DISS. ETH NO. 29460

FINE-GRAINED ACCESS CONTROL FOR SENSORS, ACTUATORS, AND AUTOMATION NETWORKS

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

PIET DE VAERE
MSc ETH in Electrical Engineering and
Information Technology, ETH Zurich

born on 25.11.1994
citizen of Belgium

accepted on the recommendation of

Prof. Dr. Adrian Perrig, examiner
Prof. Dr. David Hausheer, co-examiner
Prof. Dr. Gene Tsudik, co-examiner
2023

Abstract

Both industrial- and consumer-oriented automation systems are pervasively deployed. Although automation is not new, current trends are causing clear security concerns. Buzzwords such as *IoT*, *smart*, *connected*, and *Industry 4.0* have two things in common: they rely heavily on connectivity, and they introduce significant new system complexity. The latter, in turn, has shown to lead to clear increases in system vulnerability. In addition, automation systems directly interact with the physical world, so compromised devices can directly cause physical harm.

Based on the above, we identify connectivity and access to the physical environment as two key risk inducers for automation systems, and we study both in this thesis. First, we discuss how fine-grained access control over sensors and actuators can isolate computational resources from their physical surroundings. Doing so significantly reduces the potential impact of a compromised automation system. Second, we present network segmentation methods that provide a high level of control over who can access which network resource. Doing so allows for access to connectivity to be tightly regulated, reducing the risk of devices being compromised in the first place.

Part I — Sensors and Actuators Today, device-level protection mechanisms for sensors and actuators are the norm, relying on the trustworthiness of automation devices themselves. However, this approach fails if devices are compromised. To mitigate the effects of device compromise, our first contribution is SA^{4}P: Sensing and Actuation as a Privilege, a framework to decouple automation infrastructure from its physical environment. When

SA4P is deployed, even a device's own software runtime must be duly authorized each time it wants to sense or actuate. This decoupling is achieved by the inclusion of an on-board component that physically guards sensors and actuators. Besides providing strong privacy and safety guarantees, our approach motivates developers to consider sensing and actuation as high-value resources.

Our second contribution considers devices with always-standby event-triggered sensors. Most significantly, smart speakers and other voice assistants. Although such devices are becoming increasingly ubiquitous, their always-standby nature continues to prompt significant privacy concerns. To address these, we propose KIMYA, a hardening framework that allows device vendors to provide strong data-privacy guarantees. Concretely, KIMYA guarantees that sensor data can only be used for local processing, and is immediately discarded unless a user-auditable notification is generated. KIMYA thus makes devices accountable for their data-retention behavior.

Part II — Automation Networks We observe that the fundamental assumptions on which current network-based automation defenses are based, are rapidly being invalidated by reality. We present an analysis of both historical and new trends to substantiate this claim, and then introduce TABLEAU, a new zoning architecture for operational technology (OT) networks. TABLEAU increases network flexibility by flattening network structures and by allowing the seamless integration of information technology (IT), OT, and cloud networks. Simultaneously, TABLEAU facilitates modern security practices and is IEC 62443 compatible, ensuring the continued secure operation of OT infrastructure.

Then, to provide protections beyond those of perimeter-based network security, we present HOPPER, an industrial automation security protocol that places each network host in its own access-controlled *nano segment*, thus further minimizing the attack surface introduced by connecting devices. Because HOPPER enforces nano segmentation in-fabric, it does not require routing modifications. Especially when combined with TABLEAU, HOPPER significantly reduces the exposure of automation devices.

Zusammenfassung

Sowohl industrielle als auch verbraucherorientierte Automatisierungssysteme sind weit verbreitet. Obwohl die Automatisierung nicht neu ist, geben die aktuellen Trends Anlass zu deutlichen Sicherheitsbedenken. Schlagworte wie *IoT*, *smart*, *connected* und *Industrie 4.0* haben zwei Dinge gemeinsam: Sie beruhen in hohem Masse auf Konnektivität, und sie führen zu einer erheblichen neuen Systemkomplexität. Letzteres, hat wiederum gezeigt, dass sie die Anfälligkeit von Systemen deutlich erhöht. Darüber hinaus interagieren Automatisierungssysteme direkt mit der physischen Welt, sodass kompromittierte Geräte direkt physischen Schaden anrichten können.

Auf der Grundlage der obigen Ausführungen identifizieren wir die Konnektivität und den Zugang zur physischen Umgebung als zwei zentrale Risikofaktoren für Automatisierungssysteme, welche wir beide in dieser Arbeit untersuchen. Zuerst erörtern wir, wie eine fein abgestufte Zugriffskontrolle für Sensoren und Aktoren, die Rechenressourcen von ihrer physischen Umgebung isolieren kann. Auf diese Weise werden die potenziellen Auswirkungen eines kompromittierten Automatisierungssystems erheblich reduziert. Danach stellen wir moderne Netzwerksegmentierungsmethoden vor, die ein hohes Mass an Kontrolle darüber bieten, wer auf welche Netzwerkressourcen zugreifen kann. Auf diese Weise kann die Konnektivität streng reguliert werden, was das Risiko einer Kompromittierung von Geräten im Vorhinein verringert.

Teil I — Sensoren und Aktoren Heutzutage sind Schutzmechanismen, die sich auf die Vertrauenswürdigkeit der Automa-

tisierungsgeräte selbst verlassen, auf Geräteebene für Sensoren und Aktoren die Norm. Dieser Ansatz versagt jedoch, wenn die Geräte kompromittiert werden. Um die Auswirkungen einer Gerätekompromittierung zu mildern, haben wir in unserm ersten Beitrag SA^{4}P (Sensing and Actuation as a Privilege) entwickelt. SA^{4}P ist ein Framework zur Entkopplung der Automatisierungsinfrastruktur von ihrer physischen Umgebung. Wenn SA^{4}P eingesetzt wird, muss sogar die eigene CPU eines Geräts jedes Mal ordnungsgemäss autorisiert werden, wenn er etwas messen oder steuern möchte. Diese Entkopplung wird durch eine integrierte Komponente erreicht, die die Sensoren und Aktoren physisch schützt. Unser Ansatz bietet nicht nur starke Datenschutz- und Sicherheitsgarantien, sondern motiviert Entwickler auch dazu, Messungen und Ansteuerungen als hochwertige Ressourcen zu betrachten.

Unser zweiter Beitrag befasst sich mit Geräten, dessen Sensoren ereignisgesteuert und immer in Bereitschaft sind. Dazu gehören vor allem intelligente Lautsprecher und andere Sprachassistenten. Obwohl solche Geräte zunehmend allgegenwärtig sind, gibt ihr ständiger Bereitschaftszustand weiterhin Anlass zu erheblichen Bedenken hinsichtlich des Datenschutzes. Um diese zu beseitigen, schlagen wir KIMYA vor, ein Härtungs-Framework, das es Geräteherstellern ermöglicht, starke Datenschutzgarantien zu geben. Konkret garantiert KIMYA, dass Sensordaten nur für die lokale Verarbeitung verwendet werden können und sofort verworfen werden, ausser es wird eine vom Benutzer überprüfbare Benachrichtigung erzeugt. KIMYA gewährleistet somit, dass die Geräte für ihr Datenaufbewahrungsverhalten Rechenschaft ablegen müssen.

Teil II — Automatisierungsnetzwerke Wir stellen fest, dass die grundlegenden Annahmen, auf denen die derzeitigen netzwerkbasierten Automatisierungsverteidigungen beruhen, durch die wandelnde Realität eingeholt werden. Wir präsentieren eine Analyse sowohl historischer als auch neuer Trends, um diese Behauptung zu untermauern, und stellen dann TABLEAU vor, eine neue Zonierungsarchitektur für OT Netzwerke. TABLEAU erhöht die Flexibilität von Netzwerken, indem es die Netzwerkstrukturen abflacht und eine nahtlose Integration von IT-, OT- und Cloud-Netzwerken ermöglicht. Zugleich erleichtert TABLEAU moderne Sicherheitspraktiken und ist mit IEC 62443 kompatibel, sodass der sichere Betrieb von Betrieb der OT-Infrastruktur gewährleistet ist.

Um einen Schutz zu bieten, der über den der Perimeter-basierten Netzwerksicherheit hinausgeht, stellen wir HOPPER vor, ein Sicherheitsprotokoll für die Industrieautomation, das jeden Netzwerk-Host in sein eigenes zugangskontrolliertes *nano segment* platziert und so die Angriffsfläche, die durch Verbindungsgeräte entsteht, weiter minimiert. Da HOPPER die Nanosegmentierung homogen in der Netzwerk-Fabric implementiert, sind keine Änderungen an der Weiterleitung von Paketen erforderlich. Insbesondere in Kombination mit TABLEAU reduziert HOPPER die Anfälligkeit von Automatisierungsgeräten erheblich.

Contents

1 *Introduction* 15

1.1 *Related Publications and Contributors* 20

2 *Background* 23

2.1 *Safety and Security in Automation Networks* 23

2.2 *TrustZone on Cortex-M* 24

2.3 *Mondrian Network Zoning* 25

I *Sensors & Actuators* 33

3 *Sensing and Actuation as a Privilege* 35

3.1 *Introduction* 35

3.2 *Preliminaries* 39

3.3 *Adversary and Trust Model* 40

3.4 SA^{4}P *Overview* 41

3.5 $\mathcal{P}$EG *Design* 43

3.6 *Security Analysis* 51

3.7 *Implementation* 56

3.8 Evaluation 60
3.9 Discussion 64
3.10 Summary 68

4 Securing Event-Triggered Sensors 69
4.1 Introduction 69
4.2 Adversary Model & Security Setting 73
4.3 Design 74
4.4 Implementation on Cortex-M 81
4.5 Security Analysis 88
4.6 Prototype Application 92
4.7 Evaluation 95
4.8 Discussion 101
4.9 Summary 108

II Automation Networks 111

5 Current Automation Networks 113
5.1 Introduction 113
5.2 Current OT Networks 114
5.3 Challenges to OT Networks 117

6 Future-Proof Zoning for OT Networks 121
6.1 Introduction 121
6.2 A Flat Zoning Architecture for OT Networks 122
6.3 Security Aspects 131
6.4 Compatibility with IEC 62443 134
6.5 Summary 134

7 Per-Device Nano Segmentation for the Industrial IoT 135
7.1 Introduction 135
7.2 Constraints in Industrial IoT Networks 138
7.3 Adversary Model & Security Goals 139
7.4 HOPPER *Protocol* 140
7.5 Practical Considerations 150
7.6 Security Analysis 157
7.7 Implementation and Evaluation 161
7.8 Relationship with TABLEAU *and IEC 62443* 169
7.9 Summary 170

8 Related Work 171
8.1 Sensors and Actuators 171
8.2 Automation Networks 175

9 Conclusion 179
9.1 Summary 179
9.2 Future Work 181
9.3 Closing Remarks 182

To my parents.

1 Introduction

Automation plays a key role in modern society. In fact, as far back as the steam age, automation has been an indispensable component of technological development. Consider, for example, the centrifugal governor shown in Fig. 1.1. This device can automatically vary an engine's throttle to keep it operating at near-constant rotational speed.[1] Without governors, steam engines would not have been able to operate without permanent human supervision.

[1] The governor was first invented to regulate the distance and pressure between millstones in windmills [72].

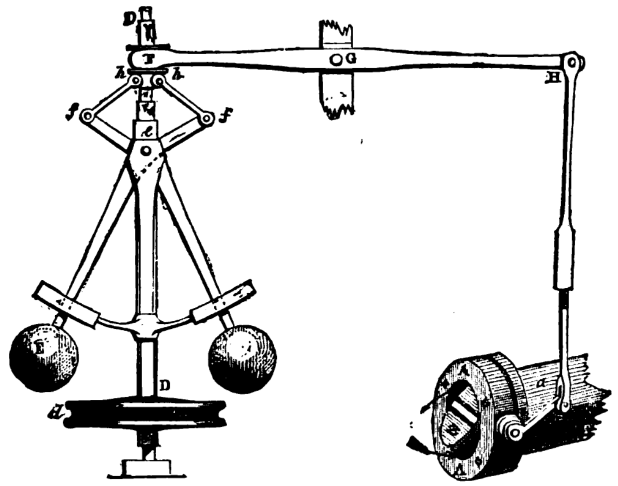


Figure 1.1: A centrifugal governor. As the rotational speed increases, the balls swing further outward, causing the throttle valve to be closed. The result is a proportional control loop.

We have come a long way since those early days of automation. The steam engine has long been replaced by increasingly sophisticated motors. The same goes for the basic governor, which has been superseded by electronic loop controllers which can have hundreds, if not thousands, of parameters. Moreover, relay logic and later programmable logic controllers (PLCs) allowed complex logical operations to be fully automated.

Simultaneously, automation has left the factory and started to enter our day-to-day lives. This too, started with rudimentary devices (e.g., the washing machine), but has grown to highly complex systems such as robot vacuums and voice assistants.

Both in and out of the factory, automation systems originated as individual, fully isolated systems. Over time, these systems have clustered together slowly and organically. Initially, clusters were small and local, but today more and more systems are being connected to the global Internet. In fact, we have reached a point where systems are not only connected to the Internet, but often critically rely on remote services to perform their functionality. The latter is especially pronounced in consumer-oriented products, such as smart speakers or other home automation systems. In such systems, a cloud-centric design typically offers lower development costs, and a shorter time to market.

Combining the ever-increasing connectivity of automation products with the general vulnerability of computer systems, results in a myriad of security risks. These are further exacerbated by an automation system's inherent ability to interact with the physical environment; whereas a compromised desktop computer might result in financial or reputational losses, a compromised control system can potentially lead to direct physical harm.

The number of networked automation systems is ever increasing. Moreover, such systems are often critical in nature, e.g., because they are deployed in a critical infrastructure, or even simply because of their pervasive nature. Therefore, robust security mechanisms for automation systems are continually gaining importance. On the surface, it might seem that modern automation systems are best secured using the same methods used to secure general purpose computation systems. However, such techniques not only run the risk of being mismatched to the problem, they can also overlook defense opportunities that are unique to the automation world. Therefore, this thesis focuses on defenses specifically tailored to automation systems.

With this approach in mind, we make two observations. First, the principal difference between automation systems and general purpose computers, is the former's ability to interact with the physical world. Although this capacity is required for automation devices to perform their tasks, it also poses significant danger if a device is compromised. Ideally, devices would only be able to access the physical world to execute their legitimate tasks. Second, although automation devices are not new, security breaches related to them have witnessed a significant rise in recent years, mainly due to enhanced device connectivity. Thus, reducing an adversaries ability to abuse connectivity is a promising defense avenue.

Having identified (i) access to the physical world, and (ii) network connectivity as critical risk inducers above, we dedicate one part of this thesis to each topic.

In Part I, we present mechanisms that regulate the access an automation device has to the physical world. In today's systems, automation devices are typically able to access their sensors and actuators at will. That is, no policies to limit their ability to interact with their physical environment are (or can be) enforced. Instead, access control is typically performed at the device (Fig. 1.2, ①) or platform (Fig. 1.2, ②) level, However, this means that if a device is compromised, the physical environment is unavoidably at risk. That is, the compromised device may violate the physical world's integrity (by performing malicious actuation), or confidentiality (by using sensors to spy).

In this thesis, we present an alternative approach: we place access control mechanisms between the computational elements of automation devices and their sensors and actuators (Fig. 1.2, ③). Doing so isolates the computational elements from the physical world. Given the typical usage patterns of consumer automation products, this approach most naturally fits the consumer setting. Therefore, we focus Part I on consumer devices.

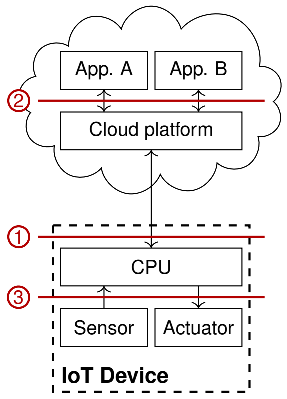


Figure 1.2: Typical IoT access control enforcement levels.

To practically achieve isolation between the computational and the physical worlds, Chapter 3 introduces the Sensing and Actuation as a Privilege (SA^{4}P) framework, the key component of which is the *Peripheral Guard* or $\mathcal{P}$EG. The $\mathcal{P}$EG is a small-footprint component that functions as an on-device policy enforcement point, allowing a remote entity to determine when a device can access its sensors or actuators. Besides providing strong privacy and safety guarantees, this approach motivates designers to consider sensing and actuation as high-value resources rather than as readily available commodities.

Chapter 4 further extends this approach by considering always-standby, event-triggered devices, e.g., smart speakers. For such devices, a natural paradox arises: they should not be able to spy on the environment, but should also always be sensing for events. Put in the more-relatable terms of smart speakers: they should not be able to listen to our conversations, but should also always be listening to hear if they are addressed (e.g., "Hey Alexa!"). Here, gate-like methods—where resource access is either granted or restricted—fall short: for the device to function, sensor access should continuously be granted, and hence, covert snooping cannot be prevented.

To address the smart speaker paradox, Chapter 4 proposes KIMYA, a hardware-assisted containerization mechanism. On a KIMYA-enabled device, application code does not, by default, have access to any protected sensing or actuation resources. However, the application can run arbitrary routines in an *isolated* and *amnestic* container. These routines have full access to the protected resources, but can neither exfiltrate this data (isolated), nor store it for longer than a predefined duration (amnestic); unless they generate an auditable notification. This notification can take multiple forms. Possible options include a visual indicator (e.g., a LED), or a cryptographically protected machine-to-machine message. In the latter case, the KIMYA enforcement mechanism can simultaneously function as a $\mathcal{P}$EG, allowing external access control policies to be enforced on top of the properties KIMYA provides.

The second line of work covered in this thesis focuses on network connectivity. As discussed above, ever-increasing connectivity is the primary catalyst for the security incidents involving automation products. Therefore, managing connectivity is a promising avenue to mitigate vulnerabilities in automation systems, and we dedicate Part II of this thesis to such efforts.

Whereas Part I focuses on consumer settings, network-based solutions are most suitable in industrial settings. The reason for this is threefold: (i) industrial networks are more complex, providing more opportunities for network-based defenses; (ii) industrial networks are managed by trained personal, requiring less focus on auto-configuration; and (iii) industrial control loops function at very small timescales and typically run continuously for significant portions of the day, reducing the relevance of isolating industrial controllers from the physical world. Therefore, Part II focuses on industrial settings.

Concretely, in Chapter 5 we present an overview of how current automation networks are structured. We then discuss how new and ongoing trends are challenging today's practices. Particularly, we argue how these trends are gradually invalidating the fundamental assumptions on which the security properties of today's automation networks are based, thereby leading to an erosion of these security properties.

To address this, Chapter 6 explores how networks can be restructured based on updated assumptions and modern technologies. We do so by proposing TABLEAU, a flat zoning architecture for automation networks, which not only facilitates more flexible zoning than current solutions, but also aims to simplify network management and increase security. TABLEAU allows for the seamless integration of plant, edge, corporate, and cloud networks, while simultaneously facilitating modern security practices.

Being a zoning architecture, TABLEAU provides its protections at the network or zone boundaries. To extend these protections to intra-zone traffic, Chapter 7 introduces the concept of per-device *nano-segmentation*, along with HOPPER, a practical nano-segmentation protocol. As the name implies, in a nano-segmented network, each individual device is placed in its own virtual *nano segment*. Contrary to classical zoning mechanisms, Hopper enforces segmentation in-fabric, and therefore does not require modifications to packet routing. This allows Hopper to be deployed without introducing network bottleneck or single points of failure. Hopper achieves segmentation by allowing each network node to verify that each packet it processes is part of a desired flow and was generated by an authorized host. Packets that fail any of these checks are dropped en route.

Although Parts I and II focus on consumer and industrial settings, respectively, all contributions presented in this thesis can fundamentally be applied to both settings. Moreover, SA^{4}P, Kimya, Tableau, and Hopper are composable and can be deployed simultaneously on a shared set of devices.

1.1 *Related Publications and Contributors*

This thesis is the synopsis of a multi-year effort and is based on multiple academic publications. Various collaborators have contributed to its content over the years. My advisor, Prof. Dr. Adrian Perrig has continuously provided his expertise and insight along the way. He deserves partial credit for the entire body of this thesis. Thank you, Adrian.

Chapter 3 is based on a paper (under submission) that is co-authored by Prof. Dr. Gene Tsudik and Felix Stöger. Gene's insights and expertise contributed significantly to the protocol and Peripheral Guard designs. Felix provided the system implementation and performed the benchmarks. He also contributed extensively to the early design and exploratory phases of the work.

Chapter 4 is based on a publication at the upcoming 2023 USENIX Security Symposium [47].

[47] *Hey Kimya, Is My Smart Speaker Spying on Me? Taking Control of Sensor Privacy Through Isolation and Amnesia*, De Vaere and Perrig (2023)

Both Chapters 5 and 6 are based on the Tableau paper published at the 2021 CRITIS conference [46]. Franco Monti's extensive experience with real-world industrial automation networks was indispensable for the analysis presented in Chapter 5. Claude Hähni's intricate knowledge of the Mondrian zoning system [92] proved highly valuable to shape the Tableau architecture described in Chapter 6.

[46] *Tableau: Future-Proof Zoning for OT Networks*, De Vaere et al. (2021)

[92] *Mondrian: Comprehensive Inter-domain Network Zoning Architecture*, Kwon et al. (2021)

Chapter 7 is based on the Hopper paper published at ASIA-

CCS 2022 [49]. As part of his master thesis, Andrea Tulimiero provided the embedded Hopper implementation and designed and performed the embedded benchmarks. His efficiency at these tasks was unparalleled.

[49] *Hopper: Per-Device Nano Segmentation for the Industrial IoT*, De Vaere, Tulimiero, and Perrig (2022)

2 Background

2.1 Safety and Security in Automation Networks

In automation systems, a distinction is made between system *safety* and *security*. As automation systems interact with the physical world, a misbehaving system can lead to direct physical damage or even harm. For example, two system components could collide with each other, chemicals could be mixed in explosive combinations, or an operator could suffer physical harm. The study of such misbehavior, its origins, and its consequences is known as *functional safety*, *operational safety*, or simply *safety* [62]. Many different aspects can affect safety, including, but not limited to, equipment failure, operator error, natural disasters, and sabotage.

[62] *Cybersecurity of Industrial Systems*, Flaus (2019)

Contrary to functional safety, the study of security explicitly considers the involvement of a malicious actor know as the adversary or attacker. The adversary actively attempts to compromise the system under consideration. In principle, any entity, both internal and external, can act as an adversary. An adversary can have the following goals:

Affecting confidentiality, e.g., by exfiltrating sensitive system parameters or user data;

Affecting availability, e.g., by shutting down the system; and

Affecting integrity, e.g., by introducing artificial latency into the automation loop thereby reducing product quality, or by enrolling a device into a botnet.

Depending on the adversary's goals and actions, a security incident might result in a safety incident, e.g., if the adversary circumvents safety systems to create an unsafe situation. In this thesis, we will primarily consider information security, i.e., the security of information systems. Whereas in traditional information systems the properties of the CIA triad are typically prioritized as they appear (i.e., first confidentiality, then integrity, and then availability), this prioritization is not equally clear in automation systems. In fact, in industrial automation settings the priorities are usually reversed: availability first, then integrity, and confidentiality last. However, for consumer settings this does not always hold. Consider, for example, a camera-equipped smart vacuum cleaner. Most home owners would likely prefer the vacuum cleaner malfunctioning over the camera stream's confidentiality being compromised.

2.2 *TrustZone on Cortex-M*

TrustZone on Cortex-M introduces two new processor security states: secure and non-secure [16]. These states are orthogonal to traditional processor states such as thread vs. handler mode and privileged vs. non-privileged mode. The active security state is determined by the instruction pointer and a security map which partitions executable addresses into secure and non-secure regions.

[16] *ARM v8-M Security Extensions: Requirements on Development Tools*, ARM Ltd. (2019)

Beyond executable memory, other microcontroller unit (MCU) resources are assigned a security attribute. Resources marked as secure are only accessible to code running in the secure state. Resources marked as non-secure are accessible to all code. Because of this separation, the security states are also referred to as the secure and non-secure *worlds*.

There are two principal ways to configure which resources are placed in which world. First, the Cortex-M core is extended with a secure attribution unit (SAU). The SAU is functionally similar to a memory protection unit (MPU) and can be used to configure specific memory regions as secure or non-secure. Second, in order to extend the concept of security states beyond the core, the data bus is extended to carry the security state of each transaction. Individual peripherals can either be made TrustZone-aware, or must be protected by a security gate, as shown in Fig. 2.1 [16]. Other bus masters (i.e., direct memory access (DMA) controllers) must also indicate the security state of their bus requests. Further relevant for our work is that the MPU is duplicated, with one instance being active when the core is in the secure state, and the other one when the core is in the non-secure state [19]. Both instances can be independently configured.

[16] *ARM v8-M Security Extensions: Requirements on Development Tools*, ARM Ltd. (2019)

[19] *Introduction to the ARMv8-M architecture*, ARM Ltd. (2017)

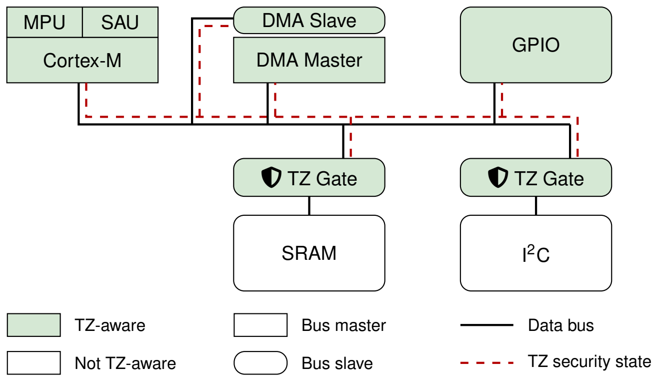


Figure 2.1: A high level overview showing how TrustZone (TZ) on Cortex-M extends beyond the processor core.

Contrary to TrustZone for Cortex-A, there is no secure monitor in the Cortex-M architecture. Transitions from the non-secure world to the secure world are facilitated through jumps to developer-defined *secure gateway* (SG) instructions, which provide a limited set of entry points into the secure world. Secure functions can also make calls to the non-secure world. After the non-secure function returns, control is then automatically returned to the secure world. To speed up transitions between the secure and non-secure worlds, some processor registers are banked.

2.3 *Mondrian Network Zoning*

Mondrian [92] is a recent zoning architecture for enterprise networks that was motivated by the need for modern network models which is arising in cloud and hybrid-cloud deployments. These new deployment scenarios are posing additional demands

[92] *Mondrian: Comprehensive Inter-domain Network Zoning Architecture*, Kwon et al. (2021)

on IT security in large corporate networks. Traditionally, information was processed within a single domain. Today, IT infrastructures are distributed across several heterogeneous systems that all need to communicate with each other. This has lead to increased complexity in the structure of IT networks, with a myriad of systems and policies that need to be managed, kept synchronized, and kept consistent. This is similar to what we are currently experiencing in OT networks. Mondrian offers a secure, flexible, and scalable network zoning architecture that alleviates these issues. One notable property of Mondrian is its capability to securely bridge geographically distributed, heterogeneous networks over untrusted infrastructure. As a result, Mondrian opens the door for many interesting deployment scenarios in which a highly secure and easy to manage zoning architecture is required.

2.3.1 *Mondrian Overview*

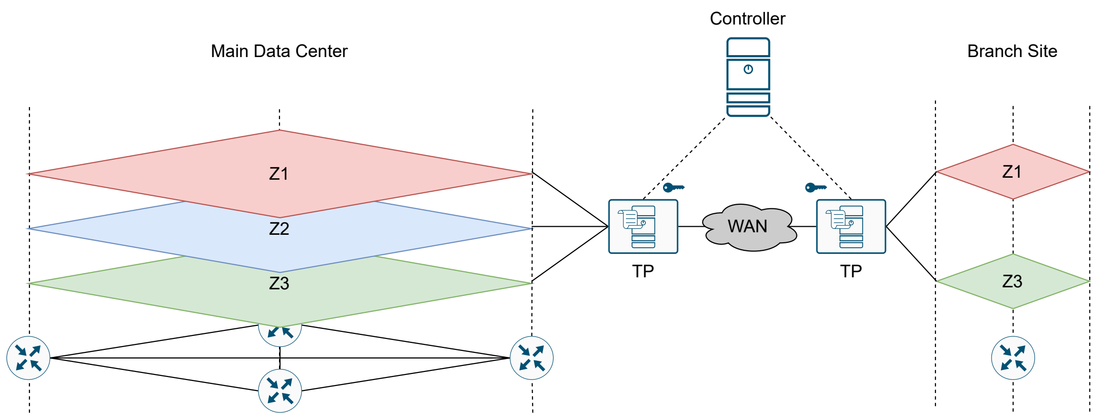


Figure 2.2: Mondrian architecture overview.

Network Zoning with Mondrian In contrast to the current, often highly-complex organization of network zones, Mondrian partitions the network into a collection of flat zones. As illustrated in Fig. 2.2, each of these zones is connected to a designated security gateway called the transition point (TP). Placing zones adjacent to each other, only separated by the TP, simplifies today's network architectures in which traffic often needs to traverse multiple layers to reach its destination. A logically centralized controller provides a comprehensive management interface for operators to orchestrate the network. Common tasks, such as zone migration and zone initialization become much easier, as the network configuration is centralized on a single system. TPs ensure source authentication, zone access authorization, and ingress/egress filtering for all connected network zones. Using the concept of an *inter-domain transit zone*, Mondrian enables network zoning across the boundaries of local networks. This is particularly useful for enterprises that operate geographically distributed branch sites or leverage the cloud as part of their infrastructure.

Flexibility and Scalability The core of Mondrian is the logically centralized controller, presenting a single interface through which network operators manage their network. Sites, zones and transition policies can all be centrally managed through this interface. The controller then takes care of distributing these policies to the TPs, which enforce the policies at the individual premises.

