extra energy and minimal overhead 2.1ms.

The rest of the paper is organized as follows. Section 4.2 briefly discusses the background information of Wasm and bare-metal IoT devices. We then overview AEROGEL in section 4.3 and explain the details of the design in section 4.4. We talk about the implementation in section 4.5 and evaluate our work in section 4.6. We next analyze the security issues and limitations and discuss the future work in section 4.7. We compare AEROGEL with the related work in section 4.8. Lastly, we conclude this paper in section 4.9.

Our source code to reproduce our results is available online: `https://github.com/nesl/Project-directory`.

4.2 Background

We first discuss the emerging field of multi-tenancy on bare-metal IoT devices. We then describe the security guarantees provided by Wasm and give a brief overview of the Wasm runtime for IoT devices.

4.2.1 Multi-tenant Bare-metal IoT Devices

Bare-metal IoT devices have shifted away from single-purpose applications as equipped sensors and actuators enable them to perform multiple tasks. For example, the battery-powered smart camera Blink XT2 [Sec] can capture images and perform on-device object detection. Further, the development ecosystem of IoT devices has enabled APIs for developers to implement applications that leverage the sensor and actuator abstractions, e.g., the Skills API for Amazon Alexa [ATA02]. Hence, we model the complex and fragmented software

and hardware IoT ecosystems as *multi-tenant* application environments. However, supporting multi-tenancy confounds the challenges of performance, sustainability, and security on resource-constrained devices.

Bare-metal characterization. We characterize *bare-metal* IoT devices with limited resources, such as small battery capacities containing few thousand mAh energy, low-end microprocessors (MCUs) with only a few hundred MHz frequencies, or small memory size with few hundred Kilo-Byte(KB) memories. These devices typically do not have complicated memory protection mechanisms such as user space and kernel space address separation through the Memory Management Unit (MMU). Moreover, those devices are designed for heterogeneous sensing and actuation workloads such as UAVs and smart home sensors. For example, Pixhawk 4 [PX4] flight control device is equipped with two ARM-M processors that have 216 MHz for flight control and 24 MHz MCUs for I/O operations, and each processor has 512KB and 8KB RAM respectively. Other popular bare-metal IoT device examples include the Nest Protect [Goo] (MCU = 100 MHz [ARMd] and RAM = 512 KB) and the EdgeReady Voice Control platform [NXP] (MCU = 600 MHz [ARMe] and RAM = 1024 KB).

Lack of multi-tenant isolation. MMU-less, bare-metal IoT devices cannot provide memory isolation among different applications. Figure 4.1 shows an example of a bare-metal UAV system supporting two different applications, i.e., tenants, that perform sensing tasks to control flight dynamics. If various entities develop the applications, the bare-metal devices would not be able to protect one safety-critical application from another application's bugs or vulnerabilities. Although researchers [LCG17, DMS14, PPR18] have proposed to leverage Memory Protection Units (MPUs) on ARM Cortex-M based IoT devices to provide memory isolation [CAB18], MPUs can only support a finite number of memory regions. Moreover, the associated applications would not be portable as they need to be rewritten under different bare-metal OSes. Thus, we require a lightweight, portable, and software-based memory

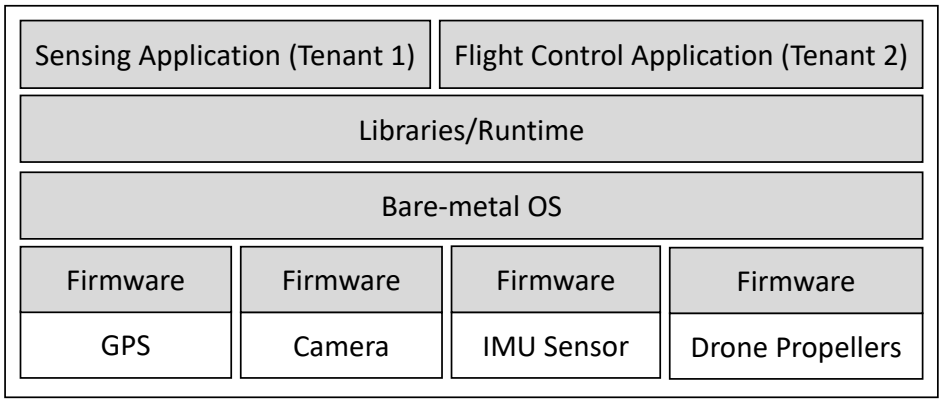


Figure 4.1: The UAV example of multi-tenant bare-metal software stacks AEROGEL targets on. All the code in the shared areas including runs on the same address space.
isolation for multi-tenant applications.

4.2.2 WebAssembly for Non-web Embeddings

Researchers have adopted WebAssembly [Com] (Wasm) to account for the bottlenecks of security, portability, and dynamic instantiation. Wasm was initially designed for web browser JavaScript applications on heterogeneous client devices to enhance the security of script isolation, improve web application execution speed, and erase the execution environment's heterogeneity requirement. Wasm has since generalized beyond web embeddings to bare-metal IoT devices due to the original problems' generality.

4.2.2.1 Tenets of Wasm.

We first motivate the major advantages of Wasm in the context of bare-metal IoT devices.

Performance. Wasm is designed as a statically-typed programming language such that the variable type is determined at compilation time. Additionally, Wasm utilizes a linear memory structure that is loaded as a compact binary format. Hence, Wasm is able to achieve

near-native speed performance [HRS17, was].

Security. Wasm provides runtime code isolation for applications by running them in sandboxed execution environments. Inside each sandbox, the Wasm application has full access to its memory. However, any access to the memory outside of the sandbox throws an exception. Moreover, each Wasm application has its own sandbox, and its sandbox cannot be arbitrarily accessed by different applications. By achieving this, Wasm runtime linearly allocates memory regions and ensures that the entry and the exit point of a function do not go beyond the sandbox bounds. If a Wasm bytecode instruction reads from or writes to a specific memory address, Wasm runtime will check whether the memory address is within the application's sandboxed memory regions.

Portability and Dynamic Instantiation. Wasm is a platform-independent binary format whose execution resides on top of its runtime. Hence, Wasm applications are portable on any device that has a Wasm runtime and can initiate the execution environment without recompiling the software stack.

4.2.2.2 Wasm Micro-runtime

Wasm bytecode is executed on a Wasm runtime. From the runtime's perspective, Wasm bytecode is a group of Wasm bytecode instructions, where each instruction is encoded with one OPCODE followed by one or more arguments. For example, `i32.add(i32.const 3)(i32.const 2)` computes the addition of 3 and 2. Wasm byte code applications have a special instruction that can execute native functions inside Wasm runtime exported through pre-registered function tables.

Wasm micro-runtime [Byt] (WAMR) is one of the most popular lightweight runtimes for Wasm bytecode on bare-metal devices. WAMR has only a few hundred kilobytes of memory

footprint. WAMR manages all the execution of Wasm bytecode. In the beginning, it allocates a contiguous memory region for the Wasm application that can only grow contiguously starting from the end address. Next, each Wasm instruction is translated into machine code by WAMR. Hence, if the Wasm opcode is a memory access opcode, it is further sanitized by WAMR to make sure such access does not go beyond the allocated memory region. To reduce the execution overhead, WAMR also allows a mixture of Wasm bytecode execution and platform-dependent execution. A mixture may be allowed for platform-dependent functionality optimization. For example, if a Wasm application tries to use pthread, WAMR allows it to choose an optimized platform-based pthread and run the native code. To maintain security enforcement, the associated native code is provided by WAMR and not the application developer. Although WAMR provides the initial framework for Wasm on IoT devices, Aerogel will aim to a provide peripheral access control framework for bare-metal IoT applications. Next, we overview the design of Aerogel.

4.3 Overview

We first describe the threat model and goals of Aerogel followed by the design workflow.

4.3.1 Threat Model and Assumptions

Aerogel trusts the software stack below the applications running on bare-metal IoT devices. More specifically speaking, Aerogel trusts the firmware, the bare-metal OS, and the Wasm runtime. Aerogel does not trust any application. We assume the entire software stack code–including the application, the bare-metal OS, the Wasm runtime, and the firmware of the hardware–is running on the same address space as there is no MMU for the memory address space separation. Side-channel attacks, including cyber-physical attacks, towards the sensors or actuators such as GPS spoofing are out of the scope of this paper.

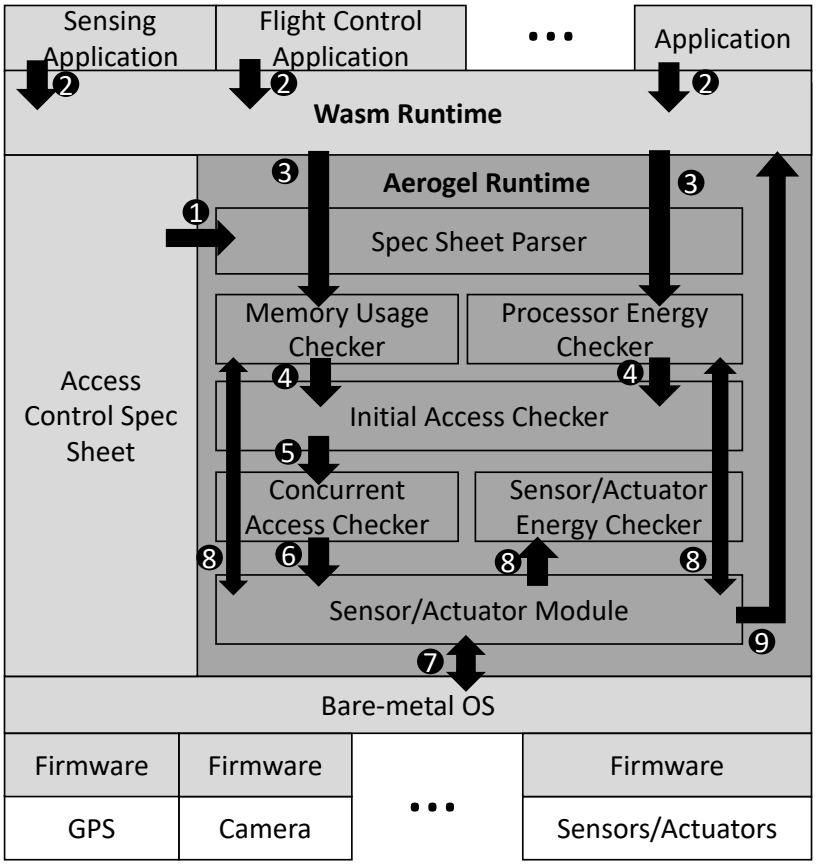


Figure 4.2: An overview and the workflow of AEROGEL. The darker grey area is the component of AEROGEL.

4.3.2 Goals

We enumerate the design goals for AEROGEL as follows:

- **Sensor and actuator access protection**. Each application is expected to be isolated from each other under a robust sandboxed execution environment. AEROGEL needs to make sure its execution environment does not allow arbitrary access to the peripherals such as sensors or actuators.
- **Fine-grained access control**. AEROGEL aims to provide a fine-grained access control

mechanism such that the users are able to define the the processor energy consumption, memory usage, as well as the energy consumption per sensor and actuator for each application.

- **Minimal overhead and memory footprint**. AEROGEL aims to provide low overhead and low memory footprint runtime support such that it has minimal execution impact for all applications on the resource-constrained bare-metal devices.

4.3.3 Workflow

Figure 4.2 provides an overview of AEROGEL's design workflow. Prior to execution, all application code needs to be compiled to Wasm bytecode. At runtime, AEROGEL parses the access control specification sheet (❶), which is the user-defined access rules whose details are discussed in section 4.4.2. After the specification sheet is processed, the Wasm runtime loads the Wasm bytecode of the applications as Wasm instructions and initiates their runtime environment(❷). Wasm runtime then attempts to execute the loaded Wasm instructions. Upon each instruction execution, the Wasm runtime makes a request to the AEROGEL runtime checker to determine if the current application has exceeded the maximum allowable processor energy and memory usage(❸). If an application's processor usage has exceed its allotment (assuming the allotment exists), AEROGEL will request the bare-metal OS schedule the current application to sleep for a user-defined period of time. The total processor energy consumption for the particular application is reset to zero by the processor energy checker after the user-defined reset time in the access control specification sheet has elapsed. Resetting the application's energy consumption prevents the application from sleeping indefinitely. If the application's memory usage has reached its allotment, the memory can no longer be increased.

If the Wasm instruction requires reading sensor data or writing data to actuator peripherals, a request is sent to AEROGEL's initial sensor and actuator permission checker to check

whether such an access is allowed in the user-defined sensor and actuator allowlist (❹). Once the initial access has been cleared, AEROGEL's runtime checks whether the maximum total number of accesses has been achieved for the requested sensors or actuators by the access monitor(❺). If either the initial permission checking or the maximum number of access checking fails, the access is denied. AEROGEL's runtime throws an exception that is handled by Wasm runtime. Otherwise, the request is passed to be registered by the sensor and actuator module(❻) that directly interfaces with the sensors and actuators. When new sensor data for or a new actuation command from a particular application needs to be handled by the sensor and actuator module(❼), the module sends requests to the energy usage and the memory usage checkers to verify whether the memory usage or the energy usage has exceeded the maximum allotment for the associated sensor or actuator (❽). If not, the sensor and actuator module executes the actuation command or sends back the new sensor data to the application (❾). Otherwise, the corresponding command or data is discarded.

We next explain the details of AEROGEL's design.

4.4 Aerogel Runtime

We first describe how AEROGEL provides memory protection for sensors and actuators. We then explain how access control policies are defined and enforced.

4.4.1 Wasm-based Peripheral Memory Isolation

AEROGEL's bare-metal peripheral access control hinges on isolating the peripheral memory locations from the application memory that resides on the same contiguous linear memory space. We describe how AEROGEL isolates the memory-mapped peripherals from application memory in two stages: application instantiation and application runtime.

4.4.1.1 Isolation at Application Instantiation.

When the Wasm runtime instantiates the runtime environment for an application, it needs to allocate the associated memory heap. The runtime searches for the first available memory region from the beginning of the linear physical memory. AEROGEL's runtime checks whether such allocation has overlapped with sensor or actuators memory-mapped addresses, i.e., by checking whether the linear regions include sensor or actuator addresses. If an overlap is detected, AEROGEL's runtime returns the first possible available memory regions that do not overlap with the I/O address of the sensors and actuators with the required size of the memory and returns the start address to Wasm runtime.

Memory collision resolution. When Wasm runtime's requested memory region overlaps with a sensor's or actuator's memory-mapped I/O address, AEROGEL's runtime starts looking for the first possible memory addresses that could fulfill with the request. AEROGEL's runtime first searches from the low address to the high address of all sensors and actuators without considering other conditions such as whether the memory is used by other applications, further checked by the Wasm runtime. For instance, a sensor and an actuator may have addresses `0x8FFFFFE0` and `0x8FFFFFF0` respectively, and the Wasm runtime needs to allocate `0x100` bytes of memory. Assuming the memory is aligned in 4-byte settings, AEROGEL's runtime first checks whether `0x8FFFFFD4` can fulfill with the request of `0x100` memory size. In this instance, the allocation overlaps with the actuator's address (`0x8FFFFFE0`). AEROGEL's runtime then checks whether `0x8FFFFFE4` can be a potential candidate and ensures there are no other actuators or sensors between `0x8FFFFFE4` and `0x900000D0` (`0x8FFFFFE4` + `0x100`), hence `0x8FFFFFE4` will be returned to Wasm runtime that will do further checks of whether the memory regions will be eligible.

4.4.1.2 Isolation at Application Runtime.

When the application's current memory size is not large enough to satisfy the needs, the Wasm runtime will enlarge the existing memory region. Wasm application's runtime memory is enlarged by extending the end address of the original memory's end address but keeping the same starting address. In other words, only one chunk of contiguous physical memory is allowed for each application. Wasm runtime will do a similar memory checking procedure as when instantiating an application to request AEROGEL runtime to check whether such extension overlaps with memory mapped I/O addresses of the sensors or actuators, but the anticipated memory size is the application's original memory heap size plus the required enlarging memory size. If a new starting address is return by AEROGEL runtime, Wasm runtime copies the contents from the old memory chunk to the new memory regions and frees the old memory trunk.

Given peripheral memory isolation, we can now explain how AEROGEL's runtime enforces access control to the devices by starting from how the users should specify the access rules through an *access control specification sheet*.

4.4.2 Access Control Specification

The AEROGEL access control specification sheet defines the permission list for each application. AEROGEL requires the user to provide two pieces of information through the specification sheet: 1) per-device specifications and 2) per-application specifications. **Per-device specification.** For each device, the user needs to define the device's manufacturer information. In particular, the user should specify the power consumption profiles for each sensor, actuator, and memory-mapped I/O addresses as well as the processor power under different power states. The user also defines the maximum number of applications that can access each sensor or actuator at a time. **Per-application specification.** For each application on the IoT device, the user provides an allowlist of sensors or actuators, the maximum energy

```
<Device-Spec>
    <!--The metadata of the device-->
    <Param name="home_camera" description="Home smart camera"/>
    <Sub-Devices>
        <Property>
            <!--Capture the image-->
            <Param name="image_sensor" address=0x8FFFFFE0/>
            <!--Power state, power unit mW-->
            <Max-Access val=10/>
            <Power-State val="active" power=800>
            <Power-State val="sleep" power=300>
        </Property>
        <Property>
            <!--Adjust the angle of the camera-->
            <Param name="angle_actuator" address=0x8FFFFFF4/>
            <Max-Access val=1/>
            <!--Power state, power unit mW-->
            <Power-State val="active" power=1000>
        </Property>
    </Sub-Devices>
</Device-spec>
```

(a) An example of per-device specification sheet. This device specification sheet is a smart home camera that has an image sensor capturing the images and a camera angle adjustment actuator.

```
<Access-Control-Spec>
    <Applications>
        <!--Application name and description-->
        <Param name="home_security" description="Home security App"/>

        <!--Allowed access devices and energy (uAh)-->
        <Access-Devices>
            <!--Set the consumed energy to be 0 after reset (ms)-->
            <Device name="image_sensor" energy=2500 reset=10000>
            <Device name="angle_actuator" energy=unlimited reset=0>
            <Device name="door_actuator" energy=1500 reset=15000>
        </Access-Devices>

        <!--Allowed processor energy and memory usage-->
        <Processor-Memory>
            <Processor val=5000 reset=15000> <!--5000 uAh-->
            <Memory val=131072> <!--128 KB-->
        </Processor-Memory>
    </Applications>
</Access-Control-Spec>
```

(b) An example of per-application specification sheet. This access control specification sheet is a smart home security monitoring application, which is allowed to access smart camera and door controller.

Figure 4.3: Example of the specification sheet needed by AEROGEL.

usage, the maximum processor energy usage, as well as the maximum memory usage. The user will also specify the *reset time* for resetting the application's total energy usage to be zero.

Figure 4.3 shows an example of the two specification sheets for AEROGEL[1]. Figure 4.3a shows a per-device specification sheet for a smart home security camera that has an image sensor and an angle adjustment actuator. In this example, up to ten applications can access the image sensor. The device has two power states: active and sleeping. The angle actuator is only accessible by one application at any time. Figure 4.3b is an example of access control per-application specification sheet. In this example, the *home_security* application is given access to the smart camera's image sensor, the angle adjustment actuator, and the door movement actuator. Access to the angle adjustment actuator allows unlimited energy usage. The total energy usage for the processor resets after 15000 ms.

Once the specification sheet policies are loaded in memory, AEROGEL's runtime starts enforcing the access control rules using a hierarchy of access checkers. The first access control checker focuses on compute resource access control.

4.4.3 Compute Resource Access Control

The first stage of AEROGEL's access control focuses on compute resource policies. AEROGEL first checks compute resource access policies before peripheral access policies since all applications will require compute resources, but not all applications will access peripherals. AEROGEL's compute access control has two components: the *memory usage checker* and the *processor energy usage checker*.

[1]The grammar template can be found here: `https://tinyurl.com/aerogel-spec-sheet`

4.4.3.1 Memory usage checker.

The memory usage checker performs the total memory usage checking when a new piece of memory region needs to be allocated by the Wasm runtime. Thus, this type of access control checking is triggered in two scenarios: 1) application instantiation and application runtime memory expansion. Because Wasm runtime allows only one chunk of contiguous memory for each Wasm application, the memory usage checker only needs to keep track of each application's start and end addresses when its memory regions are changed. The memory usage checker computes the total memory usage of a specific application by subtracting the application's end address from the application's start address. The difference is compared with the user-specified memory usage threshold. Hence, the performance of checking the memory usage is always constant, i.e., $O(1)$.

4.4.3.2 Processor energy usage checker.

An application's processor energy usage is defined as the processor energy consumed by executing its Wasm instructions and invoked native functions. For example, suppose a Wasm instruction adds two numbers with the opcode ADDITION followed by two numbers as the arguments. In that case, the processor energy consumed is the processor energy that needs to add those two numbers, including loading them to the registers and storing the result back to the memory.

For each Wasm instruction or native function invocation, AEROGEL's runtime records the total execution time under different processor states and computes the energy consumption using the following formula:

$$E_{total} = \sum_{i=i}^{n} P_i * t_i \tag{4.1}$$

where $P_1, P_2, ..., P_n$ are the different power states of the processor and $t_1, t_2, ..., t_n$ are the corresponding execution times.

After the execution of one Wasm instruction, the processor energy usage checker checks

whether the application's total energy cost has exceeded the maximum allowable value. Suppose the total energy cost is more than the allowed maximum. In that case, the application is scheduled to sleep for a period defined by the user. Once the period has passed, the total energy cost resets to zero.

Performance optimizations.

Since one application might have many Wasm instructions, it is inefficient to compute the energy for every instruction. We introduce two optimization methods to reduce the overhead of the processor energy checking procedures.

- For the static instructions whose execution times do not change under different applications, e.g., addition and subtraction, AEROGEL's runtime stores the value of their associated energy cost. When AEROGEL encounters those instructions, the checker retrieves the value from the first computation.
- Similarly, when instructions have the same processor execution cycles, we only need to compute the processor energy consumption for one of them and reuse the calculated value for the rest instructions. For example, loading a 32-bit float to a register has the same number of execution cycles as loading a 32-bit integer.

After passing the compute resource access control checkers, the application can issue requests to access sensor and actuator peripherals. The requests are forwarded to the sensing and actuation specific application access checkers.

4.4.4 Sensor And Actuator Access Control

Unlike the compute resource access controls checkers, the sensor and actuator access control checkers only enforce the access control rules when an application requires access to the sensors or actuators. AEROGEL's sensor and actuator access control consists of three

components: the *sensor and actuator initial permission checker*, the *access monitor*, and the *sensor and actuator energy usage checker*.

4.4.4.1 Sensor and actuator initial permission checker.

The initial permission checker is triggered when a new application requests AEROGEL's runtime to read the sensor data or write data to actuator peripherals. When such a request is received, the initial permission checker checks whether the requested sensor or actuator is in the allowlist of sensors and actuators for the application parsed from the specification sheet. If the requested sensor or actuator is on the list, the initial access checker will allow the request to advance to the next stage. Otherwise, a denial will be sent back to the application.

4.4.4.2 Access monitor.

The access monitor verifies that only a certain (user-defined) number of applications are accessing the sensors and actuators, i.e., AEROGEL enforces the user-defined counting semaphores for sensor and actuator peripherals. When an application needs to register with the sensor and actuator module, the access monitor finds the current total number of applications accessing the sensor or actuator. If the access is less than the total number of allowed accesses, the peripheral access will be granted. AEROGEL's runtime then increments the total number of accesses. When an application dispatches from the sensor and actuator module, AEROGEL's runtime will decrement the total number of applications accessing the sensor or actuator.

4.4.4.3 Sensor and actuator energy usage checker.

When there is a new peripheral event to be handled such as new sensor data or new actuation command, the sensor and actuator module sends the power states of the sensor or actuator

and the duration of that application on each power state to the sensors and actuators *energy usage checker*. The checker looks up the power information of each power state sent from the module according to the previously parsed per-device specification sheet. The checker then computes the energy usage of this event using a similar formula as Equation 4.1 and adds this energy to the total sensor or actuator energy consumed by the application for that particular sensor or actuator.

The energy usage checker also compares whether the energy usage has exceeded the maximum allowable value. If yes, the access checker will request AEROGEL's runtime to virtually dispatch the corresponding application from the sensor and actuator module, i.e., the application cannot read or write to peripherals. The application's total energy consumption for the given sensor or actuator is only reset to zero when the user-defined reset period has passed. If the application was previously dispatched from the sensors and actuators module, it would be virtually registered back with the module. All virtual dispatches or registrations do not decrease or increase the number of total accesses for that sensor or actuator.

4.5 Implementation

In this section, we will discuss the implementation of AEROGEL.

4.5.1 System Setup

We prototyped our design AEROGEL as depicted in figure 4.2 with the Wasm micro-runtime (WAMR) [Byt]–which is implemented in a mixture of C and assembly on both a bare-metal dev board (Nordic nRF52840 [Nor]) and a simulator (QEMU [QEM]). We use the Zephyr real-time OS [Zep] as the bare-metal operating system. The nRF52840 dev board is equipped with a 32-bit ARM Cortex-M4 MCU whose running frequency is 64 MHz with 1MB flash and 256 KB RAM. The nRF52840 is mainly utilized by wireless IoT devices such as wireless security cameras. Because we need to measure the overhead of AEROGEL under different

processor frequencies, we also set up the QEMU simulator with various MCU frequencies from 10 MHz to 110 MHz with 512 KB RAM.

AEROGEL and the associated runtime are implemented with a total of 2321 significant Lines of Code (sLoC): 1399 sLoC for the AEROGEL runtime implementation, 108 sLoC for testing and debugging purposes, and 814 sLoC for evaluation.

4.5.2 Aerogel Runtime

We now describe how we implement the three major components of the AEROGEL runtime: the access control specification sheet parser, the sensor and actuator module, and the access control checkers. Moreover, we describe how we augmented our implementation to support Just-in-Time (JIT) compilation for the Wasm applications enabled by the WAMR.

4.5.2.1 Access control specification sheet parser

The parser needs to initiate two types of state variables when parsing the specification sheet: global state variables and per-application state variables. The global state variables are shared among all the applications. In particular, all variables extracted from the per-device specification sheet information are considered global variables, e.g., the address of each sensor and actuator, the power states, and the maximum allowable concurrent access to a particular sensor or actuator. The access control specification sheet initiates the global state variables only once. We implement the `parse_per_device()` function to parse the per-device access control specification sheet at the beginning of the `wasm_env_create()`–which creates the Wasm environment for all Wasm applications.

Per-application state variables are parsed from per-application specification sheets and vary for different applications. We implement the `parse_per_app()` function to parse the per-application access control specification sheet such that each application's variables are initialized. These variables include information about the allowable set of sensors and

Table 4.1: APIs implemented of the sensor and actuator module exposed to the Wasm application developer.

APIs	**Description**
`sensing (id, freq, duration, cb_func)`	Register the application to the sensor with *id* for given *frequency* and *duration*. The results are sent back by the *callback* function.
`actuation(id, *value, cb_func)`	Send the actuation command with the expected *value* to the actuator with *id*. The returned values if any from the actuator are sent back through the *callback* function.

actuators, the energy allowed, and the associated reset time. The access control checker will use the variables after each application is instantiated by the `wasm_instantiate()` function–which initiates the Wasm application runtime for a particular application.

4.5.2.2 Sensor and Actuator Module

We implement the sensor and actuator module as native functions that are exported and exposed to the Wasm applications. When a Wasm application calls the sensor and actuator module functions, the Wasm runtime looks into a function table pre-registered with all native functions and finds the symbols of the sensor and actuator module functions. The symbols are linked with the Wasm application at runtime.

We implement two APIs for application developers, as summarized in table 4.1. The sensing API is used to register the application to listen to any sensing events, and the actuation API is used to send an actuation command to the actuators from the application. When a Wasm application invokes either of these APIs, the sensing and actuating functions will first call the `access_control_checking()` function of the AEROGEL runtime to ensure

such a request is legitimate. If the request is to periodically send actuation commands or receive sensing data, the sensing or actuating functions will call the energy and memory usage checking functions to ensure the total energy and memory usage has not exceeded the application's allotment.

4.5.2.3 Access Control Checkers

The sensor and actuator access checkers' implementation is integrated with the sensor and actuator module. In particular, the sensor and actuator initial permission checker and the access monitor are called at the beginning of the functions `sensing (id, freq, duration, cb_func)` and `actuation(id, *value, cb_func)` before executing the sensing or actuation requests. The sensor and actuator energy usage checker is implemented at the sensor and actuator module before the peripheral request is executed. The reset timer used by the Aerogel access control checkers to reset the total peripheral energy is realized with the Zephyr up-time timer, which is the time relative to the board's boot-up time.

The compute resource access checkers are implemented where a Wasm native function or bytecode instruction is called. For the memory usage checker, it is invoked when extra memory is needed for the Wasm application's runtime. In particular, the memory usage checker is implemented at the beginning of `wasm_instantiate()` function that instantiates the Wasm runtime environment for the application and the `wasm_enlarge_memory()` function that requests extra memory when the current memory is not large enough. The processor energy usage checker is implemented at the end of the execution of each Wasm bytecode instruction or native function.

4.6 Evaluation

We evaluate our design Aerogel on both nRF5840 dev board and QEMU. We first explain the benchmarks we used for our evaluation, followed by the experimental results.

Table 4.2: Simulated sensors and actuators for Unmanned Aerial Vehicles (UAVs) and smart home for AEROGEL evaluations.

Category	Device Name	Peripherals	Description	Index	Power	Max Concurrent Access
Unman. Aerial Vehicle (UAV)	Camera	Camera image sensor	Capturing the images	①	2W	5
	GPS	GPS sensor	Sensing GPS positioning signals	②	1W	10
	Propellers	Motor actuators	Controlling UAV propeller motors	③	10W	1
Smart Home	Camera	Camera image sensor	Capturing images	④	5W	5
		Angle control actuator	Controlling camera angles	⑤	4W	1
	Door Control	Motion Sensor	Detection moving objects	⑥	0.2W	10
		Door motor actuator	Controlling door opening and closing	⑦	3W	1
		Battery usage sensor	Detecting battery capacity	⑧	0.2W	10
	Speaker	Speaker actuator	Playing sound from the speaker	⑨	4W	1
	Microphone	Microphone sensor	Sensing acoustic signals	⑩	1W	2

Table 4.3: Benchmark applications running on nRF52840 board and its access configurations.

	App Name	Description	Devices Used	# of Wasm inst.	Wasm file size (Bytes)	Allowed Devices	Device Energy Allowed	MCU Energy Allowed	Memory Usage Allowed
Regular Access	uav_ctrl	UAV flight control system	①②③	75	542	①②③	100 mAh	20 µAh	Unlimited
	uav_sense	UAV image capturing based on different locations	①②	41	362	①②	50 mAh	Unlimited	250 KB
	home_monitor	Voice control to get home info and play it via the speaker	④⑧⑨⑩	86	607	④⑧⑨⑩	Unlimited	Unlimited	255 KB
	home_security	Door opening after image identity verification	④⑤⑥⑦	70	565	④⑤⑥⑦	100 mAh	Unlimited	230 KB
Restrict. Access	uav_shortage_mcu	Exceeds max allowed MCU power usage on UAV	①	41	374	①	10 mAh	**0.5 µAh**	250 KB
	home_shortage_cam	Exceeds max allowed home camera power usage	④	41	393	④	**1.5 µAh**	60 µAh	240 KB
	uav_max_access	Exceeds max allowed access to UAV propellers	①③	77	549	①③	60 mAh	70 µAh	250 KB
	home_init_denial	Access to some smart home sensors denied	④⑧⑩	55	490	④	60 mAh	50 µAh	200 KB

4.6.1 Benchmarks

To evaluate our design, we first implemented several simulated sensors and actuators for Unmanned Aerial Vehicles(UAVs) and smart home environments. For the UAVs, we simulated a camera, a GPS, and the motor for the propellers. For the smart home scenario, we simulated four different devices that have more than one sensor or actuator, e.g., a smart home camera and a door controller. Table 4.2 summarizes all of the simulated devices.

We evaluated eight different Wasm sensing and actuation applications, as summarized in table 4.3 based on the sensors and actuators of the UAVs and the smart home. Among

these eight Wasm sensing and actuation applications, four of them have regular access to the sensors or actuators. The other four are restricted to evaluate denial for certain access requests.

The regular access Wasm applications for UAVs are the *uav_ctrl* that is designed to be the UAV flight control system, and *uav_sense* that is used to capture an image through the camera of the UAV. In the smart home scenario, we proposed a *home_monitor* application that monitors the home status through the available sensors and a *home_security* application that protects the safety of the home. The four restricted access applications are used to evaluate the four different access control checkers under extreme conditions such as a shortage of processor energy consumption and sensor energy consumption, maximum concurrent accesses to peripherals, and initial access to peripherals denial. For each sensor or actuator the applications try to access, we set the duration for one second.

4.6.2 Results

We analyze the results of the benchmarks on the nRF52840 board and QEMU. We combined the overhead of the initial access checker, the memory usage checker, and the maximum concurrent access checker for all of our results. The total overhead in the worst case is less than 0.07% of the execution time.

4.6.2.1 Latency overhead

We run all benchmark Wasm applications on the nRF52840 board, as summarized in table 4.4. The reported overhead in this table does not include the specification sheet parsing since this is done only at the system boot up time, where the average overhead of parsing the specification sheet is 450ms.

Our results show that for the regular access applications, Aerogel runtime introduces at most 1.04% overhead. Most of the overhead comes from the sensor and the actuator energy

Table 4.4: AEROGEL overhead of benchmark applications running on nRF52840 dev board.

Aerogel Overhead Breakdown

Percentage over Total time	Processor Energy Checker	Sen./Act. Energy Checker	Other Checkers	**Total Aerogel Overhead**	App Execution	Total time (ms)
uav_ctrl	0.01%	1.02%	0.01%	1.04%	98.96%	3166
uav_sense	0.00%	0.17%	0.01%	0.19%	99.81%	1056
home_monitor	0.00%	0.92%	0.02%	0.94%	99.06%	4127
home_security	0.00%	0.58%	0.02%	0.61%	99.39%	4092
uav_shortage_mcu	0.01%	0.12%	0.01%	0.14%	99.86%	1530
uav_shortage_cam	0.01%	0.52%	0.02%	0.55%	99.45%	1171
uav_max_access	0.01%	0.17%	0.06%	0.24%	99.76%	1064
home_init_denial	0.01%	0.09%	0.07%	0.16%	99.84%	1054

checker, whose energy checking happens more frequently than the other checkers. For the UAV sensing application, the overhead is only 0.19% of the total execution. This reduction comes from the shorter execution time–which implies lesser sensor and actuator energy usage checks. When examining the runtime overhead of the restricted access applications, we found the sensor energy shortage application has the most overhead at 0.55%. This overhead is the result of the applications that require frequent access to sensors and, thus, more energy checks when new sensor data is available. On the other hand, the lowest MCU application's overhead is 0.14%. When the application's energy usage is denied, the application is scheduled to sleep immediately–resulting in fewer checks than other applications.

We examined the overhead percentage of different access checks relative to the total AEROGEL overhead. The results–depicted in Figure 4.4–show that the sensor and the actuator energy usage checker consumes the most overhead. The energy usage checker is triggered when a new sensor event or actuation command needs to be handled. Some applications, e.g., the *uav_control* that sends more than 1000 actuation commands, trigger thousands of sensor and energy usage checking procedures. In contrast, the processor energy usage checker triggers only tens of times, as shown in table 2.1. Hence, the overhead of the sensor and actuator energy usage checker is significantly higher than that of the processor energy usage checker. Other access control checkers, such as the initial access control checker, consume

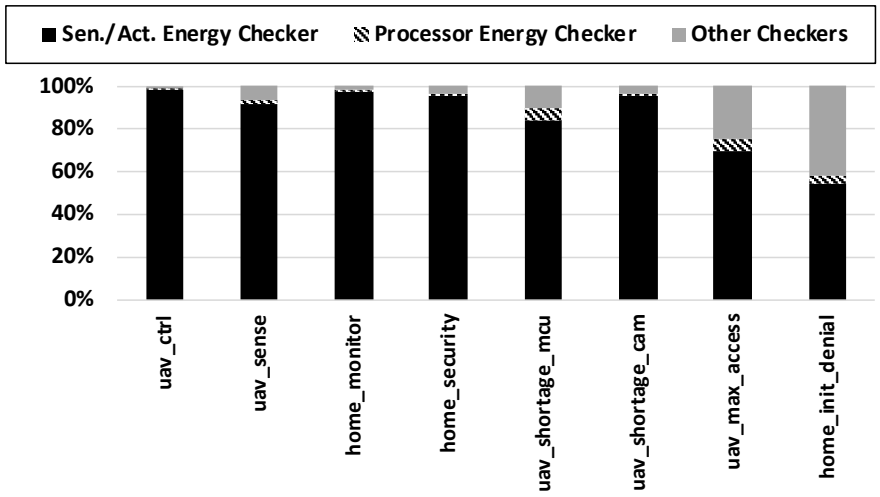


Figure 4.4: The overhead percentage of different AEROGEL runtime access control checkers under each Wasm application on nRF52840 board.

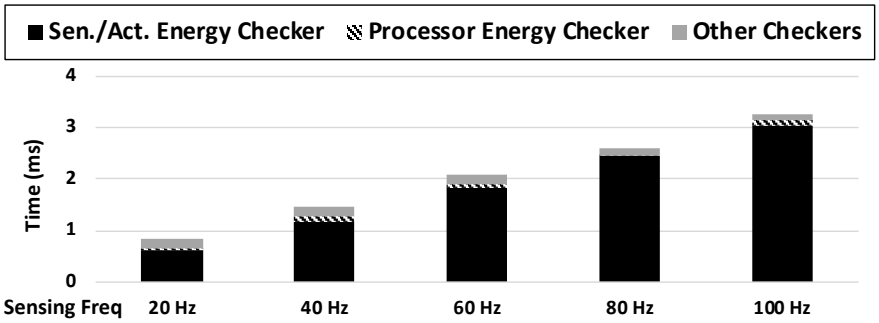


Figure 4.5: The overhead of AEROGEL for *uav_sense* on nRF52840 board under various camera sensing frequency.

a large portion of the overhead only in restricted access applications, especially the initial access denial app. This overhead is due to the high-frequency sensor's access denial, resulting in fewer energy usage checks.

We ran the experiment under different sensing frequencies on the nRF52840 dev board and different processor frequencies on the QEMU emulator. Figure 4.5 shows the different

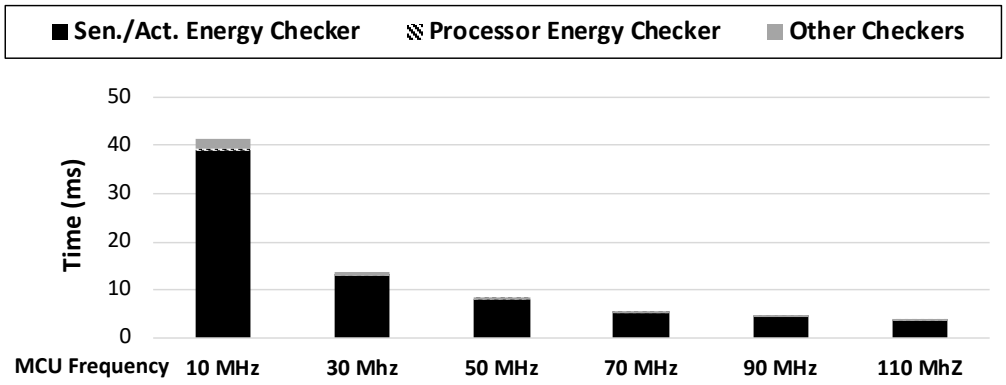


Figure 4.6: The overhead of AEROGEL for *home_security* on QEMU simulator under various MCU frequency.

camera sensing frequencies of the *uav_sense* application. The overhead of the sensor and actuator energy usage checker increases with respect to the increase of the sensing frequency. In contrast, the overhead of other access control checker does not increase. The sensing frequency change only increases the number of sensing events that trigger the sensor energy usage checking. We also show the overhead of AEROGEL runtime under different MCU frequencies on the QEMU simulator, as illustrated in Figure 4.6. AEROGEL runtime has very minimal overhead when the MCU frequency is greater than or equal to 70 MHz.

4.6.2.2 Memory overhead

We next evaluated AEROGEL's SRAM and flash memory overhead. We only evaluated the memory overhead for the four regular access applications since the restricted access applications cannot provide full execution paths. The flash memory size is the size of all the compiled code–including the OS, the applications, and the AEROGEL runtime. For the flash memory, AEROGEL runtime only introduces a marginal overhead that is less than 5KB–independent of the application. The AEROGEL runtime overhead of SRAM is also minimal around 0.1KB. This minimal overhead is due to the fact that no significant amount

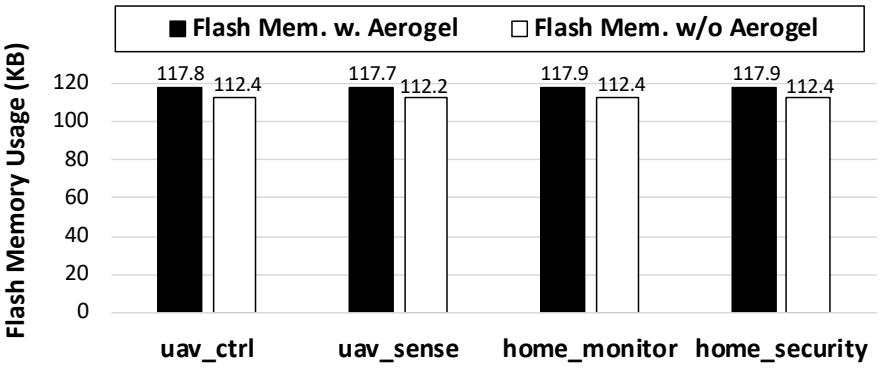


Figure 4.7: The flash memory usage of nRF52840 board when running different applications.

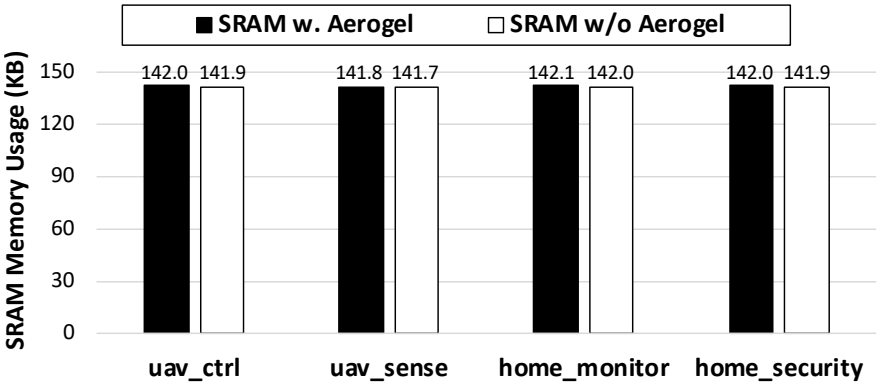


Figure 4.8: The SRAM usage of nRF52840 board when running different applications.

of temporary data is stored in memory.

4.6.2.3 Energy consumption

Finally, we evaluated our benchmark on nRF52840 for energy consumption. To measure the board consumption, we used the Monsoon power monitor [Mon] to connect to the dev board. The monitor provides a 3V external battery for the board. Our results show that the AEROGEL runtime costs a maximum of 0.65 μAh and a minimum of 0.39 μAh for all regular access applications. For the restricted access control applications, the energy con-

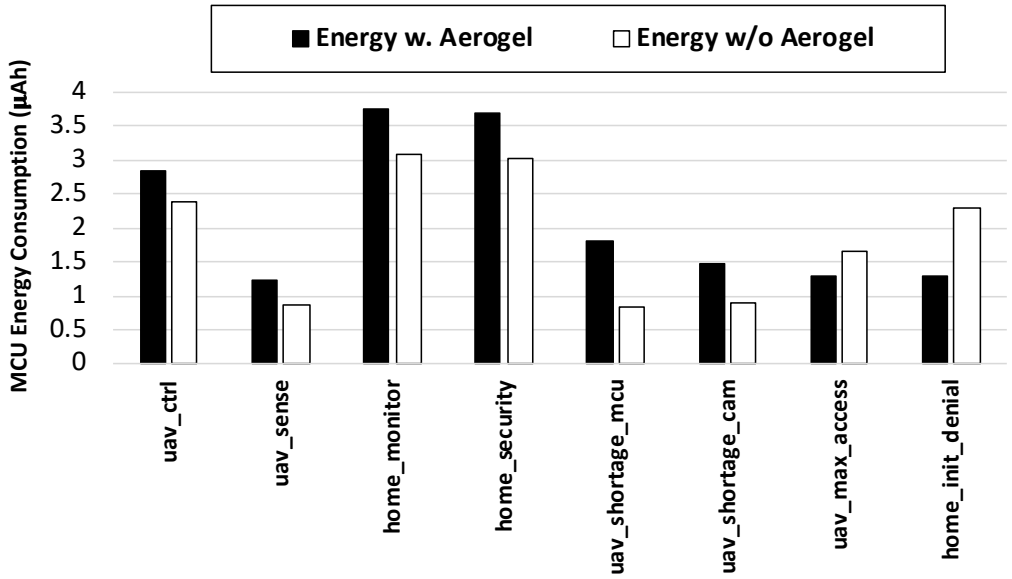


Figure 4.9: The energy consumption of nRF52840 board when running different applications.

sumption with Aerogel runtime enabled for some applications such as *home_init_denial* and *uav_max_access* is less than those without enabling Aerogel runtime. This is because when such access is denied, Aerogel runtime does not need to have further energy usage check of the sensor or actuators. The application is not allowed to further execute to read the sensor data or to send the actuation commands.

4.7 Security Analysis

We will do the security analysis and discuss the future works.

Attack Prevention. Due to the memory protection characteristics of Wasm, Aerogel can build a secure runtime that provides software memory isolation for all peripherals even when the devices lack the memory address space separation from MMU. Aerogel also protects the bare-metal IoT devices from malicious applications that try to drain the device's resources such as the battery and memory. Moreover, Aerogel can protect the sensors and

actuators of the device from being accessed without the user's authorizations. Further, AEROGEL's processor energy usage checker can mitigate a Denial-of-Service (DoS) attack from a malicious Wasm application because it has a restricted energy and memory usage allotment.

Limitations. AEROGEL cannot protect against side-channel memory attacks [LKP20]. AEROGEL also cannot protect against cyber-physical attacks on the peripherals such as GPS spoofing. Furthermore, AEROGEL cannot prevent the application from misusing the sensor data or sending dangerous actuation commands if it is granted access to the sensor or the actuator. Next, AEROGEL cannot provide access control for sensors or actuators that are not memory-mapped I/Os, e.g., interrupt-based sensors, because Wasm runtime only provides the isolation of the memory. Finally, RAM energy usage control is out of the scope of this paper.

4.8 Related Work

We now compare the existing work with AEROGEL.

Wasm on Edge. OneOS [JGP19] designs a single-image universal edge OS for heterogeneous IoT devices using JavaScript that can enable Wasm execution. Hall et al. [HR19] utilizes Wasm to execute serverless functions on edge to reduce the hardware resources usage with respect to traditional edge serverless computation systems. Wasmachine [WW20] uses Wasm to host an edge operating system with kernel written by Rust to speed up the applications running on top of it. Jeong et al. [JSS19] proposes a system offloading Wasm functions mingled with JavaScript to edge server from the mobile devices to reduce execution latency. However, unlike AEROGEL, none of the above focuses on access protections to use peripherals such as sensors and actuators on bare-metal IoT devices where MMU is not

available.

Memory Protection on Bare-metal Devices. ACES [CAB18], uXOM [KSK19], MicroGuard [SHC19] and Clements et al. [CAS17] use the compiler to achieve the memory compartmentalizing on bare-metal devices based on ARM Cortex-M's Memory Protection Unit (MPU). PicoXom [SC20] uses MPU to provide read-only memory execution environment on ARM Cortex M environment. μRAI [ACB] enforces Return Address Integrity (RAI) by modifying the compiler to move the return address from writable memory to read- and execute-only memory regions. Moreover, ARMor [ZLD11] uses formal verification methods of software fault isolation (SFI) to ensure the memory safety and the control flow integrity of applications by inserting dynamic check before certain instructions. Unlike them, Aerogel is able to achieve memory isolation including the protections to the peripherals at runtime without pre-inspecting the applications.

Access Control on IoT Devices. Access control on IoT devices. AccTee [GNK19] uses Intel SGX with Wasm to enforce access control usage for the memory and CPU usage for cloud applications. Several prior works [AKA19, PDK19, OBS18, HCK18, LMN18] adopt blockchain techniques to achieve decentralized access control for IoT devices. Atlam et al. [AAW17] builds an access control model based on the context of the environment to decide whether granting the access request exposes the security risks of the data usage. However, although the above works propose access control framework for IoT networks, these works fail to provide fine-grained access control for applications running on an individual devices unlike Aerogel.

4.9 Conclusion

In this paper, we propose AEROGEL, a lightweight access control framework to provide the users capabilities of having fine-grained access control for Wasm-based bare-metal IoT devices. AEROGEL leverages the security features of Wasm runtime to protect the access and usage of peripherals. We prototype AEROGEL on nRF52840 dev board, and the results show that AEROGEL only introduces 0.19% to 1.04% overhead.

Part II

Securing Inferencing from Sensor Data at the Edge

CHAPTER 5

SecDeep: Secure and Performant Deep Learning Inference Framework for Edge Devices

5.1 Introduction

Deep learning (DL) has enabled applications that require complex reasoning about the raw sensor data stemming from the proliferous Internet of Things (IoT) [LOD18b]. Traditionally, to account for the memory and computation constraints of edge devices, resource-intensive tasks such as deep learning (DL) inferences have been offloaded to cloud-based resources [OSS20, WCY18, WZB18]. With the increasing demands for better application latency and privacy, recent efforts have pushed such inference tasks closer to the edge devices where the data are collected. Thanks to emerging on-device technology with low-power processors and accelerators, edge devices are now capable of performing advanced computation tasks such as moderate deep learning tasks [WBC19, XDH18, OG18, LZC18]. Unfortunately, larger computation tasks entail a larger attack surface, and securing DL inference models from leaking private information in untrusted operating systems is a daunting challenge even for cloud environments.

Recent advances in trusted execution environments (TEEs) have provided an opportunity to revisit the security mechanisms for protecting computation and private user information. A TEE is a secure area in the processor that enables the mapping of sensitive data and associated computation into trusted memory before interacting with an untrusted operating system. TEEs such as Intel SGX [Inta] and AMD SEV [AMD] are enabled in the latest

Intel and AMD CPUs to provide secure computations for applications. ARM CPUs—which are ubiquitous in mobile and IoT edge devices—are enabled with ARM TrustZone [ARMa] TEEs that can map sensor peripherals to trusted memory. These hardware-based solutions enable a new opportunity to build trusted DL inference systems from the digitization of sensor values to the output DL inference.

Indeed, we are not alone in recognizing this promise to secure DL inferences, and there are concurrent efforts to protect offloaded inference tasks using TEEs [LLP19, OSF16b, GHZ18, TGS18]. However, these solutions are specifically designed for cloud environments and are not viable solutions for low-power, low-memory edge devices. Moreover, these solutions also fail to interface the TEEs with available accelerators securely (e.g., GPUs). Accordingly, they suffer from fundamental issues in achieving security and performance:

- Large secure memory usage and TCB: To protect the entire DL inference, we can simply try to place everything—framework, data, and model, into the TEE. Unfortunately, this is not a feasible solution due to a large trusted computing base (TCB) and inherent memory constraint [LLP19]. Current efforts for securing DL on the cloud [LLP19, OSF16b, GHZ18, TGS18] leverage Intel SGX with 128MB of total secure memory, which would be a significant amount of secure memory for edge devices (as in §5.2.2). Moreover, as typical edge deep learning frameworks have hundreds of thousands of lines of code, placing such a large codebase within a TEE contradicts the notion of minimizing the TCB and attack surface.
- Limited inference performance: TEEs have yet to be interfaced with on-device accelerators to enable secure *and* performant DL inference, completely neutralizing the benefits of the accelerators. Our experiments demonstrate that on-device accelerators can improve the inference latency by more than two orders of magnitude.

In this paper, we address these limitations by presenting SecDeep. SecDeep aims to protect data confidentiality during the entire end-to-end DL inference on edge devices, from the digitization of the raw sensor data until obtaining the inference results from accelerators.

We further extend the guarantees toward the integrity of the inference model and the underlying computation. SecDeep demonstrates that we can adequately secure GPU-accelerated DL inference frameworks on edge devices.

In designing SecDeep, there are several technical challenges to address. (1) First, we must minimize the TCB and the secure memory of large DL model inference frameworks while not significantly degrading the performance or utility. This first challenge implies that we need to identify a portion of the DL model inference frameworks that can reside outside of the TEE while being able to leverage available accelerators. (2) Second, if we are to run a portion of the DL model inference framework outside of the TEE, we must ensure the integrity of the associated code. (3) Finally, we must further ensure the confidentiality of any data that needs to be passed this code residing outside of the TEE. We tackle these challenges as follows:

- DL model computation split: to reduce the TCB size of the DL inference model, we split the model computation base into a *confidential computing base* and a *nonconfidential computing base*. The confidential computing base is comprised of any code that interacts with the plaintext input data, e.g., matrix multiplication in a convolutional layer of a deep learning model. The nonconfidential computing base is comprised of any code that does not need to interact with the plaintext tensor data, e.g., GPU configuration code.
- Runtime integrity checker inside TEE: to verify any parts of the model and related code running in the untrusted environment, we utilize code signing. At compilation time, our enhanced compiler will sign the nonconfidential computing base with cryptographic hashing. When the model is loaded, the integrity checker sanitizes any access request to the nonconfidential computing base to preserve integrity.
- Secure runtime data management: although the nonconfidential code and data are exposed to the untrusted OS, the data are adequately encrypted when they go outside of the TEE. SecDeep adopts similar techniques from [YZ19, ANS14] in which the kernel page table is treated as a user-space process page table to protect the code's integrity. SecDeep uses

Format-Preserving Encryption (FPE) to hide the values of the data.

We implement SecDeep prototype using an edge device enabled with ARM TrustZone TEE [ARMa] and interface it with an embedded GPU through the ARM NN SDK [ARMk]. We minimize the TCB of the ARM NN framework from 232K sLoC to 1K sLoC. Our evaluation shows that SecDeep achieves 16× to 127× better inference latency than CPU-only solutions for a representative set of on-device deep learning models (SqueezeNet [IMA16], MobileNet V1 [HZC17], MobileNet V2 [SHZ18], GoogleNet [SWY15], YoloTiny [RF16], ResNet50 [HZR15], and Inception [IS15]) while ensuring confidentiality and integrity of the data and code.

We make the following contributions in this chapter.

- We present SecDeep, the first system to our knowledge that provides secure and private deep learning model inference framework for edge devices. (§5.3)
- We develop a technique for edge devices to minimize the TCB via proper DL computation split and ensure the integrity of the code and the associated split through secure bootup. (§5.4)
- We show how SecDeep can maintain the performance of DL inference on the edge by securely interfacing on-device accelerators with TEEs. (§5.5)
- We implement SecDeep for ARM-enabled edge devices by interfacing the ARM TrustZone TEE with the ARM NN deep learning computation framework. We minimize the TCB of the ARM NN from 232K sLoC to ˜1K sLoC. (§5.6)
- We evaluate SecDeep on a representative set of deep learning inference models and demonstrate that deep learning on the edge can be secure *and* performant. (§5.7)

5.2 Background and Motivation

This section will discuss the background and motivation of our SecDeep framework.

5.2.1 DL Inference Framework on the Edge

Due to the high demand for edge computing needs resulting from data privacy, network latency, and network bandwidth concerns, the most popular deep learning frameworks provide support to run deep learning model inferences at the edge directly from the raw input sensor data. For example, Caffe2 [Caf], PyTorch [PyT], and TensorFlow Lite [Ten] provide developers with an efficient way to perform deep learning inference at the edge before sending them to the cloud. More generic platforms, such as ARM NN [ARMk], have emerged from hardware vendors, allowing the aforementioned deep learning frameworks to target common platforms with the same underlying computation base. ARM NN currently supports both Caffe and Tensorflow[1] for a more generic optimized performance on ARM devices. These frameworks expose a common design: a framework composed of a neural network parser along with computation libraries that are optimized for specific operating systems such as Android [And] or iOS [Appb].