Supporting fine-grained zone transition policies offers great flexibility for operators to cover a diverse set of use cases. The centralized interface simplifies today's complex infrastructure with potentially many systems and their respective configurations that need to be updated for every change to the network. As a result, Mondrian is less susceptible to configuration errors and makes policy reviews more efficient. In concert, these properties significantly enhance management scalability.

Deployability Mondrian supports multiple deployment methods that can be used in conjunction with each other. The primary method uses TPs in the form of all-in-one gateways which perform routing, packet authorization, and tunneling, all without requiring any changes to end hosts. This method reduces the number of security middleboxes that need to be maintained in

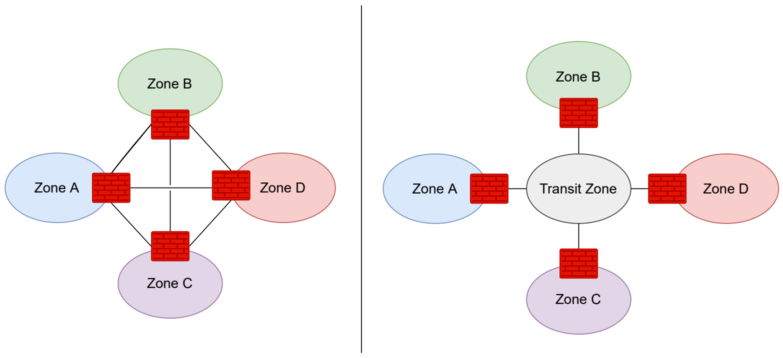


Figure 2.3: On the left: a network topology using dedicated links to connect each pair of zones. On the right: The same network organized in a hub-spoke configuration using a transit zone as central element.

networks. When using this method, Mondrian can also assume a supportive role in which traffic is prefiltered before it gets handled by security middleboxes.

Alternatively, Mondrian can be deployed purely in software on commodity computing devices. Similar to a VPN, this allows individuals to remotely access network assets from their personal devices in a secure and authenticated manner. When using this method, a TP runs as virtual gateway on a computer and tunnels packets from the device to a remote TP in the enterprise. In contrast to a traditional VPN, a software TP is part of the regular Mondrian deployment and seamlessly integrates with the rest of the architecture.

2.3.2 *Mondrian in Detail*

Inter-domain transit zone One of the main building blocks that allows Mondrian to achieve the properties introduced above, is the concept of the inter-domain transit zone. Transit zones are commonly used within local networks to facilitate zone transitions. Concretely, they are special zones that do not contain any end hosts, but merely exist to interconnect other zones. Put differently, a transit zone is the hub in a hub-spoke network topology, providing connectivity between all the other zones. Hub-spoke configurations allow physically separated network zones to access shared services without the need for dedicated links between each pair of zones (see Fig. 2.3). Mondrian scales transit zones to inter-domain networks. The inter-domain transit zone spans across a wide area network (WAN), connecting the branch sites of enterprises. At every site, local zones are directly attached to the inter-domain transit zone, thus creating a collection of disjoint, parallel network zones. Such a network requires packets to traverse fewer security middleboxes as all zone transitions can be checked already at the border of the inter-domain transit zone. Inside the transit zone, the Mondrian protocol is used to transport zone information across the inter-domain transit zone, allowing remote destinations to easily verify zone transitions, even if the the underlying network is untrusted. Additionally, the Mondrian protocol is independent from the internal protocols used at each site, which means it is able to bridge networks that operate on otherwise incompatible internal protocols.

Transition points & controller At each network site, Mondrian deploys a dedicated security gateway, called the transition point (TP). Network zones (subnets) at every branch site are directly connected to the TP, creating a flat network structure (see Fig. 2.2). This means that all inter-zone traffic needs to pass at least one TP. Together, TPs span the inter-domain transit zone. The main task of a TP is twofold: (i) it ensures that traffic does not violate the zone transition policy. For that, TPs check all zone transitions against a policy they receive from a logically centralized controller. On an abstract level, this transition policy is a matrix which defines for each ordered pair of zones (A, B) which traffic is allowed to flow from zone A to zone B. The controller has the full view over the entire distributed network and makes sure that all sites operate with the latest security policy. (ii) For zone transitions that cross the inter-domain transit zone, the second task of TPs is to attach cryptographically secured zone information to each packet before encrypting and forwarding the packet over the WAN. This way, Mondrian achieves integrity and confidentiality of information being sent over a potentially untrusted network. Because the complete original packet, including headers, is encrypted, internal addresses are prevented from leaking. Upon receiving a packet, the remote TP can verify the

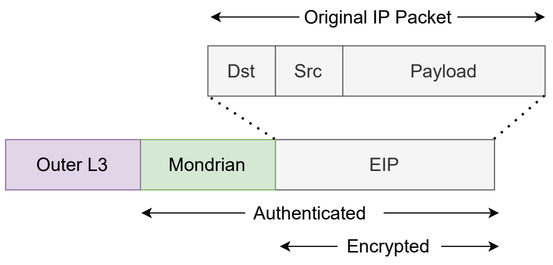


Figure 2.4: A depiction of the Mondrian encapsulation. For packets traversing the inter-domain transit zone, a Mondrian header including a zone authenticator is attached to the encrypted original IP packet (EIP). Finally, the Mondrian packet is wrapped in an outer Layer-3 header.

zone information, decrypt the packet and, if all checks succeed, forward the packet into the local network. The latency overhead introduced by each TP is less than 5 μs [92].

[92] *Mondrian: Comprehensive Inter-domain Network Zoning Architecture*, Kwon et al. (2021)

Packet life-cycle The life-cycle of a packet in a Mondrian network is as follows.

1. An end host in a source zone Z_S sends an IP packet towards an end host in a destination zone Z_D by creating a regular IP packet with the usual source and destination addresses.
   (a) If $Z_S = Z_D$, the packet is delivered directly by the Layer-2 protocol.
   (b) Otherwise, the packet needs to be forwarded via a Mondrian TP.
2. The TP analyzes the packet, retrieving Z_S and Z_D based on the source and destination address of the packet, ensuring that the zone transition Z_S to Z_D is allowed.
   (a) If not, the packet is dropped.
   (b) If yes, the packet is forwarded towards the destination.
3. Next, based on the destination address, the TP evaluates if the packet is destined for an end host in the same branch site.
   (a) If yes, the TP forwards the packet towards the destination in the internal network.
   (b) In case the destination is in a different network across the inter-domain transit zone, the TP looks up the remote TP, creates a cryptographic authenticator, encrypts the original

IP packet, and encapsulates the encrypted packet together with the Mondrian header in an outer Layer 3 header (see Fig. 2.4). The exact outer layer depends on the protocol used within the inter-domain transit zone. This packet is then forwarded to the remote TP.

4. Finally, the receiving TP decapsulates the payload, verifies the authenticator and, if all checks succeed, decrypts the payload back into the original IP packet which it then forwards to the destination inside the internal network.

Part I

Sensors & Actuators

3 SA^{4}P: *Sensing and Actuation as a Privilege*

3.1 *Introduction*

Specialized embedded devices of various types and uses have made their way into almost every aspect of our lives. Commonly, these devices are referred to as Internet of Things (IoT) devices, and they differ from general-purpose computers (e.g., servers, desktops, laptops, tablets, and smartphones) in that their primary purpose is to interact with their physical environment. This can mean either sampling data from the physical world (i.e., sensing), or actively influencing that world (i.e., actuation).

With some IoT devices we interact directly (e.g., home automation), others are hidden to most of the population (e.g., factory automation). In either case, many of today's *smart* devices are an evolution of devices without (or with highly limited) computational or communication abilities. Practical examples of devices with historic, *dumb*, counterparts include light switches, garage doors, and locks, but also PLCs, motor drives, and irrigation pivots.

Regardless of their type, history, or purpose, IoT devices are well known to pose *privacy* risks, e.g., through the leakage of sensed data or actuation commands. Less well known are the potential *safety* risks that can arise when sensing or actuation data is spoofed or corrupted. Both type of risks are not imagined or exaggerated; they are in fact very real, as demonstrated by numerous examples in the real world. In 2010 the Stuxnet malware targeting the Iranian nuclear program was discovered [95]. In 2015 the Ukrainian power grid came under attack [191]. In 2016 the Mirai botnet targeted consumer devices at an unprecedented scale [9]. There are also examples more directly relevant to consumers. For example, it has been shown that smart door locks can be bypassed [13], and loose privacy policies for vacuum cleaners have lead to intimate pictures being leaked [68].

[95] *Stuxnet: Dissecting a Cyberwarfare Weapon*, Langner (2011)

[191] *Inside the Cunning, Unprecedented Hack of Ukraine's Power Grid*, Zetter (2016)

[9] *Understanding the Mirai Botnet*, Antonakakis et al. (2017)

[13] *Nuki Smart Lock Vulnerabilities Allow Hackers to Open Doors*, Arghire (2022)

[68] *Roomba testers feel misled after intimate images ended up on Facebook*, Guo (2023)

Moreover, past work has shown that IoT attacks have the potential to extend beyond what we are used to from attacks on classical IT systems. Ronen et al. have shown that IoT attacks have the potential to propagate without relying on classical network infrastructure, circumventing network-based defenses [141]. Additionally, even seemingly innocent data leaks have the potential to pose serious privacy risks. For example, a compromised humidity sensor can leak information about room occupancy [70], and a compromised gyroscope signal can expose a user's location [116].

[141] *IoT Goes Nuclear: Creating a ZigBee Chain Reaction*, Ronen et al. (2017)

[70] *Don't Sweat Your Privacy: Using Humidity to Detect Human Presence*, Han et al. (2007)

[116] *Inferring User Routes and Locations Using Zero-Permission Mobile Sensors*, Narain et al. (2016)

The key characteristic of an IoT device is its ability to interact with the physical world. These interactions take place through the device's sensing and actuation hardware peripherals. Today, access control to those peripherals is typically enforced at the device or platform level. That is, from an access control perspective, the entire device is considered to be a single, trusted, entity. Access control enforcement and decision-making are then performed either by the application code on the device itself, or by a higher-level entity (e.g., a control hub or cloud platform). These approaches work well as long as the device is not compromised or otherwise malicious. However, if the device itself is infected, device-, and platform-level approaches become ineffective; if the adversary has a presence inside of the device, it is free to sense and actuate at will, leaving the higher-level control mechanisms powerless.

One approach to handle such situations, is to quarantine infected devices, rendering them unable to communicate with the external adversary. Approaches to detect infections include network monitoring and remote attestation. Network monitoring techniques aim to identify suspicious traffic patterns or packet signatures. However, detection is usually not instantaneous, meaning that malicious peripheral use can have already occurred. Moreover, network-based detection techniques are probabilistic in nature, so an infected device could shape its traffic to delay detection. Remote attestation schemes require close integration with the device and its specific software version, are typically resource intensive, and will only report compromised devices after the fact. Moreover, all quarantine-based approaches assume that an adversary relies on classical network infrastructure. When attacks are infrastructure-independent (i.e., because the malicious code is self-contained, or because the attack uses infrastructure-free communication [141]), quarantine is ineffective.

[141] *IoT Goes Nuclear: Creating a ZigBee Chain Reaction*, Ronen et al. (2017)

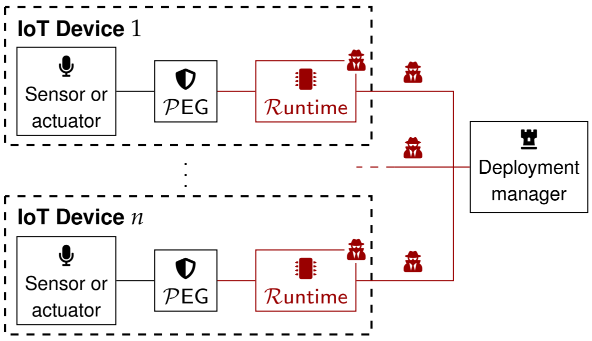


Figure 3.1: Conceptual SA⁴P architecture. indicates an untrusted component.

Given the limitations of quarantine-based approaches, this thesis introduces the use of peripheral-level access enforcement. We present SA^{4}P: Sensing and Actuation as a Privilege, an IoT security architecture working at the sub-device level. As illustrated in Fig. 3.1, SA^{4}P's key feature is the placement of a Peripheral Guard ($\mathcal{P}$EG) between a device's software $\mathcal{R}$untime and its sensing and actuating peripherals, thereby decoupling the $\mathcal{R}$untime from the physical environment. The $\mathcal{P}$EG is operated centrally, by a *deployment manager*, a trusted entity that makes all access control decisions for an IoT deployment. The deployment manager can run on a remote or a local server, e.g., a home router. Together with the $\mathcal{P}$EGs, the deployment manager effectively constitutes a distributed reference monitor wherein a centralized entity (the deployment manager) controls all physical-world interactions occurring across an entire IoT deployment.

By default, the $\mathcal{P}$EG disallows all sensor and actuator access. Each time the code running on the device $\mathcal{R}$untime wants to interact with the physical environment, it must first send a request to the deployment manager. The main benefit of this approach is its preventative nature: data that is not sampled cannot be leaked, and attempted (yet not performed) actuation causes no harm. In addition to allowing the deployment manager to make fine-grained access control decisions, SA^{4}P enables centralized logging and auditing of all interactions with the physical world.

This centralized activity overview enables additional security mechanisms which are currently limited to individual devices, to be utilized on a deployment-wide scale. For example, 6thsense [153], which monitors sensor access patterns to detect adversarial activity, could be extended to observe cross-deployment sensor access information. SA^{4}P also supports work on automated IoT privacy assistants [45], by providing a robust and efficient mechanism to both collect sensor activity information, and enforce user privacy policies.

[153] *6thSense: A Context-aware Sensor-based Attack Detector for Smart Devices*, Sikder, Aksu, and Uluagac (2017)

[45] *Personalized Privacy Assistants for the Internet of Things: Providing Users with Notice and Choice*, Das et al. (2018)

In order for SA^{4}P to be deployable in as many device classes as possible, the $\mathcal{P}$EG is designed to be lightweight and modular. The former ensures compatibility with constrained environments, while the latter allows the $\mathcal{P}$EG to be efficiently adapted to individual deployment settings. Moreover, both the $\mathcal{P}$EG's lightweight structure and its modularity facilitate formal verification.

On devices with trusted execution environment (TEE) support (e.g., those with ARM TrustZone), a $\mathcal{P}$EG can be instantiated with no additional hardware requirements. On simpler devices, a stand-alone $\mathcal{P}$EG component can be added. We provide implementations for both device types. Furthermore, because SA^{4}P is orthogonal to both RA and traffic monitoring systems, these techniques can be combined with SA^{4}P to achieve even higher levels of security.

3.2 *Preliminaries*

Devices The term "IoT" refers to a wide variety of devices, ranging from tiny sensors and actuators to powerful industrial controllers. In this chapter, we only consider devices that interact with the physical environment. We exclude purely computational or general-purpose devices, e.g., smartphones. Also, although we aim for compatibility with as many device classes as possible, some are simply too constrained for our purposes. We use RFC 7228 [28] to define a set of minimum requirements for targeted devices.

[28] *Terminology for Constrained-Node Networks*, Bormann, Ersue, and Keränen (2014)

RFC 7228 lists six types of constraints for IoT devices:

- code complexity (i.e., flash memory size),
- size of state (i.e., size of RAM),
- processing power,
- user interfaces,
- connectivity, and
- available electrical energy.

In this work, we consider all devices that can:

- execute a simple state machine;
- store tens of bytes of persistent state;
- perform basic cryptographic operations;
- capture a user's indication of intent (e.g., through a button press);
- communicate with a central entity on a regular basis; and
- have enough energy to support the demands listed above.

The period of communication can range from days to tenths of seconds, depending on the required security properties. Because industrial applications tend to have a much stronger focus on network security (rather than device security) than consumer devices, we focus our design on consumer hardware. Nonetheless, our approach is also applicable to industrial settings and devices. Targeted device examples include: smart speakers, motion sensors, CO_2 sensors, door locks, and appliances.

Deployment context Although IoT devices are deployed in many diverse settings, we focus on private or semi-private ones, such as:

1. private homes,
2. office spaces, and
3. hotel rooms and short-term private rentals (e.g., AirBnB).

In each such space, people have expectations of personal safety and privacy. For example, a homeowner might want to be assured that security cameras are disabled when they are home, or that an adversary cannot turn on the stove when they are sleeping. Similarly, participants of a sensitive meeting in a hotel room want to ensure that the audio system is not snooping.

Deployment manager We assume that each deployment has a trusted *deployment manager*. The deployment manager is a logical entity that runs on a local or remote server, e.g., on an existing home gateway. It acts on behalf of the primary user of the deployment space (e.g., home owner, renter or hotel guest). The precise implementation of the deployment manager is orthogonal to this paper and is out of scope. We refer to prior work on IoT policy management [137] and privacy assistants [94, 45].

[137] *A Survey on Access Control in the Age of Internet of Things*, Qiu et al. (2020)

[94] *A Privacy Awareness System for Ubiquitous Computing Environments*, Langheinrich (2002)

[45] *Personalized Privacy Assistants for the Internet of Things: Providing Users with Notice and Choice*, Das et al. (2018)

3.3 *Adversary and Trust Model*

Considering the deployment settings above, our adversary's goal is to access a sensor or an actuator without the user's permission.

The SA4P trust model considers IoT device manufacturers to be trusted, while the code running on IoT devices is untrusted. We refer to this code—which comprises both application(s) and OS/hypervisor—as the *software runtime* or just $\mathcal{R}$untime. Its untrusted status is motivated by (i) a myriad of vulnerabilities seen in IoT devices [44, 139], and (ii) widespread use of third-party software components [194]. Nonetheless, we assume that the manufacturer can add a (small) trusted component to the device, either in the form of additional hardware, or take advantage of a TEE already present on the device, e.g., ARM TrustZone. Given its limited footprint, the code running inside this component is trusted.

[44] *A Large-Scale Analysis of the Security of Embedded Firmwares*, Costin et al. (2014)

[139] *An Experimental Security Analysis of an Industrial Robot Controller*, Quarta et al. (2017)

[194] *One Bad Apple Spoils the Barrel: Understanding the Security Risks Introduced by Third-Party Components in IoT Firmware*, Zhao et al. (2022)

We consider two types of adversaries:

The base adversary controls all network traffic to/from a device, on all of its interfaces. It may be local, remote, or both. It can arbitrary record, drop (including jam), modify, and insert packets. Moreover, it has control over the device's application code, obtained either through an exploit, or through infection at production or deployment/provisioning time. However, it does not have physical access to the device.

The non-invasive physical adversary has all capabilities of the base adversary. Additionally, it has physical access to the device. It can use the device's standard interfaces, e.g., press buttons, read any external markings, and use any wired (e.g., USB) interfaces. It can also physically remove the device. This adversary models malicious visitors to the private space. We do not consider physically invasive adversaries, as they could simply install their own malicious sensors or actuators in the private space.

Attacks against the TEEs (if present) or the code running therein are out of scope. We also do not consider attacks against the deployment manager, since, we expect it to be secure and not resource-constrained.

Our high-level security goals are:

SA^{4}P-1 A base adversary cannot access device sensing/actuation peripherals, unless when permitted by the deployment manager.

SA^{4}P-2 A non-invasive physical adversary cannot access device sensing/actuation peripherals without its presence being detected.

3.4 SA^{4}P *Overview*

A SA^{4}P deployment consists of one or more IoT devices and a trusted *deployment manager*. When a device enrolls in a deployment, its $\mathcal{R}$untime is decoupled from the physical environment.

That is, by default, SA4P-enabled devices cannot access their sensors or actuators. Upon request, the deployment manager *may* grant a device temporary access to its sensing or actuation peripherals. The manager keeps a log of all granted and denied access requests. For conciseness, we refer to sensors and actuators collectively as *interaction peripherals*, or, depending on context, simply as *peripherals*.

The key enabler of SA4P is the Peripheral Guard or $\mathcal{P}$EG, an architectural component that enables the deployment manager to enforce peripheral access policies within a device, even if that device is compromised. Concretely, the $\mathcal{P}$EG, trusted and lightweight, is placed between interaction peripherals and $\mathcal{R}$untime, as Fig. 3.2 depicts.

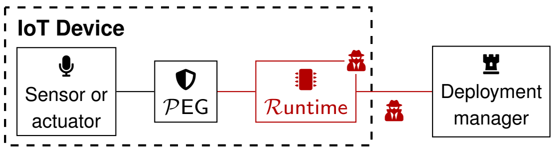


Figure 3.2: Conceptual SA4P architecture (single device). 🕵 indicates an untrusted component.

The $\mathcal{P}$EG responds to commands from the manager, allowing the latter to enforce access control policies. Although SA4P does not require synchronized clocks, the $\mathcal{P}$EG needs to have a notion of elapsed time.

To make SA4P widely applicable, the $\mathcal{P}$EG design must be *inclusive*, i.e., compatible with as many device classes as possible. This is especially important for highly constrained devices. SA4P achieves inclusiveness by applying two design strategies: complexity reduction and functional modularization.

Complexity reduction Throughout the design process, we aim to minimize complexity of the $\mathcal{P}$EG. This allows it to operate even in highly challenging environments and on constrained hardware platforms, Also, its trusted computing base (TCB) size is small. Furthermore, lower complexity simplifies formal verification and security audits. This is especially important as IoT devices, once deployed, might never experience a firmware update.

Functional modularization Given the large variety of IoT devices, no single $\mathcal{P}$EG design can cover all use-cases. Therefore, we identify four core $\mathcal{P}$EG system functions. By specifying the interface between these functions, we can implement them in separate and independent *modules*. These four functional modules can then be individually tailored to the requirements of a specific deployment scenario.

3.5 $\mathcal{P}$EG *Design*

We now overview the four $\mathcal{P}$EG modules, their functionalities, and interactions. We then discuss them in more detail and provide concrete designs. Fig. 3.3 illustrates the high-level architecture of a $\mathcal{P}$EG-enabled device, and the functionality of the four $\mathcal{P}$EG modules.

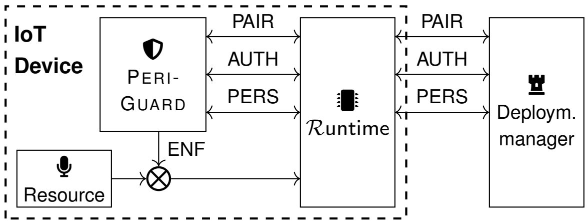


Figure 3.3: An overview of the $\mathcal{P}$EG architecture.

First, we need a mechanism to associate a $\mathcal{P}$EG (and its host device) with the deployment manager. This mechanism is implemented by the *pairing* module (PAIR, Section 3.5.1), which sets up an association in the form of shared cryptographic keying material.

The manager can then receive access requests from the $\mathcal{P}$EG-enabled device, and return access grants based on its policies. The format and behavior of access requests and access grants are defined by the *authorization* module (AUTH, Section 3.5.2). As shown in Fig. 3.3, communication between the $\mathcal{P}$EG and deployment manager is proxied by the $\mathcal{R}$untime, reducing $\mathcal{P}$EG complexity and TCB size. However, since $\mathcal{P}$EG traffic is exposed to both the (untrusted) $\mathcal{R}$untime and the network, it must be cryptographically protected to provide integrity and origin authenticity. AUTH relies on the keys established by PAIR to do so.

The access grants inform the $\mathcal{P}$EG about when it should provide access to the protected peripherals. It is then the *enforcement* module's (ENF, Section 3.5.3) responsibility to provide the actual, signal-level, enforcement of this policy.

Finally, the *persistence* module (PERS, Section 3.5.4) ensures that devices cannot be stealthily removed from SA^{4}P deployment. As discussed in Section 3.5.4, PERS is especially important with respect to physical adversaries.

3.5.1 Pairing Module

PAIR is responsible for establishing shared keying material between a $\mathcal{P}$EG and its manager.

Because of the wide variety of IoT devices, no single pairing scheme can satisfy the requirements of each deployment setting. Nonetheless, for the sake of completeness, we design and implement *semi-identified, PSK-aided authentication (SIPA)*, a 3-message pairing protocol inspired by the Noise framework [131]. Since SIPA is applicable to even highly constrained settings, it satisfies the requirements of most deployment scenarios. We present SIPA below, and assume its use for the remainder of this paper.

[131] *The Noise Protocol Framework*, Perrin (2018)

SIPA is illustrated in Fig. 3.4. In order to participate in a SIPA pairing, each $\mathcal{P}$EG must be provisioned with (i) a static public-private key-pair, and (ii) a pre-shared key (PSK) at the time of manufacturing. These keys must be stored in non-volatile, read-only memory (ROM). Each device containing a $\mathcal{P}$EG, must have both the public and pre-shared keys of its $\mathcal{P}$EG printed or etched on the outside of the device, e.g., using a 2D barcode. Additionally, each device must have a button or similar interface that can be used to instruct the $\mathcal{P}$EG to enter the pairing mode.

A SIPA handshake is executed as follows:

1. The user, acting on behalf of the deployment manager, scans the public and pre-shared keys on the device.

2. The user pushes the $\mathcal{P}$EG button of the device. This enables the $\mathcal{P}$EG pairing mode.

3. The $\mathcal{P}$EG and deployment manager perform a Noise protocol-

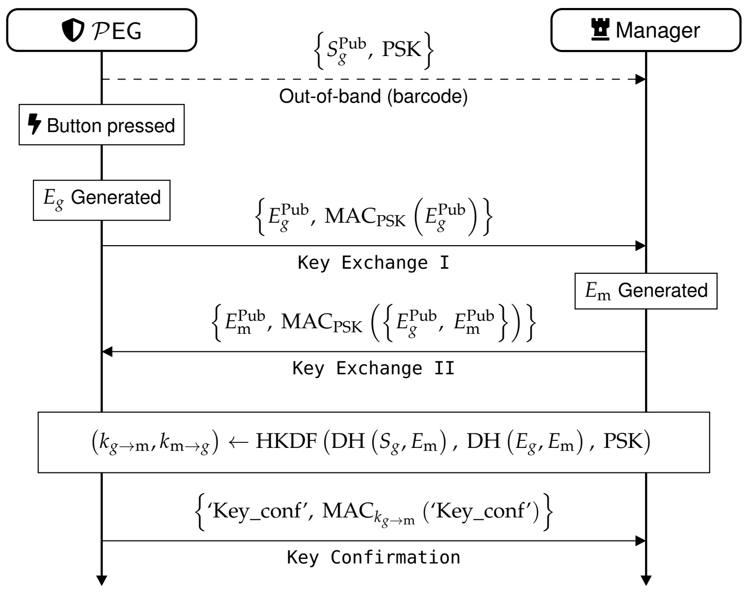

$\mathcal{P}$EG
Manager
$\left\{S_g^{\text{Pub}}, \text{PSK}\right\}$
Out-of-band (barcode)
Button pressed
E_g Generated
$\left\{E_g^{\text{Pub}}, \text{MAC}_{\text{PSK}}\left(E_g^{\text{Pub}}\right)\right\}$
Key Exchange I
E_{m} Generated
$\left\{E_{\text{m}}^{\text{Pub}}, \text{MAC}_{\text{PSK}}\left(\left\{E_g^{\text{Pub}}, E_{\text{m}}^{\text{Pub}}\right\}\right)\right\}$
Key Exchange II
$(k_{g\to\text{m}}, k_{\text{m}\to g}) \leftarrow \text{HKDF}\left(\text{DH}\left(S_g, E_{\text{m}}\right), \text{DH}\left(E_g, E_{\text{m}}\right), \text{PSK}\right)$
$\left\{\text{'Key\_conf'}, \text{MAC}_{k_{g\to\text{m}}}(\text{'Key\_conf'})\right\}$
Key Confirmation

Figure 3.4: The SIPA handshake.

inspired handshake as follows:

4.(a) Both the $\mathcal{P}$EG and manager generate ephemeral keypairs and exchange public keys.
(b) A Diffie–Hellman key exchange is performed based on each of the two-key pairs (static and ephemeral) of the $\mathcal{P}$EG, and the ephemeral keys of the manager.
(c) The result of the two Diffie-Hellmann operations and the PSK, are fed into a HKDF function, which yields two uni-directional session keys.

5. If successful, the $\mathcal{P}$EG erases all old pairing information and adopts the new keys. It then sends a key confirmation message to the manager.

6. Upon receipt of the key confirmation message, the manager adopts the new keys.

A SIPA pairing module provides the following properties:

***PAIR-1**: $\mathcal{P}$EG authentication* Whenever a manager believes to have established a session key with a $\mathcal{P}$EG G, the $\mathcal{P}$EG G

established the same session key. Hence, the $\mathcal{P}$EG is authenticated to the manager, and by extension to the user. This ensures, even while pairing over wireless interfaces, that the $\mathcal{P}$EG that performed the pairing is, in fact, the $\mathcal{P}$EG embedded in the scanned device.

***PAIR-2**: Weak manager authentication* Whenever a $\mathcal{P}$EG G establishes a key with a manager M, M (or its delegate) must have scanned the data inscribed on the device in which G is embedded. This is accomplished by mixing the PSK in the handshake, thereby verifying that the party performing the pairing had, at some point in the past, physical access to the device.

***PAIR-3**: Key secrecy* Whenever a manager establishes a key with an uncompromised $\mathcal{P}$EG, that key is not known to the adversary.

***PAIR-4**: Verification of intent* No new keys can be established without a time-adjacent physical interaction. This verifies that an entity with current physical device access intends to pair the device.

As its name suggests, SIPA does not fully authenticate both parties involved in the pairing process: while the $\mathcal{P}$EG is strongly authenticated using its long-term private key, the deployment manager is only weakly authenticated using the PSK. The motivation behind this asymmetry is twofold. First, there is the practical information limit on the $\mathcal{P}$EG: given its restricted user interfaces, providing the $\mathcal{P}$EG with the identity of the deployment manager is cumbersome and unnecessary. Second, there is no need for stronger authentication: At the end of a SIPA handshake, the deployment manager, acting on behalf of the user, is assured about the identity of $\mathcal{P}$EG. Since the $\mathcal{P}$EG adopts the newly established keying material *before* sending out the key confirmation message, it has already established a session with the deployment manager. This one-sided knowledge is sufficient to satisfy the required high-level security properties.

3.5.2 *Authorization Module*

After shared keying material has been established by the pairing module, the $\mathcal{P}$EG must be able to receive instructions from its manager on when to allow or restrict access to the protected peripheral. This functionality is provided by the authorization module.

The $\mathcal{P}$EG instantiation uses a two-message challenge-response based protocol illustrated in Fig. 3.5. When access to a protected peripheral is requested by the $\mathcal{R}$untime, the $\mathcal{P}$EG generates an authenticated challenge consisting of a counter, an access type, and a MAC using the session key over the first two fields. The counter achieves replay protection, while the access type allows one $\mathcal{P}$EG to protect multiple peripherals. This challenge is then sent to the $\mathcal{R}$untime, which forwards it to the manager.

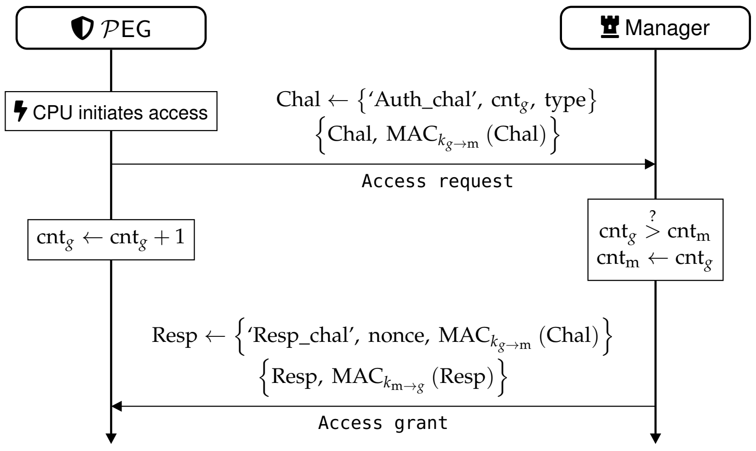

$\mathcal{P}$EG
Manager
CPU initiates access
$\text{Chal} \leftarrow \{\text{'Auth\_chal'}, \text{cnt}_g, \text{type}\}$
$\{\text{Chal}, \text{MAC}_{k_{g\to m}}(\text{Chal})\}$
Access request
$\text{cnt}_g \leftarrow \text{cnt}_g + 1$
$\text{cnt}_g \overset{?}{>} \text{cnt}_m$
$\text{cnt}_m \leftarrow \text{cnt}_g$
$\text{Resp} \leftarrow \{\text{'Resp\_chal'}, \text{nonce}, \text{MAC}_{k_{g\to m}}(\text{Chal})\}$
$\{\text{Resp}, \text{MAC}_{k_{m\to g}}(\text{Resp})\}$
Access grant

Figure 3.5: The authorization protocol flow.

Upon receipt of the challenge, the manager checks the counter value against previous values, evaluates the request against its policy, and, if access is granted, returns an authenticated response. The latter consists of a random nonce, the original authentication tag of the challenge, and a new MAC using the session key over the first two fields. The nonce prevents the manager from being used as a MAC oracle.

When the $\mathcal{P}$EG receives a valid response within a fixed time T_{chal}, it grants access to the protected peripheral for T_{auth}. Both T_{chal} and T_{auth} are design parameters. T_{chal} can be different for each protected peripheral. Potentially, multiple access types can refer to the same peripheral, but with different associated values of T_{auth}.

The authorization module described above provides the following properties:

***AUTH-1**: Bounded-window authorization* It is guaranteed that access to a protected peripheral is only granted within well-defined time windows. Each time window starts at the moment that the manager transmits a grant and ends at most $T_{\text{chal}} + T_{\text{auth}}$ time units later.

***AUTH-2**: Bounded-duration authorization* The protected peripheral can be accessed for at most T_{auth} time units per grant issued by the manager.

3.5.3 *Enforcement Module*

The enforcement module implements the low-level access control to the peripherals. Because of the large variety of peripheral interfaces, different authorization modules must be used for different types of interfaces. We provide implementation examples for three types of interfaces.

Open collector (Fig. 3.6) Communication over open collector-based interfaces can be interrupted by pulling the signal lines to ground potential using transistors. The $\mathcal{P}$EG forces all signal lines to ground, unless when peripheral access is granted.

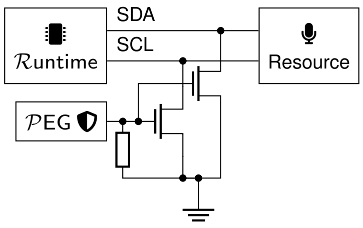


Figure 3.6: Example enforcement module implementation for open collector (e.g., I2C) communication.

Push-Pull (Fig. 3.7) Contrary to open collector type interfaces, forcing the signal lines of push-pull-based interfaces to ground may create physical damage. Instead, tri-state digital buffer elements can be used to decouple the signal lines on both sides of the interface.

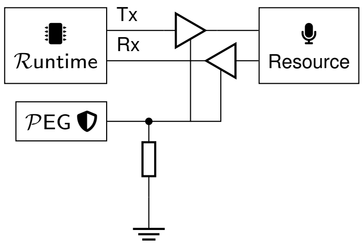


Figure 3.7: Example enforcement module implementation for push-pull (e.g., UART) communication.

Analog (Fig. 3.8) Digital buffers cannot be used for analog interfaces. However, similar functionality can be achieved using an operational amplifier in a non-inverting configuration. A transistor in the feedback circuit can be used to clamp the analog output to the positive supply bus voltage, inhibiting communication.

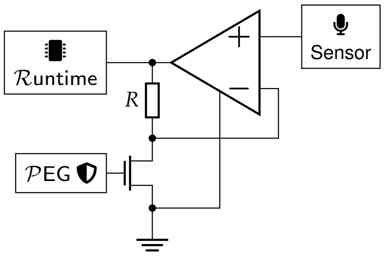


Figure 3.8: Example enforcement module implementation for analog signals.

Regardless of its implementation, an enforcement module should provide the following property:

***ENF-1**: Peripheral isolation* The enforcement module ensures that the $\mathcal{P}$EG can isolate the protected peripheral from the $\mathcal{R}$untime. In more concrete terms: the enforcement module can prevent the $\mathcal{R}$untime from interacting with the protected peripheral.

3.5.4 *Persistence Module*

Although the pairing module described above provides guarantees about the pairing state of the $\mathcal{P}$EG at the time of the pairing handshake, it does not prevent the $\mathcal{P}$EG from being re-associated[1] with another manager at a later point in time. Nevertheless, when combined with the pairing module's verification of intent (i.e., the requirement for a button to be pressed), this suffices to meet our security goals for the base adversary (**SA^{4}P-1**).

[1] We use the term *re-associated* in favor of *repaired* for clarity.

However, when considering the non-invasive physical attacker, additional guarantees are needed. Indeed, the non-invasive physical attacker could press the pairing button on the device, and re-associate the device to a malicious manager. The owner would not be informed of this, leading to a potential violation of security goal **SA^{4}P-2**.

The persistence module mitigates this attack by providing guarantees about the continued pairing state of a $\mathcal{P}$EG. Different mechanisms can be used to achieve this. For example, the persistence module could require the current manager's approval before a $\mathcal{P}$EG can be unpaired. Doing so would ensure that the manager, and by extension the user, would always be informed when a device leaves the SA^{4}P deployment. Although this might be a desirable strategy in some cases, it also introduces a significant availability risk: if a $\mathcal{P}$EG can no longer communicate with its manager, (e.g., because the manager's keying material was lost), it can no longer be un- or re-associated, and the device in which this $\mathcal{P}$EG is embedded effectively becomes a paper weight.

Instead we propose a persistence module based on liveness. Concretely we propose the design illustrated in Fig. 3.9. In this design, the manager periodically sends an authenticated nonce to the $\mathcal{P}$EG, which re-authenticates and echoes back the nonce last received from the manager. Although this protocol is depicted as stand-alone in Fig. 3.9, its messages can be piggy-backed onto the messages of the authentication protocol.

The persistence protocol provides the following properties:

***PERS-1**: Retroactive proof of pairing state* Whenever the man-

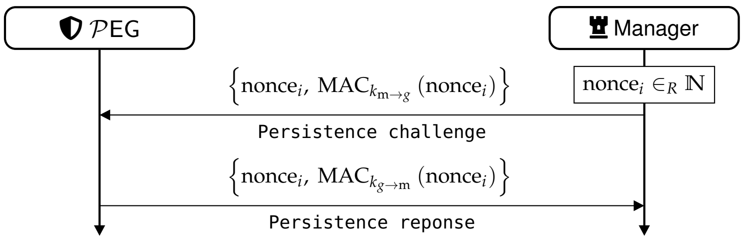

𝒫EG
Manager
$\text{nonce}_i \in_R \mathbb{N}$
$\left\{\text{nonce}_i, \text{MAC}_{k_{m\to g}}(\text{nonce}_i)\right\}$
Persistence challenge
$\left\{\text{nonce}_i, \text{MAC}_{k_{g\to m}}(\text{nonce}_i)\right\}$
Persistence reponse

Figure 3.9: The persistence protocol flow.

ager M receives back a nonce N from a 𝒫EG G, it is confirmed that the 𝒫EG G was paired to the manager M at the point that the nonce N was transmitted by the manager M.

This protocol thus provides retroactive confirmation of pairing state. Moreover, when this protocol is repeated with period P, an absence of responses from the 𝒫EG indicates that an unpairing event has occurred, meaning that devices that leave the SA4P deployment will be detected within a bounded amount of time. Further, when the 𝒫EG disallows unpairing for a period $P' > P$ after each received persistance challenge, even stronger properties are attained, albeit at the cost of reduced useability.

3.6 *Security Analysis*

Key to any SA4P deployment is high trust in the access enforcement mechanism. To this end, we leverage a combination of formal and circuit analysis methods to analyze the security of the 𝒫EG modules we introduced above. The formal methods are used to analyze the protocol-based modules, and circuit analysis is used to evaluate the various proposed enforcement modules.

Thanks to the 𝒫EG's modular design, each module can be analyzed independently: the properties provided by the authentication and persistence modules are only dependent on the session keys being correctly established by the pairing module. The enforcement module has no dependencies. Therefore, when individual modules are updated or replaced, only those modules need to be re-verified.

3.6.1 Pairing Module

We analyze the pairing module using the Tamarin protocol prover [107]. Our model assumes that all long-term public keys are registered at, and can be retrieved from a public key server. We model the scanning of the public and pre-shared keys inscribed on the device using two rules: the `Scan_device` rule, which generates a fact representing a manager's knowledge of the inscribed keys, and the `Scan_device_adversary` rule, which adds the inscribed keys to the adversaries knowledge. Additionally, a rule that leaks a $\mathcal{P}$EG's internal long-term secrets is included in the model.

[107] *The TAMARIN Prover for the Symbolic Analysis of Security Protocols*, Meier et al. (2013)

Further, the model does not limit the number of $\mathcal{P}$EGs or managers in a trace and supports re-associating after a completed handshake. The model also allows for an ongoing pairing operation to be aborted when a new pairing operation is started. Moreover, for each of these operations, the internal key management of the $\mathcal{P}$EG is modeled.

Using the model described above, we verify the validity of the properties **PAIR-1** through to **PAIR-4**. We find that all of them hold.

3.6.2 Authorization Module

We model the authorization protocol using Tamarin. Our model accounts for re-association events, key disclosure, counter reveals, timer expirations, and guard reboots.

Because Tamarin's discrete time system only models order, but not elapsed time, the properties **AUTH-1** and **AUTH-2**, cannot be proven directly. Instead, we apply a hybrid approach, in which we prove base statements using Tamarin, from which we then derive the properties **AUTH-1** and **AUTH-2** using conventional logic.

To this end, we instrument the model with action facts that indicate when the T_{chal} and T_{auth} timers are started or when they expire. We then prove the following properties, referring to Table 3.1 for notation.

T-1 A T_{auth} timer cannot start after its associated T_{chal} timer has

Symbol	Description
T_x	The name of a timer or duration
t_x	The time point at which event x occurs
$t_x^{\text{start}}, t_x^{\text{expire}}$	The time point at which timer x is started or expires, respectively
S_x, E_x	The static and ephemeral key pairs of entity x, respectively
$S_x^{\text{Pub}}, E_x^{\text{Pub}}$	The static and ephemeral public key of entity x, respectively

Table 3.1: An overview of symbolic notation.

expired: $t_{\text{auth}}^{\text{start}} < t_{\text{chal}}^{\text{expire}}$.

***T*-2** A T_{chal} timer must be started before the manager can issue a corresponding grant: $t_{\text{chal}}^{\text{start}} < t_{\text{grant}}$.

***T*-3** A T_{auth} timer can only start after the manager has issued the corresponding grant: $t_{\text{auth}}^{\text{start}} > t_{\text{grant}}$.

***T*-4** A peripheral interaction can only occur between the start and expiry of a T_{auth} timer: $t_{\text{auth}}^{\text{start}} < t_{\text{interaction}} < t_{\text{auth}}^{\text{expire}}$.

***T*-5** At most one T_{auth} timer can be started per grant issued by the manager.

From these properties, we can then prove **AUTH-1** as follows. From **T-1** we know

$$t_{\text{auth}}^{\text{start}} < t_{\text{chal}}^{\text{expire}} = t_{\text{chal}}^{\text{start}} + T_{\text{chal}}$$

Therefore,

$$t_{\text{auth}}^{\text{end}} = t_{\text{auth}}^{\text{start}} + T_{\text{auth}} < t_{\text{chal}}^{\text{start}} + T_{\text{chal}} + T_{\text{auth}}$$

Combined with **T-2** this yields

$$t_{\text{auth}}^{\text{end}} < t_{\text{grant}} + T_{\text{chal}} + T_{\text{auth}}$$

Or, in words, the T_{auth} timer expires at most $T_{\text{chal}} + T_{\text{auth}}$ after the manager issued the corresponding grant.

Similarly, **T-3** states that the T_{auth} timer must be started after the manager's grant. Combining these two results with **T-4** proves property **AUTH-1**. □

We prove **AUTH-2** by combining **T-5** and **T-4**, which leads directly to **AUTH-2**. □

3.6.3 *Enforcement Module*

Unlike the other three modules, the enforcement module is not protocol-based. Therefore, we cannot use the Tamarin prover to perform our security analyses. Instead, **ENF-1** is verified using circuit analysis.

Open Collector (Fig. 3.10) When the $\mathcal{P}$EG pulls the data lines to ground, the drain source resistance R_{ds} of the MOSFETs in Fig. 3.6 will induce a small voltage drop across each MOSFET. Assuming a common 2N7000 MOSFET [125] with $R_{ds} = 6.0\,\Omega$, a system voltage $V_{cc} = 3.3\,\mathrm{V}$, and a data pull-up resistor value $R_{bus} = 2\,\mathrm{k}\Omega$, the resulting voltage on the data lines will be $\frac{6\,\Omega}{2\,\mathrm{k}\Omega+6\,\Omega} \cdot 3.3\,\mathrm{V} \approx 10\,\mathrm{mV}$. Assuming the line driver to have perfect switching capabilities, it could thus toggle the data lines between 0 and 10 mV.[2] When connected to a digital input, both values will be read as low, so no communication is possible. However, care must be taken to ensure that no other, more sensitive sampling hardware can be connected to the bus, e.g., be reassigning the data pins to an analog-to-digital converter (ADC).

[125] *2N7000G: Small Signal MOSFET 200 mApms, 60 Volts*, ON Semiconductor (2011)

[2] Most components will not do this. E.g., standard-complying I2C implementations will interpret the low bus voltage as an ongoing transmission and refrain from sending [122].

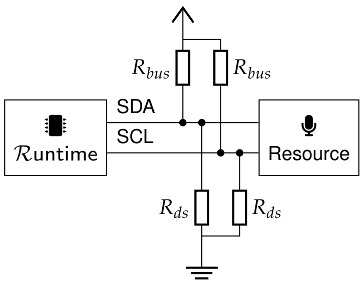


Figure 3.10: Circuit analysis of the enforcement circuit for open collector communication shown in Fig. 3.6.

Push-Pull (Fig. 3.11) Digital buffers provide much better isolation than the circuit in Fig. 3.6. For example, the MC74VHC541 buffer [126] lists a maximum tri-state leakage current of 0.25 µA. This makes it extremely unlikely that signal could be recovered using IoT device hardware.

[126] *MC74VHC541: Octal Bus Buffer*, ON Semiconductor (2014)

Analog (Fig. 3.12) When the $\mathcal{P}$EG disables the signal, the feedback factor of the opamp is $b = \frac{R_{ds}}{R_{ds}+R}$. Again assuming the

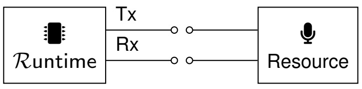


Figure 3.11: Circuit analysis of the enforcement circuit for push-pull communication shown in Fig. 3.7.

2N7000 MOSFET [125], and $R = 10\,\mathrm{k}\Omega$ this results in $b \approx 6 \cdot 10^{-4}$. Assuming a LMV358 opamp [166] with a large-signal differential voltage gain $A_{vd} = 10^4$ the closed loop gain becomes $A = b^{-1} \cdot \left(1 + (A_{vd} \cdot b)^{-1}\right)^{-1} \approx 1429$ [84]. Assuming 0 V and 3.3 V supply rails, this means that any input signal above 2.3 mV would be clamped to the 3.3 V rail, rendering the output meaningless.

[125] *2N7000G: Small Signal MOSFET 200 mApms, 60 Volts*, ON Semiconductor (2011)

[166] *LMV3xx Low-Voltage Rail-to-Rail Output Operational Amplifier*, Texas Instruments (1999)

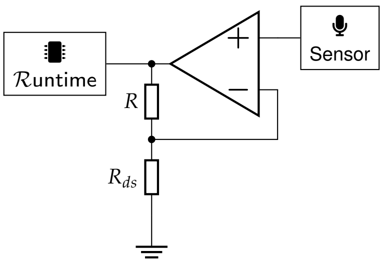


Figure 3.12: Circuit analysis of the enforcement circuit for analog signals shown in Fig. 3.8.

[84] *Application Report: Understanding Operational Amplifier Specifications*, Karki (2021)

3.6.4 *Persistence Module*

We model the persistence protocol using the Tamarin protocol prover. Our model covers repairing events, key disclosure, multiple $\mathcal{P}$EG transmissions with the same nonce, and a loss of nonce state on the guard (for example, caused by a reboot). We find that property **PERS-1** holds.

3.6.5 *Reference Monitor Properties*

We show that SA4P satisfies the three defining properties of a reference monitor: (1) non-bypassability, (2) tamper proofness, and (3) verifiability. Non-bypassability prevents unauthenticated peripheral access, tamper proofness prevents manual or programatic modifications to the reference monitor, and verifiability prescribes a sufficiently small TCB that can be analyzed and proven correct. We focus this analysis on the $\mathcal{P}$EG.

Non-Bypassability Peripheral access is restricted by the enforcement module, unless permitted by the authorization module (**ENF-1**). Access is only permitted by the authorization module for T_{auth} seconds in response to a fresh and authentic grant from the manager. Freshness is ensured by the request's counter and the T_{chal} window, while authenticity is ensured by the MAC tag. The deployment manager only grants access to fresh and authentic requests permitted by its policy. Thus, the authorization module only allows peripheral access if it shortly before (at most $T_{\text{chal}} + T_{\text{auth}}$ seconds) created a corresponding request that was granted by the manager.

Tamper proofness The trusted $\mathcal{P}$EG is implemented either inside ARM TrustZone on the $\mathcal{R}$untime's CPU, or on a physically separate MCU. We only consider software and non-invasive physical tampering, invasive physical tampering is not considered under our adversary model. TrustZone, like the physically separate MCU, provide strong isolation between the secure $\mathcal{P}$EG and the untrusted $\mathcal{R}$untime. By connecting any physical interface directly to pins controlled by the TrustZone secure world, or directly to the separate MCU, physical interactions cannot be tampered with by the $\mathcal{R}$untime. **PERS-1** ensures attempts to re-associate a device are detected.

Verifiability All four $\mathcal{P}$EG modules are formally verified, and can be implemented using simple primitives. The deployment manager is more complex as it implements the access-control policy. When deployment managers are implemented, it is important to integrate verifiability in the design and implementation process.

3.7 Implementation

To assess SA^{4}P's practicality, we implement two $\mathcal{P}$EG variants. The first uses a dedicated MCU to implement a stand-alone $\mathcal{P}$EG and the second uses a TrustZone-enabled Cortex-M MCU as the $\mathcal{R}$untime, placing the $\mathcal{P}$EG logic in the secure TrustZone partition of the same MCU.

We instantiate both variants on the ultra-low-power STM32L552ZE MCU. For the stand-alone $\mathcal{P}$EG, we fully disable TrustZone. The STM32L552ZE runs at 110 MHz, has 256 kB of SRAM, and 512 kB of flash memory. We set $T_{\text{chal}} = 20\,\text{ms}$ and $T_{\text{auth}} = 10\,\text{s}$.

3.7.1 Stand-alone Peripheral Guard

The stand-alone $\mathcal{P}$EG is implemented on a dedicated MCU. Although it would be possible to independently connect this $\mathcal{P}$EG to the network, we refrain from doing so, and opt to proxy all commands over the $\mathcal{R}$untime using a UART connection. This has two advantages: (i) it removes the need for a (complex) networking stack on the $\mathcal{P}$EG, significantly reducing its TCB size; and (ii) it makes it easier for the $\mathcal{R}$untime to stay in sync with the $\mathcal{P}$EG. Although this design allows the $\mathcal{R}$untime to perform a denial-of-service (DoS) attack against the $\mathcal{P}$EG, this would be strictly against its own interests.

Message Format To facilitate communication over the serial link, we implement a serial message format to carry the protocol messages from Figs. 3.4, 3.5 and 3.9. For the serial transmission, a 3 B header consisting of a 1 B type field and a 2 B length field is added to each message. To reduce the number of messages, we combine authorization and persistence protocol messages.

Cryptography Asymmetric key exchanges are realized using the compact25519 library [93] on Curve25519. All symmetric keys are 256 bit in size, MACs are implemented using HMAC with SHA256. HMAC operations and randomness generation are hardware-supported.

[93] *compact25519: A compact portable X25519 + Ed25519 implementation*, Landman (2022)

Counters and Persistent Memory The authorization protocol uses a monotonic counter to prevent message replays. It can either be implemented using persistent memory or a secure clock. Since the latter requires complex hardware support, and because persistent memory is already required to store the session keys, we opt for the former. However, counter update events are expected to be frequent (unlike re-association events). Thus, care must be taken not to wear out the MCU's memory prematurely by generating excessive writes. For example, the flash memory of the STM32L552ZE MCU is specified as being able to withstand at least 10 000 writecycles. This number is unlikely to ever by reached via re-association events. However, only five counter updates per day suffices to reach this number in 5 years.

To avoid memory wear, we use a hybrid 8 B counter consisting of two 4 B subcounters: upper and lower, referred to as the *reboot* and *request* counters, respectively. The latter functions as expected, incrementing each time a challenge is generated. However, instead of being stored in persistent memory, it is stored in RAM and initialized to zero at each device boot. Meanwhile, the reboot counter *is* stored in persistent memory, and incremented each time the device boots. Hence, the combined counter remains strictly incremental, and it only requires a write to persistent memory once per device boot cycle, thus solving the memory constraints. To ensure the integrity of the reboot counter in case of boot interruptions, it is replicated three times in memory. A corrupted value can thus always be recovered using majority voting. Alternatively, we could use a specialized memory, e.g., ferroelectric RAM (FRAM), for counter storage, though such memory types are significantly less common.

Timeouts Because the state of the T_{chal} timer only needs to be checked when processing an incoming message, it is implemented using system ticks. Conversely, the T_{auth} uses a dedicated hardware timer that triggers an interrupt when it expires. This ensures that access to the protected peripheral is always promptly removed.

Enforcement Module Our $\mathcal{P}$EG implementation writes a binary signal representing the current peripheral enforcement state to a General Purpose Input/Output (GPIO) output pin. An external enforcement circuit (e.g., one of the circuits in Section 3.5.3) can be connected to this pin.

3.7.2 TrustZone Peripheral Guard

Unlike the dedicated $\mathcal{P}$EG, the TrustZone variant shares processing hardware with the the $\mathcal{R}$untime. Concretely, one TrustZone enabled Cortex-M MCU is flashed with two binaries: secure and non-secure. We use the former for the $\mathcal{P}$EG functionality, and the latter for $\mathcal{R}$untime code. Isolation of the $\mathcal{P}$EG is provided by TrustZone.

Implementing the $\mathcal{P}$EG using TrustZone has the primary advantage of not requiring additional hardware. Also, TrustZone-based $\mathcal{P}$EGs might be applicable to existing products with TrustZone-capable MCUs.[3]

[3] No general statements can be made about the feasibility of this approach, since it is largely dependent on the design of individual products.

API A further advantage of a TrustZone-based $\mathcal{P}$EG is that no (slow) communication bus between the $\mathcal{P}$EG and $\mathcal{R}$untime is needed. Instead, the TrustZone $\mathcal{P}$EG exposes a narrow API to the $\mathcal{R}$untime, consisting of one call per incoming message type. Messages are passed as function arguments and return values.

Enforcement Module Contrary to the dedicated $\mathcal{P}$EG, the TrustZone-based $\mathcal{P}$EG does not need external enforcement circuitry. Instead, the enforcement module can dynamically modify the MCU's TrustZone configuration to provide or revoke the $\mathcal{R}$untime's access to the interface to which the protected peripheral is connected [158].

[158] *Embedded Systems Security and TrustZone*, Slamaris (2022)

Preventing Interference Having the $\mathcal{R}$untime and $\mathcal{P}$EG share a computation platform increases the risk of interference by the $\mathcal{R}$untime into $\mathcal{P}$EG operations. Although default TrustZone behavior suffices to protect the $\mathcal{P}$EG's secrets (i.e., keying material), additional care must be taken so that the untrusted $\mathcal{R}$untime code cannot interfere with the $\mathcal{P}$EG's time base. Most significantly, the `PRIS` bit in the Application Interrupt and Reset Control Register (AIRCR) must be set to ensure that the code running in the non-secure TrustZone partition (i.e., the $\mathcal{R}$untime) cannot prevent the $\mathcal{P}$EG code in the secure partition from executing by masking it's interrupt service routines [14].

[14] *ARM Cortex-M33 Devices Generic User Guide*, ARM Ltd. (2020)

3.8 Evaluation

To evaluate our $\mathcal{P}$EG implementations, we first perform a macro benchmark which focuses on the overall system performance of a $\mathcal{P}$EG-enabled device. To gain further insights, a series of micro benchmarks that evaluate the performance of the $\mathcal{P}$EGs in isolation is performed as well. We also report on binary sizes.

3.8.1 Macro Evaluation

Modifying devices to adopt the SA^{4}P philosophy will inevitably have an impact on system operations. Concretely, the requirement to obtain permission ahead of peripheral interactions introduces both bandwidth overhead and access latency. Given that pairing events are expected to be rare, we focus this evaluation on access requests and grants.

Bandwidth overhead, access event The communication overhead in terms of message sizes can be determined directly from the protocol specification (Sections 3.5.1, 3.5.2 and 3.5.4) and the serial message format (Section 3.7.1). The resulting message sizes are shown in Table 3.2.

Message	**Generated by**	**Size [B]**
Pairing		
Key exchange I	$\mathcal{P}$EG	67
Key exchange II	Manager	67
Key confirmation	$\mathcal{P}$EG	35
Authorization & persistence		
Access initialization	$\mathcal{R}$untime	3
Access request	$\mathcal{P}$EG	61
Access grant	Manager	51
Access confirmation	$\mathcal{P}$EG	6

Table 3.2: Sizes of messages exchanged between the $\mathcal{P}$EG and $\mathcal{R}$untime, including the 3 B serial encapsulation header.

3.8.2 Macro Evaluation

Latency overhead, access event To quantify latency overhead, we performed end-to-end system measurements using the setup shown in Fig. 3.13. The setup consists of a proof-of-concept $\mathcal{P}$EG-enabled device that communicates with a dummy deployment manager over a standard Ethernet network. The $\mathcal{P}$EG-enabled device consists of a button functioning as a rudimentary sensor, a STM32 as dedicated $\mathcal{P}$EG, and a second, identical, MCU functioning as the $\mathcal{R}$untime. Communication between the $\mathcal{P}$EG and the $\mathcal{R}$untime uses a UART bus running at 921 kbaud. The $\mathcal{R}$untime is connected to the Ethernet network using a WIZnet W7500S2E-R1 UART to Ethernet bridge, running at 461 kbaud. The dummy manager simply grants every request. It is implemented in Golang and runs on a commodity laptop.

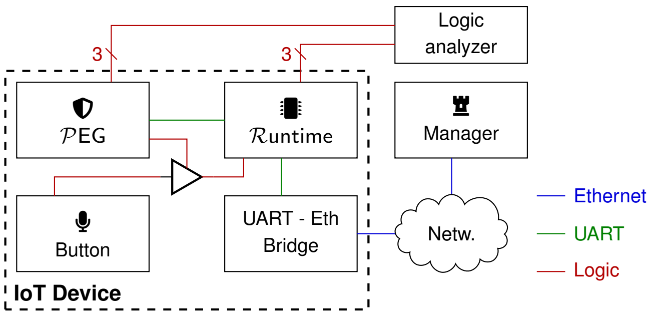


Figure 3.13: Schematic overview of the macro evaluation setup.