Neural network parser. Most on-device deep learning frameworks consist of a neural network parser. Given a model that is generated using a supported framework such as Caffe or Tensorflow, the parser compiles the model into a graph representation that interfaces with the underlying computation libraries. This graph is constructed in a way that can be optimized for the backend execution.

Computation optimization. On-device deep learning frameworks also typically have backend execution frameworks to optimally execute the associated neural network graph representations depending on the available computation resources. For example, in ARM NN [ARMk], if multiple backends are available simultaneously, the graph will be established such that multi-computing can be achieved efficiently. The resource optimizer validates the correctness of the input model and optimizes the resources needed for the model. It can remove redundant operations, reshape the data if necessary, reorder the graph constructed

[1]Support for other frameworks are currently under development.

by the parser, and determine which acceleration methods to use.

Given these computation frameworks for deep learning inference on the edge, we now discuss how to secure the computation on these devices.

5.2.2 Secure Computation on the Edge

Although software-based cryptographic mechanisms allow for the protection and sanitization of this digitized data on edge devices, the data can still be leaked either prior to being encrypted or at the time of computation when the data are decrypted [YZ19]. To protect the computation, most mainstream CPU manufacturers provide hardware-assisted TEEs. For example, Intel provides extension guarding instructions (Intel SGX [Inta]) to establish per-application TEE, and AMD also provides secure execution environments (Secure Encrypted Virtualization [AMD]) to protect the application's data confidentiality. However, the most popular trusted execution environment that provides access protection to peripherals equipped on edge devices is ARM TrustZone [ARMa].

Trusted execution environments. Trusted execution environments (TEEs) are hardware protection mechanisms that isolate the memory into secure memory and unsecure memory. The secure memory can only be accessed by privileged code running inside the TEE while any code can implicitly access the unsecure memory. In ARM TrustZone, the secure memory code resides in secure memory—referred to as the *Secure World* (SW), whose high privilege is designated by setting a special ARM instruction *SMC*. The unsecure code resides in unsecure memory—referred to as the *Normal World* (NW). The context switch between SW and NW is done through a Secure Monitor (SM) [ARMh].

Although one may trivially assume that computation for a large model such as a deep learning model could be placed within the secure world of a TEE, we discuss the several reasons why this is a strawman solution.

Strawman solution for secure model inference. A naive approach to provide such

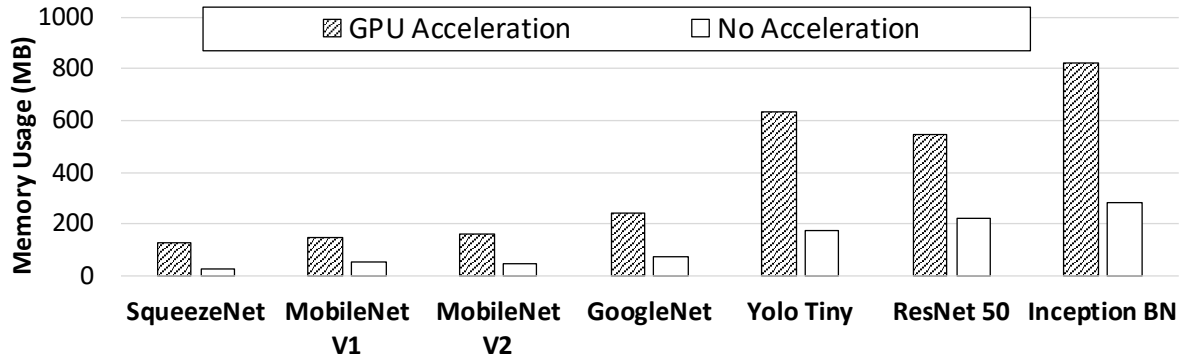


Figure 5.1: Maximum memory consumption of different Caffe models on an ARM device using ARM NN.

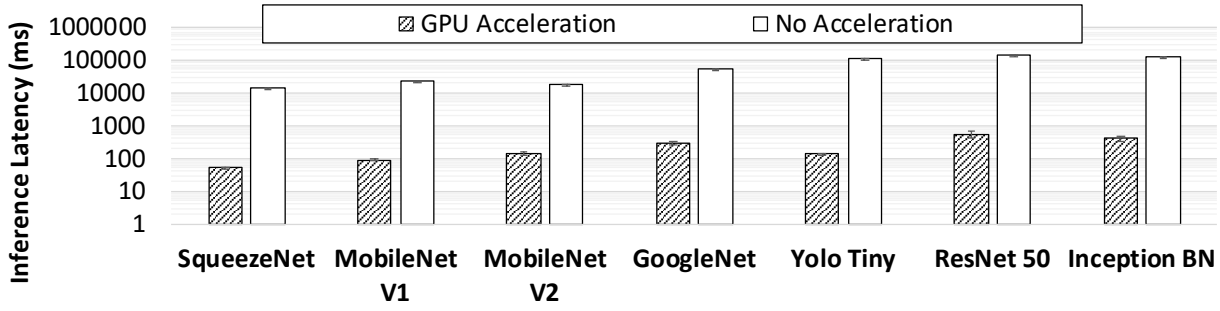


Figure 5.2: The average inference latency (log scale) of different Caffe Models on an ARM device using ARM NN.

protections such as data confidentiality and code integrity is to put the entire deep learning framework inside the trusted execution environment. However, this approach has two critical flaws: 1) excessive secure memory usage is required (e.g., up to 821MB), 2) a large trusted computing base will increase the attack surface of the application.

Mobile and IoT devices are typically memory-constrained on the order of a few Gigabytes. Secure memory is generally limited to tens of Megabytes per application as the initial

allocation is deducted from the normal operating system's unsecure memory allocation. For example, in ARM TrustZone, the memory configuration is done at boot time, so the more secure memory, the less unsecure memory for an already resource-constrained device. In this paper, we follow the lead of prior works [AS19] and allocate only tens of Megabytes of secure memory to provide the same level of protections—limiting the feasibility of running large deep learning inference models within secure memory. For instance, Figure 5.1 shows the maximum memory consumption of running different Caffe models on an ARM device using ARM NN. Without any acceleration, the smallest model (SqueezeNet) consumes 28 MB of memory. With accelerators enabled, the memory consumption for a model inference could shoot up to 821MB. Of course, without accelerators, the performance of on-device deep learning inferencing applications is degraded by several orders of magnitude—as depicted in Figure 5.2.

Moreover, Table 5.1a shows that edge deep learning frameworks could bring hundreds of thousand lines of code. If the whole framework is placed inside TEE, the total trusted computing base (TCB) size will be tremendously large and introduce unnecessary attack surfaces. Upon analysis of the ARM NN deep learning inference framework as shown in Table 5.1b, we found that, without acceleration, typically 90.2% of the framework code is for tensor computation preparation or for performance optimization, and only about 9.8% is dedicated to mathematical tensor computation that changes the values of the input tensor data and yields the values of the output tensors. With acceleration enabled, the computation and preparation code makes up about 99% of the code. Furthermore, the output of the computation preparation and the performance optimization code only depends on the size of the input rather than the values of the input. Thus, our design aims to leverage the hardware-assisted execution environment to reduce the TCB size while still sanitizing the access to accelerators with high security level by only putting the tensor value computation code inside the TEE while leaving the tensor preparation code, the resource configuration

[2]including GPU acceleration backends in ARM ComputeLibrary

Table 5.1: Lines of code for different deep learning frameworks on edge as well as a breakdown for the ARM NN deep learning inference framework. The table highlights the small percentage of code dedicated to the privacy-sensitive tensor computation.

Framework Name	sLoC
TensorFlow Lite [Ten]	404K
Caffe2 [Caf]	368K
PyTorch [PyT]	191K
Deeplearning4j [DL4]	690K
ARM NN[2] [ARMk]	232K

(a) Lines of code for different deep learning frameworks on edge.

ARM NN No Accel.	sLoC	Pct.
Tensor Preparation	109.9K	90.2%
Tensor Computation	11.95K	9.8%

ARM NN GPU Accel.	sLoC	Pct.
Tensor Preparation	232K	99.95%
Tensor Computation	114	0.05%

(b) Lines of code breakdown for ARM NN deep learning inference framework.

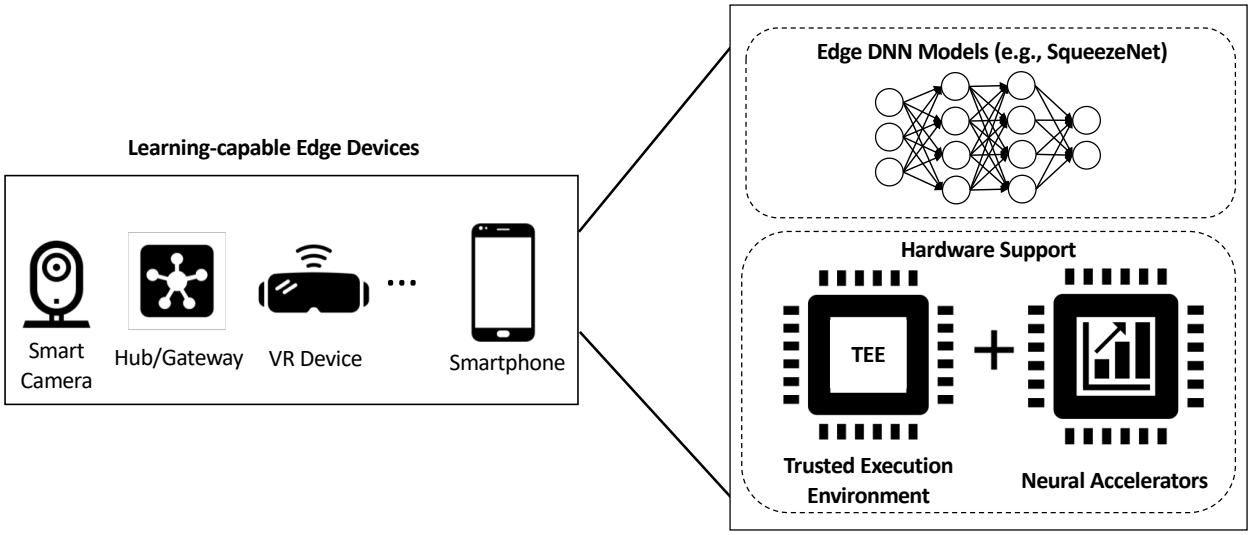


Figure 5.3: The system model of SecDeep. We aim to secure DL model inferencing on IoT edge devices that are enabled with a TEE and possibly on-device accelerators.

code, and the optimization code outside of TEE.

5.3 Overview

In this section, we describe the scope and workflow of SecDeep before discussing the main technical challenges and insights.

5.3.1 Problem Scope

System Model: SecDeep is a general secure framework residing on the edge of IoT networks to protect deep learning inference tasks, as shown in Figure 5.3. We assume the devices are enabled with hardware support for TEEs as well as on-device neural accelerators. We further assume that the DL model provider does not care about the confidentiality of the DL model and its underlying computation framework, e.g., the model may be a publicly available DL model such as SqueezeNet [IMA16] with an open-source computation framework

such as ARM NN [ARMk]. To verify the integrity of the DL models in SecDeep, we expect the provider will supply the hashes of the authentic models using cryptographic hash functions (e.g., SHA-3 [Dwo15] and BLAKE3 [JZ]). During inference tasks, SecDeep is designed to achieve (1) user data integrity and confidentiality, (2) the integrity of deployed DL models, and (3) the integrity of supporting codebase, e.g., TensorFlow Lite [Ten], ARM NN [ARMk].

We envision IoT device vendors and IoT cloud operators being early adopters of such a framework, given the supporting evidence that inference tasks are being pushed closer to the edge. As such, we consider two system scenarios to deploy SecDeep. (1) In the first scenario, an IoT cloud backend needs the inference information from a mobile or IoT device on the edge. The backend sends the request to the edge devices, and the edge device only returns the final inference output from SecDeep instead of the raw, potentially large sensor information to preserve user privacy and reduce network latency. (2) The second system scenario is a mobile or IoT edge device that needs to perform end-to-end local inferencing without needing to access or share any information with the IoT cloud backend. Given this system model, we now consider the threat model for SecDeep.

Threat Model: SecDeep considers a strong adversary that aims to compromise the operating systems in order to intrude, forge, and modify the inference tasks, as well as to steal user data from the non-protected processes. Thus, we cannot trust any part of the software stack—including the OS—that resides outside of a TEE. And although there are many ways to defend against side-channel attacks and Denial-of-Service (DoS) attacks targeting the program and data inside of a TEE, we assume that side-channel attacks, DoS attacks, and cyber-physical attacks on sensors are outside of the paper's scope. Given the system and threat models of SecDeep, we now summarize the goals of its design.

Goals: SecDeep aims to protect *data confidentiality* during inference, starting from the digitization of the raw sensor data until obtaining the inferenced results. This protection implies that the confidentiality of any intermediate, generated metadata will also be protected.

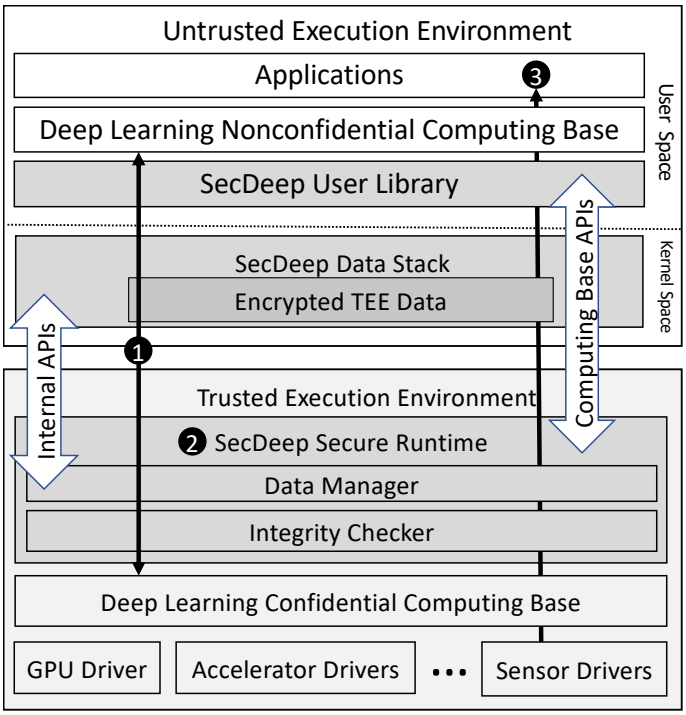


Figure 5.4: The architecture overview of SecDeep. The Dark shaded areas are the components of SecDeep framework and the light shaded areas are the TEE.

Further, SecDeep aims to ensure the *integrity* of the inference code and the associated model. Finally, SecDeep aims to utilize a *minimal trusted computing base* size with *reasonable inference latency and energy consumption* while incurring *no inference accuracy loss*. We illustrate how these design goals are achieved by walking through the SecDeep workflow.

5.3.2 SecDeep Workflow Design

Figure 5.4 shows the design architecture of SecDeep. The workflow of SecDeep can be broken down into three steps: ❶ transforming the deep learning inference computation base for trusted execution, ❷ secure, confidential, and performant execution of the deep learning inference model, and ❸ securing the inference result.

DL model computation transformation for trusted execution. To identify which components of the DL inference code and data should reside within the TEE, we first split the deep learning libraries into two parts: a *confidential computing base* that executes in a TEE, and a *nonconfidential computing base* that executes in the untrusted execution environment. Generally, any code that only requires access to tensor metadata, e.g., tensor shapes, rather than the plaintext tensor data, will be designated to the nonconfidential computing base. Otherwise, the code will be designated as confidential. The confidential and nonconfidential computation base are annotated at a functional level with preprocessor directives to enforce this computation split at compile-time (as is done in Figure 5.5a). Given a split computation base, the SecDeep system is initialized through a secure boot that ensures the integrity of the entire SecDeep code base [She15, AFS97, Win08, Win12] as well the confidentiality of the designated code. With a secure boot in place, we can now describe how SecDeep handles the aforementioned split computation base at runtime.

Secure execution. After SecDeep properly loads the framework code, SecDeep starts to load user data and performs the inference tasks in the following steps.

- First, the user data (e.g., sensor data) are securely loaded into the TEE via protected drivers. Thus, the confidentiality and integrity of the data are guaranteed.
- Second, once the data are inside the TEE, SecDeep's *data manager* decides if any nonconfidential code or data needs to be exported to the nonconfidential computing base due to memory footprint limitation. When some data are set to be exported, SecDeep encrypts them inside the TEE to ensure confidentiality.
- Third, to perform the inference, the nonconfidential computing base uses encrypted data with the model parameters for the current neural network layer to configure the tensor information and send it back to the confidential computing base. Inside the TEE, SecDeep then decrypts the data and begins executing the current layer collaboratively using a protected accelerator (e.g., an embedded GPU). After the results have been computed for the current layer, SecDeep will repeat the same procedure until the inference process is

complete.

Secure output: Because the inference process up until the output is secure and confidential, securing the output is trivial, e.g., the results can be signed or encrypted before being sent back to the requester. Therefore, we focus on the challenges and design of the first two components of the SecDeep framework.

5.3.3 Challenges and Key Insights

Given this workflow, we highlight the key design challenges and our associated approach for each.

Challenge 1: Managing the TCB size for the TEE. Performance with limited secure memory is constrained. SecDeep needs to use limited secure memory to provide protections for the input data along with any data generated throughout the inference process while providing a performant deep learning inference framework. SecDeep utilizes Format-Preserving Encryption (FPE) along with an on-demand table to fulfill this requirement, as described in section 5.5.2. Although the on-demand table requires more memory, our experiments show that it significantly reduces the necessity for encryption and decryption of often-used values and, thus, significantly reduces overhead.

Challenge 2: Ensuring code integrity outside of the TEE. Any code cannot be modified by the compromised OS after it is loaded into memory outside of the TEE. SecDeep treats the kernel page table as a user-space process page table. This allows SecDeep to forward every modification from the kernel page table after the system boots up to the TEE to ensure the integrity of the inference code running outside of the TEE as described in section 5.5.1.

Challenge 3: Ensuring data confidentiality outside of the TEE. Because some of the code, i.e., the nonconfidential computing base, will reside outside of the TEE, there is an inherent risk when computing with confidential data. Because this code only requires

information about the *properties* of the confidential data, e.g., data size, the TEE generates a placeholder value for the confidential tensor data. In particular, SecDeep uses format-preserving encryption (FPE) to generate encrypted data with the same size and length as the plaintext tensor data.

5.4 Transforming Inference Computation for Secure Execution

We now describe the design of the first major component of the SecDeep workflow. We describe how the deep learning computation base is split into confidential and nonconfidential code. We then briefly describe how we ensure the integrity of the code and the associated split at bootup.

5.4.1 Splitting Deep Learning Computing Base

As discussed in Section 5.2, placing the entire DL inference computation framework within a TEE is infeasible and only increases the attack surface of the TEE. Thus, given a DL inference computation framework, we need to identify a minimal set of code that needs to be protected inside TEE. In this case, we aim to protect only code that is designated as *confidential*. This code will be annotated at development time so that it can be separated from the nonconfidential code at compile-time and loaded into the TEE.

Deep learning confidential computing base. To minimize the code running inside TEE, we design the confidential computing base to be composed of the deep learning inference computation that requires access to the unencrypted, plaintext values of the tensor data. For example, Figure 5.5a shows a snippet of code for different activation functions for a neural network. The functions require access to the tensor values (Line 10) to calculate the activation output for the next layer of the neural network. The variable `inputTensor` cannot be replaced by any placeholder value without losing the fidelity of the original computation. However, as per Section 5.2, we find that this confidential base typically makes up a very small

```
1.  /* Sample Confidential Computing Base Code */
2.  float[] CPUActivation[CONFIDENTIAL] (inputTensor,
3.                                      tensorInfo,
4.                                      activationFunc,
5.                                      float weight, float bias)
6.  {
7.      int tensorSize = tensorInfo.getTensorSize();
8.      float[tensorSize] results;
9.      for (int i = 0 ; i < tensorSize ; i++) {
10.         float tensor_value = inputTensor.getData(i);
11.         switch(activationFunc) {
12.             case Linear:
13.                 results[i] = weight * tensor_value + bias;
14.                 break;
15.             case SoftReLu:
16.                 results[i] = log(1.0f + exp(tensor_value));
17.                 break;
18.             // Other Activation Function Computations
19.             ...
20.         }
21.     }
22.     return results;
23. }
```

(a) Confidential computing base code.

```
1.  /* Sample Nonconfidential Computing Base Code */
2.  void configureGPUActivationLayer(inputTensor, *layerInfo)
3.  {
4.      // Build config options. e.g. DataType
5.      int dimension = inputTensor.data().dimension();
6.      DataType type = inputTensor.type();
7.      ...
8.
9.      String configured_option = build_options(dimension, type, ...);
10.     String layer_id = "activation_layer_1";
11.     String gpu_kernel = "activation_layer";
12.     // Set the GPU configuration
13.     layerInfo.setGPUConfigure(configured_options, layer_id, gpu_kernel);
14. }
```

(b) Nonconfidential computing base code.

Figure 5.5: Example snippets of confidential and nonconfidential deep learning inference computation code. Confidential code requires access to the plaintext tensor data, while nonconfidential code only requires information about the tensor metadata, e.g., the dimensions.

percentage of the overall computation base. Line 2 also shows an example of how a developer may annotate a function as confidential with a preprocessor directive (`[[CONFIDENTIAL]]`[3]).

[3]This syntax is similar to the syntax used by a previous work that annotated code that can parrot-

Deep learning nonconfidential computing base. Any code that does not require access to the plaintext, unencrypted tensor data is designated as *nonconfidential* and will reside outside of the TEE in the untrusted software stack. For example, Figure 5.5b shows a snippet of code that observes the input tensor shape and configures the GPU accordingly. The only interaction with the confidential variable `inputTensor` involves extracting the variable dimensions and data type on Lines 5 and 6. The GPU configuration call does not require access to the tensor values. Therefore, this code can easily be refactored such that the tensor data is replaced with a placeholder variable that has arbitrary values with the same shape, size, and data type. This maintains the fidelity of the original function and would not compromise the integrity of the overall computation.

Despite this code being designated as nonconfidential, its integrity is still imperative to the overall computation base. We describe how we maintain its integrity in Section 5.5.1. Before doing so, an underlying assumption is that SecDeep's base confidentiality enforcement mechanism, along with the peripherals, has been secured upon booting the system. We describe how a secure path can be established from sensor peripherals to accelerators in the following subsection.

5.4.2 Securing the Path from Sensor Peripherals to Accelerators

SecDeep needs to create a secure path such that the raw sensor data along with any generated, intermediate metadata are protected when interfacing with accelerators. To achieve such protections, SecDeep utilizes the properties of TEE to disable access to the protected sensors and accelerators from the untrusted OS. SecDeep configures the memory-mapped IO addresses of the sensors and accelerators into the secure memory of TEE such that, upon booting up, those TEE-protected memory-mapped IO addresses can only be accessed by the privileged code inside TEE, but not the untrusted OS. For example, in ARM TrustZone, if

transformed (approximated) by a neural network [ESC12].

the memory-mapped IO addresses for the sensors are configured as secure memory addresses before the system boots, any access to those addresses from the untrusted will be trapped to a higher level execution (i.e., Boot Loader Stage 3 (BL3) [ARMf]) through exceptions, and the secure world in the ARM TrustZone is able to decide whether such access requests should be granted or not.

SecDeep furthers leverages secure boot features supported by most edge devices to establish the initial mapping of sensor peripherals within the TEE [She15, AFS97, Win08, Win12]. With secure boot, we can guarantee that the boot configuration, such as the TEE configuration is properly enforced. Moreover, the secure boot also ensures the integrity and confidentiality of any code that is designated as part of the confidential computing base. However, although a TEE along with a secure boot mechanism allows us to provide a secure path from raw sensor data to accelerators, SecDeep's secure runtime execution is incomplete as we still need runtime mechanisms for checking the integrity of the code as well as data management between the confidential and nonconfidential computing bases.

5.5 Secure and Performant Inference Execution with Accelerators

In this section, we describe how the SecDeep secure runtime provides runtime protection for the entire deep learning inference framework. SecDeep collaboratively works with the data stack to serve as a secure storage outside of TEE when any datum is exchanged between the TEE and the untrusted execution environment. The SecDeep secure runtime is comprised of two major components: a runtime integrity checker as well as a data manager. To enable both components, we later detail the secure APIs exposed by SecDeep that facilitate the confidential data exchange between trusted and untrusted computing bases. The corresponding data sanitization of the secure runtime enables SecDeep to securely leverage available accelerators without leaking private data.

5.5.1 Runtime Integrity Checker

To confirm the integrity of the code running in the deep learning nonconfidential computing base as well as the deep learning model, we design a checker located inside of the TEE for verification. The key intuition of the integrity checker's design is the verification of the hash value of both the model and the code in a trusted mode. The integrity checker works together with an enhanced compiler that signs the nonconfidential computing base code at compilation time to make sure the integrity is preserved when loading the code and the model. After the code and the model have been loaded, the integrity checker sanitizes any access request to the memory of the nonconfidential computing base to make sure the untrusted OS cannot modify the code in the nonconfidential computing base after the secure boot. This sanitization is enforced by trapping the modification of the kernel page table into a higher-level model such as BL3 in ARM. We next describe the design details of the integrity checking mechanisms for both the nonconfidential computing base code and the associated DL model.

Nonconfidential computing base code integrity. To detect the code integrity before any code is loaded into the memory of the nonconfidential computing base, we first modify the associated compiler to hash the nonconfidential computing base code running outside of the TEE. To hash the code at compilation time, the compiler identifies what code belongs to the nonconfidential deep learning computing base by excluding any code that has been annotated as confidential (e.g., Figure 5.5a). The extracted nonconfidential computing base is then hashed accordingly at compile time. At runtime, SecDeep's integrity checking service temporarily stores the hash value into the secure memory using any key exchange algorithm (e.g., as Diffie–Hellman algorithm). The integrity checker then allocates memory regions for the nonconfidential computing base. After the code has been loaded, the integrity checker computes the hash of the loaded code and compares it with the hash value supplied at compilation time to verify the code's integrity.

To ensure that the integrity of the loaded code is protected from the modifications by the untrusted OS, we hide the code pages from the OS kernel—as depicted in Figure 5.6. We first configure all kernel page tables to be *read-only* during the aforementioned secure boot [She15, AFS97, Win08, Win12]. At runtime, if the OS needs to modify a kernel page table, e.g., modifying the page table base registers (PTBR) in ARM, such a request will be trapped into the TEE. Within the TEE, SecDeep's integrity checker will ensure that the page table modification will not result in mapping a kernel page table into a nonconfidential computing base memory regions via a table walking attack [JLK14, BWK17].

In this scenario, another potential attack is to swap the page table with a compromised one such that the nonconfidential computing base code will be accessed through the new kernel page table. To protect against such attacks, SecDeep disables the base registers that modify the kernel page table by removing the page swapping instructions and trapping the write instructions into TEE. SecDeep then checks the access to ensure that the new page table will be mapped into sensitive memory regions when a new kernel module has been loaded. Although similar approaches have been used in prior works [ANS14, YZ19], these approaches need additional mechanisms to perform data confidentiality verification at the same time. Because SecDeep has pre-processed the data confidentiality issue, we simplify their approach to obtain better performance with the same level of security.

Finally, SecDeep needs to make sure the exception handler outside of the TEE is not able to make modifications to the kernel page table when an exception has been trapped by a higher privileged code. Similar to its code integrity protection techniques, SecDeep modifies the exception handler such that any exceptions will be trapped and forwarded to the TEE. The TEE will examine the code to ensure that the code does not contain any modifications in the memory regions from either the nonconfidential computing base or the secure buffer. If the exception does not violate the code's integrity check, the exception will be returned back to the untrusted TEE. The saved registers will be restored for further execution. However, if an exception contains any modifications to the sensitive regions, the

exceptions will never be returned and the user will be notified of the malicious behavior.