To extract timing information, both the $\mathcal{P}$EG and $\mathcal{R}$untime are connected to the same logic analyzer sampling at 16 MHz, each using a 3-bit logic bus. The code running on both platforms is instrumented to write code points to the bus, providing insight into internal operations and timing.

We use the setup described above to measure the overall system performance of an access request (including the piggy-backed persistence exchange). The results are shown in Fig. 3.14.

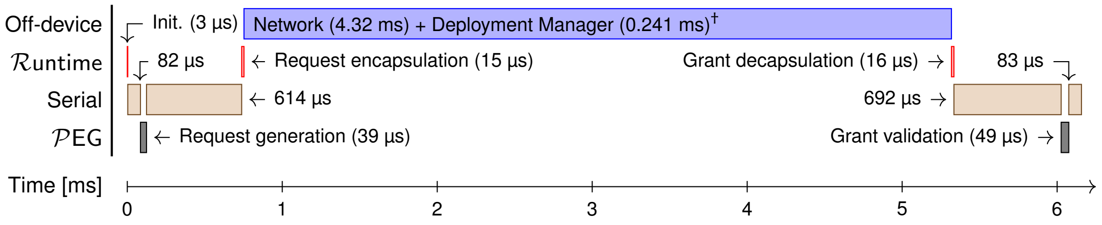


Figure 3.14: Timeline of an access requests to a dedicated $\mathcal{P}$EG. Averaged values, $N = 468$. †Deployment manager processing times were measured during a separate experiment with $N = 100$ and an independent time base.

As can be seen in Fig. 3.14, it takes, on average, 6.16 ms from the point where the $\mathcal{R}$untime initiates an access request to the point at which the $\mathcal{R}$untime is informed that peripheral access has been provided by the $\mathcal{P}$EG. This duration is primarily dominated by serial (1.47 ms) and network[4] (4.32 ms) delays, which sum up to 5.79 ms. We attribute the majority of the network delays to the relatively slow WIZnet module.

Processing times on the protocol endpoints ($\mathcal{P}$EG and deployment manager) are an order of magnitude shorter, with a combined processing time of 0.09 ms measured on the $\mathcal{P}$EG, and 0.24 ms measured on the deployment manager. The latter being higher due to the use of a general purpose operating system on the deployment manager. The forwarding overhead on the $\mathcal{R}$untime was measured to sum up to 0.03 ms, which we consider to be negligible.

[4] Network delays include serial transmissions to the UART to Ethernet bridge.

Pairing events The sizes of pairing messages are shown in Table 3.2. We informally measured a pairing event to take around 9 s. More detailed measurements are given in the following section on micro evaluation.

3.8.3 Micro Evaluation

To gain further insight into $\mathcal{P}$EG performance, we continue with a set of micro benchmarks. Similar to the macro evaluation setup, we connect the $\mathcal{P}$EG to a logic analyzer using a 3-bit bus, and instrument the $\mathcal{P}$EG code to write code points to this bus at significant points in its program.

Dedicated $\mathcal{P}$EG, access event timing We start by taking a more detailed measurement of the processing times required to generate an access request and validate a grant on the dedicated $\mathcal{P}$EG. The results of this measurement are shown in Table 3.3. As can be seen from the table, the processing times are highly deterministic, with low spread. The small discrepancy between the values in Fig. 3.14 and Table 3.3 can be attributed to slight differences in code instrumentation and compile-time optimization.

	Mean	Std. Dev.	Min	Max
Dedicated $\mathcal{P}$EG (N=288)				
Request generation	40.34	0.11	40.25	41.25
Grant validation	50.28	0.18	50.17	51.17
TrustZone $\mathcal{P}$EG (N=324)				
Request generation	33.52	0.34	32.62	34.63
Grant validation	49.98	0.54	48.63	52.87

Table 3.3: Processing times for access events. Values in µs.

Dedicated $\mathcal{P}$EG, access event load profile To further break down the numbers shown in Table 3.3, Fig. 3.15 shows relative load profiles for the $\mathcal{P}$EG. We see that for both messages, the workload is dominated by the cryptographic tag verification, which is to be expected.

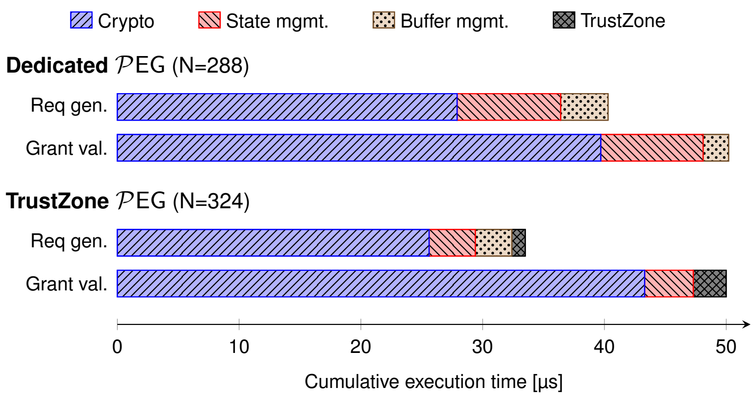


Figure 3.15: $\mathcal{P}$EG load profiles for access-event messages.

TrustZone $\mathcal{P}$EG, access event timing Next, we compare the performance of the dedicated $\mathcal{P}$EG to the TrustZone-based $\mathcal{P}$EG. Table 3.3 and Fig. 3.15 show the execution times of `get_challenge()` and `put_response()` $\mathcal{P}$EG API calls as observed by the $\mathcal{R}$untime code. These calls respectively initiate an access request, or supply the $\mathcal{P}$EG with a response from the manager. We see that the latencies of the two implementations are similar. However, the TrustZone numbers already include the $\mathcal{P}$EG to $\mathcal{R}$untime communication, whereas the dedicated $\mathcal{P}$EG relies on time-intensive serial communication. Therefore, the system performance of the TrustZone $\mathcal{P}$EG is superior. That said, as is shown in Table 3.3, the response times of the TrustZone $\mathcal{P}$EG are less stable than those of the dedicated $\mathcal{P}$EG. This is to be expected, as the TrustZone implementation shares a processor core with the $\mathcal{R}$untime.

Dedicated $\mathcal{P}$EG, pairing event timing Although pairing events are expected to be rare, we performed measurements to get an understanding of their performance. Table 3.4 shows the time required to process the pairing messages on the $\mathcal{P}$EG. Table 3.5 lists the measured duration of individual cryptographic operations, providing more insight in the composition of the delays shown in Table 3.4. We see that (as expected) the processing times are dominated by the cryptographic operations. We note that two Diffie-Hellmann operations take place for each key confirmation message.

Outgoing message	Mean	Std. Dev.	N
Key exchange I	2891	0.30	281
Key confirmation	5737	0.92	155

Table 3.4: Dedicated $\mathcal{P}$EG processing times for pairing message processing and generation. Values in ms.

Operation	Mean	Std. Dev.	N
Ephemeral key pair generation	2891	0.30	281
Diffie-Hellmann derivation	2868	0.46	155
HKDF computation	0.08	$<$ 0.01	155

Table 3.5: Dedicated $\mathcal{P}$EG processing times for pairing-related cryptographic operations. Values in ms.

3.8.4 Memory Footprint

We measure the memory footprint of both $\mathcal{P}$EG implementations. The dedicated $\mathcal{P}$EG has a binary size of 14.97 kB and a RAM footprint of 3.18 kB. The TrustZone-based $\mathcal{P}$EG has a binary size of 6.62 kB and a RAM footprint of 1.9 kB.

3.9 Discussion

3.9.1 Peripheral Access Latency

The results from Section 3.8 confirm the feasibility of the SA^{4}P approach. Although the newly introduced peripheral access latency ($\leq$ 6 ms) is non-negligible, it is not of a prohibitive nature for the applications we envision; system response times below 100 ms are generally perceived as instantaneous by users [118]. Moreover, significant performance improvements are still achievable. Concretely, we expect that when a TrustZone-based $\mathcal{P}\mathsf{EG}$ is used, and our proof-of-concept networking setup is replaced by a networking stack running directly on the $\mathcal{R}\mathsf{untime}$, an access latency below 1 ms is attainable. Additionally, $\mathcal{P}\mathsf{EG}$ operations have shown to be both fast and deterministic, which facilitates access request scheduling.

[118] *Usability Engineering*, Nielsen (1993)

3.9.2 *T_{chal}, T_{auth}, and Network Overhead*

The timer durations T_{chal} and T_{auth} should be set based on the characteristics of the deployment: T_{chal} should be set to sum of the largest expected round trip time (RTT) from the $\mathcal{P}\mathsf{EG}$ to the deployment manager, and the highest expected deployment manager response time. In most deployments this value will be on the order of milliseconds or less.

Setting T_{auth} is more complicated, as it requires a trade-off between control and network overhead: setting T_{auth} high results in coarse access control and low overhead, whereas a low T_{auth} provides fine-grained control at the cost of higher network overhead. To quantify this overhead, consider a deployment with $T_{\text{auth}} = 10\,\text{s}$, and in which access requests are sent with a 100 ms overlap to ensure access continuity. Assuming our SA4P instantiation, the resulting network overhead during peripheral access events would average $\frac{1}{10-0.1}\text{req/s} \times (61 + 51)\text{B/req} = 90\,\text{bps}$. Although we expect this data rate to be acceptable in most application settings, this might not hold for some highly constrained scenarios. Strategies to reduce the network overhead include:

- Increasing T_{auth}. In some deployment scenarios significantly longer values of T_{auth} can be considered. For example, in wireless sensor networks where the primary goal of SA4P deployment is to remove peripheral access when the deployment is compromised, values of T_{auth} up to multiple hours may be appropriate.

- The length of the various protocol fields can be reduced, albeit at the cost of weakened cryptographic properties.

- Multiple access types associated with the same peripheral, but with different associated T_{auth} values, can be used.

3.9.3 *High-Level Context*

Although the low-level nature of the $\mathcal{P}$EG makes the design inclusive and robust, it limits the amount of high-level contextual information that is inherently available to base access decisions on. This means that the deployment manager must actively collect such information from across the deployment. How this collection should be implement is beyond the scope of this paper. Additionally, the SA^{4}P design philosophy is not intended to replace existing high-level access control mechanism, but rather to complement them.

Example: combination with attestation mechanisms One type of high-level context that could be gathered, is the attestation state of the $\mathcal{P}$EG-enabled devices. By requiring the $\mathcal{R}$untime to present an attestation proof together with each access request, the deployment manager can ensure that peripheral access is rapidly removed after a device has been compromised. Doing so would remove significant adversarial utility from compromised devices, effectively rendering them to be generic network-connected computational nodes.

3.9.4 *Non-Binary Enforcement Modules*

In Section 3.6.3 we only considered *binary* enforcement modules, i.e., enforcement modules that either allow for full peripheral access or none at all. However, it is also possible to design non-binary enforcement modules, e.g., by integrating a low-pass filter into their design. Doing so would allow the $\mathcal{P}$EG to provide more fine-grained access control. For example, it has been shown that the high frequency components in a humidity signal can reveal information about human presence [70]. A low-pass filter enabled $\mathcal{P}$EG could selective provide access to the average room

[70] *Don't Sweat Your Privacy: Using Humidity to Detect Human Presence*, Han et al. (2007)

humidity while filtering out sensitive room occupancy information.

3.9.5 *Protecting Complex Actuators*

For simple actuators, our $\mathcal{P}$EG design is directly applicable. In fact, as part of this research project we have created a demo that uses our $\mathcal{P}$EG design to control access to an electric strike plate, giving the deployment manager control over when a door can be opened.

For other actuators (e.g., lights), a designer might opt to place a latch or a flip flop behind the $\mathcal{P}$EG, so that an access permission is only required to toggle the actuator. More generally, a setpoint-based control loop could be used, thus only requiring manager authorization for setpoint adjustments. Such an architecture is depicted in Fig. 3.16. The primary challenge associated with this approach is that all components placed behind the $\mathcal{P}$EG must be trusted.

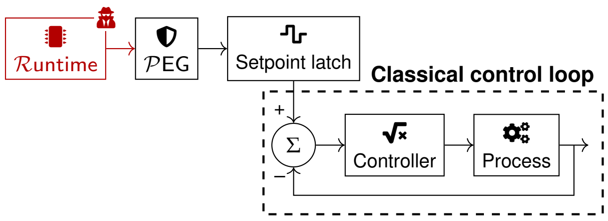


Figure 3.16: A SA^{4}P design only requiring authorization for setpoint adjustments. 🕵 indicates untrusted components.

3.9.6 *Information Leakage*

The $\mathcal{P}$EG design does not encrypt network traffic, since, unlike data authenticity, data secrecy is not required for our security goals. However, this means that passive network adversaries can observe access requests in cleartext, including the access type field. This exposure can be addressed by creating an encrypted tunnel between the $\mathcal{R}$untime and the deployment manager. However, encryption offers limited protection: previous work has shown that even encrypted IoT traffic leaks significant private information [12, 174, 2].

[12] *Spying on the Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic*, Apthorpe et al. (2017)

[174] *IoTAthena: Unveiling IoT Device Activities From Network Traffic*, Wan et al. (2022)

[2] *Peek-a-boo: I see your smart home activities, even encrypted!*, Acar et al. (2020)

3.10 Summary

As IoT devices are becoming increasingly ubiquitous and intertwined with many aspects of everyday life, they are being plagued by vulnerabilities, creating a security and privacy nightmare. The proposed SA4P framework helps address this ongoing challenge: by controlling a device's ability to interact with the physical environment, SA4P can significantly reduce the risks of rogue IoT devices. We demonstrate, via actual implementations, that SA4P is technically feasible and practical, thus making it a promising approach for securing certain common types of IoT devices.

4 Kimya: *Securing Event-Triggered Sensors Through Isolation and Amnesia*

4.1 Introduction

Over the past five years, we have witnessed a massive rise in the popularity of voice assistants such as Apple's Siri, Amazon's Alexa, or Google Assistant. In fact, in January 2021, more than one in three adults living in the U.S. owned a smart speaker [173]. Moreover, voice assistants are increasingly being integrated into everything from headphones to glasses and cars, and recent technologies such as ARM Helium promise to facilitate voice assistants even on the smallest devices [188].

[173] Voicebot Research (2021)

[188] Yiu (2020)

Despite their success, the "always on" nature of voice assistants has given rise to significant concerns about the privacy and societal implications of these devices [102]. In particular, a significant fraction of people, including voice assistant users, are worried about voice assistants being hacked [90, 30], or that they are covertly recording and being used to spy on them [96, 104, 108]. This fear has been further amplified by reports that voice assistant interactions—including false positives—are recorded and reviewed by humans [23]. Moreover, current hardware capabilities and attacks go well beyond speech recognition [155]. For example, smart-speaker hardware is sufficiently precise to

[102] Lynskey (2019)

[90] Kumar et al. (2018)

[30] Bräunlein and Frerichs (2019)

[96] Lau, Zimmerman, and Schaub (2018)

[104] Manikonda, Deotale, and Kambhampati (2018)

[108] Meng, Keküllüoğlu, and Vaniea (2021)

[23] BBC News (2019)

[155] Sikder et al. (2021)

perform sonar ranging and thereby detect movements as small as a heartbeat [175].

[175] *Using smart speakers to contactlessly monitor heart rhythms*, Wang et al. (2021)

Today, voice assistant vendors are addressing these privacy issues by allowing users to mute device microphones, and by adding status indicators which show when devices are actively recording audio. Although these methods can provide some level of protection, they also have significant shortcomings. Concretely, (i) muting a voice assistant removes almost all of its (useful) functionality, and (ii) both microphone muting and status indicator mechanisms are typically implemented as opaque features, making it unclear how strong the privacy guarantees they aim to provide really are. To illustrate the latter, while Amazon requires devices to "implement *(...)* a hardware-based microphone on/off control" [4], it only requires "a dedicated microphone status indicator" [4], without any further security requirements. If such a status indicator is controlled by standard software, it can potentially be disabled by a remote adversary.

[4] *Alexa Voice Service (AVS) Security Requirements*, Amazon.com (2022)

On a more general level, the proliferation of sensors in our daily lives is making it increasingly more difficult for individuals to know when they are (not) being monitored. We consider this trend to be undesirable, and believe that people deserve the assurance that they are not being unknowingly observed when they are in a private space [129].

[129] *Nineteen Eighty-Four*, Orwell (1949)

Focusing on voice assistants, this leads us to the following research question: How can we ensure that voice assistants only record when spoken to, even when they have been compromised? However, voice assistants are just one instantiation of a more general class of devices: those with always-standby, event-triggered sensors. That is, devices that contain sensors that are continuously on *standby* (i.e., processing the sensor data for event detection), but only rarely *triggered*. Although voice assistants are currently the dominant device in this class, others, such as "always-on" cameras for smartphones, are already on the horizon [138]. When considering this more general device class, our research question generalizes to: How can we ensure that always-standby devices only record when triggered, even when they have been compromised?

[138] *Snapdragon 8 Gen 1 Mobile Platform*, Qualcomm Technologies (2021)

Answering this question requires us to unify the apparently conflicting requirements of a device that is always sampling its sensor, but should only record when triggered. By itself, the SA^{4}P framework (see Chapter 3) does not provide a satisfactory solution: at any point in time, it either blocks all access to a sensor, which is not compatible with always-standby behavior; or it allows all access to a sensor, and therefore cannot prevent snooping.

Similarly, past work on sensor privacy under adversarial settings has either focused on restricting *all* access to sensors [29, 98], or on generating a notification for *any* sensor sampling activities [112, 111]. However, such approaches can also not differentiate between an always-standby device in standby mode (i.e., waiting for its trigger event), or in triggered mode (i.e., actively processing sensor data), and therefore do not address our research question.

[29] *Regulating ARM Trust-Zone Devices in Restricted Spaces*, Brasser et al. (2016)

[98] *SeCloak: ARM trustzone-based mobile peripheral control*, Lentz et al. (2018)

[112] *Viola: Trustworthy Sensor Notifications for Enhanced Privacy on Mobile Systems*, Mirzamohammadi and Sani (2018)

[111] *Ditio: Trustworthy Auditing of Sensor Activities in Mobile & IoT Devices*, Mirzamohammadi et al. (2017)

To move forward, we make the following observation: the key challenge is to guarantee that always-standby devices only locally process sensor data, and, if no event has occurred, immediately discard the sampled data. If this guarantee is met, the device cannot eavesdrop and its privacy implications are minimal. Applied to voice assistants, they must immediately discard sampled audio if no wake word is detected.

To enable vendors to guarantee this property, we propose KIMYA[1], a hardening framework that restricts direct access to sensor data, but provides an isolated, *amnestic* execution container, inside of which applications can execute event-detection routines. When using KIMYA, application code maintains access to all sensor data, but can neither store nor transmit it without generating a user-auditable *notification*.

[1] Swahilli for “silence”.

By using user-auditable notifications, KIMYA can allow application code to self declare when an event has occurred. This is necessary, as generally no ground truth data about event occurrences is available. After all, if such data were available, no event detection would have to be performed in the first place. Yet, the device user will often intuitively know when an event has occurred. Hence, KIMYA's notifications make a device accountable for its detection behavior.

When a user notices that notifications are generated when no trigger event has occurred, this strongly indicates that a device either (i) has been compromised and is being used to eavesdrop, or (ii) generates excessive false positives and is therefore not privacy-preserving. Based on this information, users can then decide to mend or, when this is not possible, to stop using the device.

KIMYA runs on commodity MCUs. It achieves isolation by partitioning memory-mapped resources into multiple regions, and restricts access to these regions based on execution *phases*. Amnesia is achieved by routinely erasing memory regions that store sensor data or derivatives thereof. We design KIMYA's erasure schedule to not affect the continuity of event-detection algorithms. KIMYA can run together with existing application code on the same MCU and does not require additional hardware.

KIMYA allows for arbitrary code to be executed inside the event-detection container and is not dependent on cryptography. This is possible because KIMYA's security properties are based on isolation and amnesia rather than software attestation. Moreover, it ensures that KIMYA does not inhibit device vendors from updating their event-detection algorithms once devices are in the field. It also ensures that KIMYA is lightweight and applicable to constrained hardware. This is important because (i) it has been shown that applications such as wake-word detection are already possible on such hardware [192], and (ii) industry is actively working to further facilitate digital signal processing (DSP) and machine learning applications on constrained devices [188].

[192] *Hello Edge: Keyword Spotting on Microcontrollers*, Zhang et al. (2018)

[188] *White Paper: Introduction to the ARM Cortex-M55 Processor*, Yiu (2020)

Further, KIMYA's lightweight design translates itself into a small TCB size when implemented. Combined with the strong properties KIMYA provides, this allows security audits to focus on a small, reusable, module with clearly defined functionality, which in turn facilitates the work of independent certification centers, such as the new Swiss National Test Institute for Cybersecurity (NTC). KIMYA can either be used as an independent system, or it can be integrated into a SA^{4}P deployment. In the later case, the SA^{4}P authorization module can function as a notification mechanism. We describe SA^{4}P integration in more detail in Section 4.8.4.

We demonstrate KIMYA's practicality by implementing it on an ultra-low-power Cortex-M33 MCU with ARM TrustZone. We design our implementation to be minimally intrusive, allowing it to coexist with existing application code on the same device. Further, we demonstrate Kimya's applicability by applying it to a wake-word detection engine running on a Cortex-M33 processor.

4.2 *Adversary Model & Security Setting*

We consider a setting in which a user has equipped a private space, such as a home or workspace, with an Internet-connected device that has an always-standby sensor. The goal of the adversary is to access data from the always-standby sensor when no trigger event has occurred, and to do so with stealth, i.e., without generating a notification and without leaving an auditable trace.

The Kimya trust model is similar to the SA4P trust model, but is reformulated to better match the Kimya application setting. Concretely, the Kimya trust model distinguishes between the platform vendor and application vendor of a device. The platform vendor constitutes the entity that produces the hardware platform and provides the Kimya firmware. The application vendor implements the device functionality. It is possible for the platform and application vendor to be the same entity, but they can also be different entities within the same company, or different entities in different companies.

Kimya requires the platform vendor to be trusted, as it provides the TCB upon which Kimya's features are based. However, the application vendor can be untrusted. As the application vendor does not contribute to the TCB, companies providing both platform and application can focus their security resources on a smaller entity and a minimal code base. Platform vendors can obtain a trusted status through reputation or auditing.

We consider an adversary with full control over the device's network connectivity who can view, inject, and drop packets. Moreover, the adversary can exploit device vulnerabilities and can execute code on the device. The adversary might have already infected the device at the time of production. However, we explicitly do not consider attacks against trusted execution environments [38] or the platform code therein [101]. We also do not consider physical attacks, as adversaries with physical access could install their own covert sensors in the private space, significantly reducing the relevance of defenses against such adversaries.

[38] *SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems*, Cerdeira et al. (2020)

[101] *On Security of TrustZone-M-Based IoT Systems*, Luo et al. (2022)

4.3 *Design*

4.3.1 *Design Goals*

Kimya's primary goal is to facilitate the privacy-preserving use of always-standby sensors. More concretely, this goal can be broken down into the following two subgoals.

***KIMYA-1**, Availability:* Sensor data must be made available to an event-detection algorithm.

***KIMYA-2**, Isolation:* It must be ensured that only sensor data related to an event can be used for purposes other than event detection. This is a high-level goal and will be refined in Section 4.3.3.

Additionally, we have the following secondary goals.

***KIMYA-3**, Lightweight:* Kimya should be lightweight and deployable on microcontrollers. Its TCB should be small.

***KIMYA-4**, Low-cost:* Kimya should not require designs to include additional hardware.

***KIMYA-5**, Non-restrictive:* Kimya should not restrict which event-detection algorithms can be used.

***KIMYA-6**, Agile:* Kimya should not prevent application vendors from pushing updates to their devices, in particular, updates to the event-detection logic.

4.3.2 *Straw-man Proposals*

Because our solution should be lightweight (**KIMYA-3**), we do not consider mechanisms that rely on dynamic code analysis. Additionally, both because of the lightweightness goal **KIMYA-3**, and to avoid the complexity of traditional attestation mechanisms, we refrain from using software attestation. Combining this restriction with the requirement for devices to be easily updatable (**KIMYA-6**), this means that we must consider all application code, including the event-detection code, to be untrusted.

Therefore, we focus on approaches whose properties are independent of the executed code, and instead are based solely on the properties of the environment in which code is executed. Concretely, we focus on approaches that directly restrict access to the always-standby sensor. Figs. 4.1 and 4.2 displays two such approaches which serve as straw men for our final design.

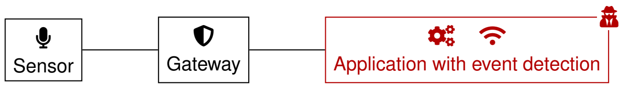


Figure 4.1: Straw-man defense using a sensor gateway. indicates an untrusted component.

The first design (Fig. 4.1) places a gateway between the always-standby sensor and the application which also contains the event-detection logic. A SA4P $\mathcal{P}$EG could be used as a gateway instantiation. This approach is most similar to previous designs for peripheral access control, e.g., SeCloak [98]. However, once the gateway provides sensor access to the application (as required by **KIMYA-1**), it can no longer control what purpose the sensor data is being used for, thus violating the isolation goal **KIMYA-2**.

[98] *SeCloak: ARM trustzone-based mobile peripheral control*, Lentz et al. (2018)

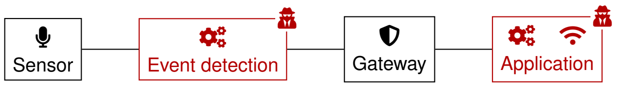


Figure 4.2: Straw-man defense using a sensor gateway and an isolated container. indicates an untrusted component.

The second design (Fig. 4.2) attempts to addresses this issue by separating event-detection functionality from the rest of the application. Event detection is then performed in an isolated environment that has direct access to the sensor. Doing so provides the event-detection code with continuous access to the sensor, while still allowing the gateway to restrict access for the other application code. Once the containerized code declares that a sensor event was detected, a notification is generated by the gateway and the application code is granted access to the event-detection container and sensor data.

Although this design represents a significant improvement over the design in Fig. 4.1, it does not provide control over the state stored in the event-detection container. This means that it cannot yet fully satisfy the isolation goal **KIMYA-2**. Concretely, the (untrusted) code running in the event-detection container could continuously eavesdrop, store captured information in the container, and exfiltrate this data to the application as soon as an event is detected.

4.3.3 *Event-Detection Timeline*

The shortcomings in the straw-man designs show that a more precise definition of the isolation goal **KIMYA-2** is needed. Specifically, it must be defined which information may be made available to the main application once an event is detected.

To this end, we introduce the notion of a sensor *interaction*. Each sensor interaction corresponds to a time window during which data relevant for the device's legitimate operation is sampled by the always-standby sensor. For example, for the voice assistant interaction "Hey Kimya, what time is it", the time window would correspond to the period during which the user is uttering this sentence. For event-triggered sensors, interactions are initiated by an event. In the case of our example, the event is the utterance of "Hey Kimya".

Ideally, KIMYA would only grant unrestricted access to data sampled during the interaction period. However, doing so is impractical. As is illustrated in Fig. 4.3, an event can generally not be detected right at the start of an interaction (e.g., the utterance "Hey Kimya" can only be detected once it has been fully articulated). Similarly, immediately after the trigger event, it is unclear how long an interaction will last.

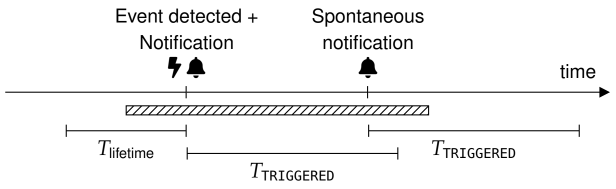


Figure 4.3: The interaction timeline. ▨ indicates an interaction.

Therefore, we introduce the durations T_{lifetime} and $T_{\texttt{TRIGGERED}}$, as illustrated in Fig. 4.3. T_{lifetime} specifies how much data from before a trigger event can be released. In the case of a voice assistant, it should be set based on the expected maximum duration of the trigger phrase, plus any required *pre-roll*[2]. We discuss this in more detail in Section 4.8.1. Similarly, $T_{\texttt{TRIGGERED}}$ defines the duration for which data can be freely accessed after an event. Because interactions might last longer than $T_{\texttt{TRIGGERED}}$, a time extension mechanism must be foreseen. This can be achieved by allowing applications to generate spontaneous notifications, each of which resets $T_{\texttt{TRIGGERED}}$. Doing so effectively visualizes sensor-access behavior with a temporal granularity of $T_{\texttt{TRIGGERED}}$. Moreover, it facilitates interaction models where user input is expected at the end of a prompt.

[2] *Pre-roll* data refers to data captured just before a trigger-event occurred. It is often used to calibrate noise levels.

Based on this model, we extend the straw-man design of Fig. 4.2 by additionally rendering the event detection container *amnestic*. That is, by ensuring that no data older than T_{lifetime} can be present in the container.

A naive approach to implement amnesia could be to periodically zero out the event-detection container's memory. However, doing so risks trigger events being segmented by a wipe. To illustrate this, consider again a wake-word detection engine that listens for the phrase "Hey Kimya". If a wipe event were to occur after a user has said "Hey", but before they said "Kimya", no trigger event would be detected. This would be a violation of the non-restrictiveness goal **KIMYA-5**.

In order to ensure data continuity while simultaneously limiting data age, KIMYA instead uses multiple buffers that are routinely wiped, and enforces an unidirectional information flow between these buffers. We present this design in more detail in the following section.

4.3.4 KIMYA *Design*

KIMYA provides two key features. First, it provides an isolated execution environment that has direct access to data from an always-standby sensor. Second, it ensures that this container is amnestic, i.e., it provides strong guarantees on the maximum age of sensor data (or information derived thereof).