Deep learning inference model integrity. Although the deep learning inference model will reside in the same untrusted memory as the nonconfidential computing base, the design of the integrity checking mechanism will require a slightly different approach. As per our system model, we assume that the provided model will be stored in the storage media or may be downloaded from the internet. In either case, we assume that we will also be provided a hash of the authentic model using cryptographic hash functions. Thus, SecDeep's integrity checker focuses only on detecting the model's integrity and not the protection of the model before it is loaded in memory. However, if one wants to protect the model from being modified in a future implementation, secure communication can be established with the cloud through a TEE [DJZ15].

Further, since our system model does not require the DL model to be confidential—as well as the aforementioned constraints of secure memory, SecDeep loads the model in the nonconfidential computing base outside of the TEE. Instead of verifying the integrity of the DL model against its hashed value within the TEE, SecDeep utilizes a mechanism provided by the TEE to set up a read-only (nonconfidential) buffer for the non-TEE code—while the TEE code has read and write privileges to the buffer. To ensure a secure buffer design, we take a similar approach as the nonconfidential computing base and hide the memory region from the untrusted OS kernel. In particular, we the mapping from the kernel page table to the buffer region[4].

Given the secure buffer design, SecDeep's integrity checking service computes the hash of the model and passes the signature to the nonconfidential computing base through the read-only buffer. When the model is loaded, the nonconfidential computing base checks whether it has been verified by the integrity checking service.

[4]Another possible design is to use hidden registers that are invisible to the untrusted environment [YZ19]. However, this design may potentially waste registers and, hence, significantly hinder the performance.

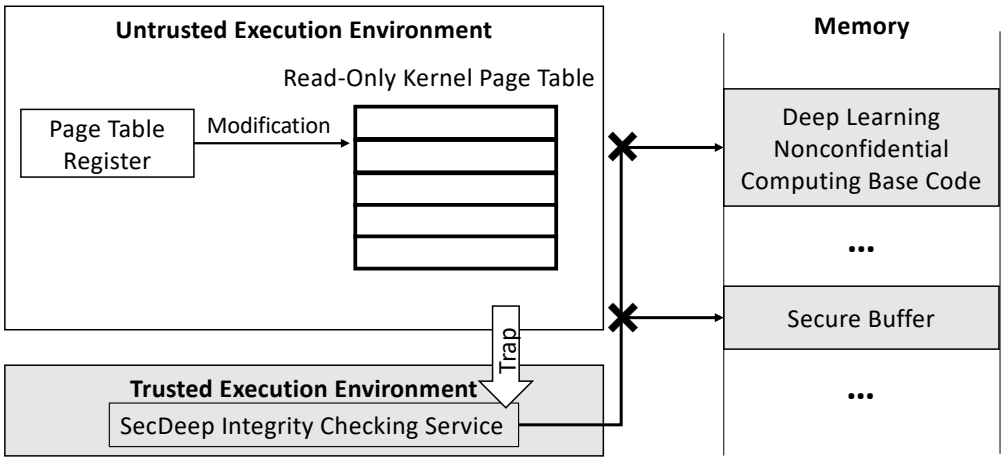


Figure 5.6: When the untrusted OS tries to modify the kernel page table, the request will be trapped to the TEE to make sure the mapping does not map to sensitive memory regions. The shaded areas are the sensitive memory regions.

5.5.2 Data Management

The final component of SecDeep's secure runtime is the data manager, which primarily manages the confidential data communication between the deep learning nonconfidential computing base and the secure memory inside the TEE. In particular, SecDeep's data manager is responsible for providing the associated data sanitization by replacing the raw data with encrypted data that has the same dimensions—as shown in the sample snippet in Figure 5.5b. Further, if the secure memory is running low, the data manager is also responsible for encrypting any data that needs to be stored outside of the TEE in the untrusted data stack. Hence, the data manager design has two requirements. First, the original data's confidentiality cannot be leaked. This means that the attackers cannot reverse engineer any secrets from the supplied encrypted data. The second requirement is that the dimensions of the original data must be the same as the supplied data. To satisfy these needs, SecDeep uses a format-preserving encryption (FPE) [BRR09] function to sanitize the data. FPE encrypts the plaintext value of each basic element of the tensor data while ensuring the

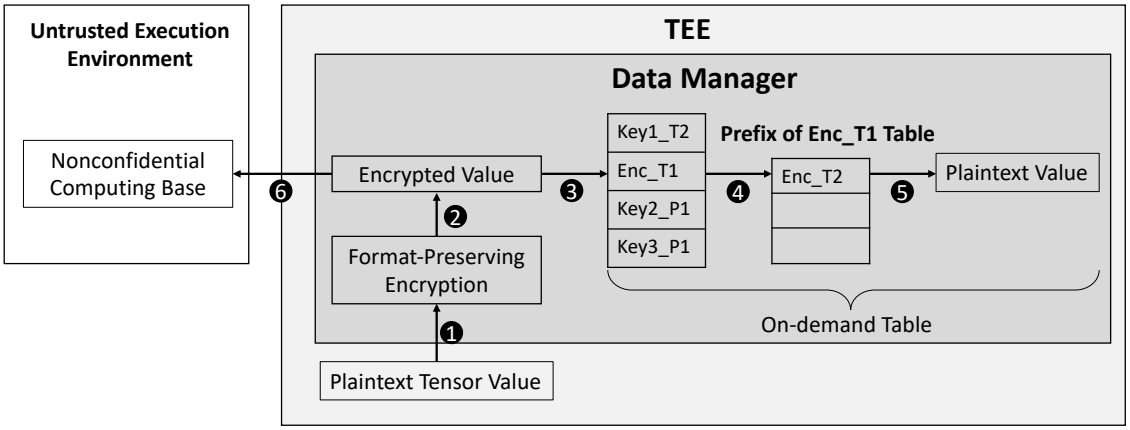


Figure 5.7: The sanitization procedures of the confidential raw data before leaving the TEE. The shaded areas are trusted.

dimensions of the data are retained. For example, if the tensor data represents an array of integers, the FPE encrypts every integer of the array to create an array of encrypted integers. This array has the same length and data size as the plaintext tensor data array.

However, if we simply encrypt and decrypt the data using FPE whenever there is a context exchange between the trusted and untrusted execution environments, this implies that the total computation for every confidential value will be doubled, i.e., the data will need to be encrypted when exiting the TEE and decrypted upon entry. A prior study [LLP19] confirmed that there is indeed a large overhead incurred from such frequent swapping. Hence, to provide efficient swapping, SecDeep utilizes a table inside the TEE to maintain the mapping of encrypted data to the raw data. This method will ensure the "decryption" time to be constant, i.e., it will have a complexity of O(1) by simply referring to the table if the encrypted datum has been created. If an encrypted datum is designated to enter to the TEE from the untrusted execution environment, SecDeep's data manager first refers to the table to retrieve the original plaintext datum. If the datum cannot be found in the table, the data manager performs a decryption method to obtain the original raw datum. This table-mapping approach is summarized in Figure 5.7: when any confidential data needs to

exit the TEE, it is first encrypted using FPE (❶-❷), and then stored in the table using the encrypted data as the key and the plaintext data as the value (❸-❺). Equivalent plaintext values will have the same encrypted values.

On-demand table maintenance. As discussed, the table will inevitably require extra usage of secure memory. There are a couple of optimizations SecDeep adopts to reduce the extra secure memory consumption. First, SecDeep's data manager is to only maintain an entry as long as it is needed. An entry is created when the raw datum needs to leave the TEE. When the raw datum is encrypted and exfiltrated, a count of the datum entry increase by one (starting from zero). When the datum is returned into the TEE to be encrypted, the counter is decreased by one. If the count becomes zero, the entry is removed from the table. Second, if the secure memory is full, SecDeep's data manager unit uses a cache evicting algorithm (e.g., least frequently used) to release more memory and move the encrypted data outside of the TEE. When one layer's computation is finished, all of the intermediate data will be destroyed unless they are needed for the next layer, which will be indicated by the nonconfidential computing base's results.

The traditional implementation of the table data structure usually reserves enough space at initiation time and increases the size by copying the existing table into a larger memory chunk. Due to the constraints of the secure memory, we design an *on-demand* table mechanism to save the mapping of the decrypted data. Inspired by traditional kernel OS design, we design SecDeep's on-demand table to be segmented into small chunks by having a multi-level table (❸-❺). The table entry will only be created when data needs to be stored but does not need to reserve a large space at the initiation time like the traditional table, which is able to ultimately save secure memory usage, and this design also does not require a large consecutive memory if the key-pairs are large.

```
1.  /* Nonconfidential Computing Base */
2.  void prepartion(model)
3.  {
4.      loaded_model = c_model_init(model);
5.      for (auto layer: loaded_model)
6.      {
7.          tensor_info = c_tensor_request(layer);
8.          // Computation such as configurations
9.          ...
10.         c_output_result(layer);
11.     }
12. }
```

(a) The nonconfidential computing base sample code that configures the input tensor for each layer.

```
1.  /* Confidential Computing Base */
2.  void model_verification(model)
3.  {
4.      loaded_model = i_model_load(model);
5.      hash_verify(loaded_model);
6.  }
7.
8.  void data_store(data)
9.  {
10.     encrypted_data = encrypt(data);
11.     i_data_store(encrypted_data);
12. }
```

(b) The confidential computing base sample code to verify the model integrity, and store the necessary data outside of TEE.

Figure 5.8: Sample code of how confidential computing base and nonconfidential computing base interacts with other components using secure APIs.

5.5.3 Confidential Data Exchange through Secure APIs

As depicted in Section 5.3.1, SecDeep's secure APIs enable confidential data exchange between various components of SecDeep, including those residing both inside and outside of the TEE. Table 5.2 summarizes the five secure APIs exposed by SecDeep. The APIs are split into two categories: 1) computing base APIs that enable communication between

Table 5.2: Summary of secure APIs in SecDeep.

Category	Name	Description
Computing Base APIs	`c_tensor_request(layer)`	Request encrypted tensor data from the deep learning confidential computing.
	`c_output_result(layer)`	Send results of the specific layer from the nonconfidential to confidential computing base.
	`c_model_init(model)`	Requests SecDeep secure runtime to load the model.
Internal APIs	`i_data_store(data)`	Store encrypted data from the deep learning confidential computing base.
	`i _model_load(model)`	Send the model to SecDeep secure runtime for integrity verification.

the confidential and nonconfidential computing bases, and 2) the internal APIs that enable communication between SecDeep's data stack and secure runtime.

Computing base APIs. The computing base APIs are used to send tensor information between the confidential computing base and the nonconfidential computing base. For instance, Figure 5.8a provides a sample code snippet for a secure API request from the nonconfidential computing base to the confidential computing base. Once the nonconfidential computing base has configured the input tensor and the resource requests to use a GPU, the code in the nonconfidential computing base will call `c_output_result(layer)` (Line 10) to pass the results to the confidential computing base via SecDeep's secure runtime data manager using a secure buffer.

Internal APIs. The internal APIs are used to exchange the data stored on the data stack in the untrusted execution environment with the data in SecDeep's secure runtime. For example, Figure 5.8b shows a sample code snippet of the integrity checker verifying and signing the deep learning model. The secure API `i_model_load (model)` is called to load the model from the data stack (Line 4). The APIs are designed to use secure monitor code (SMC) to establish a secure buffer such that a malicious OS cannot modify the contents as described in section 5.5.1 by properly hiding the memory region from the kernel page table.

5.6 Implementation

In this section, we discuss how we prototype the design of SecDeep.

Table 5.3: Lines of code implemented for SecDeep.

Model Name	**Repo**	**sLoC**
User Library	ARM NN	694
	ComputeLibrary	126
Integrity Checker	ATF	897
	OP-TEE	69
	Linux	457
	LLVM Compiler	70
Data Manager	OP-TEE	1635
Total		3948

5.6.1 System Setup

We prototype the design of SecDeep using ARM NN with a Caffe deep learning model on a HiKey960 Android reference board [96b]. The Hikey960 board was enabled with an embedded GPU–a Mali G71 MP8 graphics processor, and ARM TrustZone support. For the TEE, we use ARM Trusted Firmware (ATF) [ARMl] with Open Portable Trusted Execution Environment (OP-TEE) [opt] within ARM TrustZone. Because the driver of Mali GPU is not fully open-source, we have to simulate the secure access in our implementation. Although we prototype SecDeep on an ARM A-series dev board, SecDeep is portable to low-end ARM processors such as Cortex-M23 [ARMb] and Cortex-M33 [ARMc] processors with ARM TrustZone.

We implement the user library inside ARM NN and the associated ARM Compute Library [ARMj] to provide the secure APIs. We implement the integrity checker within the OP-TEE and modify the Linux kernel. We also modify the LLVM compiler to perform the confidential code extraction and provide the signatures for the codebase. We imple-

ment the data manager inside OP-TEE. Table 5.3 summarizes the 3.9K lines of code for the implementation.

To evaluate our framework, we built a deep learning inference application using ARM NN to perform image classification using different Caffe models supplied for the Hikey960 reference board. We now detail how each component of the SecDeep is implemented.

5.6.2 Deep Learning Computing Base Split Annotation

We leverage ARM NN and ARM Compute Library with the support of Caffe inference framework to build the confidential computing base and nonconfidential computing base. Upon analysis, we found that most of the ARM NN code is for resource preparation such as building the computation nodes of a special graph or the model parser that loads the model into memory. The ARM NN documentation revealed that the tensor data can only be accessed through the function `Map()` inside the structure `ITensorHandle`. Thus, we were able to script the identification of all the code that uses these functions and analyze whether they are using the sensitive tensor data. We found that the only functions of the ARM NN code that needed the aforementioned confidential designation were the functions associated with the tensor input layers that process all the input data and other tensor metadata such as padding.

5.6.3 Secure Runtime

We build the secure runtime inside ARM TrustZone using both OP-TEE OS and ARM Trusted Firmware (ATF). The OP-TEE OS is responsible for processing the model integrity checking. The ATF traps all of the kernel page table modifications and computes the code hashing. The ATF is also responsible for checking whether the kernel modification will map to a memory region that holds nonconfidential computing base code and data and the secure buffer.

Integrity checker. We implement the runtime integrity checker for both the deep learning model and the code inside the nonconfidential computing base. We implement an MD5 hash mechanism inside OP-TEE to compute whether the model has been tampered with while loading it onto the inference framework.

For the nonconfidential computing base code integrity checker, we first modify the Linux kernel page table entry functions such as clear_pte_bit() and set_pte() so that, every time these functions are called, they will be trapped to BL3 through SMC. When the BL3 handler functions in ATF receive such requests, the BL3 handler functions determine which request they need to handle. If the OS tries to load code into the nonconfidential computing base, the BL3 handler functions use SHA1 to compute the hash of the code and compare it with the compiler-supplied hash. If the kernel page table modification request should not load the code into nonconfidential computing base, the BL3 handler functions walk through all of the page tables to make sure the modification does not map to the nonconfidential computing base nor the secure buffer.

Offline signature generation for nonconfidential computing base code. To generate the signature for the nonconfidential computing-based code, we extract any code that was not designated as confidential using the aforementioned annotations. We modified the LLVM compiler such that during the code emission stage, when the LLVM compiler detects the confidential designation, it computes the hashing for the code block for that function. We use a SHA-1 hashing algorithm to do the hashing computing and verification for the instructions within the designated code blocks. We then sign the hash values and store the signature into the data segment of the program. The program is later loaded into secure memory for verification.

Table 5.4: Benchmark models used for evaluation.

	Model Name	**Model Size**
Light	SqueezeNet	5 MB
	MobileNet V1	17.1 MB
	MobileNet V2	14.4 MB
Med.	GoogleNet	28.3 MB
	Yolo Tiny	65.6 MB
Heavy	ResNet-50	102.5 MB
	Inception BN	137.8 MB

5.6.3.1 Data Manager

We implement the runtime data manager inside OP-TEE. We use Advanced Encryption Standard (AES) with Counter (CTR) mode as the format-preserving encryption (FEP) method because AES-CTR provides the same length of the output as the input. We also implement a two-level table for our on-demand hash table, where the key is the encrypted data and the value is the plaintext data. We maintain the table using the least frequently used (LFU) mechanism. We also evaluate different maximum table size values allowed before adding a new entry in the next section.

5.7 Evaluation

In this section, we will discuss how we evaluated SecDeep with various parameters to show that (1) SecDeep achieves secure DL model inferencing with superb latency (e.g., 172× better than CPU) via secure acceleration, and (2) incurs only a minimal TCB size and computation/energy overhead.

Benchmark Models. Our implementation of SecDeep supports Caffe models. Thus, we

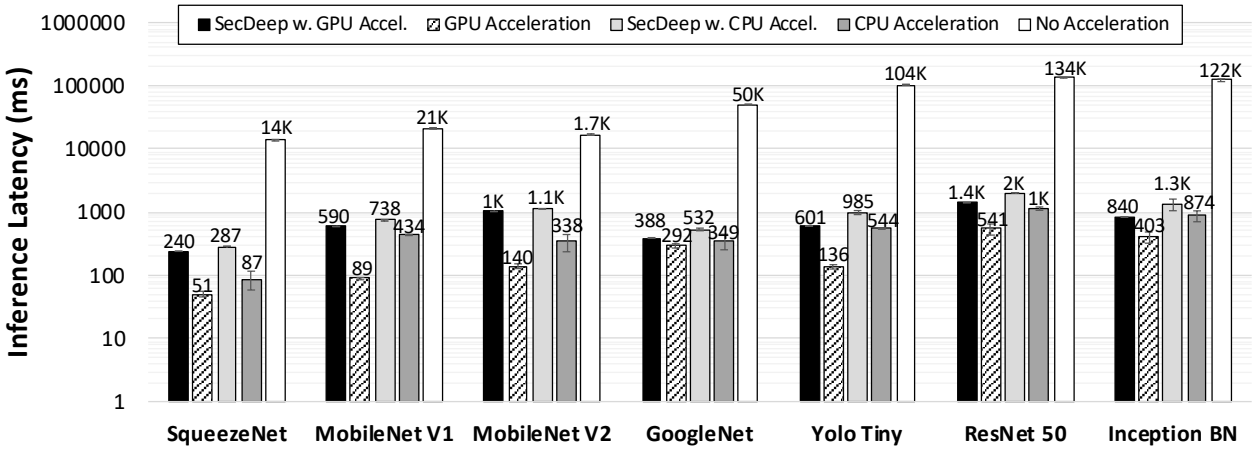


Figure 5.9: The overall latency introduced by SecDeep and the comparison with CPU acceleration and GPU acceleration.

chose to evaluate popular Caffe models with varying size as listed in Table 5.4. We first chose 3 popular, lightweight (less than 20 MB) models: SqueezeNet [IMA16], MobileNet V1 [HZC17], and MobileNet V2 [SHZ18]. We then evaluated 2 mediumweight (less than 100 MB) models: GoogleNet [SWY15] and Yolo Tiny [RF16]. Finally, we used 2 heavyweight (greater than 100 MB) models: ResNet50 [HZR15] and Inception BN [IS15]. These are a representative set of on-device models with varying capabilities.

Trusted Computing Base Size. Based on our implementation, the current TCB size is 1015 sLoC, where 901 sLoC comes from OP-TEE and 114 sLoC comes from ARM NN. Compared with the total computing base size 232K[5] of ARM NN, SecDeep has provided a minimal TCB size. All of the GPU kernel code resides outside of the TEE because the kernel code is only configured by the CPU.

5.7.1 Inference Latency

We run inferencing experiments for all of our benchmark models at least 10 times to measure the average latency using 16MB of secure memory. This is a sufficient amount of memory to support the minimal TCB within the TEE (OP-TEE's kernel is typically a few MB [opt]) as well as an on-demand table with 1M entries.

Overall latency. We run the inference experiments for each model using SecDeep with GPU acceleration, unsecure GPU acceleration, unsecure CPU acceleration, and no acceleration. We compute the average inference latency for each model under each of these scenarios and summarize the results in Figure 5.9. Our experiment shows that, although SecDeep is slower than unsecure GPU acceleration, it is still comparable to unsecure CPU acceleration—it is even faster than unsecure CPU acceleration for the Inception BN model. Most importantly, SecDeep is significantly faster than the case where no acceleration is enabled. This result also shows that the inference latency is not fully proportional to the size of the model. For example, MobileNet V2 is only about one-fifth of Yolo Tiny's model size, but MobileNet V2's inference latency is slower than Yolo Tiny. This is because MobileNet V2 generates more intermediate results and the table hit ratio is lower, resulting in more decryption computations.

Acceleration ratio. In terms of acceleration ratio, Figure 5.10 shows that, for the best case (ResNet-50), SecDeep is able to accelerate up to 172 times more than the case with no acceleration. Even in the worst case (MobileNet V2), SecDeep is able to accelerate 16 times faster than when no acceleration is enabled. Our results have shown that using SecDeep can achieve both acceptable latency and enable secure protection.

Overhead breakdown. We further break down the overhead of SecDeep introduced for different layers of a deep learning model. As shown in Figure 5.11, we accounted for the overhead in each of SqueezeNet's layers. The results show that the overhead of SecDeep

[5]This only counts the GPU acceleration code.

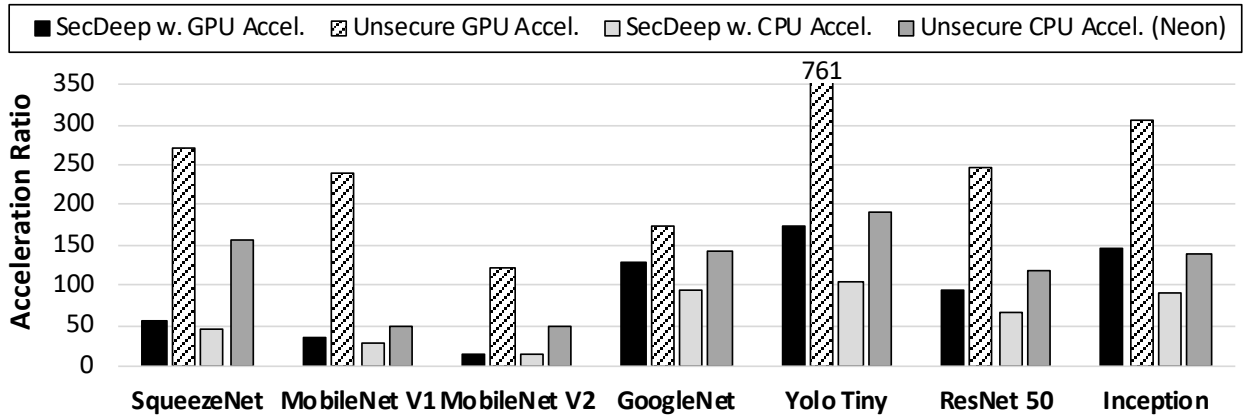


Figure 5.10: The acceleration ratio of each acceleration methods and SecDeep in respect of execution the model without acceleration

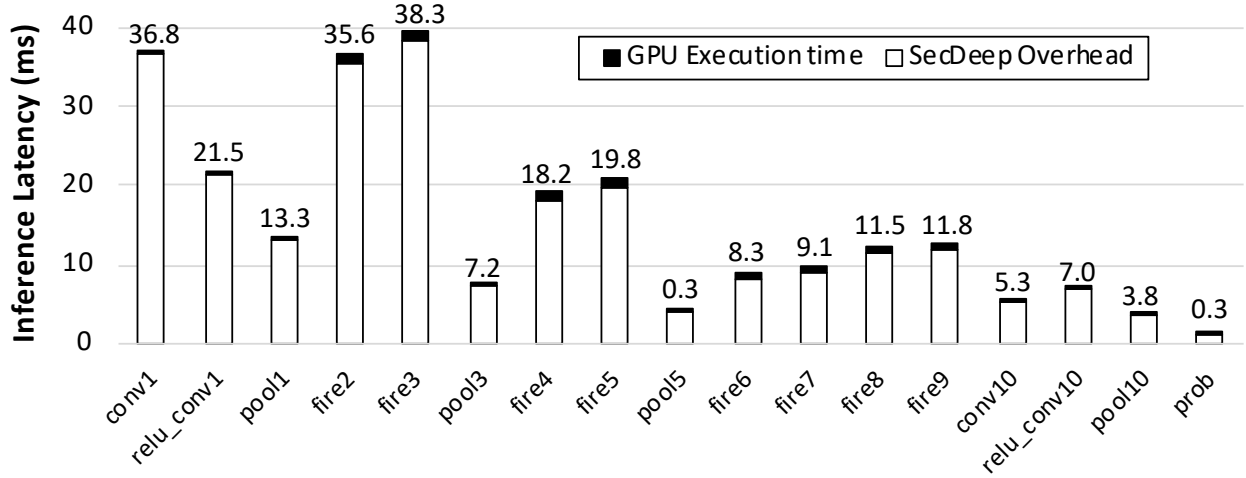


Figure 5.11: The SecDeep overhead breakdown for each layer of SqueezeNet.

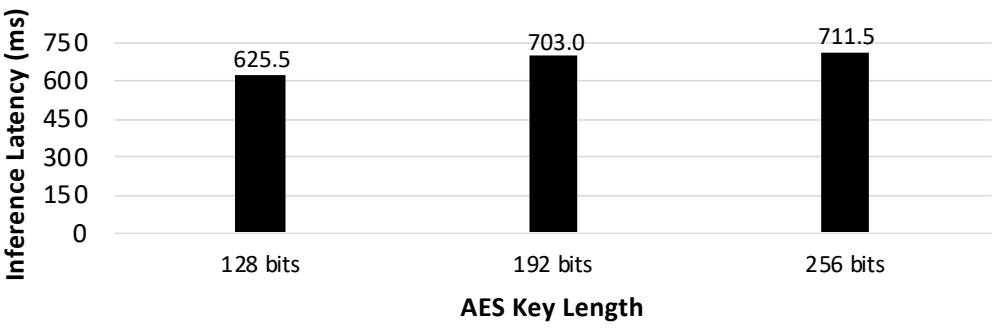


Figure 5.12: SecDeep latency of different AES key lengths for MobileNet V1.

mainly comes from the TrustZone execution, i.e., the world context switch time and the encryption. Furthermore, the first few layers have higher overhead than the last several layers. This discrepancy is due to the fact that the last few layers generate less intermediate data and because the cached table hit rate is high. Further, the overhead of SecDeep is not completely proportional to the size of the tensor input of each layer. For example, if the input tensor size of `fire4` and the input tensor size of `fire3` are both $55 \times 55 \times 128$, but the latency of `fire4` is significantly smaller than the latency of `fire3`. These results also demonstrate that future optimizations could focus on how to store the values of the encrypted data in a table such that the table hit rate can be high enough to benefit the overall performance.

Varying the encryption key length for FPE. Our experiments use AES with CTR mode as the format-preserving encryption method. Although our key length is 128-bits, we run the inference experiment with all three different sizes of AES key lengths, i.e., 128 bits, 192 bits and 256 bits, for MobileNet V1. Our results—summarized in Figure 5.12—have shown that even if we use the highest AES security standard with a 256-bit key, the overall latency has only increased 13.7% with the lowest AES security standard that uses 128-bit keys. Our SecDeep design has validated that the data confidentiality can be very robust without sacrificing much computation latency.

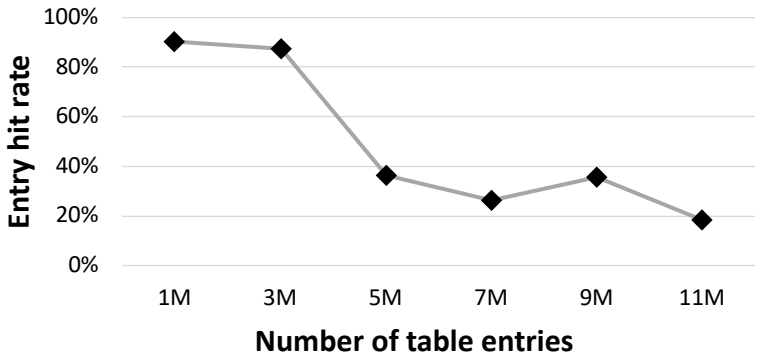


Figure 5.13: The hit rate of table contents in varied table entries for MobileNet V2.

5.7.2 Table size options.

Although the table is dynamically created, we observed that some of the table contents are hardly hit in the future. Hence, they waste the already-limited secure memory. In this experiment, we test the hit rate for tables with varying upper limits for the table size as shown in Figure 5.13. The results show that the hit rate significantly reduces after 3 million entries. Hence, for the best performance, 3 million entries should be used to ensure that the memory will not be significantly wasted while providing good performance.

5.7.3 Model Loading Latency

We evaluate the model loading time using and without using SecDeep. The model loading time refers to loading the model into memory and converting the model to a graph that ARM NN understands for further execution. We implement the MD5 hashing mechanism to check whether the model has been tampered with before loading it to the nonconfidential computing base. Our results in figure 5.14 show that the hash checking has introduced marginal overhead when loading the model in comparison to the unsecure GPU acceleration's model loading time. For the lightweight and mediumweight models, the model loading overhead of SecDeep is less than 4%, while the model loading overhead for the heavyweight

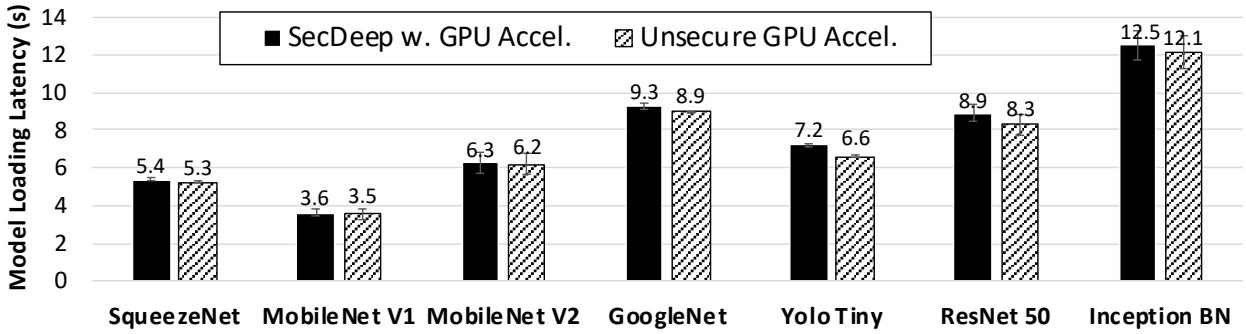


Figure 5.14: Comparing model loading latencywith and without SecDeep's integrity checker on a GPU-enabled device.

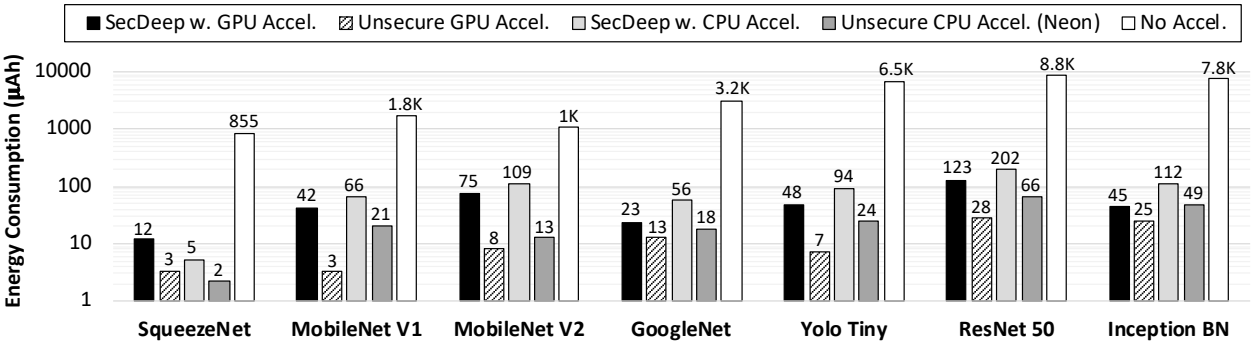


Figure 5.15: Comparing energy consumption when inferencing with and without SecDeep.

models is less than 7.8%.

5.7.4 Energy Consumption

The last evaluation we characterized is power consumption. Because these are mobile and IoT devices, energy management is a critical concern. To measure the power consumption of the inference process under the aforementioned scenarios, we connect the Monsoon power monitor [Mon] to our HiKey960 reference board. Figure 5.15 summarizes the results for each benchmark model. Our experiments show that although GPU is more power-hungry,

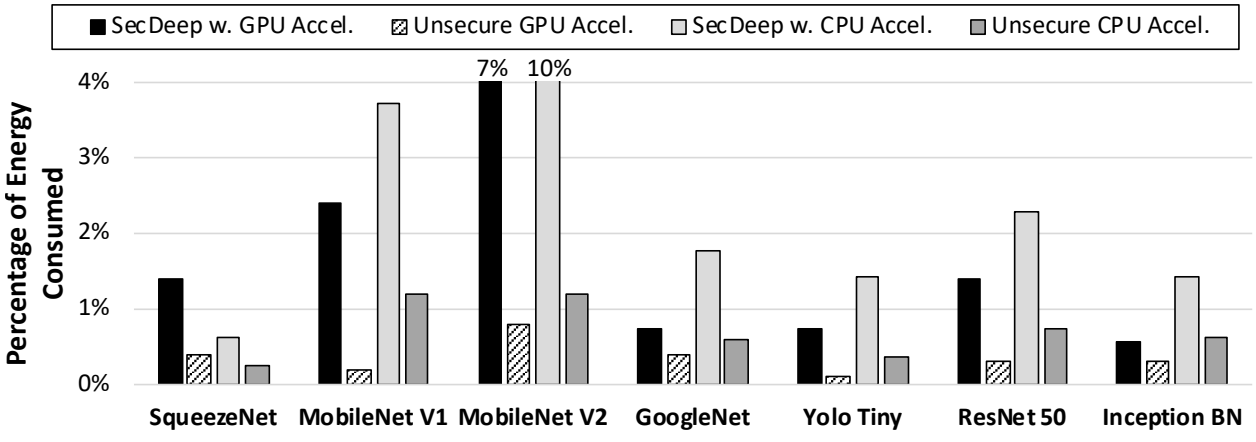


Figure 5.16: The overhead percentage of the energy consumption with and without SecDeep, CPU acceleration, and GPU acceleration.

the overall latency of GPU acceleration is much smaller. Hence, less power consumption is achieved when using GPU acceleration. Although SecDeep consumes slightly more energy than unsecure CPU accelerations in most cases, it still consumes significantly less energy than without using acceleration as illustrated in Figure 5.16. Our experiments demonstrate that SecDeep can achieve acceptable performance and energy consumption.

5.8 Discussion and Future Work

In this section, we will analyze the security guarantees, discuss the limitations of SecDeep as well as the future research directions.

5.8.1 Security Analysis

SecDeep is able to protect the attacks from stealing raw sensor data to infer private user information from malicious applications and operating systems. Specifically speaking, when an application needs to get the inference results from the sensor data, it is only fed with

the final results from a deep learning rather than the raw data. In this situation, the risk of leaking private user data is mitigated. Moreover, when the OS is malicious, SecDeep utilizes the characteristics of hardware TEE to protect the sensors such that the OS cannot obtain the sensor data.

Since SecDeep does not trust the operating system, any attacks that can maliciously obtain privileged access to the OS (e.g., CVE-2018-8781, CVE-2018-14634,CVE-2019-8635, CVE-2019-17666, CVE-2018-1087, CVE-2018-1068, CVE-2019-0808, and CVE-2019-1159) are diminished in terms of the attacking deep learning execution on the edge—where both the data confidentiality and accuracy can be guaranteed from the sensor digitization to the DL results via TEEs.

5.8.2 Limitations

One of the assumptions SecDeep makes is that the TEE is always secure and trusted. Possible side-channel vulnerabilities for TEEs may hinder the assumptions of SecDeep. For example, the CPU cache attacks towards Intel SGX [KHF19, LSG18, HCP17, GES17, SGF17, BMD17] and ARM TrustZone [KHF19, LSG18, AL17, CZK18, BLL18, ZSS16] can diminish the protections of SecDeep. Although side-channel attacks are out of this paper's scope, extra protection mechanisms [OTK18, CZR17, KKS19, KKS18] could be implemented to reduce the effects from side-channel attacks.

5.8.3 Future Work

Training at the Edge. Although SecDeep focuses on deep learning inference at the edge, another direction SecDeep is targeting is secure training on the edge [ZLD20]. Given that sensors are increasingly deployed at the edge, models need to be updated frequently to improve accuracy.

Pruning the Model. To increase performance, future works can focus on optimizing deep

learning models to adapt to the associated hardware, e.g., such as deep compression [HMD15] or AutoML techniques [HLL18]. However, although the model size is reduced, such pruning does not solve the biggest constraints of running a secure deep learning inference framework on the edge—limited secure memory.

Generalizing to cloud settings. Although SecDeep mainly targets the mobile and IoT edge devices, the design concepts of SecDeep can be extended to cloud settings. Our framework could theoretically leverage accelerators in the cloud as long as the cloud service providers are able to provide TEEs that can restrict access to the accelerators–as is done in ARM TrustZone.

Autonomous confidential annotation. Ideally, developers would employ annotations for functions as confidential or nonconfidential from the start. However, as was done in this paper, we envision existing frameworks would have to be retroactively annotated. Future work can focus on autonomously or semi-autonomously annotating the code that requires access to plaintext tensor values. An analogous semi-autonomous solution has been presented in the context of annotating secure labels for code that interacts with *secrets* [WRP19].

5.9 Related Work

In this section, we will discuss the related works of SecDeep.

Secure machine learning. Previous works have explored securing deep learning frameworks algorithmically when the machine learning models are offloaded to cloud environments. Occlumency [LLP19] leverages Intel SGX to secure deep learning inferencing in cloud environments to preserve data privacy without trusting the cloud service provider. Similarly, Ohrimenko et al. [OSF16b] use trusted enclaves to collect sensitive data from distributed clients and run oblivious machine learning training processes. Chiron [HSS18] also leverages SGX to host microservices for machine learning (ML-as-a-Service) in the cloud to both train and deploy machine learning models in a light and easy fashion. Slalom [TB19] uses

Intel SGX by partially offloading linear layers of DNNs to untrusted CPUs to obtain high performance without sacrificing the data privacy. DeepEnclave [GHZ18] uses cloud-assisted SGX for inferencing to overcome the shortage of secure memory on the edge and, hence, reduce the secure memory consumption on the edge devices. Privado [TGS18] uses Intel SGX to load different models into an enclave and defend the side-channel attack through access patterns. However, unlike SecDeep, the above works do not provide a secure path to use the available accelerators such as GPUs for the inference. They were also designed for cloud environments that are less resource-constrained.

TrustZone applications on the edge. ARM TrustZone has been widely adopted in different designs to achieve the security requirement of an app or a system in the research field. Ginseng [YZ19] uses secure registers to hide sensitive variables. However, it's NP computation complexity is problematic for our design that has many sensitive variables. TZ-RKP [ANS14] uses ARM TrustZone to monitor whether the OS is compromised. However, TZ-RKP is unable to protect the integrity of the applications running on the OS. Liu et al. [LSW12b], AdAttester [LLC15], TLR [SRS14], SeCloak [LSD18b], [BKL16b], TrustOtp [SSW15a], VirtSense [LS18] split applications into sensitive code and insensitive code. PROTC [LS17], uses ARM TrustZone to sanitize drone control commands running inside the TEE. These approaches only trust the sensitive code and do not provide protections for the insensitive code. TrustShadow [GLX17] runs an entire secure application inside a TEE, but is not feasible for the large DL models we are considering.

5.10 Conclusion

We propose the SecDeep DL model computation framework that securely uses available accelerators to provide performant on-device inferencing on the edge.SecDeep leverages the benefits of TEEs to achieve both high performance and a small TCB size with limited secure memory. We prototype the design SecDeep on a HiKey 960 dev board using ARM

TrustZone technology, and our experiments show that SecDeep can achieve up to 172X model inference acceleration while using only 16MB of secure memory and while minimizing the TCB by 92.4%.

Part III

Providing Remedial Actions for Actuation Conflicts

CHAPTER 6

RemedIoT: Remedial Actions for Internet-of-Things Conflicts

6.1 Introduction

The proliferation of the Internet-of-Things (IoT) has enabled sensors and actuators across several facets of society for the purpose of automating and optimizing our daily tasks. As humans settle into smart environments, the collection of data and automation of associated processes draw concerns of safety, security, and privacy [Sta14, LYZ17, ABC12]. IoT programming platforms such as IFTTT [SAB17a], Zapier [Zap], and Samsung SmartThings [Sam] have eased the configuration process of smart environment devices, but they placed the burden of addressing those aforementioned concerns on both the IoT application developers and the application users who are configuring the environment.

Previous works have attempted to alleviate this burden by checking potential IoT configurations against *policies*, which are rule specifications of a particular system with respect to some objective function such as safety or utility. An example policy may aim at preventing racing conditions between two applications that may turn a device on or off. Policy-based mechanisms have been proposed to enforce policies that avoid unsafe or unsecure states for smart environments. Solutions such as IoTGuard [CTMa] will raise an alert to the user if a particular policy is violated and block the associated action. The general focus of these mechanisms has been to determine inter or intra-app safety, security, and privacy conflicts for either explicitly defined relationships amongst devices [LKL15b, DH18, FPR16] or for im-

plicit relationships that are learned based on the device behavior [MMH17, BVN15, HOW13].

Although these solutions provide promising approaches to conflict detection, they typically do not have a means of taking the right course of action once a conflict is detected–they usually only block the conflicting actions [CFP18]. Several of these conflicts may be in the context of a safety-critical application and require immediate remediation, i.e., an automated solution to resolve the conflict without deadlocking any critical applications. For example, an IoT application may specify that if the concentration of CO_2 is high, an immediate ventilation action needs to be performed by opening the windows. Meanwhile, another thermostat application ensures that the windows remain closed in cold weather to save energy. If the indoor CO_2 accumulates to a dangerous level in winter, a conflict may arise. The current state of the art conflict protection mechanisms, however, may block the safety-critical ventilation actuation. In this case, a *remedial action* would resolve the conflict by finding an alternative or redundant path that implements the same utility, e.g., an HVAC system would be used instead of the smart window for ventilation. However, finding alternative or redundant realizations for commodity IoT utilities can be difficult. Implemented as traditional device drivers, IoT device interfaces are typically *monomorphic* and, as such, cannot be easily realized by alternative means. We, therefore, need a method to provide a polymorphic interaction interface such as polymorphic programming abstractions for those devices that facilitate such remedial actions.

Polymorphic abstractions for sensors and actuators have been employed by IoT macroprogramming frameworks that suggest high-level commands for distributed IoT frameworks [NTG19]. These frameworks rely on the notion that they can specify what should be sensed or actuated as opposed to how sensing or actuation is carried out. Preliminary works show the possibility of raising abstractions via inference graphs for sensors[SSP16], e.g., a "fitness activity" sensor abstraction may be realized by either an inertial movement sensor or a heart rate monitor. However, the same notion has yet to be realized for actuation as it is difficult to reason about the utility of an actuator based on the user's intention.

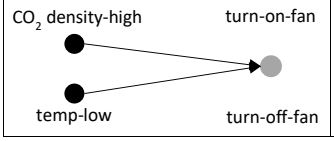
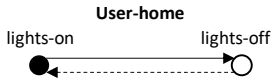
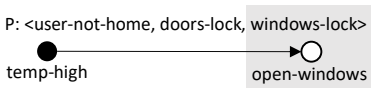
In this chapter, we present RemedIoT, a remedial action engine framework for resolving the conflicts (i.e., policy violations) of IoT-based smart environments. RemedIoT helps the application users remediate conflicts when they configure the smart environment through smart application platforms. We introduce a key component, actuation graphs, that allows raised, polymorphic abstractions of both IoT actuators and sensors. RemedIoT leverages the IoT actuation graphs to provide alternative realizations of sensors and actuators for smart environments. Safety, security, and privacy policies can then be specified with respect to these raised high-level sensor and actuator abstractions. RemedIoT then uses a conflict detector to determine if a given IoT actuation or control command triggers any conflicts. If a conflict is detected, the Remedial Action Engine suggests alternative realizations of a user's intended application that can be installed to avoid such a conflict. Once a remedial action has been selected, the same conflict runtime executes the remedial action scheme.

We evaluate RemedIoT on Samsung SmartThings applications and IFTTT applets by extracting 195 possible automation logics and generating 74 sample applications with 11 policies. For these applications, we utilized state of the art conflict detector approaches to detect 102 possible conflicts. We show that RemedIoT is able to remediate $\sim 80\%$ of the conflicts that would normally be blocked by prior solutions. We show how remedial actions can be optimized against abstraction cost functions such as power consumption and device profiling. We further discuss the efficacy and scalability of RemedIoT in the context of smart city environments.

Contributions. Our contributions are summarized as follows.

- We provide actuation graph abstractions for actuators in IoT smart environments.
- We present RemedIoT, a remedial action engine framework that utilizes alternative realizations of IoT applications to provide remedial actions for a given IoT policy conflict.
- We evaluate RemedIoT on a set of Samsung SmartThings applications and IFTTT

Table 6.1: The two general categories of policies: the policies defined by Celik et al. [CTMa] and user-defined policies.

Types of Policies		
P1: Mutually exclusive states must not exist in the environment.		**P2: User-defined rules.**
Racing Events	Cyclic Events	**E.g.: Doors and windows must be locked is user is not home**
E1: CO_2 density-high -> turn-on-fan E2: temp-low -> turn-off-fan	E1: user-home, lights-on -> lights-off E2: user-home, lights-off -> lights-on	E1: user-away -> user-away-mode-on E2: user-away-mode-on, temp-high -> windows-on
		

applets and show how REMEDIOT can remediate $\sim 80\%$ of conflicts that would normally be blocked by previous solutions.

- We show how remediation can be optimized for providing automated remedial actions based on abstraction metadata.

Our source code and datasets are available online:

`https://github.com/nesl/buildsys-19-code`

6.2 Background

In this section, we provide the preliminary information necessary to understand the rest of this paper. We first discuss the state-of-the-art for IoT event service platforms. We then discuss the state-of-the-art for conflict detection across different IoT services along with their limitations.

6.2.1 IoT Event Service Platforms

The current programming paradigms for commodity IoT smart environments typically facilitate IoT device interaction through a control hub, e.g., Samsung SmartThings [Sam], Apple

HomeKit [Appa], and the Microsoft Azure IoT Edge [Mic].

Recently, programming paradigms have emerged that allow users to interact with IoT devices through event services. Instead of having the user manually control IoT devices through a control hub, these services abstract away the complexities of automation. These mechanisms automate user control processes through *events*,i.e., user-defined trigger-action schemes for IoT environments. For instance, event service platforms such as IFTTT (If-this-then-that) [SAB17a] and Zapier [Zap] provide a more convenient way to execute an action when a user-defined condition or set of conditions is satisfied. The user only needs to set an "if" condition (i.e., event trigger condition), and the corresponding IFTTT rule will interact with the control hub to execute the associated event action. Although these paradigms generally provide a convenient means of programming IoT domains, we will show that their limitations reside in their sensor and actuator programming abstractions.

Generalized system model. The system model we consider in this chapter has a control hub that may be running one of the hub services mentioned above. A user may have direct access to a device via an application interface or can write an *event* for the smart environment using a commodity IoT event service platform. Each event consists of a condition or a set of conditions and an associated action(s). The condition is an expression that specifies the state of the smart environment, and the action is the actuation command that changes the environment state(s). The device state-space representation for both conditions and actions is domain-specific and depends on the API provided by the IoT event service platform, e.g., a smart bulb being on or off. A *policy* will also be domain-specific as it defines the sets of allowed and disallowed state-space transitions for an IoT system [CTMa]. Our model allows a policy to be specified by whoever is configuring the IoT system. In this paper, we seek to provide conflict remediation, i.e., policy violation resolution, for the system model and a number of defined events. The remediation, in essence, is a set of new events suggested for replacing some old ones in order to resolve conflicts in the event set. We first enumerate the limitations of current conflict detection methods and formalize the conflicts considered

in this paper.

Conflict types. We define a *conflict* as a violation of policy. We consider two general categories of policies: user-defined policies and the class of policies introduced by Celik et al. that do not allow mutually exclusive states to exist in an environment [CTMa]. The latter category can be decomposed into racing events and cyclic events. Racing events stand for two or more events that are triggered at the same time while having conflicting actions, and cyclic events represent two or more events consist of a set of conditions and a set of actions that mutually trigger each other continuously. Table 6.1 summarizes the types of policy violations we consider with illustrative examples.

As was previously pointed out, any forms of conflict resolution from prior works have been limited to simply blocking a prospective action or reporting the conflict to the user[CFP18]. The bottleneck for providing remedial actions for such conflicts resides in the current state-of-the-art for obtaining the user's intention since the current IoT actuation programming abstractions hide such information. These premises allow us to provide an overview of the RemedIoT framework.

6.2.2 Related Work

We now discuss the relevant works directly related to RemedIoT. **Conflict Detection.** Several works have already focused on the problem of detecting conflicts between IoT events. BuildingRules [NRB18] proposes an occupant customized building configuration system. Surbatovich et al. [SAB17b] builds an information-flow model to analyze how IFTTT recipes violate the integrity, and it then categorizes what damages the IFTTT recipes could cause for the user. IoTSAT [MAH16] models the cyber-physical behaviors of IoT devices based on the factors including the network, device configurations, and user policies to analyze the possible or potential vulnerability of the IoT network. Danger-system [PPN15] detects the smart building environment conflicts through mobile crowd sensing. Ma et al. [MPT] and CityGuard [MPS17] provide runtime detection for the specific conflicts in the smart

city through an intermediate layer of watchdog. IoTMon[DH18] uses data mining and NLP-based technology to analyze how the applications affect physical environments, which further suggests the risks of the applications. IoTGuard [CTMa] enforces a set of policy rules on IoT applications by injecting monitor code into the target app, and it blocks any unsafe states. Soteria [CMT18] builds a static analysis system from the application descriptions to infer whether there are potential safety or security issues of the app. SIFT (Liang et al.) [LKL15b] proposes a safety-centric programming platform for connected devices in IoT environments to safely verify that there are no conflicts when the developers compile the applications. Miettinen et al. [MMH17] introduced a technique for IoT *device-type* identification used for security enforcement based on device-fingerprinting. Trimananda et al. [TAC20] introduces a cyber-physical modeling method to exhaustively check the conflicts among smart home applications.

All of the above works have focused on analyzing or detecting conflicts or security violations. However, none of them has yet provided a way to resolve such conflicts or violations except for simply reporting them to the user. Nextly, we introduce some other works showing how conflicts can be *debugged*.

Conflict Debugging. CityResolver [MSF18] uses Integer Linear Programming (ILP) method to find the most optimal conflict resolution in a given set of resolutions. Meanwhile, RemedIoT provides a means of automatically generating the resolutions. Liang et al. [LBL16] provides an automated debugging tool for the IFTTT platform by adjusting the clauses or the parameters of IFTTT rules. However, such modifications might not lead to the original desired state that the user wants. RemedIoT focuses on providing an alternative path to get to the exact original desired states without violating any policies or conflicting with other events.

IoT programming abstractions Beam [SSP16] abstracts sensors into modules based on the inference that can be made from the sensory data. Such abstraction is realized using inference graphs, and then the developers only need to focus on sensing capabilities rather

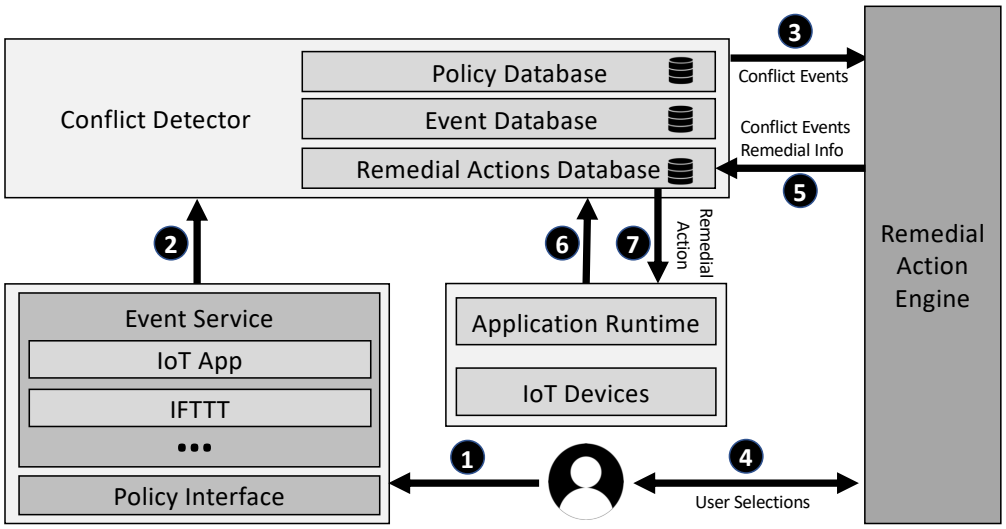


Figure 6.1: An overview of the RemedIoT framework.

than choosing particular sensors. However, Beam does not deal with the abstraction of actuators that might cause conflicts among the devices. ExPat [YPH19] proposes a formal specification programming language to ensure that the rules in the smart environment can be varified by the user's intention without conflicting with other rules. However, it does not provide any solutions to the conflicts. HomeOS [DMA12, DMA10] and BOSS [DKT13] provide an operating system for smart home environments so that programming or accessing the IoT devices at home can be done through an OS-like interface, i.e., device drivers. However, like all other traditional operating system, HomeOS simply puts a lock on the device when other processes are trying to access it. It does not make attempts to find conflicts or resolve them.

6.3 RemedIoT Overview

RemedIoT is a dynamic and context-based mediator for IoT event services. RemedIoT aims to provide a set of meaningful remedial actions for conflicting actions rather than

merely block the actions. For example, an application user may configure an event that opens the window when the temperature is too high, but this configuration may violate a safety policy which states that all windows and doors should be closed when the user is not home. RemedIoT would provide a remedial action to suggest the user turn on the HVAC instead of opening the window as the former provides the same utility. Figure 6.1 shows the design overview and the workflow of how RemedIoT interacts with the aforementioned system model.

At configuration time, a user may attempt to install a third-party IoT safety policy through a policy interface or configure an event service (❶) that can be analyzed by the conflict detector (❷). The conflict detector will check if the prospective event conflicts with any installed events or policies. If conflicts do not exist, the new event will be installed. If a conflict exists, it will send the set of conflicting events to the RemedIoT Remedial Action Engine (❸) that will analyze the conflict and see if any remedial actions can be installed to prevent conflicting actions from being triggered at the same time. The remedial actions will then be suggested to the user (❹). After the remedial actions are selected by the user, they will then be written back to a database for future use (❺). When such a conflict happens during runtime, the remedial action from the database will be executed. At runtime, all events from the installed event services are examined in case an event is conflicting with a policy(❻). If a conflict exists, the appropriate remedial action is carried out if available. Otherwise, the action is blocked (❼).

Although conflict detection is a part of the system, the core contribution of this chapter resides in the Remedial Action Engine. However, in order to realize such a tool, we need to design appropriate programming abstractions first.

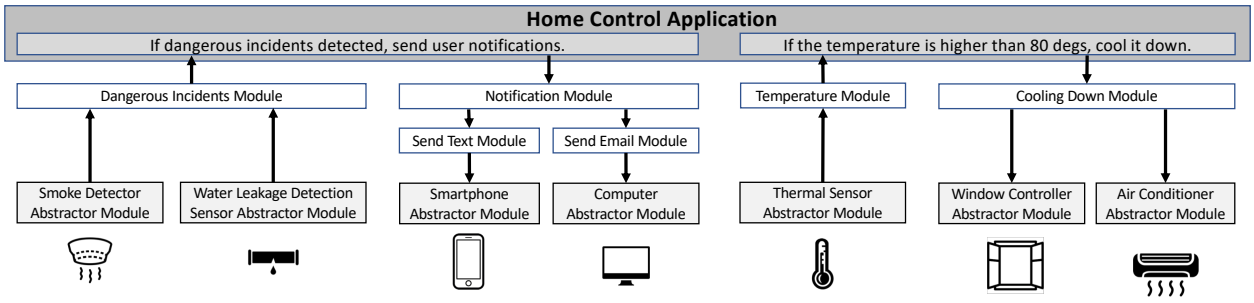


Figure 6.2: A sample smart home control application and its associated actuation graph.