In order to achieve these features, KIMYA segments its host MCU into five memory *regions*, and introduces four execution *phases*. Because on modern MCUs most peripherals are memory-mapped, these memory regions can also be considered to be *resource* regions. Concretely, the five memory regions, as depicted in Fig. 4.4, are:

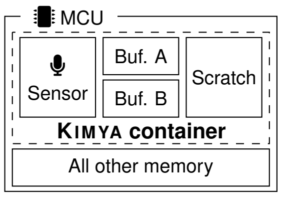


Figure 4.4: The KIMYA container memory regions.

Sensor: A region containing the always-standby sensor to which access should be restricted.

Buffer A and Buffer B: Two memory regions forming a pair of alternating buffers to store sensor data.

Scratch: A memory region that can be used to store state for the event-detection algorithm.

All other memory: All memory-mapped resources not included in the other four regions. This includes GPIO, timers, and communication peripherals.

KIMYA's four execution phases are listed below. Execution starts in the `IDLE` phase, and phase transitions are requested by the application code.

`IDLE`: As long as no trigger event has been detected, all application code that is not related to event detection is executed in the `IDLE` phase.

`ACQUIRE`: Used to acquire sensor data, to perform data pre-processing, and to store the result in the alternating buffer formed by the Buffer A and Buffer B memory regions.

`PROCESS`: After acquiring fresh sensor data, the event-detection is performed in the `PROCESS` phase. Event-detection state can be stored in the Scratch region.

`TRIGGERED`: When a trigger event has occurred, the application code is executed in the `TRIGGERED` phase (instead of in the `IDLE` phase) for a preset duration $T_{\texttt{TRIGGERED}}$. Before the `TRIGGERED` phase can be entered, a notification (see Section 4.3.5) must be generated. Regenerating this notification

while the TRIGGERED phase is active extends the duration of this phase to $T_{\texttt{TRIGGERED}}$ after the notification was generated.

	MCU memory region				
KIMYA phase	Sensor	Buf. A	Buf. B	Scratch	All other mem.
IDLE	-	-	-	-	👁 ✏
ACQUIRE (A)	👁	👁 ✏	-	-	-
ACQUIRE (B)	👁	-	👁 ✏	-	-
PROCESS	-	👁	👁	👁 ✏	-
TRIGGERED	👁 ✏	👁 ✏	👁 ✏	👁 ✏	👁 ✏

Table 4.1: Memory access rights in the four KIMYA execution phases. '👁' indicates read access, '✏' indicates write access.

Depending on the active execution phase, KIMYA restricts access to the various memory regions according to the permissions shown in Table 4.1. In the IDLE phase, no memory regions related to the always-standby sensor and event detection are accessible. During the ACQUIRE phase, sensor data can be sampled and written to one of the alternating buffers, but no other memory can be accessed. During the PROCESS phase, there is read-only access to both buffers and full access to the Scratch memory. Finally, during the TRIGGERED phase, all memory is fully accessible.

Enforcing the Table 4.1 access map has two desirable effects. First, the ACQUIRE and PROCESS phase form an isolated container in which event detection can be performed. Second, a unidirectional data path, as illustrated in Fig. 4.5 is created.

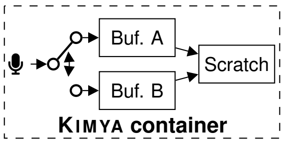


Figure 4.5: The KIMYA container data flow.

In order to guarantee amnesia, KIMYA must enforce strong limits on the maximum age of the data inside the KIMYA container, that is, in the Buffer A, Buffer B, and Scratch memory regions.[3] To this end, the following buffer management schedule is executed every $T_{\text{lifetime}}/2$ seconds: (i) zero out the Buffer A or B that is not currently accessible from the ACQUIRE phase; (ii) zero out the Scratch memory region; and (iii) alternate Buffer A and Buffer B.

[3] Data in the Sensor region is always fresh.

T_{lifetime} is a static value representing the permissible lifetime of sensor data. For most applications, this value will be on the order of seconds or less. Because Buffer A and B are not simultaneously erased, data continuity is provided.

Combining this buffer management schedule with the unidirectional data path, ensures that at any point in time, the data inside the KIMYA container cannot be older than T_{lifetime}, unless a notification is generated. Intuitively, this is the case because both the Scratch region and the alternating buffers are erased at least every T_{lifetime}, and the unidirectional data path ensures that no data can be transferred between Buffer A and Buffer B to circumvent the erasure schedule. A proof of this property is provided in Section 4.5.1.

4.3.5 *Notification Design*

KIMYA has no event-detection logic built-in. Therefore, KIMYA cannot rely on ground truth information to regulate access to the `TRIGGERED` phase. Instead, KIMYA places control over the active execution phase with the application itself. Concretely, execution starts in the `IDLE` phase, from which the application can request specific functions to be executed in the `ACQUIRE` or `PROCESS` phases. Upon returning, these functions indicate if they want to return the MCU to the `IDLE` phase, or, they can request a transition to the `TRIGGERED` phase. In the latter case KIMYA will generate a user-auditable notification before returning control back to the application code.

We do not prescribe a specific notification mechanism in this work. Instead, KIMYA provides a flexible platform upon which different notification mechanisms can be build. The design of an effective privacy notification mechanisms is orthogonal to our work, and has been studied before [145, 146].

[145] *Designing Effective Privacy Notices and Controls*, Schaub, Balebako, and Cranor (2017)

[146] *A Design Space for Effective Privacy Notices*, Schaub et al. (2015)

To illustrate the flexibility KIMYA provides, we briefly discuss three types of notification below.

LED indicators. Similar to current smart-speaker products, a LED can be used to indicate when the device is in the `TRIGGERED` phase. KIMYA then provides strong guarantees that this indicator LED cannot be circumvented. Note that when using a visual indicator, a careful design is needed to ensure its effectiveness [103, 136].

[103] *Understanding Sensor Notifications on Mobile Devices*, Ma, Mirzamohammadi, and Sani (2017)

[136] *Somebody's Watching Me?: Assessing the Effectiveness of Webcam Indicator Lights*, Portnoff et al. (2015)

Bluetooth beacons. Devices could broadcast Bluetooth beacons containing information about the sensor that is being accessed. Other devices receiving these beacons could then visualize which information is being sampled from their surroundings.

Centralized logging. KIMYA could require a central server to be contacted before granting access to sensor data. This server can log all sensor activity and make it available for later auditing. When integrating a KIMYA-enabled device into a SA^{4}P deployment, the $\mathcal{P}$EG and deployment manager can be combined with the KIMYA gateway and central server, respectively. We explore this application setting in more detail in Section 4.8.4.

In order to guarantee the availability and integrity of the notification mechanism, it should be protected. This protection can be achieved through isolation or by using cryptographic techniques, depending on the notification mechanism that is in use. For example, an LED indicator can be efficiently protected by isolating the control over the GPIO pin to which it is connected. Conversely, a centralized logging scheme is best protected by establishing a cryptographic channel between the KIMYA gateway and logging server.

When notifications are generated in a machine-readable format (e.g., Bluetooth beacons or logs on a central server), a *privacy assistant* [45, 94] can be used to aid the user with the auditing of notifications.

[45] *Personalized Privacy Assistants for the Internet of Things: Providing Users with Notice and Choice*, Das et al. (2018)

[94] *A Privacy Awareness System for Ubiquitous Computing Environments*, Langheinrich (2002)

4.4 Implementation on Cortex-M

We leverage TrustZone to implement KIMYA on a Cortex-M33 MCU. Specifically, we prototype our implementation on an STM NUCLEO-L552ZE-Q development board with an ultra-low-power STM32L552ZE MCU running at 110 MHz with MPU and a floating-point unit (FPU) [163].

[163] *Ultra-low-power ARM Cortex-M33 32-bit MCU with TrustZone*, STMicroelectronics (2020)

We implement KIMYA as a gateway running in the secure world. All application code (including event detection) runs in the non-secure world and interacts with the secure-world gateway to request KIMYA phase transitions. We illustrate the basic KIMYA control flow in Fig. 4.6.

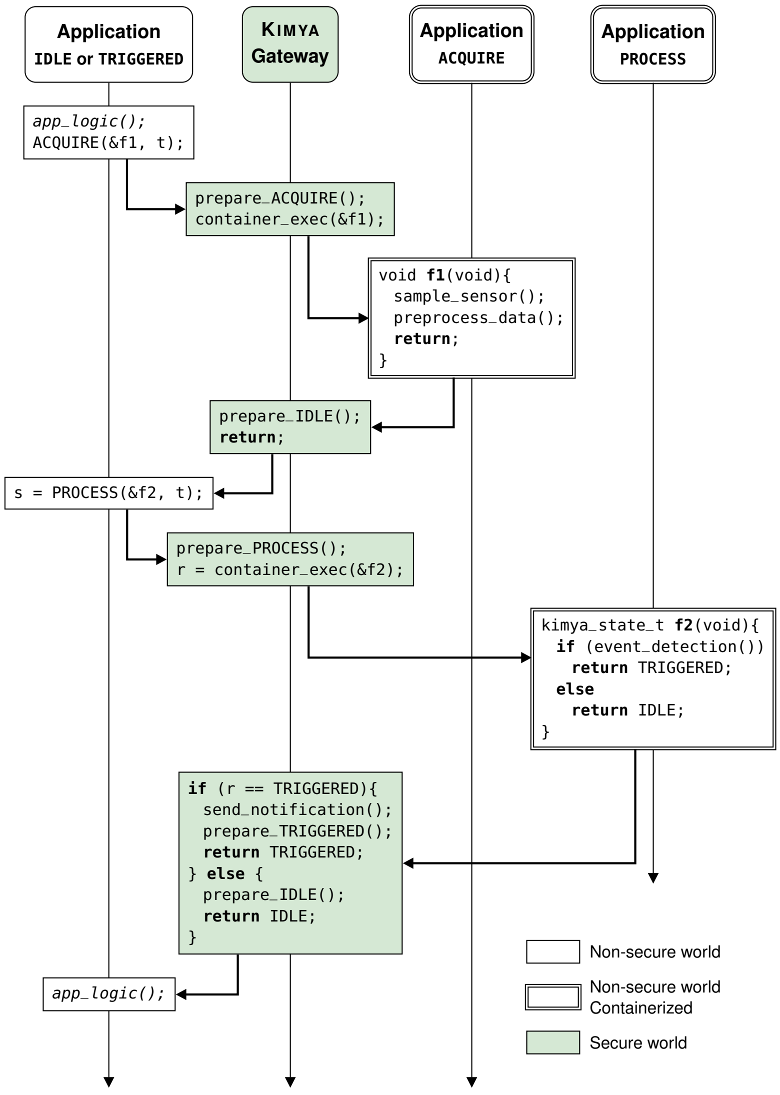


Figure 4.6: Simplified Kimya execution flow. The parameter `t` is introduced in Section 4.4.4. The notification process is executed in the secure world.

In order to transition to the `ACQUIRE` and `PROCESS` phases, the application makes a call to the gateway, passing a function pointer. The gateway function then transitions the MCU to the desired phase and executes the specified function in the non-secure world. When a function call to the `ACQUIRE` phase returns, the gateway transitions the MCU back to the `IDLE` phase and returns control to the non-secure world. Functions executing in the `PROCESS` phase can specify if the MCU should be transitioned to the `IDLE` or `TRIGGERED` phase before control is returned to the application. In the latter case the gateway will generate a notification (by calling `send_notification()`, Fig. 4.6) before executing the phase transition. In either case the non-secure world is informed about the currently active phase when control is returned to it.

4.4.1 *Enforcing the* Kimya *Access Map*

The core of our Kimya implementation is the enforcement of the access map in Table 4.1. We leverage a combination of the MPU, the TrustZone configuration of peripherals, and the TrustZone security gates (see Section 2.2) for this purpose. Concretely, during the `IDLE` phase, all Kimya container-related memory regions are marked as secure in the TrustZone configuration, preventing the non-secure application from accessing them. Therefore, full MPU control can be granted to the application, ensuring compatibility with existing OSs.

When the application makes a call to the `ACQUIRE` or `PROCESS` phases, the Kimya gateway calls `prepare_ACQUIRE()` or `prepare_PROCESS()` (Fig. 4.6) to take control of the MPU for the duration of that call. The required container resources are temporarily marked as non-secure, making them accessible to the application code. The MPU is then configured according to Table 4.1 to enforce the necessary access restrictions.

Before control is returned to the main application code, the gateway either transitions the MCU back to the `IDLE` configuration described above (by calling `prepare_IDLE()`, Fig. 4.6), or to the `TRIGGERED` configuration. In the latter case, the `prepare_TRIGGERED()` call marks all container-related memory regions as non-secure, and grants MPU control to the application. The application now has direct access to all resources needed to process the trigger event.

4.4.2 *Enforcing Amnesia*

In order to guarantee the amnestic property of the KIMYA container, the Buffers A/B and Scratch memory must be periodically erased and the one-way dataflow illustrated in Fig. 4.5 guaranteed. While the former could be achieved using a secure-world, timer-driven, interrupt, our implementation instead checks if buffer maintenance is needed each time before entering the `ACQUIRE`, `PROCESS`, or `TRIGGERED` phase. This ensures that all data older than T_{lifetime} is erased before it can be accessed, while avoiding long interrupt routines.

We further extend the KIMYA API for the non-secure world with a `maintain_buffers(t)` call. This call prepones buffer maintenance by up to `t` time units. This allows the non-secure application to schedule buffer maintenance ahead of time, thereby eliminating the need for buffer maintenance to complete before the KIMYA container can be accessed and thereby reducing timing jitter.

4.4.3 *Non-Secure Calls*

Although we can rely on standard toolchain behavior to secure calls to the secure gateway (i.e., the application's `ACQUIRE()` and `PROCESS()` calls in Fig. 4.6), this is not the case for the gateway's calls into the non-secure world (i.e., the `container_exec()` calls in Fig. 4.6). The reason for this is twofold. First, upon entering the `ACQUIRE` or `PROCESS` phases, the MPU will be in a highly restrictive state. Because the main application's stack will not be accessible, a new stack must be set up for the containerized function. Second, the official toolchain requirements for TrustZone on Cortex-M [16] are designed to provide isolation between the secure and non-secure worlds, but do not isolate non-secure function calls made by the secure world from the main non-secure application thread. Concretely, standard toolchain behavior does not clear all non-secure world registers before and after non-

[16] *ARM v8-M Security Extensions: Requirements on Development Tools*, ARM Ltd. (2019)

secure function calls, and would thus allow for communication from the KIMYA container to the main application.

To address these issues, we implement `container_exec()` as an assembly function that performs the following tasks.

1. Push all general-purpose, special, and floating-point processor registers to the secure stack. In the case of banked registers, the non-secure register is pushed.
2. Clear all registers saved on the stack.
3. Move the non-secure stack pointer to a memory region that will be writable in the container. When transitioning to the `ACQUIRE` phase, the stack pointer is moved to the top of the currently active Buffer A/B. For the `PROCESS` phase the top of the Scratch region is used.
4. Branch and link to the containerized function in the non-secure world.
5. Once the containerized function has returned, erase and restore all saved registers. This includes the non-secure stack pointer.

4.4.4 *Ensuring Container Isolation*

Beyond the core aspects described above, a number of other measures must be taken to ensure no data can be leaked from a KIMYA container. We discuss these practical measures here and present a more theoretical discussion in Section 4.5.2.

Container execution time Using a timer, real-time clock (RTC), or similar peripheral, it is possible for the main application to measure the execution time of a KIMYA container. When no special care is taken, the event-detection code running in the KIMYA container could modulate its execution time to establish a uni-directional communication channel out of the container. This channel could have a capacity of up to $\log_2(f_{cpu})$ bps, where f_{cpu} is the core frequency in Hz.

In order to prevent this source of information leakage, we require the application to specify a desired execution time for each call to the ACQUIRE or PROCESS phases. The called function *must* return before the specified duration has passed. The gateway will then wait for the remainder of that duration before control is returned to the application. A read-only timer is made available in the container to allow the event-detection code to know how much execution time is left. This timer can also be used as a relative time base for event-detection tasks.

TRIGGERED execution time When an event has been detected and a notification generated, the MCU is transitioned to the TRIGGERED phase. By default the MCU can stay in this phase for up to $T_{\texttt{TRIGGERED}}$ seconds. $T_{\texttt{TRIGGERED}}$ is an implementation defined value. Before this duration has passed, the application *must* either (i) yield back to the IDLE phase, or (ii) request another notification to be generated, extending the yield deadline to $T_{\texttt{TRIGGERED}}$ seconds beyond that request. This deadline is enforced using an interrupt triggered by a secure-world timer. Setting the PRIS bit in the Application Interrupt and Reset Control Register (AIRCR) ensures that the non-secure application cannot mask this interrupt [14].

[14] *ARM Cortex-M33 Devices Generic User Guide*, ARM Ltd. (2020)

Caches The STM32L552ZE MCU used for our implementation has a built-in instruction cache [163]. In order to prevent cache-timing attacks, the KIMYA gateway clears this cache whenever leaving the ACQUIRE or PROCESS phases. There are no other caches present on the STM32L552ZE.

[163] *Ultra-low-power ARM Cortex-M33 32-bit MCU with TrustZone*, STMicroelectronics (2020)

DMA As shown in Fig. 4.7, the MPU is part of the Cortex-M33 core, and therefore does not affect the operation of the DMA controller. Because KIMYA container resources are marked as non-secure during the ACQUIRE and PROCESS phases, this means that the application could preprogram the DMA controller to steal sensitive information while the MCU is in one of those phases. We propose two mechanisms to prevent this attack. First, the gateway can disable the DMA controller before moving container resources to the non-secure world. Second, the DMA controller can be moved to the secure world, removing the non-secure application's ability to program it directly. A thin secure-world DMA configuration shim can then verify that no DMA operations affect the KIMYA container memory regions. Our implementation uses the former strategy.

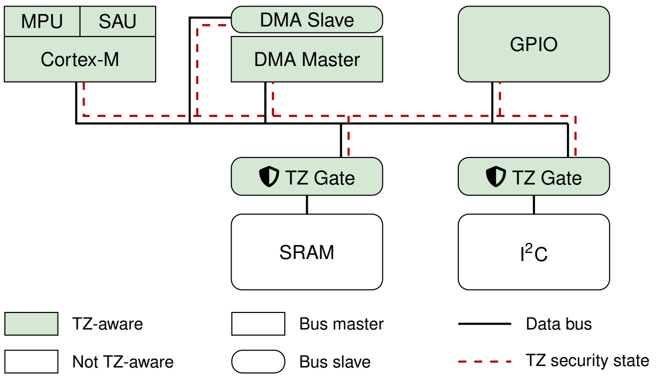


Figure 4.7: A high-level architecture overview of a TrustZone-enabled MCU.

Peripheral use In some cases, peripherals must be accessible in the KIMYA container. For example, our wake-word detection prototype (see Section 4.6) requires the cyclic redundancy check (CRC) peripheral to be available for intellectual property management.[4] In such cases, it must be ensured that (i) container isolation cannot be violated by using peripherals as communication channels with the main application, and (ii) that container amnesia cannot be violated by storing information in peripheral registers during memory wipe events. This can be achieved by setting all writable peripheral memory locations to a well-known value after the function call from the `ACQUIRE` or `PROCESS` phase returns. In the case of our prototype, this is done by resetting the CRC peripheral.

[4] The proprietary STM X-CUBE-AI neural network library uses the CRC peripheral to verify that it is running on STM hardware.

Handling policy violations Whenever event-detection or main application code violates a KIMYA policy (e.g., a container executing longer than was requested by the main application), a notification is generated. This ensures that misbehaving applications are detected and can be mended.

4.5 *Security Analysis*

KIMYA's security guarantees are derived from the two main properties it provides: amnesia and isolation. We provide a theoretical proof of KIMYA's amnesia property in Section 4.5.1. Isolation is discussed in Section 4.5.2.

4.5.1 *Amnesia*

In Section 4.3.4 we claimed that no data in the KIMYA container can be older than T_{lifetime}. We will now prove this property.

Proof model In order to facilitate the proof, we introduce the variables t_{sensor}, $t_{\text{buffer A}}$, $t_{\text{buffer B}}$, t_{scratch}, which keep track of the genesis time of the oldest data that can be present in each memory region inside the KIMYA container. Because only the latest sample can be read of the sensor, we assume $t_{\text{sensor}} = t$ at all times t. At $t = 0$, all buffers are zeroed out, so we initialize the other variables as $t_{\text{buffer A}} = t_{\text{buffer B}} = t_{\text{scratch}} = 0$.

Whenever
$$t = n \cdot \frac{T_{\text{lifetime}}}{2}, n \in \mathbb{N},$$
buffer maintenance is performed. This is modeled using the following sequential operations:
$$t'_{\text{buffer B}} = t_{\text{buffer A}},$$
and
$$t'_{\text{buffer A}} = t_{\text{scratch}} = t,$$
where we model the alternating buffer using a renaming operation to simplify notation. Additionally, at any point in time, the following two operations may be performed arbitrarily often:
$$t'_{\text{buffer A}} = \min(t_{\text{buffer A}}, t_{\text{sensor}}), \tag{α}$$
and
$$t'_{\text{scratch}} = \min(t_{\text{buffer A}}, t_{\text{buffer B}}, t_{\text{scratch}}). \tag{β}$$
α and β model the `ACQUIRE` and `PROCESS` phase, respectively.

Proof We now prove that at any time t, it holds that

$$t_{\text{min}} = \min(t_{\text{sensor}}, t_{\text{buffer A}}, t_{\text{buffer B}}, t_{\text{scratch}}) \geq t - T_{\text{lifetime}} \,.$$

To this end, we first proof that neither operation α nor β can change t_{min}. This holds because both operations assign the minimum value from a subset of variables considered for t_{min} to a variable from that same set, thus not changing t_{min}.

Next, we show that neither α nor β can change the variables $t_{\text{buffer A}}$ and $t_{\text{buffer B}}$. For β this holds trivially, as it does not write to either of those variables. α writes only to $t_{\text{buffer A}}$, so cannot affect $t_{\text{buffer B}}$. Moreover, because time is monotonically increasing, we know that $t_{\text{buffer A}} \leq t$, so

$$\begin{aligned} t'_{\text{buffer A}} &= \min(t_{\text{buffer A}}, t_{\text{sensor}}) \\ &= \min(t_{\text{buffer A}}, t) \\ &= t_{\text{buffer A}} \,. \end{aligned}$$

We know that

$$t_{\text{min}} \geq t - \frac{T_{\text{lifetime}}}{2} \text{ at } t = 0$$

because all variables are initialized at 0. Now assume that

$$t_{\text{min}} \geq t - \frac{T_{\text{lifetime}}}{2} \text{ at } t = k \cdot \frac{T_{\text{lifetime}}}{2}, k \in \mathbb{N} \,,$$

then at $t = (k+1) \cdot \frac{T_{\text{lifetime}}}{2}$, after performing buffer maintenance it holds that

$$t_{\text{sensor}} = t \text{ (by assumption)} \,,$$

$$t_{\text{buffer A}} = t_{\text{scratch}} = t \text{ (because of the wipe event)} \,,$$

and

$$t_{\text{buffer B}} = t_{\text{buffer A}} \Big|_{t=k\cdot T_{\text{lifetime}}/2} = k \cdot \frac{T_{\text{lifetime}}}{2}$$

because $t_{\text{buffer A}}$ was set at $t = k \cdot \frac{t_{\text{lifetime}}}{2}$ and cannot have changed since then (see above). Thus, it holds that

$$t_{\text{min}} \geq t - \frac{T_{\text{lifetime}}}{2} \text{ at } t = (k+1) \cdot \frac{T_{\text{lifetime}}}{2}, k \in \mathbb{N} \,.$$

Therefore, by induction it holds that

$$\forall n \in \mathbb{N} : t_{\text{min}} \geq t - \frac{T_{\text{lifetime}}}{2} \text{ at } t = n \cdot \frac{T_{\text{lifetime}}}{2} .$$

Finally, because neither operations α nor β can change t_{min}, and because buffer maintenance is performed at least every $\frac{T_{\text{lifetime}}}{2}$, it must hold that $t_{\text{min}} \geq t - T_{\text{lifetime}}$ at any time t. $\square$

4.5.2 *Isolation*

In order for KIMYA's isolation (and by extension amnesia) property to hold, it must be ensured that no covert channels exist. This section presents an overview of the considerations made during the design of KIMYA. As motivated in Section 4.2, we do not consider adversaries with physical access.

4.5.2.1 *Storage Channels*

Under storage channels we consider channels that transmit data by explicitly writing it to a storage location from which it can later be read back. Two types of storage are available on our target MCU: memory-mapped and register storage.

Memory-mapped storage Memory-mapped storage comprises all storage locations that have a memory address and are accessed over the data bus using load or store instructions. As described in Section 4.4, our KIMYA implementation dynamically manages the MPU, TrustZone configuration, and the DMA controllers, and combines this with (re)setting memory location to well-known values to ensure that no communication is possible using these resources.

Processor registers Some storage locations on the Cortex-M33 core are not memory-mapped, but can be directly accessed using dedicated instructions. To prevent covert channels that leverage these registers, we analyze the Cortex-M33 ISA to identify all writable registers. As discussed in Section 4.4.3, we constructed the `container_exec()` function to ensure that all these registers are set to a well-known value after the containerized call returns.

4.5.2.2 *Other Channels*

Beside channels that directly write data to storage locations, other, indirect, channels must be considered as well. We inspect the architecture [17] and reference manual [162] of our target MCU to identify potential covert channel and present tailored defences below.

[17] *Armv8-M Architecture Reference Manual*, ARM Ltd. (2021)

[162] *Reference manual: STM32L552xx and STM32L562xx advanced ARM-based 32-bit MCUs*, STMicroelectronics (2020)

Timing channels A containerized call could leak information by modulating its execution time. As discussed in Section 4.4.4, the application must therefore specify an exact execution time for each containerized function call.

Caches Our target MCU has an instruction cache. Timing analysis on this cache could be used to establish a covert channel. As specified in Section 4.4.4 we flush the cache after every containerized call to prevent this.

Counters Our target MCU has multiple debug and performance counters, e.g., a cache-miss counters. The containerized code could attempt to actively influence these counters. We ensure that these counters are not readable by the non-secure world application.

Physical channels Channels communicating using physical properties could be established. For example, the event-detection code could attempt to influence the temperature of the MCU package, which could then be measured using the MCU's built-in temperature sensor. We did not explicitly consider such channels, but if deemed necessary, they could be avoided by restricting non-secure access to specific resources, such as, the on-board temperature sensor.
Although the relative simplicity of our target MCU compared to high-end processors facilitates a covert-channel analysis, we consider strong claims about the total covert-channel capacity to be beyond the scope of this work. However, we do note that Kimya's amnesia property strictly limits the attacker's window to exfiltrate information from the Kimya container. This signif-

icantly reduces the utility of low-capacity covert channels, as any information not exfiltrated within at most T_{lifetime} from its genesis is lost to the attacker.

4.6 Prototype Application

In order to demonstrate KIMYA's practicality, we implement a proof-of-concept *keyword-spotting* pipeline on an ultra-low-power STM32L552ZE MCU, simulating the wake-word detection functionality of a voice assistant. We train the pipeline using recordings of the author of this thesis speaking the same words as used in the Google Speech Commands dataset [177]. The word "cat" is used as keyword. We then apply KIMYA to this detection engine.

[177] *Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition*, Warden (2018)

4.6.1 Keyword-Spotting Pipeline

A typical keyword-spotting pipeline first extracts high-level speech features from the audio signal and then feeds these features to a neural network classifier [192]. Running inferences on speech features instead of raw audio considerably reduces the input dimensions of the classifier, thereby significantly simplifying the classifier's network and training process.

[192] *Hello Edge: Keyword Spotting on Microcontrollers*, Zhang et al. (2018)

A commonly used feature set for speech processing are mel-frequency cepstral coefficients (MFCCs) making up the mel-frequency cepstrum [58]. Given the constrained nature of our target platform, we instead use the mel spectrum, a less processed version of the cepstrum. In order to obtain a mel spectrum of an audio segment, the following steps must be taken. First, the audio segment is segmented into shorter *chunks* to which a fast Fourier transform (FFT) is applied. The resulting coefficients represent the Fourier spectrum of each chunk. Plotting these spectra against time results in the spectrogram of the audio segment. This spectrogram shows how the frequency components in the segment change over time. The frequency coefficients in each chunk's Fourier spectrum are then binned using the mel scale [160], a scale based on the sensitivity of the human ear. The end result is a mel spectrogram, showing how

[58] *Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between*, Fayek (2016)

[160] *A Scale for the Measurement of the Psychological Magnitude Pitch*, Stevens, Volkmann, and Newman (1937)

the human-perceived spectrum of the audio segment changes over time.

We implement the keyword spotting pipeline shown in Fig. 4.8. The input to the pipeline is 16-bit mono audio sampled at 16 kHz from a SPH0645LM4H microphone with digital Inter-IC Sound (I2S) output. The audio is processed in chunks of 1024 samples or 64 ms. These chunks are non-overlapping and a rectangular sampling window is used. Because the spectrum of each chunk is independent, each audio chunk must be processed only once, and the result can be appended to the previous mel spectrogram from which the oldest chunk is simultaneously dropped. From each chunk, a 13-coefficient mel spectrum is extracted. 15 such spectrums form the mel spectrogram that is used as input for the classifier network.

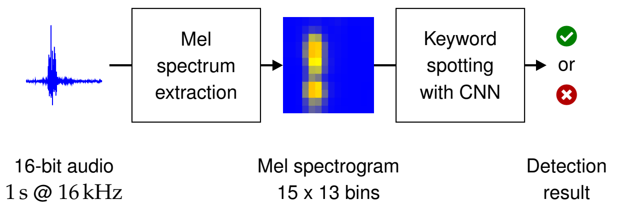

Mel spectrum extraction
Keyword spotting with CNN
or
16-bit audio 1 s @ 16 kHz
Mel spectrogram 15 x 13 bins
Detection result

Figure 4.8: The keyword-spotting pipeline.