6.4 RemedIoT Programming Abstractions

The main goals of RemedIoT's programming abstraction design are to not only increase the usability of distributed applications for dynamic smart environments, but to also ease the development effort for application developers with respect to safety and security. The abstractions enable the application developers to specify high-level programming intentions of an application instead of focusing on which devices will realize the application utility. For example, if a developer wants to develop an application that lowers the temperature of a space, he or she only needs to specify "cooling down" rather than selecting an HVAC or a smart fan realization. RemedIoT maintains the dynamic realization of these high-level intentions through lower-level device abstractions–where the lowest level abstractions will be the monomorphic device drivers. Such a design provides an inherent redundancy for abstractions that provide the core of RemedIoT's remedial action engine. Previously, Beam [SSP16] presented a framework to provide such a hierarchical structure for sensors called an inference graph. RemedIoT couples this notion of an inference graph with an actuation graph to abstract the actuators of the system.

6.4.1 Actuation Graph

RemedIoT's actuation graph–provided by the programming environment developers –maintains how an actuation should be realized. The graph is built as a directed graph where each node stands for an *actuation module* that indicates *how* to perform actuation or sensing tasks by combining other submodules or devices either collaboratively or independently. Each actuation module can have different *implementation units*–which are hidden from the application developers–to realize the module. The direction of the actuation graph corresponds to the dependency relationship among the actuation modules. With the assistance of the actuation graph, RemedIoT can easily infer the intentions of an application user by looking at which module is used. The actuation graph subsumes the notion of Beam's inference graph [SSP16] because it encapsulates the sensing dependencies of actuators. Moreover, the actuation graph is highly customized and developed differently for each smart environment.

Although abstraction modules enable RemedIoT to reason about a user's high-level intentions as well as to provide alternative actuation realizations, RemedIoT conversely needs a mechanism to expose the capabilities of physical devices. It is necessary to build device-level abstractions that are then used to realize the higher-level modules. As such, we introduce a *device abstractor module* that allows developers to interface devices with abstraction modules. For clarity, we will present the aforementioned actuation graph components in an illustrative example.

Motivating example: home control application. Figure 6.2 shows an actuation graph used by a home control application. The home control application has two installed events:

1. *If dangerous incidents are detected, then send notification to the user.*

2. *If the temperature of a room is higher than 80 degrees, then cool it down.*

The first rule will require the actuation graph to provide a detection module for "dangerous incidents" and a "notification" module such that the user can be appropriately notified.

The second rule needs to have a "temperature" module to sense the ambient temperature as well as a "cool down" module that lowers the ambient temperature. In the associated actuation graph, the sensing module *temperature* requires the data from a smart thermal sensor installed in the room to infer the temperature. Those data are hidden from the application developer's perspective. The actuation module *cooling down* needs to access either a smart window or the air conditioner–where the application developers need not worry about which device to use. However, in order to be compatible with existing IoT systems such as SamSung SmartThings and Apple HomeKit, our programming model also allows event services to directly access the devices without going through high-level sensing or actuation modules. In any case, the abstraction modules are designed to account for all possible conflicts that may exist in the network.

We now describe the attributes of an abstraction module required to maintain the actuation graph.

6.4.1.1 Abstraction Module

The abstraction module is a programming interface of a sensing or actuation functionality performed by specific implementations. An abstraction module may also be realized by other abstraction modules. Each module is implemented through *implementation units*, where the module provides a template of the attributes to be defined. The implementation units encapsulate the actuation or sensing algorithms of the associated modules. For instance, the aforementioned *cool down* abstraction module is generally associated with lowering the ambient temperature of a space, while the actuation graph developers may be able to develop different cooling down *implementation units* for a specific smart home environment, such as cooling down through an air conditioner, or cooling down by opening the windows or doors. The following attributes are necessary to realize an abstraction module for our actuation graph:

- **Module name.** The module name is the label for the abstraction modules, which summarize the characteristics of that module, e.g., "cooling down".

- **Implementation unit name.** The implementation unit name summarizes the function of an actual implementation unit that may realize the associated abstraction module. For instance, "turn on AC" or "open the window". Each abstraction module may correspond to multiple implementation units.

- **Module type.** The module type indicates whether the abstraction module is a sensing module, an actuation module, or both.

- **Implementation prerequisites.** The prerequisites refer to the implementation unit's dependent modules. Each implementation unit needs to specify what it may depend on in order to be realized. For instance, one of the implementation units of the *notification* module might depend on a *send text* module, and another implementation unit of the *notification* module might depend on a *send email* module. Different implementation units can have different prerequisites under the same module.

- **Cost and utility metrics.** Each abstraction module may be associated with a cost or utility metric that will establish how implementation units should be prioritized when RemedIoT is suggesting remedial actions. For instance, the utility metric of a "cooling down" abstraction can be defined as to how efficiently an implementation unit may lower the ambient temperature, while a cost metric may be associated with the power consumption of a unit.

Defining each of the attributes mentioned above will be domain specific, and it is up to the programming environment developer to ensure that all of the components of the actuation graph are compatible with each other. The first step in doing so is to implement a *device abstractor module*.

6.4.1.2 Device Abstractor Module

The device abstractor is a type of special abstraction module that handles the details of a particular device's metadata so that the high-level information is exposed to the developer. In particular, the abstractor module characterizes and implements the connection to the device through the device driver and the network. Every physical device corresponds to an implementation unit, but each device abstractor module might have one or more device implementation units. For example, if there are three installed cameras, each of them will have one corresponding implementation unit, and only one device abstractor module "RGB camera" is needed to encapsulate all the three implementation units. Similar to HomeOS [DMA12], RemedIoT dynamically handles the connections and disconnections of devices in the network. When a device is disconnected from RemedIoT, RemedIoT updates the actuation graph accordingly. Moreover, if a new type of device is connected, RemedIoT creates a new device abstractor instance and updates the graph if other abstraction modules depend on it.

To provide context of these abstractions, we describe how a an example abstraction module may be implemented.

6.4.2 Programming Example

Listing 6.1: Sample Inference Module

```
# Import the existing modules
from ActuationGraph import *

# Implementation of cooling down module via AC
class ACTurnOn(ImplementationUnit):
    def __init__(self):
        ImplementationUnit.__init__(...)
        acDA = ACDeviceAbstractor() #device abstractor
        super(ACTurnOn, self).appendChildDeviceAbstractor(acDA)

    def performFunc(self, *args):
```

```
        # Perform the specific action to turn on AC
        ...

# Cool Down Module
class CoolDownModule(Module):
    def __init__(self):
        Module.__init__(...)
        acTurnOn = ACTurnOn() # Implementation Unit
        fanTurOn = FanTurnOn() # Implementation Unit
        super(CoolDownModule, self).addImplUnit(acTurnOn)
        super(CoolDownModule, self).addImplUnit(fanTurnOn)
```

To develop an implementation unit of a module in the actuation graph, the actuation graph developers need to follow the template of the module and implementation units. Moreover, each actuation module can only allow prerequisites on either device abstractor modules or other general actuation modules for its implementation units. The device abstractor modules and the general actuation modules cannot simultaneously be the prerequisites of one actuation module. For instance, in Figure 6.2, the send text module and *computer abstractor module* cannot be the prerequisites of *notification module* at the same time. This is to ensure that all prerequisites of one module have the same level of hierarchy.

Listing 6.1 shows an example implementation of how a `CoolDownModule` class may be implemented as a base abstraction module associated with device modules that can cool down the ambient temperature. We show an example implementation unit of this module called `ACTurnOn` that may define the behavior associated with turning on an air conditioning unit.

Given the aforementioned programming abstractions, we now show how prior notions of a policy file can be augmented to incorporate abstraction modules.

⟨*policy-set*⟩ ::= ⟨*statement*⟩; (⟨*statement*⟩;)* **EOF**

⟨*statement*⟩ ::= ⟨*restrict-clause*⟩ | ⟨*allow-clause*⟩

⟨*restrict-clause*⟩ ::= **RESTRICT :** ⟨*state-transition*⟩

⟨*allow-clause*⟩ ::= **ALLOW:** ⟨*state-transition*⟩

⟨*state-transition*⟩ ::= ⟨*transition*⟩ (, ⟨*transition*⟩)* **:** ⟨*state*⟩ (,⟨*state*⟩)*

⟨*state*⟩ ::= ⟨*abstraction-module*⟩ **.** ⟨*abstraction-attr-id*⟩ (.⟨*abstraction-attr-id*⟩)*
⟨*logical-operator*⟩ ⟨*attr-value*⟩

Figure 6.3: Policy grammar for IoTGuard [CTMa] augmented with RemedIoT's programming abstractions.

6.4.3 Policy Grammar Definition

We show how policy grammars for conflict detectors in previous works can be augmented to incorporate our abstraction modules. For instance, IoTGuard [CTMa] defined a Backus-Naur Form (BNF) policy grammar to allow developers to define rules regarding the sets of allowed and restricted state transitions for IoT devices. The same policy grammar can be augmented to define the allowed and restricted state transitions for abstraction modules, as shown in Figure 6.3. The semantics would remain the same as long as abstraction module attributes are syntactically correct.

These programming abstractions, along with a conflict detector that utilizes an associated augmented policy grammar, allow us to realize the RemedIoT remedial action engine.

6.5 Remedial Action Engine

As discussed in Section 6.3, RemedIoT's runtime remedial action engine takes as its input a conflict generated by a conflict detector and attempts to resolve the the conflict by utilizing a configured remedial action for the particular conflict. If no remedial action exists, the conflicting events will be blocked. To provide the design details of this system, we first discuss how these remedial actions are configured and installed prior to runtime.

6.5.1 Remedial Action Configuration

Remedial actions are configured upon encountering a new conflict from the conflict detector. New conflicts may arise during the installation of either event services or policies. Algorithm 1 describes the remedial action configuration process for a candidate event e. The first step is to detect whether the candidate event conflicts with any existing event services or policies.

Conflict detection. RemedIoT ensures that any candidate events or policies are first passed through the conflict detector (Line 4) to see if any conflict exists with other installed events or policies. Because the state space for event services considered by RemedIoT is limited to two states (*on* or *off*), the set of conflicts that arise, E_{conf}, can then be divided into two sets: the singleton set with the candidate event and the set of all events that have a conflicting action with e. Further, we only care about remediating the conflicts for the set of events with lower priority, where "priority" refers to the aforementioned cost and utility metrics of an abstraction module. The higher priority set should always take precedence for executing a particular action at runtime. Therefore, RemedIoT will first generate the full set of conflicting events E_{conf} (`Line 4`), sort the events based on priority (`Line 5`), and then extract the lower priority set of conflicting events E'_{conf} (`Line7`). Given this extracted set, RemedIoT will attempt to find a remedial action for each conflict.

Remedial action generation. RemedIoT first iterates through each conflicting event service, e_{conf}, and prompts the user (`Line 10`) for the intended abstraction module of the

Algorithm 1 Remedial Action Configuration

1: **Input:** candidate event e, actuation graph G_{act}, policy database DB_{policy}, event database DB_{event}, remedial action database DB_{rem};

2: **Output:** updated event database DB'_{event}, updated remedial action database $DB'_{remedial}$;

3: //Get conflicting events for given event.

4: $E_{conf} \leftarrow$ conflictDetector(e, DB_{event}, DB_{policy}, DB_{rem});

5: Sort conflict events E_{conf} based on priority;

6: // Extract lower priority set of conflicts to remediate:

7: $E'_{conf} \leftarrow$getLowerPrioritySet(E_{conf})

8: **for all** $e_{conf} \in E'_{conf}$ **do**

9: // Get user to select the intended non-conflicting module:

10: $intendedModule \leftarrow$ getIntentionFromUser(e_{conf},G_{act});

11: // Given intended module with pruned impl. units, prompt

12: // user to select a non-conflicting rem. action impl. unit:

13: $a_{rem} \leftarrow$getRemActionFromUser($intendedModule$, G_{act});

14: // Map each conflict pair to its remedial action in database:

15: **for all** $e^{\complement} \in E_{conf} \backslash E'_{conf}$ **do**

16: $DB_{rem} \rightarrow$add($\langle (e_{conf}, e^{\complement}), a_{rem} \rangle$);

17: **end for**

18: **end for**

19: //Add candidate event to event database:

20: $DB_{event} \rightarrow$add(e);

conflicting implementation unit by utilizing the actuation graph, G_{act}. If there is more than one conflict, RemedIoT ensures each of the possible options prompted to the user does not introduce new conflicts by using the conflict detector. For instance, in Figure 6.2, if the conflicting implementation unit was the "Window Controller", the user may confirm that the intended abstraction module was the parent "Cooling Down" module. If the user declines to use the parent abstraction module, then RemedIoT will block this action upon detecting a conflict at runtime. RemedIoT ensures that any parent abstraction module that is presented to the user will have a set of non-conflicting implementation units, i.e., any implementation units from the actuation graph G_{act} that will cause a new conflict are pruned. Given the intended abstraction module, RemedIoT then prompts the user to select an alternative implementation unit (`Line 13`), e.g., the aforementioned example will prompt the user to select the "Air Conditioner" implementation unit. If the user selects a suggested remedial action, the action is stored in a remedial action database $DB_{remedial}$ by mapping each conflicting event pair, i.e., e_{conf} and the complementary $e^{\complement}$ for all events in the higher priority set $E_{conf} \backslash E'_{conf}$, to its associated remedial action a_{rem} (`Line 15`). If no remedial action is chosen, then the default action will be to block the conflicting event. Once all remedial actions have been generated, RemedIoT can now add the candidate event e to the event database DB_{event}.

6.5.2 Runtime Remediation

At runtime, RemedIoT runs alongside the event service platform. IoT event service platforms such as IFTTT typically check the trigger conditions of all installed event services periodically. When a new event is activated, RemedIoT will utilize the conflict detector mentioned in Algorithm 1 to check if the new event has conflicts with other events that are currently active. If a conflict exists, it will query the remedial action database DB_{rem} to select the appropriate action.

Each of the aforementioned components will have engineering challenges which we address

in the subsequent section.

6.6 Implementation

In order to implement RemedIoT's core utilities, we first demonstrate how an existing conflict detector can be instrumented to interface with the RemedIoT remedial action engine. We first describe the conflict detector instrumentation and then discuss the implementation for both the abstraction graph module support as well as the RemedIoT action engine.

6.6.1 Conflict Detector Instrumentation

We implement the framework for the conflict detector as a directed graph–as was done in the conflict detector for IoTGuard [CTMa][1]. Within the graph, an event can be described as two nodes and one directed edge. Each node stands for a set of device (or module) states and each directed edge indicates a trigger-action relationship. Note that one node may contain a cascade of conditions (actions) instead of only one module state. A conflict arises when a collection of compatible conditions, i.e., conditions that can be satisfied at the same time, can simultaneously cause a set of incompatible actions, i.e., lead to mutually exclusive states of the same device.

When a new event is added, the conflict detector[2] traverses the graph nodes to determine all the *compatible conditions* nodes as well as all the *incompatible actions* nodes. We then perform a reachability analysis to determine if a path exists between these two sets. Any detected path will be considered as a potential conflict and removed from the graph. If any conflicts arise, these conflicts, along with the new event, are delivered to the remedial

[1]We implemented our own conflict detector inspired by IoTGuard since the source code was not available at the time this paper was written.

[2]The conflict detector module was implemented in Python with 400 LoC and is available in the repository.

action engine to generate the associated remedial actions. If no direct conflicts are observed in this stage, the new event will be added to the graph. Finally, we employ a depth-first search-based algorithm to find all the loops inside the dependency graph. All discovered conflicts and loops are forwarded to the Remedial Action Engine.

Given a conflict detector, we now describe how we implement the core components of RemedIoT. Before we can describe the implementation of the remedial action engine, we first describe the implementation of the actuation graph module support that enables the remediation.

6.6.2 Actuation Graph Module Support

We implement a generic API for IoT programming environments to support the aforementioned actuation graphs. The graph support is implemented as a base abstract class in Python with a domain-specific implementation. We illustrate how an actuation graph can be constructed in Section 6.7. The actuation graph support enables the remediation of conflicts.

6.6.3 Remedial Action Engine

The Remedial Action Engine is also implemented in Python based on Algorithm 1. We implement a proof-of-concept interface to allow the user to select the proper remedial action, as illustrated in figure 6.4. The remedial action, event, and policy databases are implemented and maintained as dictionaries in Python. The Remedial Action Engine also utilizes the aforementioned conflict detector both at configuration time and at runtime. We now detail our evaluation with illustrative examples.

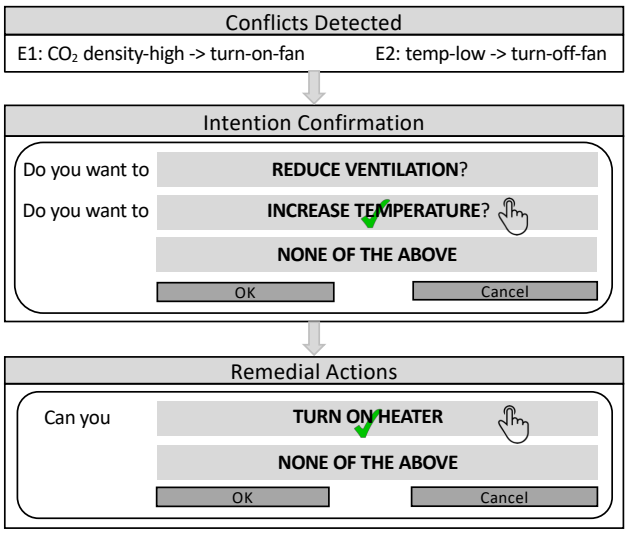


Figure 6.4: An example interface for RemedIoT that suggests a remedial action to the user when a conflict is detected.

6.7 Evaluation

To illustrate a domain-specific implementation, we evaluate RemedIoT on Samsung SmartThings and IFTTT applets. We first describe how we manually extract events and policies for the platforms in order to perform an evaluation.

6.7.1 Event and Policy Realization

We manually extract a large set of events from both programming platforms. Although we do not obtain all possible events from both platforms, we try to choose the most representative actions from each application[3]. When we extract the events from the platforms, we format the input events as "IF conditions THEN actions[,priority]", where the priority can be one of three categories (highest to the lowest): safety, energy, and utility. By default, each event is designated as a utility. The priority is used to determine which event is more important

[3]All the events we select are available in the github repo.

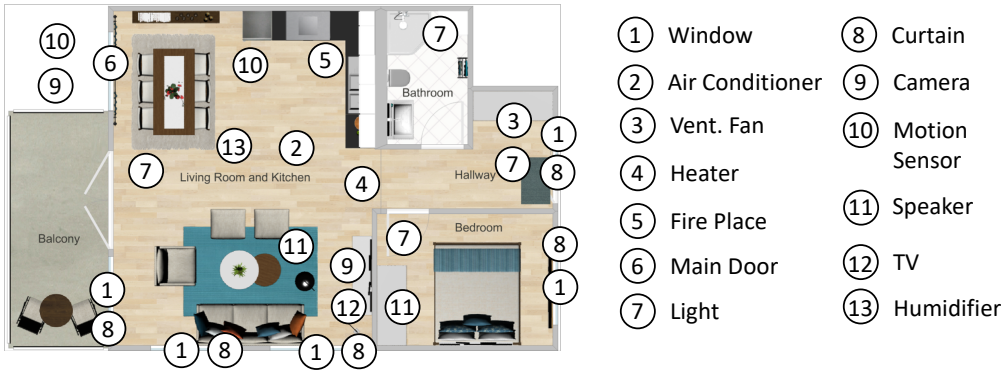


Figure 6.5: The simulated smart home environment with artificially placed IoT devices utilized in our evaluation.

when a conflict is detected. For example, in a Samsung SmartThings safety application, we manually extract an event as "`if co2.density >= 50 then vent_fan.state = on, safety`". The extracted event is then parsed and passed to the conflict detector as an input.

Further, all policies are implemented as special events to our system. For our evaluation, each specific policy has been assigned to a category – *house safety*, *energy economy*, or *user comfortability*. For example, P.1 in table 6.3 is implemented as "if user.home = 0 then doors.state = 1 and windows.state = 1,safety". We now discuss the environmental setup for our experiments.

6.7.2 Environment Setup

We first describe how we set up our experimental environment. We simulate a smart home environment, as shown in figure 6.5. It has 13 different smart devices. We run our experiment on an Intel Nuc desktop equipped with an Intel Core i7-6770HQ. For this smart environment, we have to construct an actuation graph for the associated IoT modules.

Actuation graph modules. We construct an actuation graph that implements 13 different devices through 8 actuation modules that cover all situations in our selected events.

Table 6.2: The actuation modules and their associated implementation units we considered in our evaluation.

Module	Description	Implementation Unit	Description
Cooling down	Cool down the house	Window Opening Low	Open the window when outside temp. is lower
		AC Turning On	Turn on Air Conditioner to cool down
		Fan Turning On	Turn on cooling fans
Heating up	Heat up the house	Window Opening High	Open the window when outside temp. is higher
		Heater Turning On	Turn on the heater to heat up
		Fireplace On	Light up the fireplace to heat up the house
Ventilation	Ventilate the air of the house	Fan Turning On	Turn on the fan to ventilate
		Window Opening	Open the windows to ventilate
		Door Opening	Open the doors to ventilate
Illumination	Brighten up the house	Light Bulbs On	Turning on the light bulbs
		Curtain Opening	Open the curtain when outside is brighter
Green Energy	Save the energy of the house	High-Power Devices Off	Turn off the high power consumption devices
		Low Power Mode On	Set low power mode for the devices
Movement	Detect movement at home	Camera Detection	Detect the motion through cameras
		Motion Sensors	Detect the motion through motion sensors
Warning Notification	Send the warning info to the user	Text Warning	Text the warning message to the user
		Speaker Warning	Play warning sounds through speaker
		Flashlight Warning	Display flashlight using the lightbulbs
Increasing Humidity	Increase the humidity	Fan Turning Off	Turn off the ventilation fan
		Humidifier On	Turn on the humidifier

Moreover, each module provides 2 to 3 different paths (implementation units) to achieve the desired state specified by the module. Table 6.2 summarizes the modules and the implementation units we provide.

The cost of each module or implementation is simply the energy it might consume using a hard-coded value. For example, turning on the AC has a higher cost than opening the window[4]. Given this environmental setup, we now describe the set of microbenchmarks used to evaluate RemedIoT.

6.7.3 Microbenchmarks

We propose a set of benchmarks extracted for real Samsung SmartThings applications and IFTTT applets. The goals of our benchmarks are to determine how many conflicts–which

[4]Although power consumption profiles can be made for devices, we simply set relative values to ensure the devices are sorted correctly according to their expected power consumption for a proof-of-concept.

Table 6.3: The policies used to evaluate RemedIoT on Samsung SmartThings and IFTTT applets.

Category	Index	Policy Description
General	P.0	Mutually exclusive states must not exist in the environment.
House Safety	P.1	The doors and the windows must be locked when the user is not home.
	P.2	The emergency alarming system must be on.
	P.3	The security cameras must be on when the users are not home.
	P.4	The electronic devices must be off when fire sprinklers are on.
Economic Energy	P.5	The heater and the AC must not be on at the same time.
	P.6	The sprinklers must be off when it rains.
	P.7	The dryer and the humidifier must not be on at the same time.
User Comfortability	P.8	The non-emergency sound system must be off when users are sleeping.
	P.9	The curtains must be closed when private mode is on.
	P.10	The lights must be off while the users are sleeping.

Table 6.4: The aggregated results for conflict detection and remediation using RemedIoT in the context of the Samsung SmartThings and IFTTT applets.

# Apps	# Events	# Conflicts	% Conflicts (Out of Events)	% Blocked Events (Out of Conflicts)	% Remedial Actions (Out of Conflicts)
74	195	102	52.31%	19.61%	80.39%

would normally be blocked by prior works–can be remediated. We achieve this by essentially trying to configure as many events as possible in our simulated smart environment. We further want to show the configuration time overhead to illustrate the efficacy and usability of RemedIoT. In total, we collect 195 representative events from 74 applications. Moreover, we propose *ten* specific policies ranging from housing safety to user comfort and one general policy, as illustrated in table 6.3. We implement *eight* different modules with 20 implementation units, as shown in table 6.2, to support the Remedial Action Engine.

6.7.4 Results

The aggregated results for installing the set of events are shown in Table 6.4. Based on 195 extracted events from 74 sample applications and 11 policies, we detect 102 conflicts, i.e., 52.31% of all events conflicted with each other upon configuration. By using the RemedIoT

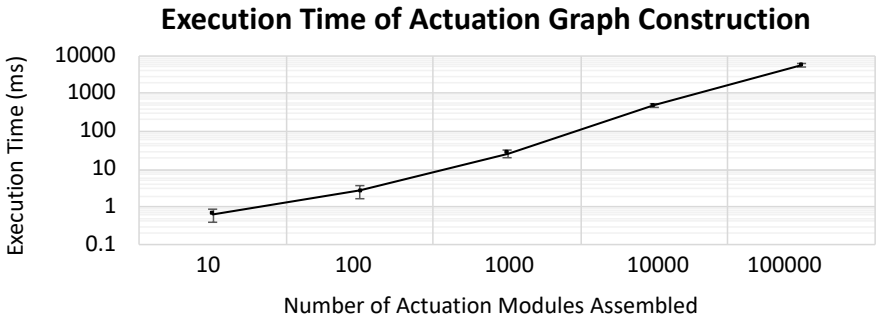


Figure 6.6: Average performance overhead of 50 iterations when constructing an actuation graph while varying the number of actuation modules.

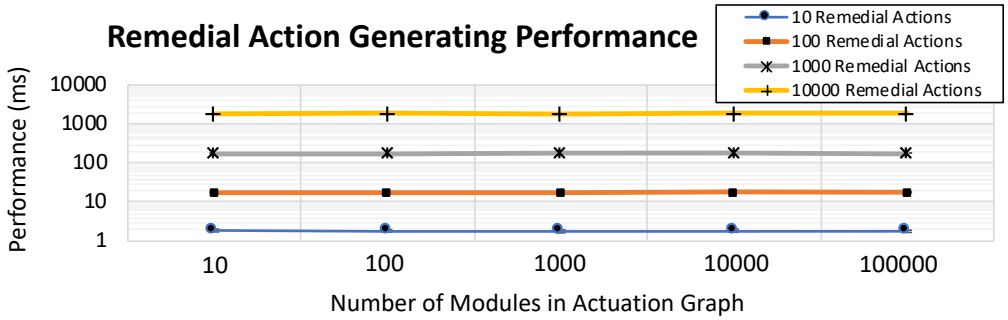


Figure 6.7: Average performance overhead of generating remedial actuations. We performed over 50 iterations while varying the size of the associated actuation graph.

Table 6.5: Ten selected remedial actions provided by REMEDIOT after running the microbenchmarks.

Index	Event	Conflict	Remedial Action
C.1	home temp >= 60 & outside temp < 60 ⇒ *open window*	window is on and off at the same time	home temp >= 60 & outside temp < 60 ⇒ vent. fan on
	user away ⇒ user away mode on		Not Applicable
	user away mode on ⇒ *close window*		Not Applicable
C.2	vacation mode on ⇒ *all lights off*	Lights on and off at the same time	Not Applicable
	motion detected ⇒ *lights on*		motion detected ⇒ text the user
C.3	home mode on ⇒ *turn on TV*	TV on and off at the same time	Not Applicable
	energy saving mode on ⇒ *turn off TV*		energy saving on ⇒ TV low power mode
C.4	home temp >= 70 ⇒ *ac on*	AC on and off at the same time	home temp >= 70 ⇒ fan on
	home temp >= 75 ⇒ *ac on*		home temp >= 75 ⇒ fan on
	user away mode on ⇒ *ac off*		Not Applicable
C.5	light lux <= 1500 ⇒ *open curtain*	P.9	Light lux <= 1500 ⇒ light on
C.6	user away mode on ⇒ camera on	P.3 ; camera on and off at the same time	Not Applicable
	user away mode on ⇒ *camera off*		user away mode on ⇒ motion sensor on
C.7	CO_2 density > 50 ⇒ fan on	Fan on and off at the same time	Not Applicable
	light on ⇒ fan on		Not Applicable
	time scheduled ⇒ *fan on*		Not Applicable
	humidity <= 50 ⇒ *fan on*		Not Applicable
	home temp >= 70 & outside temp <= 65 & ac off ⇒ *fan on*		Not Applicable
	home mode on & home temp > 75 ⇒ *fan on*		Not Applicable
	smoke detected ⇒ *fan on*		Not Applicable
	user shower ⇒ *fan on*		Not Applicable
	humidity <= 50 ⇒ *fan off*		humidity <= 50 ⇒ humidifier on
C.8	home temp >= 71 ⇒ *heater off*	Heater on and off at the same time	Not Applicable
	home mode on ⇒ *heater on*		home mode on ⇒ window open
C.9	motion detected ⇒ *lights on*	Lights on and off at the same time	Motion detected ⇒ play sound on speaker
	sleep mode on ⇒ light off		Not Applicable
C.10	arrive home ⇒ door open	Door open and close at the same time	Not Applicable
	CO_2 high ⇒ door open		Not Applicable
	home mode ⇒ door close		home mode ⇒ fan on

framework, 80.39% of the conflicts can be sought for remedial actions. These remedial actions highlight the contributions of our paper: in previous conflict detecting systems, theses conflict events are simply blocked. RemedIoT provides a set of substitution events that can realize the same function for 80.39% of the conflicts, i.e., 82 of the 102 events could be configured without worrying about possible conflicts. Table 6.5 describes *ten* selected remedial actions suggested by RemedIoT framework. For instance, for conflict C.1, one event is trying to close the window when the user is away, and the other is trying to open the window to cool down the house. Because the former event has a higher priority for safety, the smart fan is chosen to realize the cooling abstraction. We now discuss the feasibility of deploying RemedIoT in a real environment.

6.7.5 Configuration Time Overhead

Because RemedIoT might be deployed in large scale smart environments such as smart cities, we evaluate the scalability of RemedIoT. In particular, we calculate the overhead induced by RemedIoT when constructing the actuation graphs. We investigate the execution time of actuation graph construction against the number of its associated actuation modules. For each different number of actuation modules, we average the execution time over 50 runs. Figure 6.6 shows a nearly logarithmically linear relationship between the number of assembled modules and the time consumed. Given the fact that the number of smart devices is limited, the time to construct the actuation graph can be ignored. Further, the actuation graph only needs to be constructed once before runtime, so configuration time may be a moot point.

We also obtain the overhead time of running our Remedial Action Engine by generating different quantities of remedial actions on different sizes of the actuation graph. We again run each experiment for 50 iterations to get the average overhead as well as the error range. As illustrated in figure 6.7, the performance of the Remedial Action Engine is irrelevant to the number of actuation modules in the actuation graph, which means that simply increasing

the number of devices will not incur more overhead. The computation time logarithmically increases with the number of remedial actions that need to be proposed. Our results[5] show that a personal desktop can easily enable constructing 100000 modules while suggesting up to 10000 remedial actions at runtime, highlighting RemedIoT's scalability. Our results also validate the scalability of RemedIoT even if more complex policies are added, i.e., more conflicts are triggered.

6.8 Future Works

There are several future directions enabled by the RemedIoT.

Automating remediation. One limitation is that RemedIoT relies on at least one redundancy path to achieve the same functionality. A future direction of this work is to automate such a process so that RemedIoT system is able to dynamically learn and make adjustments based on the application user's behaviors. Techniques such as reinforcement learning, crowdsourcing, and collaborative filtering can facilitate the usability of RemedIoT.

Modeling sensors and actuators. Currently, the state space for sensors and actuators are binary, i.e. on or off. Cyber-physical modeling techniques can be utilized to better model different types of sensors and actuators. Further, we can dynamically optimize how an implementation unit implements an abstraction module at runtime. For instance, if an abstraction module performs human detection, a camera would be more effective than a microphone for certain environments.

[5]Although our implementation of the conflict detectors cannot detect large scale conflicts, we use a simulated program to generate many conflicts for evaluation purposes manually.

6.9 Conclusion

In this chapter, we proposed RemedIoT, a remedial action framework for resolving IoT conflicts. We evaluated the efficacy of RemedIoT on Samsung Smarthings and IFTTT applets in the context of a simulated smart home environment and showed that for a large set of applications, RemedIoT is able to resolve $\sim 80\%$ of conflicts. We further show the scalability of RemedIoT for generalizing to smart city environments.

CHAPTER 7

Conclusion and Future Research

This dissertation study and present a secure and safe framework for the usage of sensors and actuators at the edge under multi-tenant edge systems. We conclude the dissertation by summarizing the works done to build the framework and discuss the possible future directions of the research works.

7.1 Conclusion

The first part of the thesis presents **PROTC** and **VirtSense** to have runtime security for sensors and actuators under the untrusted execution environment with the help of hardware TEE. Furthermore, this part only introduces **Aerogel** that can achieve the same level of security for bare-metal IoT devices that have no hardware TEE by using Wasm. The second part of the thesis presents **SecDeep** that secures an edge-accelerators enabled deep learning inference framework at the edge. The third part of this dissertation presents **RemedIoT**, a runtime framework that can solve IoT actuation conflicts.

7.1.1 Securing Sensors and Actuators at Runtime

7.1.1.1 PROTC

PROTC is the framework for IoT devices that can protect the essential actuating services under an untrusted environment using hardware TEE, such as ARM TrustZone. The decisive part of this framework is to confirm that an actuation command is from the actual user

without being modified but not from a malicious application or OS. We implement PROTC for the UAV system on Raspberry Pi 3 with OP-TEE OS, and our results show that the average overhead of PROTC is about 143 ms.

7.1.1.2 VirtSense

VirtSense is a secure framework to protect IoT devices' essential sensing services under an untrusted execution environment through hardware TEE. The core design of VirtSense is to have a virtual sensing management unit inside TEE such that only authorized sensing requests can get the sensor data. Meanwhile, since different sensing requests might need the data at different frequencies, the virtual sensing management unit also satisfies those requests by reconstructing the sampled sensor data. We implement VirtSense on Raspberry Pi 3 using OP-TEE OS, and our results indicate an average overhead of 16ms.

7.1.1.3 Aerogel

Aerogel provides the runtime sensors and actuators protection for bare-metal IoT devices where the resources are constrained, and the hardware TEE is not available. Aerogel utilizes the memory isolation characteristics of Wasm to provide memory protection for sensors and actuators at runtime. Moreover, Aerogel also enables fine-grained access control to use resource-constrained bare-metal IoT devices such as the power consumption of sensors and actuators. We evaluate Aerogel using Wasm Micro-runtime and Zephyr OS on nRF52840, a cortex-M4 based development board. Our results show that Aerogel introduces at little as 0.19% to 1.04% runtime overhead and consumes 18.8% to 45.9% extra energies.

7.1.2 Securing Inferencing from Sensor Data at the Edge

7.1.2.1 SecDeep

SecDeep presents a new secure and performant deep learning inference framework at the edge. SecDeep utilizes TEE to secure the data confidentiality from the sensors. Moreover, SecDeep also uses available edge accelerators to speed up the deep learning inference. We implement and validate SecDeep by interfacing the ARM NN DL framework with ARM TrustZone on Hikey 960 board that has ARM Cortex A74 CPUs with ARM Mali G71 MP8 embedded GPU. Our evaluation shows that we can securely run inference tasks with 16× to 172× faster than CPU-based approaches by leveraging edge-available accelerators.

7.1.3 Providing Remedial Actions for Actuation Conflicts

7.1.3.1 RemedIoT

We provide remedial actions for IoT actuation conflicts through RemedIoT. In RemedIoT, we design an actuation graph that represents the purposes of using IoT devices. Based on the information provided by the actuation graph, RemedIoT can understand the user's intentions of using different IoT devices. Hence, RemedIoT can suggest possible remedial actions when there are actuation conflicts based on policies that define the allowable and restricted state-space transitions. We evaluate RemedIoT on Samsung SmartThings applications and IFTTT applets and show that for 102 detected conflicts across 74 sample applications with 11 policies, RemedIoT can remediate $\sim$ 80% of the conflicts found in the environment, which would normally be blocked by prior solutions. We further demonstrate the efficacy and scalability of our approach for smart city environments.

7.2 Future Research

Although this dissertation has addressed several challenges of multi-tenant edge systems from the aspects of security and safety, there still exist many interesting open research questions that can be answered in the future. Next, we list some possible future directions for the multi-tenant edge systems.

7.2.1 Prevention From Side-channel Attacks

Our design towards securing sensors and actuators heavily depends on the security features of hardware TEE. However, side-channel attacks such as Spectre [KHF19] and Meltdown [LSG18] towards TEE are still severe threats for data's confidentiality. Moreover, cyber-physical attacks such as GPS spoofing attack directly target on the sensors or actuators will affect data accuracy that might lead to severe consequences. In the future, we should have a more secure system that can protect the confidentiality of sensor data and the integrity of both sensors and actuators against possible side-channel attacks.

7.2.2 Prevention From Machine Learning Adversary Attacks

SecDeep protects edge deep learning inference, where SecDeep trusts the machine learning model that will not leak the privacy information from its inferenced results. However, researchers have found that adversary attacks can reverse the raw input data from the inferenced results in some deep learning models. Future research could utilize both TEE and adversary-attack-proof algorithms to strengthen raw data confidentiality and user privacy.

7.2.3 Increase Automation Capabilities of Actuation Conflict Resolution

Although our designed RemedIoT framework can provide remedial actions for actuation conflicts, RemedIoT still requires the users' interceptions to choose their intentions correctly.

Ideally, if the framework can learn user's habits from their past events, it will not require the user to choose his or her intentions, but make the decisions automatically. The future research direction of RemedIoT should be integrated with machine learning algorithms such as complex events detection to learn what the user's true intentions are when a conflict is detected. Moreover, we still need manual configurations of safety policies for RemedIoT. Future work could be focusing on learning the unsafe behaviors from the past experience through transfer learnings such that the framework can automatically generate safety policies in any smart environment.

[96b] 96boards. https://www.96boards.org/product/hikey960/.

[AAW17] H. F. Atlam, A. Alenezi, R. J. Walters, G. B. Wills, and J. Daniel. "Developing an Adaptive Risk-Based Access Control Model for the Internet of Things." In 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 655–661, 2017.

[ABC12] Pandarasamy Arjunan, Nipun Batra, Haksoo Choi, Amarjeet Singh, Pushpendra Singh, and Mani B. Srivastava. "SensorAct: A Privacy and Security Aware Federated Middleware for Building Management." BuildSys '12, pp. 80–87, 2012.

[ACB] Naif Saleh Almakhdhub, Abraham A Clements, Saurabh Bagchi, and Mathias Payer. "μRAI: Securing Embedded Systems with Return Address Integrity.".

[ACN10] S. Alam, M. M. R. Chowdhury, and J. Noll. "SenaaS: An event-driven sensor virtualization approach for Internet of Things cloud." In 2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications, pp. 1–6, Nov 2010.

[AFS97] W. A. Arbaugh, D. J. Farber, and J. M. Smith. "A secure and reliable bootstrap architecture." In Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097), pp. 65–71, 1997.

[AG01] A. Aldroubi and K. Gröchenig. "Nonuniform Sampling and Reconstruction in Shift-Invariant Spaces." SIAM Review, **43**(4):585–620, 2001.

[AKA19] 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), pp. 1375–1392, Santa Clara, CA, August 2019. USENIX Association.

[AL17] Na-Young Ahn and Dong Hoon Lee. "Countermeasure against Side-Channel Attack in Shared Memory of TrustZone." CoRR, **abs/1705.08279**, 2017.

[AMD] AMD. "AMD Secure Encrypted Virtualization." https://developer.amd.com/sev/.

[AMT18] Ilias Avramidis, Michael Mackay, Fung Po Tso, Takaaki Fukai, and Takahiro Shinagawa. "Live migration on ARM-based micro-datacentres." In 2018 15th

IEEE Annual Consumer Communications & Networking Conference (CCNC), pp. 1–6. IEEE, 2018.

[And] Android. "Android Neural Networks API." https://developer.android.com/ndk/guides/neuralnetworks.

[ANS14] Ahmed M. Azab, Peng Ning, Jitesh Shah, Quan Chen, Rohan Bhutkar, Guruprasad Ganesh, Jia Ma, and Wenbo Shen. "Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World." In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, pp. 90–102, New York, NY, USA, 2014. ACM.

[Appa] Apple. https://www.apple.com/ios/home/.

[Appb] Apple. "Machine Learning on iOS." https://developer.apple.com/machine-learning/.

[Ard] ArduPilot. http://www.ardupilot.org/.

[ARMa] ARM. https://developer.arm.com/ip-products/security-ip/trustzone.

[ARMb] ARM. https://developer.arm.com/ip-products/processors/cortex-m/cortex-m23.

[ARMc] ARM. https://developer.arm.com/ip-products/processors/cortex-m/cortex-m33.

[ARMd] ARM. "ARM Cortex M4." https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4.

[ARMe] ARM. "ARM Cortex M7." https://developer.arm.com/ip-products/processors/cortex-m/cortex-m7.

[ARMf] ARM. "ARM Trusted Firmware Design." https://chromium.googlesource.com/external/github.com/ARM-software/arm-trusted-firmware/+/v0.4-rc1/docs/firmware-design.md.

[ARMg] ARM. "ARM TrustZone." https://www.arm.com/products/security-on-arm/trustzone.

[ARMh] ARM. "Development of TEE and Secure Monitor Code." https://www.arm.com/why-arm/technologies/trustzone-for-cortex-a/tee-and-smc.

[ARMi] ARM. "The Memory Management Unit (MMU)." https://developer.arm.com/architectures/learn-the-architecture/memory-management/the-memory-management-unit-mmu.

[ARMj] ARM Compute Library. https://developer.arm.com/ip-products/processors/machine-learning/compute-library.

[ARMk] ARM NN (Neural Network). https://github.com/ARM-software/armnn.

[ARMl] ARM Trusted Firmware. https://github.com/ARM-software/arm-trusted-firmware.

[AS19] Julien Amacher and Valerio Schiavoni. "On the Performance of ARM TrustZone." In José Pereira and Laura Ricci, editors, Distributed Applications and Interoperable Systems, pp. 133–151, Cham, 2019. Springer International Publishing.

[ATA02] Abdulaziz Alhadlaq, Jun Tang, Marwan Almaymoni, and Aleksandra Korolova. "Privacy in the Amazon Alexa skills ecosystem." Star, **217**(11), 1902.

[ATG16] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. "SCONE: Secure Linux Containers with Intel SGX." In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 689–703, Savannah, GA, 2016. USENIX Association.

[BCC16] Endri Bregu, Nicola Casamassima, Daniel Cantoni, Luca Mottola, and Kamin Whitehouse. "Reactive Control of Autonomous Drones." In Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '16, pp. 207–219, New York, NY, USA, 2016. ACM.

[Beb] Parrot Bebop. https://www.parrot.com/us/Drones/Parrot-bebop-2.

[BGA15] Sunanda Bose, Atrayee Gupta, Sriyanjana Adhikary, and Nandini Mukherjee. "Towards a Sensor-Cloud Infrastructure with Sensor Virtualization." In Proceedings of the Second Workshop on Mobile Sensing, Computing and Communication, MSCC '15, pp. 25–30, New York, NY, USA, 2015. ACM.

[BKL16a] Ferdinand Brasser, Daeyoung Kim, Christopher Liebchen, Vinod Ganapathy, Liviu Iftode, and Ahmad-Reza Sadeghi. "Regulating ARM TrustZone Devices in Restricted Spaces." In Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '16, pp. 413–425, New York, NY, USA, 2016. ACM.

[BKL16b] Ferdinand Brasser, Daeyoung Kim, Christopher Liebchen, Vinod Ganapathy, Liviu Iftode, and Ahmad-Reza Sadeghi. "Regulating ARM TrustZone Devices in Restricted Spaces." In Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '16, p. 413–425, New York, NY, USA, 2016. Association for Computing Machinery.

[BLL18] Sebanjila Kevin Bukasa, Ronan Lashermes, Hélène Le Bouder, Jean-Louis Lanet, and Axel Legay. "How TrustZone Could Be Bypassed: Side-Channel Attacks on a Modern System-on-Chip." In Gerhard P. Hancke and Ernesto Damiani, editors, Information Security Theory and Practice, pp. 93–109, Cham, 2018. Springer International Publishing.

[Blo] Bloomberg. `https://www.bloomberg.com/news/articles/2016-04-04/drones-are-the-new-threat-to-airline-safety`.

[BMD17] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. "Software Grand Exposure: SGX Cache Attacks Are Practical." CoRR, **abs/1702.07521**, 2017.

[BRR09] Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. "Format-Preserving Encryption." In Michael J. Jacobson, Vincent Rijmen, and Reihaneh Safavi-Naini, editors, Selected Areas in Cryptography, pp. 295–312, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[BSP16] Helena Brekalo, Raoul Strackx, and Frank Piessens. "Mitigating Password Database Breaches with Intel SGX." In Proceedings of the 1st Workshop on System Software for Trusted Execution, SysTEX '16, pp. 1:1–1:6, New York, NY, USA, 2016. ACM.

[BVN15] Bharathan Balaji, Chetan Verma, Balakrishnan Narayanaswamy, and Yuvraj Agarwal. "Zodiac: Organizing Large Deployment of Sensors to Create Reusable Applications for Buildings." BuildSys '15, pp. 13–22, New York, NY, USA, 2015. ACM.

[BWK17] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. "Telling Your Secrets without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution." In 26th USENIX Security Symposium (USENIX Security 17), pp. 1041–1056, Vancouver, BC, August 2017. USENIX Association.

[Byt] BytecodeAlliance. "Wasm Micro-Runtime." `https://github.com/bytecodealliance/wasm-micro-runtime`.

[BZP10] C. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. Hart. "Enabling multi-tenancy: An industrial experience report." In 2010 IEEE International Conference on Software Maintenance, pp. 1–8, 2010.

[CAB18] Abraham A Clements, Naif Saleh Almakhdhub, Saurabh Bagchi, and Mathias Payer. "ACES: Automatic Compartments for Embedded Systems." In 27th USENIX Security Symposium (USENIX Security 18), pp. 65–82, Baltimore, MD, 2018. USENIX Association.

[Caf] Caffe2 on Smartphone. https://caffe2.ai/docs/mobile-integration.html.

[CAS17] A. A. Clements, N. S. Almakhdhub, K. S. Saab, P. Srivastava, J. Koo, S. Bagchi, and M. Payer. "Protecting Bare-Metal Embedded Systems with Privilege Overlays." In 2017 IEEE Symposium on Security and Privacy (SP), pp. 289–303, 2017.

[CDO17] Vittorio Cozzolino, Aaron Yi Ding, Jorg Ott, and Dirk Kutscher. "Enabling Fine-Grained Edge Offloading for IoT." In Proceedings of the SIGCOMM Posters and Demos, SIGCOMM Posters and Demos '17, p. 124–126, New York, NY, USA, 2017. Association for Computing Machinery.

[CFP18] Z Berkay Celik, Earlence Fernandes, Eric Pauley, Gang Tan, and Patrick McDaniel. "Program Analysis of Commodity IoT Applications for Security and Privacy: Challenges and Opportunities." arXiv preprint arXiv:1809.06962, 2018.

[CGK16] Ioannis P. Chochliouros, Ioannis Giannoulakis, Tassos Kourtis, Maria Belesioti, Evangelos Sfakianakis, Anastasia S. Spiliopoulou, Nikolaos Bompetsis, Emmanouil Kafetzakis, Leonardo Goratti, and Athanassios Dardamanis. "A Model for an Innovative 5G-Oriented Architecture, Based on Small Cells Coordination for Multi-tenancy and Edge Services." In Lazaros Iliadis and Ilias Maglogiannis, editors, Artificial Intelligence Applications and Innovations, pp. 666–675, Cham, 2016. Springer International Publishing.

[Cla20] Lin Clark. "Announcing the Bytecode Alliance: Building a secure by default, composable future for WebAssembly." https://bytecodealliance.org/articles/announcing-the-bytecode-alliance, 2020. Last accessed: 2020-07-29.

[CMG15] S. Cherrier, Z. Movahedi, and Y. M. Ghamri-Doudane. "Multi-tenancy in decentralised IoT." In 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), pp. 256–261, 2015.

[CMT18] Z. Berkay Celik, Patrick McDaniel, and Gang Tan. "Soteria: Automated IoT Safety and Security Analysis." In 2018 USENIX Annual Technical Conference (USENIX ATC 18), pp. 147–158, Boston, MA, 2018. USENIX Association.

[Com] WebAssembly Community. "WebAssembly." https://webassembly.org/.

[CTMa] Z Berkay Celik, Gang Tan, and Patrick D McDaniel. "IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT.".

[CTMb] Z Berkay Celik, Gang Tan, and Patrick D McDaniel. "IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT.".

[CZG15] Patrick Colp, Jiawen Zhang, James Gleeson, Sahil Suneja, Eyal de Lara, Himanshu Raj, Stefan Saroiu, and Alec Wolman. "Protecting Data on Smartphones and Tablets from Memory Attacks." In Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, pp. 177–189, New York, NY, USA, 2015. ACM.

[CZK18] Haehyun Cho, Penghui Zhang, Donguk Kim, Jinbum Park, Choong-Hoon Lee, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. "Prime+Count: Novel Cross-World Covert Channels on ARM TrustZone." In Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC '18, p. 441–452, New York, NY, USA, 2018. Association for Computing Machinery.

[CZR17] Sanchuan Chen, Xiaokuan Zhang, Michael K. Reiter, and Yinqian Zhang. "Detecting Privileged Side-Channel Attacks in Shielded Execution with Déjà Vu." In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, ASIA CCS '17, p. 7–18, New York, NY, USA, 2017. Association for Computing Machinery.

[DH18] Wenbo Ding and Hongxin Hu. "On the safety of iot device physical interaction control." In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 832–846. ACM, 2018.

[DJI] DJI. `http://www.dji.com/`.

[DJZ15] Weiqi Dai, Hai Jin, Deqing Zou, Shouhuai Xu, Weide Zheng, Lei Shi, and Laurence Tianruo Yang. "TEE: A virtual DRTM based execution environment for secure cloud-end computing." Future Generation Computer Systems, **49**:47 – 57, 2015.

[DKT13] Stephen Dawson-Haggerty, Andrew Krioukov, Jay Taneja, Sagar Karandikar, Gabe Fierro, Nikita Kitaev, and David Culler. "BOSS: Building Operating System Services." In Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), pp. 443–457, Lombard, IL, 2013. USENIX.

[DKW11] Karthik Dantu, Bryan Kate, Jason Waterman, Peter Bailis, and Matt Welsh. "Programming Micro-aerial Vehicle Swarms with Karma." In Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems, SenSys '11, pp. 121–134, New York, NY, USA, 2011. ACM.

[DL4] DL4J. `https://deeplearning4j.org/`.

[DMA10] Colin Dixon, Ratul Mahajan, Sharad Agarwal, A. J. Brush, Bongshin Lee, Stefan Saroiu, and Victor Bahl. "The Home Needs an Operating System (and an App Store)." In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX, pp. 18:1–18:6, New York, NY, USA, 2010. ACM.

[DMA12] Colin Dixon, Ratul Mahajan, Sharad Agarwal, A. J. Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. "An Operating System for the Home." In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, NSDI'12, pp. 25–25, Berkeley, CA, USA, 2012. USENIX Association.

[DMS14] D. Danner, R. Müller, W. Schröder-Preikschat, W. Hofer, and D. Lohmann. "SAFER SLOTH: Efficient, hardware-tailored memory protection." In 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 37–48, 2014.

[Dwo15] Morris J Dworkin. "SHA-3 standard: Permutation-based hash and extendable-output functions." Technical report, 2015.

[EGC10] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. "TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones." In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, pp. 393–407, Berkeley, CA, USA, 2010. USENIX Association.

[EM09a] P. Evensen and H. Meling. "SenseWrap: A service oriented middleware with sensor virtualization and self-configuration." In 2009 International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), pp. 261–266, Dec 2009.

[EM09b] Pål Evensen and Hein Meling. "Sensor Virtualization with Self-configuration and Flexible Interactions." In Proceedings of the 3rd ACM International Workshop on Context-Awareness for Self-Managing Systems, Casemans '09, pp. 31–38, New York, NY, USA, 2009. ACM.

[Ens16] Thomas Ensergueix. "Cortex-M23 and Cortex-M33 - Security foundation for billions of devices.", October 2016.

[ESC12] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. "Neural acceleration for general-purpose approximate programs." In 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 449–460. IEEE, 2012.

[eve] `https://event38.com/drone-data-management-system/`.

[FAA] FAA. "FAA aerospace forecast fiscal year 2016 - 2036.".

[FGT16] Julien Fleureau, Quentin Galvane, Francois-Louis Tariolle, and Philippe Guillotel. "Generic Drone Control Platform for Autonomous Capture of Cinema Scenes."

In Proceedings of the 2Nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use, DroNet '16, pp. 35–40, New York, NY, USA, 2016. ACM.

[FPR16] 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, pp. 531–548, 2016.

[GES17] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. "Cache Attacks on Intel SGX." In Proceedings of the 10th European Workshop on Systems Security, EuroSec'17, New York, NY, USA, 2017. Association for Computing Machinery.

[GGG04] Frida Gunnarsson, Fredrik Gustafsson, and Fredrik Gunnarsson. "Frequency Analysis Using Non-Uniform Sampling With Application To Active Queue Management.", 2004.

[GHZ18] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. "Securing input data of deep learning inference systems via partitioned enclave execution." arXiv preprint arXiv:1807.00969, 2018.

[GLX17] Le Guan, Peng Liu, Xinyu Xing, Xinyang Ge, Shengzhi Zhang, Meng Yu, and Trent Jaeger. "TrustShadow: Secure Execution of Unmodified Applications with ARM TrustZone." In Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '17, p. 488–501, New York, NY, USA, 2017. Association for Computing Machinery.

[GNK19] David Goltzsche, Manuel Nieke, Thomas Knauth, and Rüdiger Kapitza. "AccTEE: A WebAssembly-Based Two-Way Sandbox for Trusted Resource Accounting." In Proceedings of the 20th International Middleware Conference, Middleware '19, p. 123–135, New York, NY, USA, 2019. Association for Computing Machinery.

[Goo] Google. "Nest Protect." `https://store.google.com/us/product/nest_protect_2nd_gen`.

[gov] govtech. `http://www.govtech.com/products/Analysis-Drones-Posing-Global-Security-Issues.html`.

[GPH11] Kevin Gudeth, Matthew Pirretti, Katrin Hoeper, and Ron Buskey. "Delivering Secure Applications on Commercial Mobile Devices: The Case for Bare Metal Hypervisors." In Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM '11, p. 33–38, New York, NY, USA, 2011. Association for Computing Machinery.

[GS03] Karlheinz Gröchenig and Harald Schwab. "Fast Local Reconstruction Methods for Nonuniform Sampling in Shift Invariant Spaces." SIAM Journal on Matrix Analysis and Applications, **2002**:24–899, 2003.

[HCK18] D. Hwang, J. Choi, and K. Kim. "Dynamic Access Control Scheme for IoT Devices using Blockchain." In 2018 International Conference on Information and Communication Technology Convergence (ICTC), pp. 713–715, 2018.

[HCP17] Marcus Hähnel, Weidong Cui, and Marcus Peinado. "High-Resolution Side Channels for Untrusted Operating Systems." In 2017 USENIX Annual Technical Conference (USENIX ATC 17), pp. 299–312, Santa Clara, CA, July 2017. USENIX Association.

[HLL18] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. "Amc: Automl for model compression and acceleration on mobile devices." In Proceedings of the European Conference on Computer Vision (ECCV), pp. 784–800, 2018.

[HMD15] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149, 2015.

[HOW13] Dezhi Hong, Jorge Ortiz, Kamin Whitehouse, and David Culler. "Towards Automatic Spatial Verification of Sensor Placement in Buildings." In Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings, BuildSys'13, pp. 13:1–13:8, New York, NY, USA, 2013. ACM.

[HPC18] J. Hochstetler, R. Padidela, Q. Chen, Q. Yang, and S. Fu. "Embedded Deep Learning for Vehicular Edge Computing." In 2018 IEEE/ACM Symposium on Edge Computing (SEC), pp. 341–343, 2018.