We use the convolutional neural network (CNN) shown in Fig. 4.9 to perform the keyword spotting task on the 15x13 mel spectrograms. The network contains two convolutional layers and one dense layer, totaling $10,454$ trainable parameters.

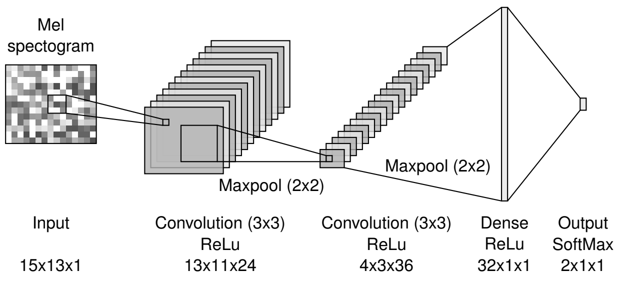


Figure 4.9: The CNN used for keyword spotting on the MCU.

The feature extraction is performed with the ARM CMSIS-DSP software library [18] using single precision floats. The neural network is executed in the STM X-CUBE-AI runtime [161],

[18] *CMSIS DSP Software Library*, ARM Ltd. (2021)

[161] *DB3788: X-CUBE-AI Databrief*, STMicroelectronics (2021)

again using single precision floats.

4.6.2 Kimya *Integration*

We implement the keyword-spotting pipeline with Kimya as shown in Fig. 4.10. Mel-spectrum coefficients are calculated in the `ACQUIRE` phase and CNN inference is performed in the `PROCESS` phase.

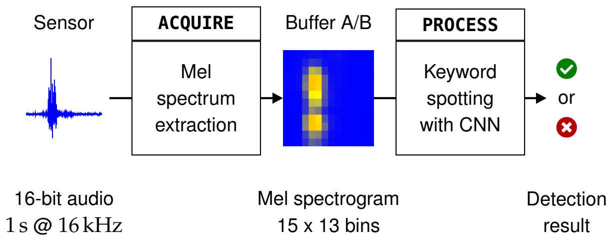

Sensor
ACQUIRE
Mel spectrum extraction
Buffer A/B
PROCESS
Keyword spotting with CNN
or
16-bit audio 1 s @ 16 kHz
Mel spectrogram 15 x 13 bins
Detection result

Figure 4.10: The keyword-spotting pipeline with Kimya integration.

In order to minimize the required number of Kimya phase changes, we use the secure world to implement a *virtual sensor*. As the microphone is sampled at 16 kHz, and the MCU's I2S peripheral only has an 8-frame FIFO buffer, the event-detection code would have to read out this buffer at least every $\frac{8}{16\,\text{kHz}} = 500\,\mu\text{s}$. As each access of the I2S peripheral require a transition to the `ACQUIRE` phase, this would be inefficient. Instead, we permanently configure the I2S peripheral as a secure-world resource, and set up a secure-world DMA stream from the I2S FIFO buffer to an alternating 1024-frame (= 64 ms) memory buffer. This memory buffer is then treated as a virtual sensor, taking the place of the I2S peripheral in the Table 4.1 access map. Additionally, a `check_for_new_data()` API call is provided for the application to check if a new buffer frame is available.

The control flow of the keyword-spotting application follows the typical Kimya flow illustrated in Fig. 4.6. If the application is in the `IDLE` phase, it checks if a new microphone frame is available. If so, it calls into the `ACQUIRE` phase, performs the feature extraction on the microphone data and stores the result in the alternating Buffer A/B. Performing this task in the `ACQUIRE` phase, ensures that it does not need to be repeated when the buffers are alternated and the Scratch memory is erased. Next, a call to the

`PROCESS` phase is made. This call copies and orders the required data from the Buffers A/B to the Scratch memory to assemble the mel spectrogram for the most recent audio data and runs the neural network inference on it. If the keyword was detected, the MCU is transitioned to the `TRIGGERED` phase, and the main application thread starts streaming microphone data to a serial port. This simulates a voice assistant streaming microphone data to a cloud service for further processing.

The non-secure application must ensure that no interrupts that violate the KIMYA access map (Table 4.1) occur while the core is in the `ACQUIRE` or `PROCESS` phase. Therefore, non-secure interrupts are masked by setting the fault mask for the duration of the container calls. Further, all functions using the Buffer A, Buffer B, and Scratch memory regions must be aware that these memory regions can be erased between calls. To this end, a canary value is placed in each buffer. When this value reads as zero, the function knows the memory was emptied and re-initializes all necessary data structures.

Configuration values We set $T_{\text{lifetime}} = 2\,\text{s}$ and $T_{\texttt{TRIGGERED}} = 5\,\text{s}$. Buffer A, Buffer B, and the Scratch memory are each 16 KiB. Two proof-of-concept notification mechanism are used simultaneously: a LED that is continuously lit when the `TRIGGERED` phase is active, and a LED that flashes upon entering the `TRIGGERED` phase or extending the $T_{\texttt{TRIGGERED}}$ deadline. We protect the notification mechanisms by limiting access to the GPIO pins that controls the LEDs to the secure world.

4.7 *Evaluation*

A basic evaluation of the on-MCU pipeline shows a precision of 100 % and a recall of 89 %. This evaluation was performed using 100 utterances of the keyword, and 100 utterances of uniformly sampled other words in the dataset. Because evaluating the performance of keyword-spotting systems is a complex task with many variables, and because we consider it to be beyond the scope of our proof of concept, we did not perform a more detailed performance analysis of the pipeline. Instead, we focus on the performance differences between the pipeline running with and without Kimya.

We use the setup shown in Fig. 4.11 to evaluate our Kimya implementation. We use two identical STM NUCLEO-L552ZE-Q development boards. One board has TrustZone activated and is running the keyword-spotting pipeline inside a Kimya container. The other board functions as a reference and has TrustZone disabled. The reference is thus running the entire keyword-spotting pipeline in the non-secure world, without Kimya. Functionality that is needed for Kimya compatibility (i.e., to reinitialize data structures or to copy and reorder data from the alternating Buffer A/B) is removed.

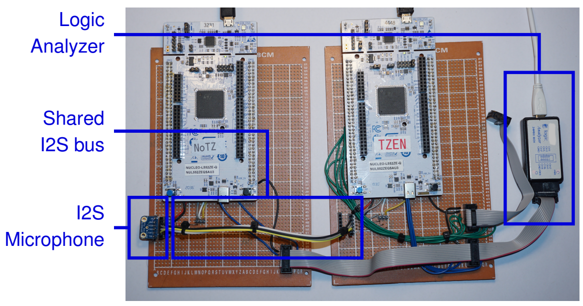


Figure 4.11: The evaluation setup. The reference is on the left and the Kimya prototype on the right.

To facilitate an accurate comparison of the two boards, they are both connected to the same I2S microphone. The microphone is configured as an audio source, the MCUs as sinks. Only the Kimya board generates a clock. This setup ensures that both boards receive identical microphone data.

Functional evaluation As a preliminary evaluation, we verified that it is not possible to access microphone data without generating a notification: doing so results in a hard fault, stalling the MCU. We also logged the status of a button to the serial port, demonstrating that Kimya does not prevent the transfer of data that did not originate in the Kimya container.

4.7.1 *Macro Benchmarks*

We evaluate Kimya using four macro benchmarks: output correlation with the reference implementation, pipeline latency, MCU duty cycle, and binary size.

4.7.1.1 *Evaluation Setup*

Each board updates the mel spectrogram and runs the neural network inference process each time a new chunk of 1024 microphone samples is available. This corresponds to one inference per 64 ms or 15.6 inferences per second. Each board exposes a `keyword_detected` signal on a GPIO pin which we sample at 16 MSamples/s using a logic analyzer. We stimulate the microphone using recordings of the keyword to generate 300 detection events per evaluation setting. This results in 600 measurable transitions of the `keyword_detected` signal. The boards also expose a signal indicating if the pipeline is running or if the MCU core is idle. We sample this signal to calculate the MCU duty cycle.

We run the macro benchmarks in three software settings. In the first setting, the application fully relies on the Kimya gateway to perform buffer management. In the second setting, the application proactively makes calls to `maintain_buffers()` (see Section 4.4.2) to ensure that no buffer maintenance must be performed when running the pipeline. In the third setting, the application additionally proactively reinitializes the neural network after the Scratch memory was erased by executing a dedicated network initialization function in the `PROCESS` phase. All code was compiled using GCC and optimized for binary size (`-Osize`).

4.7.1.2 *Results*

Output correlation When compensating for the additional delay introduced by Kimya, both `keyword_detected` signals have a correlation coefficient of 1 in all settings. That is, they are identical. We confirm this in an additional experiment in which we allow both keyword-spotting pipelines to print their prediction and confidence score to a serial port. These values are also identical. This is expected and confirms that KIMYA does not introduce data loss and does not affect computational results.

Latency Because voice commands do not have well-defined boundaries, the absolute latency of a keyword-spotting pipeline is ambiguous. Instead, we measure the relative latency of the KIMYA-enabled pipeline compared to the reference implementation. The results are shown in Fig. 4.12. We see that in the basic setting, a median 1.43 ms of additional delay is incurred because of the KIMYA containerization. In about 5 % of cases an additional 0.5 ms of delay is incurred on top of that. This corresponds to the pipeline runs where the gateway performs buffer maintenance before either call to the KIMYA container. In the setting where the application performs buffer maintenance proactively, this tail is not present. Finally, we see that when the application proactively reinitializes the neural network, the median delay is reduced to 1.19 ms. This improvement can be seen uniformly across all detection event runs, because when the application specifies a duration for a KIMYA container call, it must always assume a worst-case execution time. Thus, if the state of the neural network is unknown, the application must budget time for reinitialization during every `PROCESS` call.

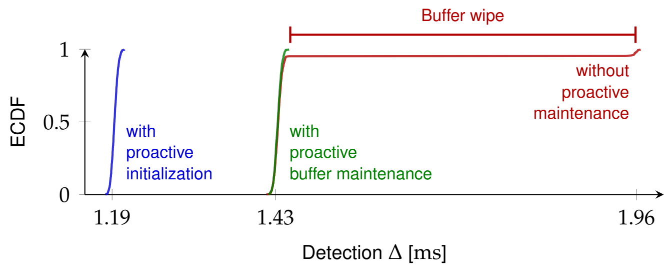


Figure 4.12: Detection delay of the KIMYA-enabled board compared to the reference implementation.

MCU duty cycle The reference implementation runs at a duty cycle of 24.5 %. The KIMYA implementation without proactive buffer management has a duty cycle of 26.8 %. Proactively calling `maintain_buffers()` did not meaningfully change the duty cycle. However, when combined with proactive reinitializations, the duty cycle was slightly reduced to 26.5 %. This improvement stems from the fact that in the latter case, time to initialize the neural network must only be budgeted in the dedicated reinitialization calls, and not in all calls to the PROCESS phase.

Binary size Table 4.2 shows the binary sizes of the Kimya and reference implementations. There were no meaningful differences between the three Kimya settings. We see that using Kimya does not measurably increase the binary of the non-secure application. In fact, the reference binary is larger than the Kimya-enabled application binary. We attribute the difference in the ROM sections to variations in compiler optimizations. The reference RAM region is larger because it includes statically allocated variables that are dynamically allocated in the Buffer A/B or Scratch regions in the Kimya implementation. The small secure-world binary size shows that the Kimya gateway has a small binary footprint.

	Secure (Gw.)		**Non-secure** (App.)	
	ROM	RAM	ROM	RAM
With Kimya	17.01	2.19	156.46	1.95
Reference	–	–	162.61	28.53

Table 4.2: Sizes of ROM and RAM sections of the benchmark binaries in KiB.

4.7.2 *Micro Benchmarks*

To better understand the origin of the additional delay observed in the macro benchmarks, we perform micro benchmarks to measure the cost of individual Kimya operations. These benchmarks use the proactive reinitialization setting.

4.7.2.1 *Container Operations*

Container entry and exit We instrument the code on the Kimya-enabled board to write well-known values to a GPIO port at key points in the execution flow. This creates an 8-bit parallel signal that can be used to profile both the secure and non-secure worlds.

As shown in Fig. 4.13, entering the KIMYA container takes between 11 and 14 µs. Leaving the container to the `IDLE` phase takes around 5 µs. Due to the additional overhead required (i.e., buffer management, setting up the $T_{\texttt{TRIGGERED}}$ timer), leaving from `PROCESS` to the `TRIGGERED` phase takes around 8 µs. These times exclude any time spent waiting to reach the specified container execution time (see Section 4.4.4).

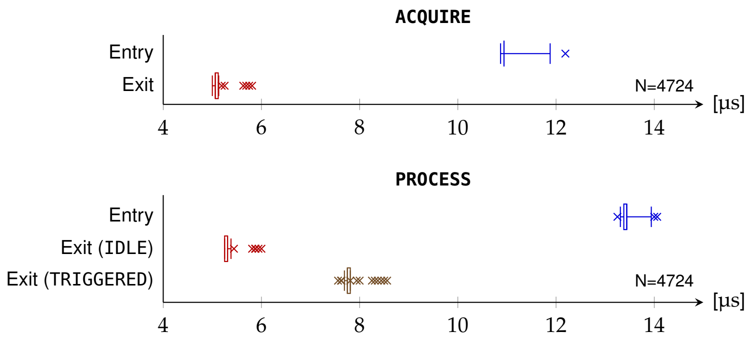


Figure 4.13: Boxplot showing the overhead of entering and exiting the KIMYA container. Exit timing excludes time spent waiting to reach the specified container execution time. Whiskers drawn at percentiles 1 and 99.

Figure 4.14 decomposes the operations shown in Fig. 4.13 into three suboperations: (i) switching from the non-secure world to the secure gateway, (ii) executing the gateway logic, and (iii) returning from the gateway to the non-secure world. We see a base cost of 1 to 2 µs to switch between the secure and non-secure worlds. KIMYA-specific logic (i.e., MPU configuration, buffer checks, timer management) adds around 10 µs per container call. Results for the `PROCESS` phase are similar and therefore not shown.

Buffer management The timing values shown in Figs. 4.13 and 4.14 include buffer management logic, but because the application performs proactive buffer maintenance, it does not include buffer erasure overhead. We separately measure buffer erasure to take 261 µs per 16 KiB memory region, resulting in an average 0.06 % core load for buffer erasure.

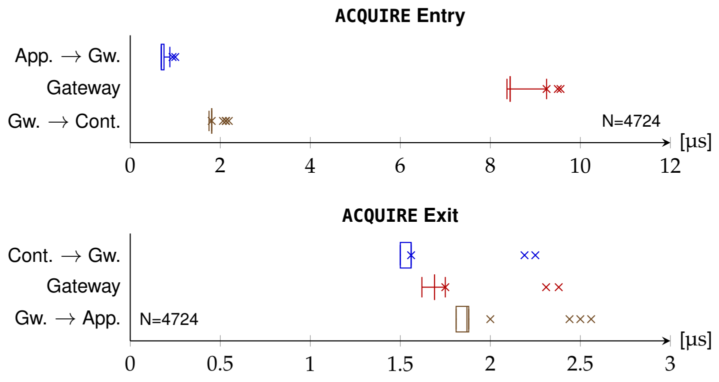


Figure 4.14: Boxplot showing KIMYA gateway overhead. Broken down into time required to enter the secure-world gateway, time spent in the gateway, and time required to return to the non-secure world. Gateway time excludes time spent waiting to reach the specified container execution time. Whiskers drawn at percentiles 1 and 99.

4.7.2.2 *Comparison to Macro Benchmarks*

Summing up the median container entry and exit times with a median 33 µs spent waiting for the specified container execution time to be reached (not shown in Fig. 4.13), results in an overhead of 70 µs per pipeline run. This number is significantly lower than the 1.19 ms measured during the macro benchmarking. To understand the origin of this discrepancy, we perform further benchmarking on the application and container code. We observe that the inference call to the (precompiled) neural network library takes 15.50 ms in the KIMYA container instead of 14.46 ms in the reference implementation. This difference of 1.04 ms makes up the majority of the 1.12 ms of unapportioned overhead. We attribute this performance gap to the different memory access pattern in the KIMYA implementation, leading to more bus contention.

4.8 *Discussion*

4.8.1 *Limitations*

Data lifetime Although KIMYA provides strong guarantees that no sensor data can be stored beyond T_{lifetime}, it requires T_{lifetime} to be configured at twice the actual useful data lifetime. For example, our prototype implementation uses $T_{\text{lifetime}} = 2\,\text{s}$, although only 1 s of data is fed into the CNN.

This is a direct consequence of using two buffers to form the alternating buffer A/B. Extending the alternating buffer to a ring buffer composed of n buffer elements, would allow T_{lifetime} to be reduced to $1 + \frac{1}{n-1}$ times the true data lifetime. Limiting factors are (i) the size of the chunks in which data is preprocessed, and (ii) the requirement to wipe the entire scratch region each time an element of the ring buffer is wiped.

Imperfect auditing KIMYA relies on device users to perform notification auditing. Past work has shown that the design of notification mechanisms is critical to ensure the audit quality [136]. However, even with an optimal notification mechanism it is possible that a user would miss some false-trigger events. When designing a notification mechanism, it is important to ensure that systematic misbehavior will eventually be noticed by the user. If this is not the case, an adversary could simply keep the KIMYA device in the `TRIGGERED` phase at all times. Care must also be taken to ensure that other factors (e.g., the location of the device) do not hinder the notification mechanism (e.g., placing a device with a light-based notification mechanism in a closed closet).

[136] *Somebody's Watching Me?: Assessing the Effectiveness of Webcam Indicator Lights*, Portnoff et al. (2015)

Instead of generating a false-trigger event, an adversary could also artificially extend the the time during which the device stays in the `TRIGGERED` phase after detecting a true-trigger event. If kept within limits, we expect users to be more likely to attribute such behavior to non-malicious technical limitations. As an example, we consider an adversary that extends each trigger by 10 s. Assuming 18 smart-speaker interactions per day [25], this would allow an adversary to maliciously capture up to 3 min of superfluous audio a day without raising suspicion.

[25] *Understanding the Long-Term Use of Smart Speaker Assistants*, Bentley et al. (2018)

Alternative event detection KIMYA does not place restrictions on which type of events the application code can detect. Therefore, the KIMYA mechanism by itself does not provide any privacy guarantees. Instead, the core KIMYA mechanism needs to be used together with suitable notification and auditing mechanisms.

For example, consider the case where an adversary writes code that detects specific information (e.g., by triggering on the keyword "password"). The attacker could then potentially export this data by generating only a small amount of low-frequency false-positive notifications. Depending on the notification and auditing mechanisms used, these notifications may go unnoticed.

The use of software attestation mechanisms to limit which code can be executing inside the Kimya container could mitigate this attack, although at the cost of reduced system agility. Alternatively, the quality of the notifications and notification auditing could be improved, for example, through the use of privacy assistants [45, 94].

[45] *Personalized Privacy Assistants for the Internet of Things: Providing Users with Notice and Choice*, Das et al. (2018)

[94] *A Privacy Awareness System for Ubiquitous Computing Environments*, Langheinrich (2002)

Multi-stage pipelines For efficiency and accuracy reasons, event-detection pipelines often use multiple stages with decreasing false-positive rates and lazy evaluation [157, 91]. In such cases, our Kimya implementation requires container execution time for all pipeline stages to be budgeted during each call. Modifications that allow the container execution time to be dependent on the result of the pipeline stages could be made, but would create a (narrow) information channel out of the container. Logging when the container execution time was extended could deter adversaries from abusing this channel. Alternatively, this channel can be eliminated by decoupling the Kimya execution time from that of the main application, for example, by executing Kimya on a dedicated core.

[157] *Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant*, Siri Team (2017)

[91] *Building a Robust Word-Level Wakeword Verification Network*, Kumar et al. (2020)

Some vendors offer cloud-based wake word verification services, intended to be used as the last stage of a keyword-spotting pipeline [83]. Kimya requires a notification to be generated each time data is sent to such a service. If the on-device pipeline stages have a high false-positive rate, this can dilute the value of Kimya notifications. However, there is an ongoing trend to move voice assistant functionality from the cloud to end-user devices [10, 31]. Therefore, we anticipate that cloud-based wake-word detection will disappear over time.

[83] *Cloud-Based Wake Word Verification Improves "Alexa" Wake Word Accuracy on Your AVS Products*, Karczewski (2017)

[10] *Apple advances its privacy leadership with iOS 15, iPadOS 15, macOS Monterey, and watchOS 8*, Apple (2021)

[31] *Bringing you the next-generation Google Assistant*, Bronstein (2019)

Reduced scheduling flexibility Calls to the `ACQUIRE` and `PROCESS` phases are fixed-duration and non-preemtable. This results in a reduced scheduling flexibility, and in certain cases, might lead to a performance degradation. However, as embedded systems are designed assuming worst-case execution times, a Kimya-enabled system should be able to meet the same deadlines as an equivalent non-KIMYA system.

TCB bloating In our KIMYA implementation, all code running in the secure world has the same security level, and is thus able to affect KIMYA's properties. Therefore, all secure world code is part of KIMYA's TCB. If additional functionalities are implemented using TrustZone, this has the potential to bloat the TCB size. A secure-world OS could be used to limit the access permissions of secure-world code. TCB bloat affects all platforms of which the hardware provides a single-world secure environment.

4.8.2 Multiple Sensors

This work focuses on regulating access to a single event-triggered sensor. However, many devices have multiple sensors. In such cases the following KIMYA deployment models are possible:

Independent The access permissions for each sensor are considered individually. Multiple sensors can be protected by creating multiple, independent, KIMYA containers.

Cross-sensor The permissions for multiple sensors can be linked together, enabling opportunities for cross-sensor activations. Consider, for example, a smart display with both a camera and a microphone. In this setting, KIMYA could be configured to link access permissions for the camera to events detected on the microphone. This way, KIMYA can ensure that camera data cannot be accessed unless the user speaks an *activation phrase*, e.g., "Hey Kimya, turn on the camera".

Virtual sensors The trigger output of a KIMYA enabled sensor can be used as a virtual sensor input for sensor access management or attack detection systems (e.g., 6thSense [154]).

[154] *A Context-Aware Framework for Detecting Sensor-Based Threats on Smart Devices*, Sikder, Aksu, and Uluagac (2020)

4.8.3 Deployability

MCU requirements Implementing KIMYA requires a mechanism that allows protected gateway code to restrict the resource access of application code. Our implementation on ARM Cortex-M uses a combination of TrustZone and the MPU for this purpose. Many modern, embedded, architectures provide similarly suitable mechanisms. These include: (i) TrustZone on ARM Cortex-A [134] in combination with a MPU or memory management unit (MMU); (ii) Physical Memory Protection (PMP) and machine mode on RISC-V [178]; and (iii) LX secure mode (XLS) on Xtensa LX [33] in combination with an MPU or MMU. In the absence of suitable hardware security extensions, KIMYA could be implemented as an OS service, although this would result in increased TCB size and significantly reduced robustness.

[134] *Demystifying Arm TrustZone: A Comprehensive Survey*, Pinto and Santos (2019)

[178] *The RISC-V Instruction Set Manual: Volume II: Privileged Architecture*, Waterman, Asanović, and Hauser (2021)

[33] *Xtensa Instruction Set Architecture (ISA) Summary*, Cadence Design Systems (2022)

Additionally, KIMYA requires one timer, and a mechanism to ensure that control is periodically returned to the gateway. The latter can be established using interrupts generated by the timer, or through a gateway-controlled watchdog timer.

Hardware and software design For KIMYA to be effective, the hardware design of KIMYA enabled products must ensure that the KIMYA gateway can restrict the application access to the event-triggered sensor. We expect this to be the case in most existing designs. Designers must implement base KIMYA functionality for their target platform analog to KIMYA for Cortex-M as described in Section 4.4. Additionally, a covert-channel analysis must be performed and countermeasures similar to those in Section 4.4.4 must be implemented.

To dimension the size of the Buffer A/B and Scratch memory regions, one must consider the amount of intermediary state that the `ACQUIRE` process will generate, and the required amount of temporary state used for signal processing. In our demo application, Buffer A and B each require 11 KiB of RAM. Of those 11 KiB, around 800 B is used to store the mel spectrograms, the remainder is used as temporary storage for signal processing. Similarly, our demo application requires a 8 KiB Scratch region, all of which is used as temporary storage for the CNN. If original sensor data is to be stored for reprocessing after a trigger event, sufficient space must be allocated for it in the Buffer A/B regions. For our demo application, this would correspond to 32 KiB per buffer.

Finally, a notification mechanism must be designed and implemented as discussed in Section 4.3.5. If cryptographically protected notifications are used, a key-establishment mechanism must also be implemented.

In-field retrofitting If an existing hardware designs make it possible to isolate the sensor that is to be protected, KIMYA can—in principle—be retrofitted to these devices using a vendor-issued software update. However, depending on the existing product configuration (i.e., TrustZone setup, MCU fuses, and exposed interfaces), the complexity of the update process can vary greatly. Some devices could be retrofitted using an over-the-air software update, others might require physical contact, and yet others could be impossible to retrofit.

4.8.3.1 Compatibility with Popular Voice Assistants

We explore the compatibility of three popular devices with KIMYA: (i) the Amazon Echo Dot (3rd generation), (ii) the Google Nest Home Mini, and (iii) the Apple HomePod. We base this analysis on publicly available data.

Amazon Echo Dot 3rd generation This device has four microphones, connected to two ADCs. The ADCs are connected to the CPU using both an I2C bus (for configuration) and an I2S bus (for audio data) [52]. The ADCs are Texas Instruments TLV320ADC3101 chips. The CPU is a Mediatek MT8516. Given that the MT8516 supports TrustZone for Cortex-A [63], implementing KIMYA should be possible. However, special care must be taken because the ADCs feature a *miniDSP* [167] which has access to the microphone stream and has (limited) storage capabilities. Hence, the miniDSP could be abused to break KIMYA isolation. Further research to determine the maximum storage duration on the ADCs, or to determine if the memory on the ADCs can efficiently be flushed would be needed. Alternatively, it could be ensured that only signed code can be loaded onto the ADC. Given that the Echo Dot already uses trusted boot [63], and that we expect the ADC configuration to be static, this is a

[52] *Echo Dot 3rd Gen Digging Deeper*, Dorey (2019)

[63] *Amazon echo dot or the reverberating secrets of IoT devices*, Giese and Noubir (2021)

[167] *TLV320ADC3101 Low-Power Stereo ADC With Embedded miniDSP for Wireless Handsets and Portable Audio*, Texas Instruments (2015)

plausible strategy.

Google Nest Home Mini Not much information is available about this device, only that it runs on a Synaptics AS370 SoC (ARM Cortex-A52) [171]. No explicit information about TrustZone support could be found, but we find it likely for support to be available. The device has 3 on-board microphones [32], but it is unclear how the audio signal is digitized. Based on the presence of a recent Cortex-A based CPU, we expect that KIMYA support would be possible on this device. Care must be taken to ensure that the SoC and ADC architectures do not break KIMYA isolation.

[171] *Google Nest Mini (H2C)*, Various Authors (2022)

[32] *Construction photos of EUT H2C*, Bureau Veritas (2019)

Apple HomePod Apple's HomePod speaker runs on a custom Apple-designed APL1011 SoC [179]. The HomePod uses seven MEMS microphones which are digitized by a Conexant CX20810 ADC [179]. This ADC does not have an on-board DSP, and is therefore unproblematic. However, we were unable to find any documentation indicating that the APL1011 silicon features the hardware security features required to support KIMYA.

[179] *Apple HomePod Teardown and Cost Comparison*, Wegner, Yang, and Waterman (2018)

4.8.4 SA4P *Integration*

It is possible to merge the functionality of the KIMYA gateway and SA4P $\mathcal{P}$EG. Doing so results in a device that (i) can perform always-standby event monitoring, and (ii) requires permission from a deployment manager before full sensor access is granted, i.e., before the `TRIGGERED` phase can be entered.

An integrated KIMYA and SA4P execution flow is shown in Fig. 4.15. Instead of immediately returning the CPU in the `TRIGGERED` phase upon event detection, the KIMYA gateway returns a *ticket*, consisting of a SA4P access request. The application (in `IDLE`) presents this ticket to the SA4P deployment manager, and receives an access grant. This grant is forwarded to the KIMYA gateway, upon which access to the protected, always standby sensor is provided. The KIMYA gateway thus acts as the SA4P $\mathcal{P}$EG.

Care must be taken to preserve KIMYA's properties when using the merged Kimya and SA^{4}P. For example, in the system shown in Fig. 4.15, the application learns about the declaration of a trigger event before the SA^{4}P manger. If the application drops the SA^{4}P ticket instead of forwarding it, a covert channel can be established by encoding data in the presence or absence of a SA^{4}P ticket. Given the phase-change overhead of about 20 µs per `PROCESS` call,[5] the capacity of the resulting channel would have an upper bound of 50 kbps.

[5] As measured in Section 4.7.

Methods to reduce the channel capacity include:

Limiting the number of calls to `PROCESS`. For example, the keyword-spotting pipeline discussed in Section 4.6 requires only one `PROCESS` call each 64 ms. Extending the gateway to only this amount of calls would reduce the channel capacity to $1\,\text{bit/call} \cdot \frac{1}{64}\,\text{call/ms} = 15\,\text{bps}$.

Always providing a SA^{4}P *ticket.* Instead of not returning a ticket when no event was detected, the gateway could return a ticket that is authenticated using an alternative key. The application would not be able to tell the difference, and would need to send the ticket to the deployment manager and only learns if access is granted or not.

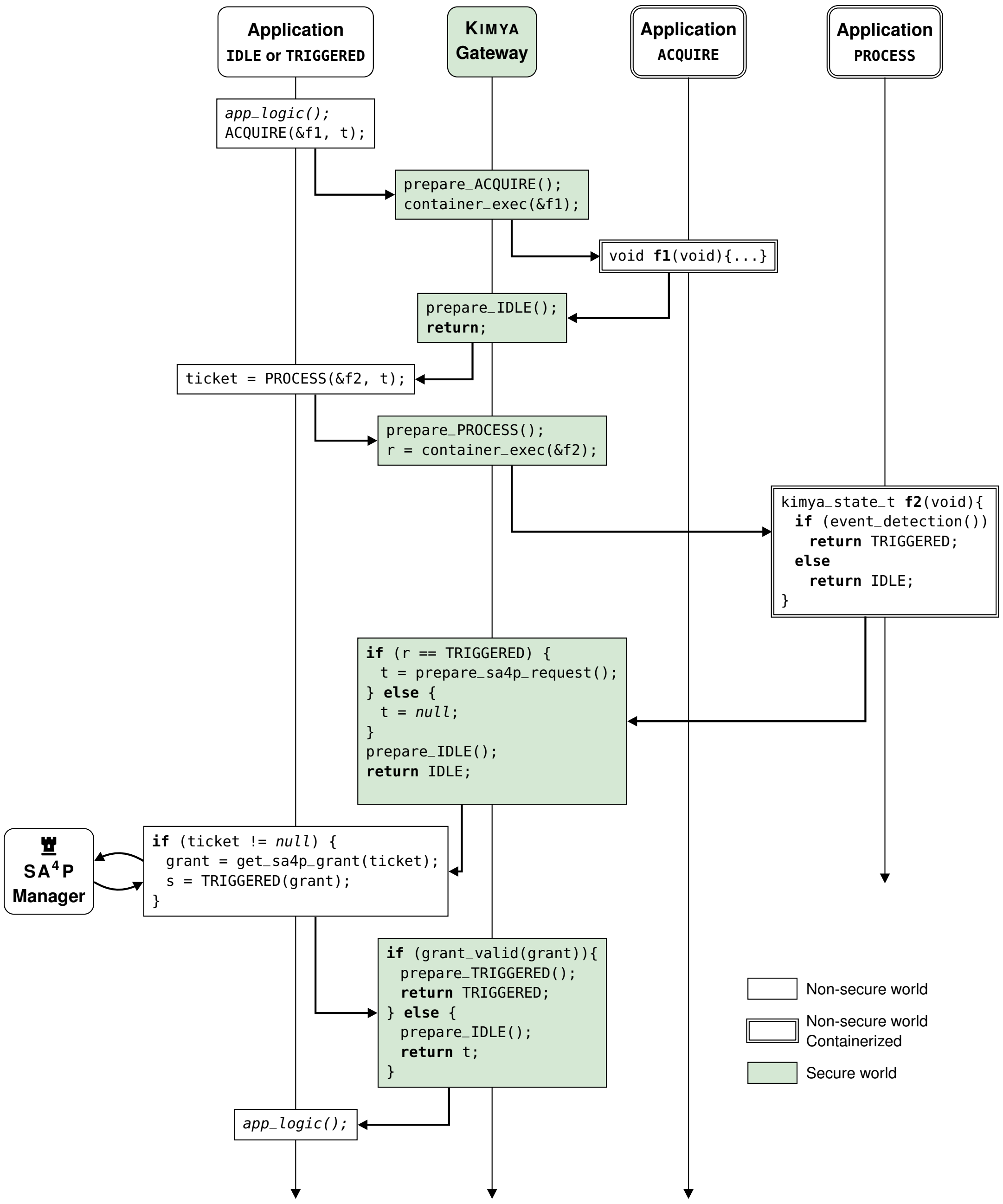
Using local notifications. In addition to the SA^{4}P request, an additional local notification could be generated[6] whenever a trigger event occurs, regardless of the deployment manager's approval for sensor access.

[6] E.g., using a led indicator.

4.9 Summary

Despite their high popularity, voice assistants continue to prompt significant privacy concerns. These concerns often focus on the always-standby nature of such assistants, and a perceived lack of transparency in how they operate.

Kimya demonstrates that it is possible to unify the functionality of always-standby sensors with strong, low-level, guarantees on privacy. Moreover, our implementation for Cortex-M demonstrates that Kimya introduces low overhead and is applicable to constrained environments.



```
app_logic();
ACQUIRE(&f1, t);
```

```
prepare_ACQUIRE();
container_exec(&f1);
```

```
void f1(void){...}
```

```
prepare_IDLE();
return;
```

```
ticket = PROCESS(&f2, t);
```

```
prepare_PROCESS();
r = container_exec(&f2);
```

```
kimya_state_t f2(void){
  if (event_detection())
    return TRIGGERED;
  else
    return IDLE;
}
```

```
if (r == TRIGGERED) {
  t = prepare_sa4p_request();
} else {
  t = null;
}
prepare_IDLE();
return IDLE;
```

```
if (ticket != null) {
  grant = get_sa4p_grant(ticket);
  s = TRIGGERED(grant);
}
```

```
if (grant_valid(grant)){
  prepare_TRIGGERED();
  return TRIGGERED;
} else {
  prepare_IDLE();
  return t;
}
```

```
app_logic();
```

Figure 4.15: Combined Kimya and SA^{4}P execution flow.

Although we have implemented Kimya for Cortex-M, its design is not platform specific, and can be implemented on other architectures (e.g., Cortex-A, RISC-V, Xtensa LX, ...) as well. Moreover, as Kimya does not require hardware modifications, it can be retrofitted to existing systems. This reduces time to market, and makes it possible to bring significant privacy enhancements to millions of devices already deployed in people's homes.

Part II

Automation Networks

5 Current Automation Networks and their Challenges

5.1 Introduction

Since the introduction of computerized control systems, automation, or operational technology (OT), networks have had a strong hierarchical structure, as commonly illustrated by the automation pyramid (see Section 5.2). For over two decades, OT network designers have successfully followed this approach.

However, in recent years, the relevance of the hierarchical model is increasingly being questioned, as the model is struggling to adapt to new realities in the automation space [67, 71, 87, 110, 113, 172], and because of the increasing convergence between IT and OT systems. In most networks, network designers already had to give up the strict air gap between IT and OT infrastructure in order to support remote management of automation systems, and new trends are further challenging the hierarchical model. Concretely, these trends can be classified as changes (i) to the network, (ii) to the the automation infrastructure, (iii) to information flows, (iv) to threat models, and (v) to operation models. For example, cloud-based predictive maintenance requires raw information to flow directly from sensors on the lowest levels of the network to the cloud, crossing all traditional network levels. This contradicts the hierarchical design principle that individual network flows should not cross more than one network level at once. We further discuss the chal-

[67] *Is the Purdue Model Still Relevant?*, Greenfield (2020)

[71] *S4x19 Panel Discussion: Is The Purdue Model Dead?*, Hegrat, Langill, and Peterson (2019)

[87] *The Purdue Reference Model outdated or up-to-date?*, Koelemij (2020)

[110] *IIoT Will Change Our View of CIM; The Purdue Model Is Becoming Dated*, Miklovic (2015)

[113] *Is the Purdue Model Relevant in a World of Industrial Internet of Things (IIoT) and Cloud Services?*, Mission Secure (2021)

[172] *Cypber-Physical Systems: Chancen und Nutzen aus Sicht der Automation*, VDI/VDE Gesellschaft Mess- und Automatisierugnstechnik (2013)

lenges created by new OT trends in Section 5.3. To address these challenges, Chapter 6 presents Tableau, an alternative way to structure industrial networks, that moves away from the hierarchical model and relies on modern base assumptions. Then, Chapter 7 presents Hopper, a nano-segmentation protocol that extends the isolation properties typically only provided at the edge of network zones uniformly across the network fabric.

5.2 Current OT Networks

Industrial automation networks are often modeled using the automation pyramid [144]. This model, shown in Fig. 5.1, is used to capture the hierarchical structure found in a broad range of industries.

[144] *The Evolution of Factory and Building Automation*, Sauter et al. (2011)

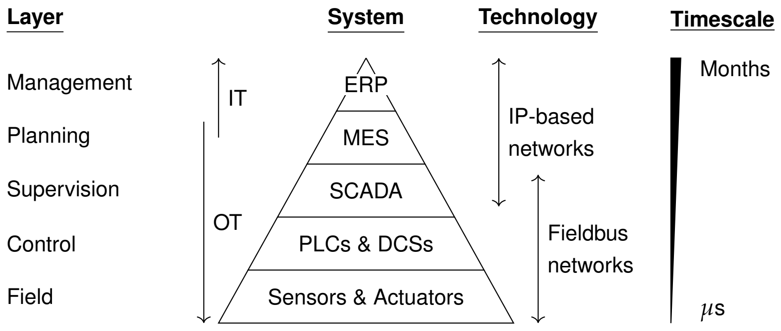


Figure 5.1: The automation pyramid.

The hierarchical structure of industrial networks has historically developed because of mainly two reasons. First, industrial processes and organizations tend to exhibit natural hierarchy. This notion was first captured in the early 1990s by Williams in the Purdue Reference Model [181], an information model for decision making and control in enterprises. Because control systems are usually placed close to the processes they control, it is logical for them to inherit the hierarchical structure of these processes. Second, using a hierarchical structure allows network designers to place security checkpoints between network levels, incrementally increasing the security level as the hierarchy descends. Because of these reasons, many OT network designs,

[181] *A Reference Model for Computer Integrated Manufacturing (CIM)*, Williams (1989)

including Cisco's Ethernet-to-the-Factory architecture [42], have adopted hierarchical structures. In fact, such architectures are widely considered to be the gold standard for designing and securing OT networks; especially in critical infrastructures such as utilities.

[42] *Ethernet-to-the-Factory 1.2 Design and Implementation Guide*, Cisco Systems and Rockwell Automation (2008)

Inspecting the automation pyramid, we see that its lowest level contains the systems that directly interact with the physical processes that are controlled, i.e., sensors and actuators. Traversing the pyramid upwards, each consecutive level adds a layer of abstraction and aggregation until finally the top level, containing the organization's management, is reached. Two common observations can be made throughout the pyramid. First, process feedback always flows upwards between the levels, while commands flow downwards; there is no direct lateral information flow. Second, the closer the distance from the process, the smaller the decision timescales become: where management decisions are taken on the scale of months or even years, field and control levels must operate at sub-second precision. Because of the high demands these short timescales place on the network, the lower levels of the automation pyramid use specialized *fieldbus* networks rather than standard Ethernet/IP. For historical reasons, many different fieldbuses are in use today, with common examples including Modbus, PROFIBUS, and EtherCAT.

Because they run on different protocol stacks, the lower levels of the automation pyramid are usually strongly decoupled from the top levels of the pyramid. This practice is referred to as the separation of information technology (IT) and operational technology (OT) systems. The IT partition of an organization consists of the systems used for business operations such as enterprise resource planning (ERP) or email. Conversely, the OT partition consists of the systems that control physical processes and infrastructure.

In practical networks, the hierarchical structure of the automation pyramid is translated to what is informally referred to as a *Purdue network*, referencing the origin on the automation pyramid. We illustrate such a Purdue network in Fig. 5.2, which shows a network as would typically be found in critical infrastructure. In a Purdue network, network zones are organized in hierarchical *Purdue* levels. Further, zones are organized in such a way that all communication between zones on the same level must traverse a zone of a higher level, and firewalls enforce security policies at each zone transition. As discussed above, communication on the lowest Purdue layers usually uses specialized *fieldbus* networks, further segregating devices deployed in the field from higher layers. Traditionally, the lower levels of the automation pyramid are considered to be part of the OT network, and the top part of the IT network, but, as we discuss in Section 5.3, this line is blurring.

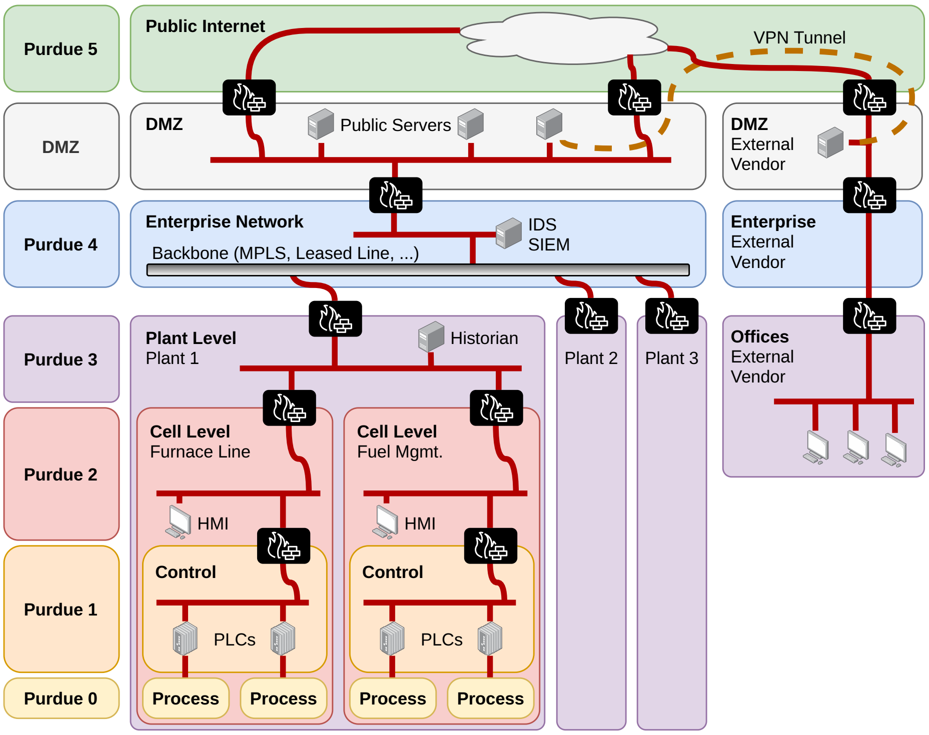


Figure 5.2: A typical Purdue network, as could be found in a critical infrastructure.

From a security perspective, the principal theory behind a Purdue network is that an adversary enters the hierarchy at the top, and needs to make its way down the levels to reach the organization's most critical assets, i.e., obtain control over the physical processes. As each level transition requires passing through a security enforcement point (e.g., a firewall), the lowest levels of the hierarchy achieve the highest level of security. In order for this property to hold, it is important to design network flows to cross as few zone boundaries as possible. After all, each permitted network flow can be used as a conduit for an attack. Thus, if a single flow crosses multiple security boundaries at once, an attacker can use this flow to bypass Purdue levels.

5.3 *Challenges to OT Networks*

The Purdue-based network design discussed in Section 5.2 has successfully served OT operators for over two decades. However, with the advent of the Industrial Internet of Things (IIoT), the "fourth industrial revolution", and the general integration of OT and IT system (IT/OT convergence), the requirements placed on the network are rapidly changing, putting pressure on the Purdue design. We discuss the most significant drivers for these changes in this section.

Changes to the network In the last decade, software-defined networking (SDN) has transformed how IT networks are being operated. So far this change has not yet significantly affected OT networks, but the ongoing convergence of IT and OT systems [41] suggests that it is only a matter of time before this will change. Moreover, recent work from the IEEE Time-Sensitive Networking (TSN) working group [76], including the specification of a TSN profile for industrial automation [77], will allow even the lowest levels of automation networks to use standard Ethernet [100, 185]. This will likely lead to a replacement of the current fieldbus protocols, and will more closely integrate field devices with higher levels of the automation system, in turn making it harder to maintain the strict separation of Purdue levels and easier for an attacker to cross from the higher levels to secondary technologies deployed in the lower levels.

[41] *IT/OT Convergence*, CISCO (2018)

[76] *Time-Sensitive Networking (TSN) Task Group*, IEEE 802.1 (2020)

[77] *IEC/IEEE 60802 TSN Profile for Industrial Automation (Draft D1.2)*, IEEE 802.1 and IEC SC65C/WG18 (2020)

[100] *A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems*, Lo Bello and Steiner (2019)

[185] *The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0*, Wollschlaeger, Sauter, and Jasperneite (2017)

Further, new intra-domain networking technologies, such as TSN and SDN, are increasingly centrally managed, which decreases both the relevance and robustness of distributed security enforcement. For example, when an SDN controller is compromised, an adversary may potentially redefine the network fabric to route packets around firewalls, effectively disabling them [147].

[147] *A Survey of Security in Software Defined Networks*, Scott-Hayward, Natarajan, and Sezer (2016)

Simultaneously, new inter-domain networking technologies, such as the SCION Internet architecture [40] are making it increasingly feasible to send traffic from lower layers of the automation pyramid across autonomous system boundaries. The current hierarchical network design is not well suited to handle such flows.

[40] *The Complete Guide to SCION. From Design Principles to Formal Verification*, Chuat et al. (2022)

Evolution of the automation infrastructure It is common that as the technological capabilities of a system start to exceed the requirements placed on that system by its users, more and more components of that system are replaced by general-purpose components. We have clearly witnessed this in the data center industry with the advent of virtualization technologies (both for end-hosts and for network functions), and also IT/OT convergence is a manifestation of this phenomenon.

Another manifestation of this phenomenon is the rise of *virtualized automation functions*, such as soft-PLC, soft-Supervisory control and data acquisition (SCADA), and soft-human-machine interface (HMI) systems. Contrary to their physical counterparts, virtualized automation functions do not need to be placed physically close to the processes they control. New network technologies (such as TSN) facilitate this further. Concretely, these virtualized computation resources can be placed at the edge (for functions in lower levels of the automation pyramid), or even in the cloud (for functions in the middle to higher levels of the pyramid). This is problematic as the hierarchy of current industrial networks tends to follow physical structures. Hence, current industrial networks are not designed to place physically distant devices logically nearby in the network.

Changes to information flows In traditional automation networks, information does not travel across more than one level of the automation pyramid without being proxied or aggregated. However, the advent of cloud-based big-data analytics for applications such as predictive maintenance has disrupted this. In order to obtain the most accurate predictions, as much raw data from the lower levels of the automation pyramid as possible is now being collected and directly uploaded to the cloud.[1] Supporting such data flows in current networks leads to high management overhead and violates the security principles of Purdue networks.

[1] Examples of industry-oriented cloud platforms include ABB's *Ability Smart Sensor* and Siemens's *MindSphere*.

Changes to threat models The security of a Purdue network is primarily based on the assumption that attackers enter the network at the top levels of the model, and have to work their way down into the lower levels with higher security. However, (i) the increased number of network flows that cross multiple Purdue levels at once, (ii) the increased complexity—and thus vulnerability—of automation devices, and (iii) the increased use of wireless and portable technologies are making it increasingly more likely for an attacker to enter the network directly at a lower Purdue level. This breaks the assumption that the security level of the network increases as one descends through the levels of a Purdue network.

Changes in the industrial target operation model Cost pressure and operational efficiency are leading to the regional cluster model, in which several geographically dispersed plants are remotely managed from a single regional node plant. This allows companies to reduce the personnel required to run plants, and to increase remote operations, sometimes even cross-border. However, such a topology of plants, besides building on an increased level of digitalization, adds complexity into the overall configuration when a Purdue-based configuration is maintained. Moreover, traffic flows between a regional node plant and its cluster plants might traverse public networks. This exposes the traffic to man-in-the-middle and spoofing attacks, which in turn can lead to a loss of control over the remotely managed plants. Hence, additional measures need to be taken to assure the integrity and availability of industrial traffic flows.

6
Tableau: *Future-Proof Zoning for OT Networks*

6.1 Introduction

The trends introduced in Section 5.2 render the current industrial network model increasingly impractical. Even worse, they incrementally erode the security properties of hierarchical network designs. Therefore, it is time to reconsider how we organize OT networks by introducing modern network management techniques to the OT environment. This will allow us to satisfy the contemporary demands placed on our networks, while achieving a high level of security.

To that end, this paper introduces Tableau, a modern zoning model for OT networks. Tableau builds on Mondrian [92], a recently developed zoning architecture for IT networks (see Section 2.3), and makes it suitable for operation on OT networks by defining a new Mondrian deployment model. By doing so, Tableau enables highly flexible network management in OT settings. Particularly, Tableau facilitates the seamless and secure integration of networked resources on the plant floor, at the edge, in the corporate network, and even in the cloud. Moreover, Tableau makes supplier access to OT infrastructures such as PLC, SCADA or HMI systems easier to configure, and reduces the impact of supply chain attacks by facilitating the creation of more, and smaller, network zones. In addition, Tableau accomplishes all of this while remaining compatible with IEC 62443,

[92] *Mondrian: Comprehensive Inter-domain Network Zoning Architecture*, Kwon et al. (2021)

the leading standard for security in industrial networks [81].

[81] *IEC 62443-3-2:2020 Security for industrial automation and control systems - Part 3-2: Security risk assessment for system design*, International Electrotechnical Commission (2020)

Because of the large number of legacy systems typically present in OT networks, Tableau was designed to be brownfield-compatible [1]. Concretely, Tableau provides the following two backward compatibility properties. First, Tableau can be incrementally deployed on subsections of the network while maintaining full network functionality. Second, it is possible to instantiate a hierarchical network structure on top of a Tableau network. Doing so enables network operators to gradually transition their network policies from the hierarchical to the Tableau model. We present the Tableau zoning architecture in Section 6.2, and we illustrate its features using examples based on a typical critical infrastructure network.

[1] Able to operate in the presence of legacy systems.

Tableau represents a significant break from the established, hierarchy-based security mindset in OT networks. We discuss the implications of this change in Section 6.3. Concretely, we argue that (i) by leveraging modern security mechanisms, and (ii) considering the changes that have occurred to OT networks since the hierarchical security models were established, Tableau not only provides much more flexibility to network administrators, but also *increases* the security of the networks in which it is deployed.

6.2 *A Flat Zoning Architecture for OT Networks*

We now introduce Tableau, a zoning architecture for OT networks that leverages Mondrian (see Section 2.3) in order to achieve flexible, future-proof network management.

Because Mondrian was originally designed for enterprise (i.e., IT) networks, we need to modify its deployment model before it can be used in an OT setting. In Section 6.2.1, we present this modified deployment model together with the remainder of the Tableau architecture using an example deployment. Then, we discuss additional Tableau features in Sections 6.2.2 to 6.2.4.

6.2.1 *A* Tableau *Production Plant*

In a standard Mondrian deployment, all the network zones at each site are connected to the same transition point (TP), which in turn is directly connected to the WAN (Fig. 6.1). Doing so results in a flat zone structure, which is one of Mondrian's key features. In order to preserve this feature when using Mondrian in OT settings, it is necessary to map the inherently hierarchical structure of industrial processes to a flat layout. Further, the use of a single transition point per site is not a well-suited approach for OT networks. The reason for this is twofold. First, using a central transition point introduces a single point of failure to the data plane. Second, the physical structure of OT networks and the spatial separation between network zones make connecting each zone to the same transition point impractical.

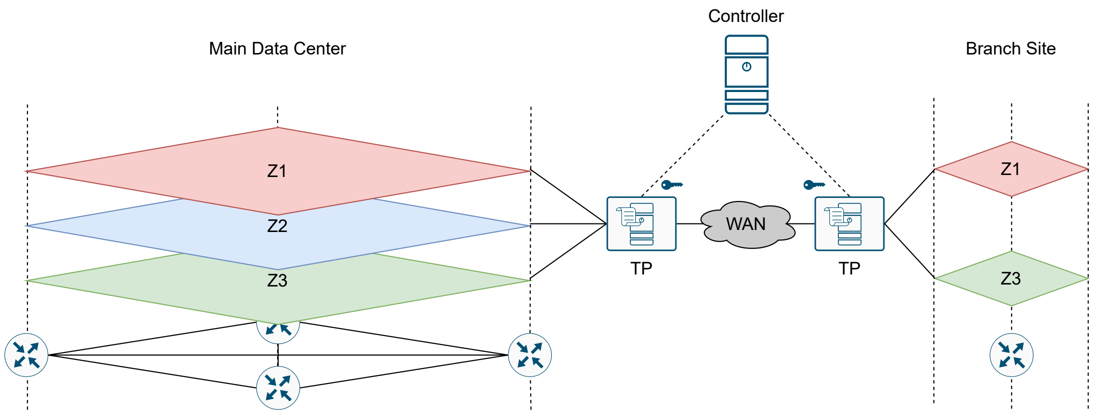


Figure 6.1: A typical Mondrian setup. See Section 2.3 for more information.

In order to flatten the structure of OT networks, we split the network into multiple host zones and a transit network that spans across all traditional network levels, as illustrated in Fig. 6.2. The separation between zones can either be physical (i.e., a zone takes the form of a dedicated physical network), or virtual (e.g., a zone consists of one or more VLANs). In either case, the introduction of a transit network ensures that no transit traffic flows through the host zones.

Next, we change the traditional Mondrian deployment model,

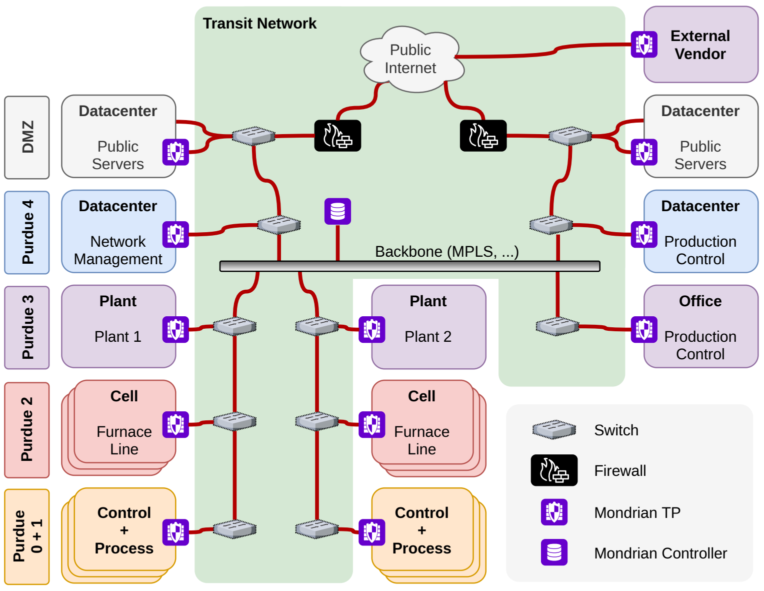


Figure 6.2: The Tableau equivalent to the Purdue-based network shown in Fig. 5.2.

and instead of connecting each zone to a central transition point, we place a transition point at the edge of each zone. Only when practical, zones share a transition point (not shown in Fig. 2.3). Each transition point is then directly connected to the transit network. When traffic leaves a zone, the transition point encapsulates it in an encrypted and authenticated tunnel and forwards the traffic over the transit network to the destination zone, where it is decapsulated before being delivered to the final destination.

Many of the zones in Fig. 6.2 can be directly mapped to one of the hierarchical zones in Fig. 5.2 (we indicated the traditional Purdue level of each zone in Fig. 6.2), but there are a number of notable exceptions. We discuss these, together with other notable Tableau features, below.

Merging Purdue levels 0 and 1 In today's industrial networks, field devices (i.e., sensors and actuators at Purdue level 0) are usually directly connected to their controllers (Purdue level 1) using a physically separated fieldbus network. Although in the future the functions of the fieldbuses might be taken over by a general-purpose network fabric, the close integration of field devices and controllers will remain critical, both for performance and safety reasons. Therefore, Tableau merges the lowest two Purdue levels and places field devices and controllers in the same zone. This captures both the traditional scenario using dedicated fieldbuses (as depicted in Fig. 5.2), as well as the future scenario where both field devices and controllers are connected to a general purpose (TSN) network fabric.

Integration of IT zones Because Mondrian was originally designed for enterprise IT networks, it can be used for the management of both IT and OT networks, greatly simplifying the management of converged networks. We demonstrate this in Fig. 6.2 by incorporating an office zone in the network map. Having this flexibility can be especially useful in highly automated or remotely operated plants, where the notion of a traditional control room is fading.

Integration of remote zones As all data is securely encapsulated during zone transit, the scope of a Tableau network does not need to be limited to a single site or domain, and also zone transitions that use the public Internet are possible without the need for additional tunneling mechanisms. In Fig. 6.2 we demonstrate this with the use case of an external vendor that needs to perform device management or security monitoring tasks on a plant's network. In a Purdue network (Fig. 5.2), a dedicated tunnel must be established and maintained between the network of the vendor and the plant operator, and firewalls or jump hosts throughout the Purdue levels must be configured to grant the required access. Evidently, this leads to high management overhead. In contrast, in a Tableau network (Fig. 6.2) the external vendor's network can be directly integrated in the networks zone plan. We discuss further benefits of inter-domain zone bridging in Section 6.2.2.

Open transit network By only allowing Mondrian encapsulated traffic to flow between network zones, Tableau largely eliminates the need for security enforcement within the transit network. We illustrate this in Fig. 6.2 by only placing classical firewalls on the Internet uplinks. An open transit network not only lowers the burden on the network administrators, but also increases the agility of the network.

Protection of transit traffic Because of the hierarchical nature of Purdue networks, zones in a Purdue network need to handle both transit and local traffic. By mixing these two network functions, transit traffic is exposed to tampering by malicious devices in the network zones the traffic traverses. In contrast, TABLEAU splits the network into device zones and a transit network, separating local from transit traffic. Moreover, all inter-zone traffic is authenticated and encrypted while passing over the transit network. Both of these factors reduce the exposure of network traffic to tampering by malicious devices.

6.2.2 *Inter-Domain Zone Bridging*

We have already shown how TABLEAU facilitates vendor access to OT networks. Not only can the same approach be used to allow remote workers to connect to the company network by running a local Mondrian instance on their laptop, but TABLEAU takes this one step further by splicing network zones across domains.