[HQS17] Hadi Habibzadeh, Zhou Qin, Tolga Soyata, and Burak Kantarci. "Large-scale distributed dedicated-and non-dedicated smart city sensing systems." IEEE Sensors Journal, **17**(23):7649–7658, 2017.

[HR19] Adam Hall and Umakishore Ramachandran. "An Execution Model for Serverless Functions at the Edge." In Proceedings of the International Conference on Internet of Things Design and Implementation, IoTDI '19, p. 225–236, New York, NY, USA, 2019. Association for Computing Machinery.

[HRS17] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. "Bringing the Web up to Speed with WebAssembly." SIGPLAN Not., **52**(6):185–200, June 2017.

[HSS18] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. "Chiron: Privacy-preserving Machine Learning as a Service." CoRR, **abs/1803.05961**, 2018.

[HZ19] M. F. A. Hamid and F. H. K. Zaman. "Hand Gesture Recognition Using Movidius Neural Compute Stick." In 2019 IEEE 9th International Conference on System Engineering and Technology (ICSET), pp. 510–514, 2019.

[HZC17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." CoRR, **abs/1704.04861**, 2017.

[HZR15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." CoRR, **abs/1512.03385**, 2015.

[HZX16] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. "Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data." In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 533–549, Savannah, GA, 2016. USENIX Association.

[II16] Michael C. Long II. "Attack and Defend: Linux Privilege Escalation Techniques of 2016." 2016.

[ILH13] Md. Motaharul Islam, Jun Hyuk Lee, and Eui-Nam Huh. "An Efficient Model for Smart Home by the Virtualization of Wireless Sensor Network." International Journal of Distributed Sensor Networks, **9**(2):168735, 2013.

[IMA16] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size." CoRR, **abs/1602.07360**, 2016.

[Inta] Intel. `https://software.intel.com/en-us/sgx`.

[Intb] Intel. "Intel Neural Stick 2." `https://ark.intel.com/content/www/us/en/ark/products/140109/intel-neural-compute-stick-2.html`.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." CoRR, **abs/1502.03167**, 2015.

[JGP19] Kumseok Jung, Julien Gascon-Samson, and Karthik Pattabiraman. "OneOS: IoT Platform based on POSIX and Actors." In 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19), Renton, WA, July 2019. USENIX Association.

[JLK14] Daehee Jang, Hojoon Lee, Minsu Kim, Daehyeok Kim, Daegyeong Kim, and Brent Byunghoon Kang. "ATRA: Address Translation Redirection Attack against Hardware-Based External Monitors." In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, p. 167–178, New York, NY, USA, 2014. Association for Computing Machinery.

[JNG17] Shubham Jain, Viet Nguyen, Marco Gruteser, and Paramvir Bahl. "Panoptes: Servicing Multiple Applications Simultaneously Using Steerable Cameras." In Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN '17, pp. 119–130, New York, NY, USA, 2017. ACM.

[JSS19] Hyuk-Jin Jeong, Chang Hyun Shin, Kwang Yong Shin, Hyeon-Jae Lee, and Soo-Mook Moon. "Seamless Offloading of Web App Computations From Mobile Device to Edge Clouds via HTML5 Web Worker Migration." In Proceedings of the ACM Symposium on Cloud Computing, SoCC '19, p. 38–49, New York, NY, USA, 2019. Association for Computing Machinery.

[JZ] Jean-Philippe Aumasson Jack O'Connor, Samuel Neves and Zooko. "BLAKE3." `https://github.com/BLAKE3-team/BLAKE3`.

[KB18] Neil Klingensmith and Suman Banerjee. "Hermes: A Real Time Hypervisor for Mobile and IoT Systems." In Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications, pp. 101–106, 2018.

[KHF19] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. "Spectre Attacks: Exploiting Speculative Execution." In 2019 IEEE Symposium on Security and Privacy (SP), pp. 1–19, 2019.

[KK15] S. H. Kim and D. Kim. "Enabling Multi-Tenancy via Middleware-Level Virtualization with Organization Management in the Cloud of Things." IEEE Transactions on Services Computing, **8**(6):971–984, Nov 2015.

[KKS18] Esmaeil Mohammadian Koruyeh, Khaled N. Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. "Spectre Returns! Speculation Attacks using the Return Stack Buffer." In 12th USENIX Workshop on Offensive Technologies (WOOT 18), Baltimore, MD, August 2018. USENIX Association.

[KKS19] K. N. Khasawneh, E. M. Koruyeh, C. Song, D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh. "SafeSpec: Banishing the Spectre of a Meltdown with Leakage-Free Speculation." In 2019 56th ACM/IEEE Design Automation Conference (DAC), pp. 1–6, 2019.

[KLL15] JeongGil Ko, Byung-Bog Lee, Kyesun Lee, Sang Gi Hong, Naesoo Kim, and Jeongyeup Paek. "Sensor Virtualization Module: Virtualizing IoT Devices on Mobile Smartphones for Effective Sensor Data Management." International Journal of Distributed Sensor Networks, **11**(10):730762, 2015.

[KSK19] Donghyun Kwon, Jangseop Shin, Giyeol Kim, Byoungyoung Lee, Yeongpil Cho, and Yunheung Paek. "uXOM: Efficient eXecute-Only Memory on ARM Cortex-M." In 28th USENIX Security Symposium (USENIX Security 19), pp. 231–247, Santa Clara, CA, August 2019. USENIX Association.

[LBL16] Chieh-Jan Mike Liang, Lei Bu, Zhao Li, Junbei Zhang, Shi Han, Börje F. Karlsson, Dongmei Zhang, and Feng Zhao. "Systematically Debugging IoT Control System Correctness for Building Automation." In Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments, BuildSys '16, pp. 133–142, New York, NY, USA, 2016. ACM.

[LCG17] Amit Levy, Bradford Campbell, Branden Ghena, Daniel B. Giffin, Pat Pannuto, Prabal Dutta, and Philip Levis. "Multiprogramming a 64kB Computer Safely and Efficiently." In Proceedings of the 26th Symposium on Operating Systems Principles, SOSP '17, p. 234–251, New York, NY, USA, 2017. Association for Computing Machinery.

[LJJ15] Renju Liu, Lintong Jiang, Ningzhe Jiang, and Felix Xiaozhu Lin. "Anatomizing System Activities on Interactive Wearable Devices." In Proceedings of the 6th Asia-Pacific Workshop on Systems, APSys '15, pp. 18:1–18:7, New York, NY, USA, 2015. ACM.

[LJL17] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. "Oblivious Neural Network Predictions via MiniONN Transformations." In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17, pp. 619–631, New York, NY, USA, 2017. ACM.

[LKL15a] 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 Proceedings of the 14th International Conference on Information Processing in Sensor Networks, pp. 298–309. ACM, 2015.

[LKL15b] 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 Proceedings of the 14th International Conference on Information Processing in Sensor Networks, pp. 298–309. ACM, 2015.

[LKP20] Daniel Lehmann, Johannes Kinder, and Michael Pradel. "Everything Old is New Again: Binary Security of WebAssembly." In 29th {USENIX} Security Symposium ({USENIX} Security 20), pp. 217–234, 2020.

[LL16] Renju Liu and Felix Xiaozhu Lin. "Understanding the Characteristics of Android Wear OS." In Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '16, pp. 151–164, New York, NY, USA, 2016. ACM.

[LLC15] Wenhao Li, Haibo Li, Haibo Chen, and Yubin Xia. "AdAttester: Secure Online Mobile Advertisement Attestation Using TrustZone." In Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '15, p. 75–88, New York, NY, USA, 2015. Association for Computing Machinery.

[LLP19] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. "Occlumency: Privacy-Preserving Remote Deep-Learning Inference Using SGX." In The 25th Annual International Conference on Mobile Computing and Networking, MobiCom '19, New York, NY, USA, 2019. Association for Computing Machinery.

[LMN18] R. C. Lunardi, R. A. Michelin, C. V. Neu, and A. F. Zorzo. "Distributed access control on IoT ledger-based architecture." In NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, pp. 1–7, 2018.

[LOD18a] H. Li, K. Ota, and M. Dong. "Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing." IEEE Network, **32**(1):96–101, 2018.

[LOD18b] He Li, Kaoru Ota, and Mianxiong Dong. "Learning IoT in edge: Deep learning for the Internet of Things with edge computing." IEEE network, **32**(1):96–101, 2018.

[LPZ20] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. "Reducto: On-Camera Filtering for Resource-Efficient Real-Time Video Analytics." In Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '20, p. 359–376, New York, NY, USA, 2020. Association for Computing Machinery.

[LS17] Renju Liu and Mani Srivastava. "PROTC: PROTeCting Drone's Peripherals Through ARM TrustZone." In Proceedings of the 3rd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications, DroNet '17, pp. 1–6, New York, NY, USA, 2017. ACM.

[LS18] Renju Liu and Mani Srivastava. VirtSense: Virtualize Sensing through ARM TrustZone on Internet-of-Things, p. 2–7. Association for Computing Machinery, New York, NY, USA, 2018.

[LSD18a] Matthew Lentz, Rijurekha Sen, Peter Druschel, and Bobby Bhattacharjee. "SeCloak: ARM Trustzone-based Mobile Peripheral Control." In Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '18, pp. 1–13, New York, NY, USA, 2018. ACM.

[LSD18b] Matthew Lentz, Rijurekha Sen, Peter Druschel, and Bobby Bhattacharjee. "Secloak: Arm trustzone-based mobile peripheral control." In Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services, pp. 1–13. ACM, 2018.

[LSG18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. "Meltdown: Reading Kernel Memory from User Space." In 27th USENIX Security Symposium (USENIX Security 18), pp. 973–990, Baltimore, MD, August 2018. USENIX Association.

[LSJ19] E. S. Lage, R. L. Santos, S. M. T. Junior, and F. Andreotti. "Low-Cost IoT Surveillance System Using Hardware-Acceleration and Convolutional Neural Networks." In 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), pp. 931–936, 2019.

[LSW12a] He Liu, Stefan Saroiu, Alec Wolman, and Himanshu Raj. "Software Abstractions for Trusted Sensors." In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, MobiSys '12, pp. 365–378, New York, NY, USA, 2012. ACM.

[LSW12b] He Liu, Stefan Saroiu, Alec Wolman, and Himanshu Raj. "Software Abstractions for Trusted Sensors." In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, MobiSys '12, p. 365–378, New York, NY, USA, 2012. Association for Computing Machinery.

[LYZ17] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao. "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications." IEEE Internet of Things Journal, **4**(5):1125–1142, Oct 2017.

[LZC18] En Li, Zhi Zhou, and Xu Chen. "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy." In Proceedings of the 2018 Workshop on Mobile Edge Communications, pp. 31–36, 2018.

[MAH16] M. Mohsin, Z. Anwar, G. Husari, E. Al-Shaer, and M. A. Rahman. "IoTSAT: A formal framework for security analysis of the internet of things (IoT)." In 2016 IEEE Conference on Communications and Network Security (CNS), pp. 180–188, Oct 2016.

[MCS17] Kai Mast, Lequn Chen, and Emin Gün Sirer. "Scaling Databases Through Trusted Hardware Proxies." In Proceedings of the 2Nd Workshop on System Software for Trusted Execution, SysTEX'17, pp. 9:1–9:6, New York, NY, USA, 2017. ACM.

[Mic] Microsoft. https://azure.microsoft.com/en-us/services/iot-edge/.

[Mic19] Microsoft. "How to provision for multitenancy." https://docs.microsoft.com/en-us/azure/iot-dps/how-to-provision-multitenant, 2019. Last accessed: 2019-04-10.

[MMH17] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. "IoT Sentinel: Automated device-type identification for security enforcement in IoT." In 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 2177–2184. IEEE, 2017.

[Mon] Monsoon Solutions Inc. https://www.msoon.com/online-store.

[mot] "Motochopper." http://hexamob.com/how-to-root/motochopper-method/.

[MPS17] Meiyi Ma, Sarah Masud Preum, and John A. Stankovic. "CityGuard: A Watchdog for Safety-Aware Conflict Detection in Smart Cities." In Proceedings of the Second International Conference on Internet-of-Things Design and Implementation, IoTDI '17, pp. 259–270, New York, NY, USA, 2017. ACM.

[MPT] M. Ma, S. M. Preum, W. Tarneberg, M. Ahmed, M. Ruiters, and J. Stankovic. "Detection of Runtime Conflicts among Services in Smart Cities." In 2016 IEEE International Conference on Smart Computing (SMARTCOMP), pp. 1–10.

[MSF18] Meiyi Ma, John A. Stankovic, and Lu Feng. "Cityresolver: A Decision Support System for Conflict Resolution in Smart Cities." In Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS '18, pp. 55–64, Piscataway, NJ, USA, 2018. IEEE Press.

[NAV] NAVIO2. https://emlid.com/introducing-navio2/.

[Nor] Nordic Semiconductor. https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840.

[NRB18] Alessandro A. Nacci, Vincenzo Rana, Bharathan Balaji, Paola Spoletini, Rajesh Gupta, Donatella Sciuto, and Yuvraj Agarwal. "BuildingRules: A Trigger-Action–Based System to Manage Complex Commercial Buildings." ACM Trans. Cyber-Phys. Syst., **2**(2):13:1–13:22, May 2018.

[NTG19] Joseph Noor, Hsiao-Yun Tseng, Luis Garcia, and Mani Srivastava. "DDFlow: visualized declarative programming for heterogeneous IoT networks." In Proceedings of the International Conference on Internet of Things Design and Implementation, pp. 172–177. ACM, 2019.

[NXP] NXP. "NXP EdgeReady." `https://www.nxp.com/applications/solutions/enabling-technologies/edgeverse/edgeready:EDGEREADY`.

[OBS18] Abdallah Zoubir Ourad, Boutheyna Belgacem, and Khaled Salah. "Using Blockchain for IOT Access Control and Authentication Management." In Dimitrios Georgakopoulos and Liang-Jie Zhang, editors, Internet of Things – ICIOT 2018, pp. 150–164, Cham, 2018. Springer International Publishing.

[OG18] Samuel S Ogden and Tian Guo. "{MODI}: Mobile Deep Inference Made Efficient by Edge Computing." In {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18), 2018.

[opt] "Open Portable Trusted Execution Environment." `https://www.op-tee.org/`.

[OSD19] Iakushkin Oleg, Ruslan Sevostyanov, Alexander Degtyarev, P. E. Karpiy, E. G. Kuzevanova, A. A. Kitaeva, and S. A. Sergiev. "Position Tracking in 3D Space Based on a Data of a Single Camera." In Sanjay Misra, Osvaldo Gervasi, Beniamino Murgante, Elena Stankova, Vladimir Korkhov, Carmelo Torre, Ana Maria A.C. Rocha, David Taniar, Bernady O. Apduhan, and Eufemia Tarantino, editors, Computational Science and Its Applications – ICCSA 2019, pp. 772–781, Cham, 2019. Springer International Publishing.

[OSF16a] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. "Oblivious Multi-Party Machine Learning on Trusted Processors." In 25th USENIX Security Symposium (USENIX Security 16), pp. 619–636, Austin, TX, 2016. USENIX Association.

[OSF16b] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. "Oblivious Multi-Party Machine Learning on Trusted Processors." In 25th USENIX Security Symposium (USENIX Security 16), pp. 619–636, Austin, TX, August 2016. USENIX Association.

[OSS20] Seyed Ali Osia, Ali Shahin Shamsabadi, Sina Sajadmanesh, Ali Taheri, Kleomenis Katevas, Hamid R Rabiee, Nicholas D Lane, and Hamed Haddadi. "A hybrid deep learning architecture for privacy-preserving mobile analytics." IEEE Internet of Things Journal, 2020.

[OTK18] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. "Varys: Protecting SGX Enclaves from Practical Side-Channel Attacks."

In 2018 USENIX Annual Technical Conference (USENIX ATC 18), pp. 227–240, Boston, MA, July 2018. USENIX Association.

[PBC14] Johann-Sebastian Pleban, Ricardo Band, and Reiner Creutzburg. "Hacking and securing the AR. Drone 2.0 quadcopter: investigations for improving the security of a toy." In IS&T/SPIE Electronic Imaging, pp. 90300L–90300L. International Society for Optics and Photonics, 2014.

[PDK19] Guntur Dharma Putra, Volkan Dedeoglu, Salil S. Kanhere, and Raja Jurdak. "Trust Management in Decentralized IoT Access Control System.", 2019.

[PIX] PIX4D. `https://pix4d.com/`.

[PMS16] Nicolas Papernot, Patrick D. McDaniel, Arunesh Sinha, and Michael P. Wellman. "Towards the Science of Security and Privacy in Machine Learning." CoRR, **abs/1611.03814**, 2016.

[PPN15] A. Piscitello, F. Paduano, A. A. Nacci, D. Noferi, M. D. Santambrogio, and D. Sciuto. "Danger-system: Exploring new ways to manage occupants safety in smart building." In 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), pp. 675–680, Dec 2015.

[PPR18] R. Pan, G. Peach, Y. Ren, and G. Parmer. "Predictable Virtualization on Memory Protection Unit-Based Microcontrollers." In 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 62–74, 2018.

[PX4] PX4. `https://dev.px4.io/`.

[PyT] PyTorch on Android. `https://pytorch.org/mobile/android/`.

[QEM] QEMU. `https://www.qemu.org/`.

[ret09] "Return-Oriented Rootkits: Bypassing Kernel Code Integrity Protection Mechanisms." In Presented as part of the 18th USENIX Security Symposium (USENIX Security 09), Montreal, Canada, 2009. USENIX.

[RF16] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger." CoRR, **abs/1612.08242**, 2016.

[Rob] Erle Robotics. `http://docs.erlerobotics.com/`.

[SAB17a] Milijana Surbatovich, Jassim Aljuraidan, Lujo Bauer, Anupam Das, and Limin Jia. "Some Recipes Can Do More Than Spoil Your Appetite: Analyzing the Security and Privacy Risks of IFTTT Recipes." In Proceedings of the 26th International Conference on World Wide Web, WWW '17, pp. 1501–1510, Perth, Australia, 2017.

[SAB17b] Milijana Surbatovich, Jassim Aljuraidan, Lujo Bauer, Anupam Das, and Limin Jia. "Some Recipes Can Do More Than Spoil Your Appetite: Analyzing the Security and Privacy Risks of IFTTT Recipes." In Proceedings of the 26th International Conference on World Wide Web, WWW '17, pp. 1501–1510, 2017.

[Sam] SamSung. `https://www.smartthings.com/`.

[Sat17] M. Satyanarayanan. "The Emergence of Edge Computing." Computer, **50**(1):30–39, 2017.

[SC20] Zhuojia Shen and John Criswell. "Fast Execute-Only Memory for Embedded Systems.", 2020.

[SCZ16] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. "Edge Computing: Vision and Challenges." IEEE Internet of Things Journal, **3**(5):637–646, 2016.

[SD16a] M. Samaniego and R. Deters. "Supporting IoT Multi-Tenancy on Edge Devices." In 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 66–73, 2016.

[SD16b] W. Shi and S. Dustdar. "The Promise of Edge Computing." Computer, **49**(5):78–81, 2016.

[Sec] Blink Home Security. "Blink XT2." `https://blinkforhome.com/products`.

[Sen] SenseFly. `https://www.sensefly.com/applications/gis.html`.

[SFH12] Fred Samland, Jana Fruth, Mario Hildebrandt, Tobias Hoppe, and Jana Dittmann. "AR. Drone: security threat analysis and exemplary attack to track persons." In Proceedings of the SPIE, volume 8301, 2012.

[SGF17] Sajin Sasy, Sergey Gorbunov, and Christopher W Fletcher. "ZeroTrace: Oblivious Memory Primitives from Intel SGX." IACR Cryptology ePrint Archive, **2017**:549, 2017.

[SHC19] M. Salehi, D. Hughes, and B. Crispo. "MicroGuard: Securing Bare-Metal Microcontrollers against Code-Reuse Attacks." In 2019 IEEE Conference on Dependable and Secure Computing (DSC), pp. 1–8, 2019.

[She15] Di Shen. "Exploiting TrustZone on android." Black Hat USA, 2015.

[SHT10] Daniel Sangorrin, Shinya Honda, and Hiroaki Takada. "Dual operating system architecture for real-time embedded systems." In Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT), Brussels, Belgium, pp. 6–15, 2010.

[SHZ18] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation." CoRR, **abs/1801.04381**, 2018.

[SIS11] Navin K. Sharma, David E. Irwin, Prashant J. Shenoy, and Michael Zink. "MultiSense: Fine-grained Multiplexing for Steerable Camera Sensor Networks." In Proceedings of the Second Annual ACM Conference on Multimedia Systems, MMSys '11, pp. 23–34, New York, NY, USA, 2011. ACM.

[Sol] 3DR Solo. `https://3dr.com/solo-drone/`.

[SRS14] Nuno Santos, Himanshu Raj, Stefan Saroiu, and Alec Wolman. "Using ARM Trustzone to Build a Trusted Language Runtime for Mobile Applications." In Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '14, pp. 67–80, New York, NY, USA, 2014. ACM.

[SSK15] Yunmok Son, Hocheol Shin, Dongkwan Kim, Youngseok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, and Yongdae Kim. "Rocking Drones with Intentional Sound Noise on Gyroscopic Sensors." In 24th USENIX Security Symposium (USENIX Security 15), pp. 881–896, Washington, D.C., 2015. USENIX Association.

[SSP16] Chenguang Shen, Rayman Preet Singh, Amar Phanishayee, Aman Kansal, and Ratul Mahajan. "Beam: Ending Monolithic Applications for Connected Devices." In 2016 USENIX Annual Technical Conference (USENIX ATC 16), pp. 143–157, Denver, CO, 2016. USENIX Association.

[SSW15a] He Sun, Kun Sun, Yuewu Wang, and Jiwu Jing. "TrustOTP: Transforming Smartphones into Secure One-Time Password Tokens." In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15, p. 976–988, New York, NY, USA, 2015. Association for Computing Machinery.

[SSW15b] He Sun, Kun Sun, Yuewu Wang, Jiwu Jing, and Haining Wang. "Trustice: Hardware-assisted isolated computing environments on mobile devices." In Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on, pp. 367–378. IEEE, 2015.

[Sta14] J. A. Stankovic. "Research Directions for the Internet of Things." IEEE Internet of Things Journal, **1**(1):3–9, Feb 2014.

[STB16] Farzad Samie, Vasileios Tsoutsouras, Lars Bauer, Sotirios Xydis, Dimitrios Soudris, and Jörg Henkel. "Computation offloading and resource allocation for low-power IoT edge devices." In 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), pp. 7–12. IEEE, 2016.

[STL20] STL. "5G and edge computing: Why does 5G needs edge compute?", Jun 2020.

[SWB16] Seung-Hyun Seo, Jongho Won, Elisa Bertino, Yousung Kang, and Dooho Choi. "A Security Framework for a Drone Delivery Service." In Proceedings of the 2Nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use, DroNet '16, pp. 29–34, New York, NY, USA, 2016. ACM.

[SWY15] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions." In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9, 2015.

[TAC20] Rahmadi Trimananda, Seyed Amir Aqajari, Jason Chuang, Brian Demsky, Guoqing Harry Xu, and Shan Lu. "Understanding and Automatically Detecting Conflicting Interactions between Smart Home IoT Applications." In Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020.

[TB19] Florian Tramer and Dan Boneh. "Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware." In International Conference on Learning Representations, 2019.

[Ten] Tensorflow Lite on Android. `https://www.tensorflow.org/lite/guide/android`.

[TGS18] Shruti Tople, Karan Grover, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. "Privado: Practical and Secure DNN Inference." CoRR, **abs/1810.00602**, 2018.

[tow] "towelroot." `https://towelroot.com/`.

[TSG18] Salman Taherizadeh, Vlado Stankovski, and Marko Grobelnik. "A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers." Sensors, **18**(9):2938, 2018.

[VFC18] D. Velasco-Montero, J. Fernández-Berni, R. Carmona-Galán, and Á. Rodríguez-Vázquez. "On-The-Fly Deployment of Deep Neural Networks on Heterogeneous Hardware in a Low-Cost Smart Camera." In Proceedings of the 12th International Conference on Distributed Smart Cameras, ICDSC '18, New York, NY, USA, 2018. Association for Computing Machinery.

[vro] "vRoot." `https://androidmtk.com/download-vroot`.

[was] `https://tinyurl.com/yxwtnnra`.

[WBC19] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, et al. "Machine learning at facebook: Understanding inference at the edge." In 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 331–344. IEEE, 2019.

[WCY18] J. Wang, B. Cao, P. Yu, L. Sun, W. Bao, and X. Zhu. "Deep Learning towards Mobile Applications." In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pp. 1385–1393, 2018.

[Win08] Johannes Winter. "Trusted Computing Building Blocks for Embedded Linux-Based ARM Trustzone Platforms." In Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing, STC '08, p. 21–30, New York, NY, USA, 2008. Association for Computing Machinery.

[Win12] J. Winter. "Experimenting with ARM TrustZone – Or: How I Met Friendly Piece of Trusted Hardware." In 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, pp. 1161–1166, 2012.

[WRP19] Conrad Watt, John Renner, Natalie Popescu, Sunjay Cauligi, and Deian Stefan. "Ct-wasm: Type-driven secure cryptography for the web ecosystem." Proceedings of the ACM on Programming Languages, **3**(POPL):1–29, 2019.

[WW20] E. Wen and G. Weber. "Wasmachine: Bring IoT up to Speed with A WebAssembly OS." In 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pp. 1–4, 2020.

[WZB18] Ji Wang, Jianguo Zhang, Weidong Bao, Xiaomin Zhu, Bokai Cao, and Philip S. Yu. "Not Just Privacy: Improving Performance of Private Deep Learning in Mobile Cloud." In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD18, p. 2407–2416, New York, NY, USA, 2018. Association for Computing Machinery.

[XDH18] Xiaowei Xu, Yukun Ding, Sharon Xiaobo Hu, Michael Niemier, Jason Cong, Yu Hu, and Yiyu Shi. "Scaling for edge inference of deep neural networks." Nature Electronics, **1**(4):216–222, 2018.

[YH] Yusnani Mohd Yussoff and Habibah Hashim. "Trusted wireless sensor node platform.".

[YPH19] Moosa Yahyazadeh, Proyash Podder, Endadul Hoque, and Omar Chowdhury. "Expat: Expectation-based Policy Analysis and Enforcement for Appified Smart-Home Platforms." In Proceedings of the 24th ACM Symposium on Access Control Models and Technologies, SACMAT '19, pp. 61–72, New York, NY, USA, 2019. ACM.

[YSB16] J. Yapp, R. Seker, and R. Babiceanu. "UAV as a service: Enabling on-demand access and on-the-fly re-tasking of multi-tenant UAVs using cloud services." In 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), pp. 1–8, 2016.

[YZ18] Min Hong Yun and Lin Zhong. "Securing Real-Time Microcontroller Systems through Customized Memory View Switching." In 22nd Network and Distributed Security Symposium (NDSS 2018), San Diego, CA, 2018.

[YZ19] Min Hong Yun and Lin Zhong. "Ginseng: Keeping Secrets in Registers When You Distrust the Operating System." In 23rd Network and Distributed Security Symposium (NDSS 2019), San Diego, CA, 2019.

[Zap] Zapier. `https://zapier.com/`.

[ZDB17] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. "Opaque: An Oblivious and Encrypted Distributed Analytics Platform." In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), pp. 283–298, Boston, MA, 2017. USENIX Association.

[Zep] Zephyr. `https://www.zephyrproject.org/`.

[ZLD11] Lu Zhao, Guodong Li, Bjorn De Sutter, and John Regehr. "ARMor: Fully Verified Software Fault Isolation." In Proceedings of the Ninth ACM International Conference on Embedded Software, EMSOFT '11, p. 289–298, New York, NY, USA, 2011. Association for Computing Machinery.

[ZLD20] Guangxu Zhu, Dongzhu Liu, Yuqing Du, Changsheng You, Jun Zhang, and Kaibin Huang. "Toward an intelligent edge: wireless communication meets machine learning." IEEE Communications Magazine, **58**(1):19–25, 2020.

[ZSS16] Ning Zhang, Kun Sun, Deborah Shands, Wenjing Lou, and Y Thomas Hou. "TruSpy: Cache Side-Channel Information Leakage from the Secure World on ARM Devices." IACR Cryptology ePrint Archive, **2016**:980, 2016.