To make this more concrete, consider the network shown in Fig. 6.3, the left side of which shows a plant network consisting of four network zones. For economic reasons, the plant oper-

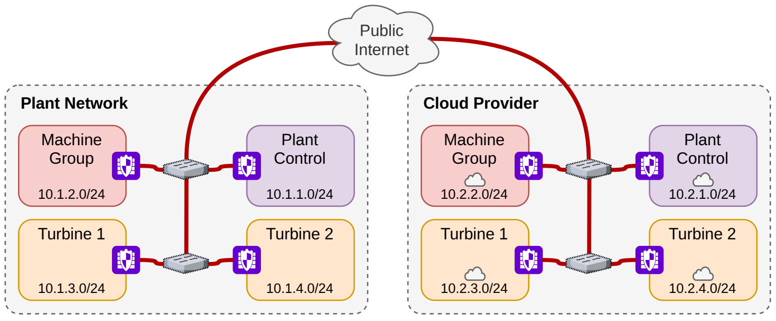


Figure 6.3: The physical layout of a hybrid plant-cloud network.

ators use multiple cloud services to support the devices in the plant. Concretely, they use a digital twin for each of the turbines, a cloud HMI for remote management of the machine group, and a cloud-based data historian for the plant. These services span across all four network zones in the plant, so in order to maintain zone isolation, the zone structure from the plant is mirrored to the cloud. In today's networks, establishing connectivity from the zones of the plant to those in the cloud would either require bundling traffic from all zones together, or setting up separate tunnels between each pair of zones. Because the former approach breaks the isolation between zones and the later approach induces high management overhead, neither of them is desirable.

In contrast, TABLEAU makes it possible to extend network zones across domains, as is shown in Fig. 6.4. This means that the physically distant zones pairs (Fig. 6.3), can be joined to form different subnets of the same logical zone (Fig. 6.4), without creating additional management overhead. Moreover, because Mondrian uses different cryptographic keys for each zone, zone isolation is maintained across the network. Further, this approach is flexible and can be adapted to network operators' needs. For example, instead of extending the same logical zone across multiple domains, the subnets can also be made logically adjacent while remaining in separate zones. This allows for smooth communication to take place between the zones, while still allowing limitations to be placed on which traffic can flow between them.

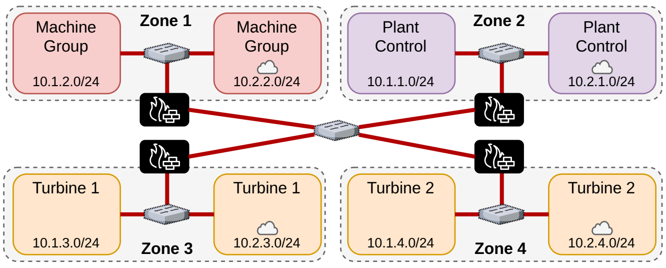


Figure 6.4: The Logical view of a TABLEAU-enabled hybrid plant-cloud network.

6.2.3 *Decoupling TP from Logical Zone Connectivity*

In a TABLEAU network, the logical connectivity between zones is decoupled from the underlying connectivity of the transition points. Besides simplifying the logical network topology, this also simplifies how redundancy and multihoming can be added to the network. Further, when using the SCION Internet architecture [40], the multi-path connectivity between transition points can be abstracted away.

[40] *The Complete Guide to SCION. From Design Principles to Formal Verification*, Chuat et al. (2022)

For a concrete example, consider Fig. 6.5, which shows a minimal TABLEAU network consisting of a plant and a remote control room. In order to ensure availability, both the plant and control room are multihomed. To highlight the separation of the logical connectivity between zones and the underlying connectivity on the transit network, we use Internet Protocol version 4 (IPv4) addresses for the former, and Internet Protocol version 6 (IPv6) addresses for the later.

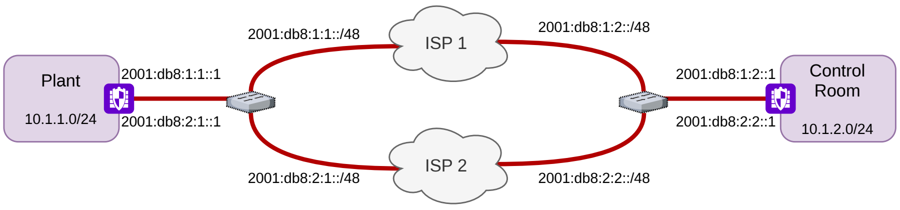


Figure 6.5: Example of a TABLEAU deployment on multihomed networks.

Because the devices inside of the TABLEAU zones are oblivious to the existence of the transit network, multihoming a zone only requires multihoming the zone's transition points. This stands in contrast to traditional multihoming, which directly affects each host in the network [22, 105]. It also means that when the connectivity between two zones breaks (e.g., because of link failure), restoring connectivity between the zones (e.g., by falling back a secondary link) only requires intervention on the transition points, and is transparent to the hosts. Although similar properties can be achieved in a Purdue architecture using VPNs, VPNs generate additional administrative overhead, whereas TABLEAU provides these properties by default.

[22] *Scalable Support for Multi-homed Multi-provider Connectivity*, Bates and Rekhter (1998)

[105] *Problem Statement for Default Address Selection in Multi-Prefix Environments: Operational Issues of RFC 3484 Default Rules*, Matsumoto et al. (2008)

6.2.4 *Backwards Compatibility*

In many cases, industrial networks are a brownfield environment. That is, any change to the network must be made while maintaining compatibility with existing devices and structures. To that end, Tableau offers two forms of backwards compatibility: partial deployment, and hierarchical overlay.

Partial deployment When it is not possible (or desirable) to convert the full network to a Tableau architecture, Tableau can be deployed on a subsection of the network instead. For example, when only a single cell in a plant is being updated, it can be desirable to deploy Tableau in this cell without changing the other parts of the plant's or organization's network. We demonstrate this scenario in Fig. 6.6, which shows the same network as Fig. 5.2, but in which one cell is converted to a Tableau architecture.

Although only a partial deployment, many of Tableau's advantages are retained. Most significantly, there is still full flexibility on how traffic can be routed across the Tableau zones. Moreover, assuming that the upstream firewalls are configured to allow Tableau traffic to pass through, inter-domain bridging remains possible. We illustrate this in Fig. 6.6 by including the external vendor in the Tableau deployment.

In order to facilitate direct communication between the Tableau-enabled cell and the plant's network, a dedicated *entry zone* is introduced. This zone acts as a gateway between the Purdue and Tableau worlds, giving it a similar function as a demilitarized zone (DMZ) in a Purdue network.

Hierarchical overlay A Tableau network provides full flexibility as to what traffic flows are permitted. This means that it is also possible to implement a policy that overlays a hierarchical network on top of Tableau. Doing so allows plants operators to convert their network to a Tableau architecture, without having to redraw all security and data-flow concepts at once. Instead, they can initially overlay the same hierarchical policies the network was operating on before, and gradually transition to new

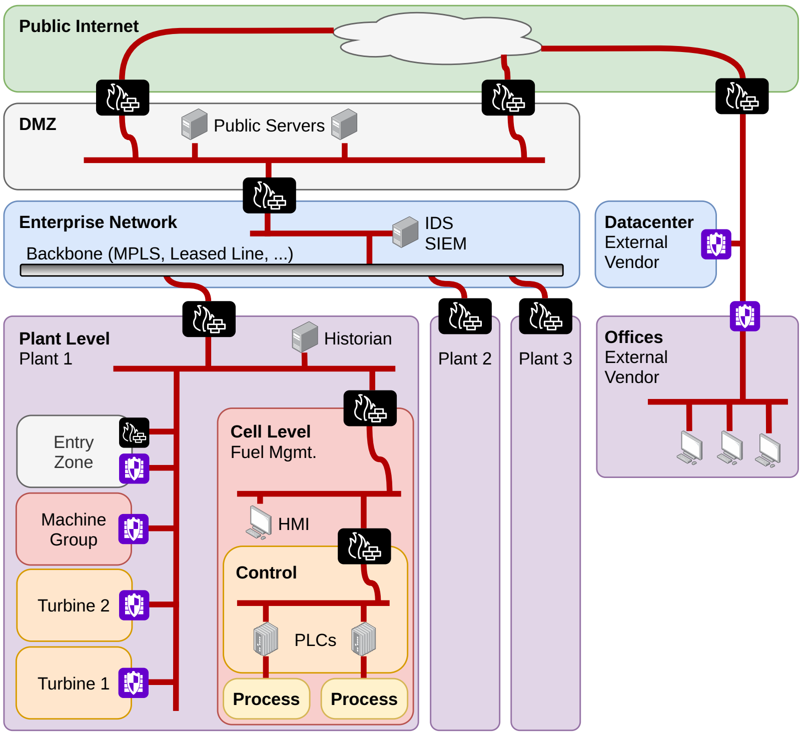


Figure 6.6: Example of a partial TABLEAU deployment.

network policies and a new security concept from there.

6.3 Security Aspects

By stepping away from the nested zone model used in today's OT networks, TABLEAU challenges a widespread design pattern in OT security. Next, we discuss the implications of this architectural change.

Hierarchical network zoning is often motivated by referring to the "defense in depth" security principle: the idea that by layering multiple defense mechanisms behind each other, the security of the system as whole is not compromised when individual defense mechanisms are found faulty. Although it is true that hierarchical network zoning can serve as a form of defense in depth, the true benefits from defense in depth cannot be obtained by using the same defense technique (i.e., firewalls) at multiple points within an organization. Instead, defense in depth requires several independent security mechanisms to be deployed throughout that organization (e.g., firewalls paired with physical security, personnel training, proper patch management, intrusion detection, etc.) [120]. In fact, past studies indicate that having complex, hard-to-maintain firewall structures in a network leads to poor policy management, and thus lowered security [150].

[120] *Defense in depth: A practical strategy for achieving Information Assurance in today's highly networked environments*, NSA (2012)

[150] *Modeling and Management of Firewall Policies*, Al-Shaer and Hamed (2004)

Moreover, as we discuss in Section 5.3, the threat model for OT networks is changing. Concretely, it is becoming increasingly more likely that attackers will not attack the network level-by-level from the top, but instead will enter the network immediately at one of the lower levels, e.g., after entering the network through a compromised software update [165]. Additionally, the centralized nature of new networking technologies (e.g., TSN and SDN) is reducing the robustness of distributed security enforcement [147]. Both these changes are further reducing the efficiency of hierarchical zoning as a defense in depth measure, and, in the medium to long term, will leave industry with a complex and hard to maintain security system, the security properties of which are based on assumptions that no longer hold.

[165] *A 'Worst Nightmare' Cyberattack: The Untold Story Of The SolarWinds Hack*, Temple-Raston (2021)

[147] *A Survey of Security in Software Defined Networks*, Scott-Hayward, Natarajan, and Sezer (2016)

In contrast, TABLEAU does not base its security properties on assumptions about the underlying system architecture, but instead simplifies and centralizes security management in order to facilitate the use of modern security tools. Concretely, by consolidating the security policy of a network into a single specification, TABLEAU facilitates policy simplification, fine-grained zoning, and automated network policy verification. We discuss each of these below.

Policy simplification Consolidating the network policy into a single specification removes much of the complexity currently encountered in firewall management. This makes policy administration less time-intensive and less error prone. Moreover, the policy becomes easier to audit.

Fine-grained zoning As discussed in Section 5.3, an increasing number of devices in the network can function as attacker entry points. In order to limit the impact that a compromised device has on the network, it is desirable to reduce the size of each network zone, thus restricting the lateral movement of an attacker [130]. TABLEAU facilitates fine-grained zoning by lowering the administrative burden required to create and manage additional network zones.

[130] *2020 Unit 42 IoT Threat Report*, Paloalto Networks (2020)

Automated network verification Not only does TABLEAU make it easier to manually audit network security policies, but aggregating the policy specification at the Mondrian controller also facilitates automated network verification [99]. Automated network verification refers to a set of techniques that make it possible to specify high-level policy goals the network should satisfy, and to automatically verify if a specific network policy satisfies these goals [99]. By doing so, network verification can provide strong guarantees on the correctness of the network policy. Moreover, when performed periodically or at every configuration change, automated network verification makes it possible to dynamically modify the network policy while maintaining a high level of confidence in the correctness of the network policies. This makes it easier and safer to update the network policy as the plant's network evolves.

[99] *A Survey on Network Verification and Testing With Formal Methods: Approaches and Challenges*, Li et al. (2019)

We anticipate that in most networks, the advantages of trading the hierarchical network model for the flexibility and simplified policy management of a Tableau network will well outweigh the disadvantages, resulting in an improved level of security for the network. Nonetheless, in some environments the use of consolidated network policy enforcement may be considered undesirable. We address this issue by introducing *structured heterogeneity*, an approach that adds diversity and redundancy to a Tableau network, without interfering with Tableau's core features.

The principal idea behind structured heterogeneity is to standardize the interfaces between the various Mondrian components (i.e., transition point, controller, and policy), and to then add diversity to each of them. Concretely, diversity is added to each component as follows:

Transition points: Different transition point implementations (e.g., from different vendors) can be deployed in different zones. This limits the consequences of an implementation bug in a specific transition point implementation to the zones in which this implementation is used.

Controller: Multiple controller implementations can be deployed in parallel. Each of these controllers connects to the same TPs, and uses the same policy specification. TPs are configured to only permit a zone transition if a threshold number of controllers approve it. This approach also improves network availability, as zone transitions remain possible if one of the controllers is unreachable.

Policy: In order not to increase policy administration overhead, a single policy specification is kept. Instead, we add diversity to the policy *verification*. By verifying the correctness of the policy using multiple methods (e.g., manual inspection combined with multiple automated network verifiers), this ensures that even if an individual verification tool fails, policy goal violations will be detected.

6.4 Compatibility with IEC 62443

IEC 62443 [81] is the leading standard series for industrial OT security. Although industrial networks are often designed using a hierarchical zone model (see Section 5.1), this approach is not mandated by IEC 62443. In fact, IEC 62443 does prescribe any specific zoning model. Instead, IEC 62443-3-2 [80], states "The organization shall group [control systems] and related assets into zones or conduits as determined by risk." (ZCR 3.1) and "[Control system] assets shall be grouped into zones that are logically or physically separated from business or enterprise system assets." (ZCR 3.2). Tableau provides a flexible tool tool to implement these zones and conduits in modern networks. Specifically, zones in a Tableau network map directly to zones as intended by IEC 62443, and conduits are defined by the zone transition policy.

[81] *IEC 62443-3-2:2020 Security for industrial automation and control systems - Part 3-2: Security risk assessment for system design*, International Electrotechnical Commission (2020)

[80] *IEC 62443 Standard Series: Industrial communication networks - IT security for networks and systems*, International Electrotechnical Commission (2020)

6.5 Summary

The rise of the IIoT and the ongoing IT/OT convergence are challenging the ways in which we defend OT networks. If we ignore this reality, the security properties of our networks will slowly erode while administrative overhead will grow. Instead, we must reevaluate the security concepts used in the OT world, and adapt them to reflect the current—and future—state of the network.

In this chapter, we introduced the Mondrian-based Tableau zoning architecture. Tableau provides the flexibility required by contemporary industrial workloads, lowers administrative overhead, is brownfield-compatible, and facilitates the use of modern security practices. Moreover, because Mondrian has its roots in IT networks, Tableau draws from the many years of experience the IT world has with managing the technologies that the IIoT and IT/OT convergence are introducing to our industrial networks.

7 Hopper: *Per-Device Nano Segmentation for the Industrial IoT*

7.1 Introduction

Over the last decade, we have increasingly witnessed industrial control systems becoming the target of sophisticated cyber attacks. These incidents range from attacks attributed to nation state adversaries, such as Stuxnet [95] and the attacks on the Ukrainian power grid in 2015 [191], to criminal ransomware attacks, such as EKANS [53] and TRISIS [54][1]. Contrary to more traditional ransomware, EKANS explicitly targets industrial control systems. TRISIS takes this approach even further by specifically targeting the industrial safety controllers which function as a last resort to prevent catastrophic process failures.

[95] *Stuxnet: Dissecting a Cyberwarfare Weapon*, Langner (2011)

[191] *Inside the Cunning, Unprecedented Hack of Ukraine's Power Grid*, Zetter (2016)

[53] *EKANS Ransomware and ICS Operations*, Dragos (2020)

[54] *TRISIS: Analyzing Safety System Targeting Malware*, Dragos (2017)

[1] Also known as "Snake" and "TRITON", respectively.

Although not many details about these attacks are public, a common trend is that they rely on lateral movement through the victim's network in order to reach the critical systems which they target. This approach is often successful because today's industrial network design practices strongly rely on perimeter protection, often having no to very little defenses in place once a perimeter has been breached. Similarly, Tableau places security enforcement points (i.e., transition point) only at the edge of network zones.

For many years this perimeter-based approach has served industry well, but, as discussed in Chapter 5, new trends are challenging this approach. For example, the increased deployment of IT systems in OT (i.e., factory automation) networks is breaking down the strong boundary between the IT and OT worlds. Moreover, trends such as (i) the increasing prevalence of cloud-assisted devices, (ii) the increasing risk of supply-chain attacks [165], and (iii) the increasing use of both long- and short-range wireless communication [141] are blurring the network boundary further. That is, every device is becoming a potential attacker entry point, rendering the sole reliance on perimeter-based defenses impractical and prompting the need for stronger defenses against lateral movement.

[165] *A 'Worst Nightmare' Cyberattack: The Untold Story Of The SolarWinds Hack*, Temple-Raston (2021)

[141] *IoT Goes Nuclear: Creating a ZigBee Chain Reaction*, Ronen et al. (2017)

Currently, industry is responding to this trend by limiting the movement of attackers by creating more—and thus smaller—network segments, a practice referred to as *micro segmentation* [130]. However, traditional segmentation mechanisms (e.g., VLANs) can only take one so far, do not scale well, and require traffic to pass through a centralized router to move between segments, thereby introducing a single point of failure in the data plane. Although TABLEAU facilitates the creation of smaller network zones, it still requires traffic to be routed over one or two transition point, introducing practical scaling limits. Thus, rather than enforcing traffic restrictions at specific points in the network, an ideal defense must enforce that only permitted communication can take place *uniformly and throughout* the network, without reducing the network's versatility.

[130] *2020 Unit 42 IoT Threat Report*, Paloalto Networks (2020)

In enterprise networks, a similar philosophy recently gained traction under the name *zero trust networking* [64]. However, as zero trust networking techniques tend to leverage enterprise-oriented security mechanisms, they are typically not suitable for the (industrial) IoT. For example, the highly constrained nature of many IoT devices renders the use of certificates challenging. Moreover, zero trust networking focuses on the protection of resources at end-hosts, whereas in industrial networks, the network fabric itself must be protected as well. For instance, being able to generate cross-traffic on a network link carrying a control signal with hard real-time requirements can increase

[64] *Zero Trust Networks*, Gilman and Barth (2017)

congestion and latency, reducing the quality of the manufactured product [89].

[89] *CPS: Driving Cyber-Physical Systems to Unsafe Operating Conditions by Timing DoS Attacks on Sensor Signals*, Krotofil et al. (2014)

To address these issues, we propose Hopper: a protocol that combines two of the best properties of zero-trust networking and classical network segmentation. Hopper is intended as an extension to perimeter-based network segmentation (e.g., using Tableau or VLANs), and provides segmentation uniformly throughout the network fabric.

On a Hopper network, each flow must be explicitly authorized, and restrictions are enforced both at end hosts, as well as in-network. Our approach de facto results in a per-device *nano segmentation* of the network, which is enforced in-fabric. That is, in a Hopper network, the attack surface is minimized by restricting the permitted communication to the minimum required for the operation of the deployment. By enforcing this restriction at each hop, Hopper effectively places each device in its own nano segment.

More concretely, Hopper achieves nano segmentation by allowing each network node to verify that each packet it processes (i) is part of an explicitly whitelisted flow, and (ii) was generated by an authorized host. Hopper is compatible with the many constraints and networking technologies encountered in industrial IoT networks, allowing it to be uniformly deployed throughout a wide range of scenarios.

Recognizing the centrally managed nature of industrial networks, Hopper has a centralized control plane, but is fully distributed in the data plane. Specifically, Hopper constructs a capability mechanism based on a hierarchical key space that is used to uniquely bind each packet to an authorized flow. Once distributed, authorized senders use their capability tokens to generate authentication tags for each transmitted packet. By distributing part of the hierarchical key space to each node, Hopper ensures that these tags can be independently verified at every network hop, while requiring only minimal local node state and using only symmetric cryptography. Further, Hopper places no assumptions on the structure of the underlying physical network, and introduces no more per-packet bandwidth overhead than a standard authentication tag. Leveraging additional common IoT characteristics allows us to keep this approach scalable while maintaining flexibility.

7.2 Constraints in Industrial IoT Networks

Whereas TSN will likely unify the wired communications in industrial environments, many industrial IoT products, especially those designed for monitoring, are using wireless communication. Contrary to the situation with TSN, it is unlikely that a universal wireless protocol will emerge in the near future. Moreover, the requirement for long (e.g., 10-year) battery lives introduces many (extreme) constraints for networks. In order to support deployment throughout industrial networks, it is important for a protocol to be able to operate under any of these constraints.

Providing a complete taxonomy of (industrial) IoT networks and devices is out of scope for this work. Instead, we discuss the diversity and possible limitations of the devices and networks that we consider. We use RFC 7228 as a basis for this discussion [28].

[28] *Terminology for Constrained-Node Networks*, Bormann, Ersue, and Keränen (2014)

Devices Industrial IoT devices range from potent, mains-powered industrial servers running standard operating systems (e.g., HPE's *Edgeline* "converged edge systems") to miniature "motes" with ten-year battery lives (e.g., National Control Devices's mesh sensors). Throughout this range of devices, the following constraints may be encountered:

***C1**: Limited state*, for program code and temporary storage.

***C2**: Limited computation*, due to low-end processors.

***C3**: Limited power*, additionally limiting computation and severely restricting wireless communication.

Networks Similar to IoT devices, networks in IoT deployments span a wide range of technologies and constraints. At one extreme, IoT devices can use gigabit-speed wired connections (e.g., KUKA's *KR C4* industrial robot controllers), while others use multi-hop mesh networks with sub-kbps data rates and highly restricting duty cycling (e.g., Analog Devices's *SmartMesh* or Digi's *DigiMesh* products). Throughout this range of network technologies, the following constraints may be encountered:

C4: *Low data rates*, due to the power constraints of the underlying devices.

C5: *High and unstable latency*, due to multi-hop networks and energy-conservation policies of the devices.

C6: *Reachability limits* of duty-cycling or default-off devices.

C7: *Lack of advanced network features* such as multicast.

C8: *Unusual routing* such as the deliberate duplication of packets for redundancy reasons, or the broadcast-like nature of flooding-based network architectures [61].

[61] *Efficient network flooding and time synchronization with Glossy*, Ferrari et al. (2011)

C9: *Unstable topologies* caused, for example, by mobile nodes or opportunistic routing.

7.3 *Adversary Model & Security Goals*

This chapter considers a network model in which networks consist of (i) links, (ii) forwarding elements, (iii) hosts, and (iv) a network controller, which are all managed by a network administrator. We use the generic term *forwarding element* to cover all network elements that forward packets. Network nodes can simultaneously be hosts and forwarding elements, e.g., in mesh networks.

Given this network model, we consider a network-based adversary which may have compromised a subset of hosts, forwarding elements, and network links. The attacker can have used any method (including physical access) to compromise these devices and links, and has full control over them. Moreover, the attacker can communicate out-of-band between the points of attack. However, the network controller and administrator must remain uncompromised.

The attacker's goal is to increase the scope of its attack by exploiting the network. Possible attacker strategies include: (i) using the network to compromise additional devices, (ii) sending spoofed packets to disrupt physical processes (e.g., impersonating a sensor and sending false readings), and (iii) performing denial- or reduction-of-service attacks against the network fabric (e.g., generating traffic in order to increase network latency in critical control loops or to reduce the lifetime of battery powered IoT deployments).

Hopper aims to mitigate such attacks by placing each device in its own access-controlled nano-segment, thus minimizing both its network exposure and its network access. Concretely, we define the following security goals:

HOPPER-1, *Least privilege:* Communication involving at least one uncompromised host can only take place on the logical flows explicitly whitelisted by the network administrator.

HOPPER-2, *Isolation:* An adversary that compromised a set of network elements can only generate network traffic between these elements, or towards the union of destinations these elements interact with under normal network operation.

HOPPER-3, *Authentication:* Each packet is source authenticated to its receiver. This prevents the adversary from spoofing packets packets that simultaneously (i) appear to be generated by a host that is not under its control, and (ii) are accepted by a host that is not under its control.

We explicitly do not list confidentiality as a security goal,[2] but we briefly discuss how it could be added to our design in Section 7.5.6.

[2] An attacker who can compromise packet integrity can de facto take over control over an industrial process, which in turn leads to a major safety risk. Because this is not the case for compromised confidentially, integrity is considered a much more important goal than confidentiality in industrial settings.

7.4 Hopper *Protocol*

We demonstrate how nano segmentation can be achieved using the example network shown in Fig. 7.1, which is based on a hydroelectric power plant. The primary section (i.e., the main plant building) of the network is Ethernet-based. The hosts on this part of the network constitute everything from servers and engineering workstations to turbine controllers and printers. Further, this section contains the network controller and the main Internet uplink. Connected to the primary network are a remote network (e.g., an upstream sluice complex) and a mesh network which contains low-power sensors mounted to the machinery in the plant. The link between the primary and remote network section may be virtual (e.g., a VPN tunnel) or physical (e.g., a long-range Wi-Fi link).

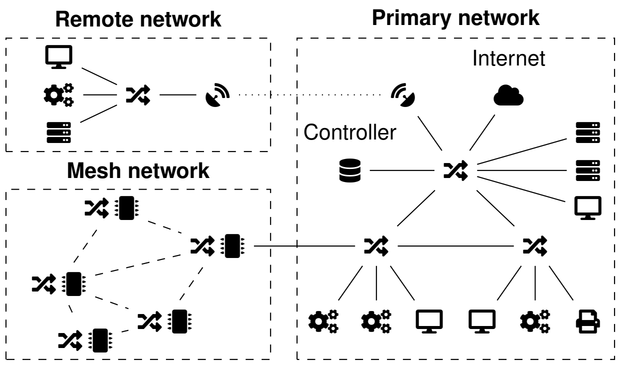


Figure 7.1: An example industrial network. 🔀 represents a forwarding element; ▮, ⚙, 🖶, ☰, and 🖵 represent hosts; and 📡 ···· 📡 represent a remote link. Dashed links are wireless.

Because the network in Fig. 7.1 covers a single plant, it is administrated by a single entity (i.e., the plant's network management team). This is typical for industrial networks, and we refer to it as *ownership centrality*. Even though the network administrator might not be the most privileged user on all devices (e.g., in the case of devices that are managed through a vendor-operated cloud), they still set the policies each device should comply with, and—in the worst case—can remove non-complying devices from the network.

Further, although industrial facilities may rely on distributed data flows, they are typically centrally orchestrated, which we refer to as *orchestration centrality*. Orchestration centrality holds for our power plant, but also, for example, for a smart factory, which will be configured by its operators to execute a specific set of tasks. Moreover, from the automation pyramid introduced in Section 5.2, we know that these tasks are temporally stable and change at most a couple of times a day.

The combination of ownership and orchestration centrality, together with a focus on a small number of temporally-stable, cyber-physical workloads, is a defining characteristic for most industrial networks. Moreover, this combination of properties makes it possible for network administrators to both compile an a priori whitelist of legitimate flows, and establish network policies that only allows packets that are part of a legitimate flow to use the network. When enforced at every hop, this effectively creates a nano segment for each device on the network, while still allowing packets to be routed on the shortest physical path between their source and destination.

Further, industrial networks—similar to enterprise networks—are strongly zone-oriented. In fact, network zoning is mandated by IEC 62443 [81], the leading standard for security in industrial networks. However, contrary to zoning in the enterprise, zoning in OT networks is typically done based on the automation pyramid (see Section 5.2), and different zones may contain radically different types of devices.[3] For example, a SCADA zone may contain standard workstations and servers, whereas a field zone can consist of a mesh of low-power sensors.

[81] *IEC 62443-3-2:2020 Security for industrial automation and control systems - Part 3-2: Security risk assessment for system design*, International Electrotechnical Commission (2020)

[3] Even though TABLEAU steps away from a hierarchical network layout, we still expect devices to be grouped based on their position in the automation pyramid.

Depending on the network zone, a different subset of the constraints listed in Section 7.2 will be encountered, changing the restrictions and requirements placed on (security) protocols. In order to cope with this diversity, we design HOPPER to be fundamentally compatible with *all* of the listed constraints, and simultaneously allow it to be adapted to specific deployment environments through parameterization. This allows a single, well-understood, security protocol to be deployed throughout the network, rather than requiring a different protocol design for each type of network zone.

As of yet, no protocols allow segmentation policies to be implemented homogeneously throughout the network while being compatible with the constraints of IoT and industrial networks. In the following subsections, we explain how we fill this void with the design of HOPPER, thereby enabling the nano segmentation throughout industrial IoT networks.

7.4.1 High-Level Design

In order to meet the least privilege goal (**HOPPER-1**) described in Section 7.3, Hopper must verify that each packet is part of a whitelisted flow at at least one point in the network: either at the receiving host or at a forwarding element. As in mesh networks or in networks using a shared medium it is not guaranteed that a flow traverses a forwarding element (**C8**), this check must be performed on the receiving host. However, to meet the isolation requirement (**HOPPER-2**), packets must be checked at each forwarding element as well. Thus, every node in the network must verify the permissions of every packet it processes.

The high availability requirements posed to industry require that no single point of failure exists in the data plane. Moreover, when the network is partitioned by the failure of a link or forwarding element, the individual partitions of the network should remain operational. For example, if the remote link in Fig. 7.1 fails, the remote network should remain in operation. This means that packet checks must take place without the online assistance from the centralized controller. Further, because transceiving data in wireless (mesh) networks consumes multiple orders of magnitude more energy than performing (symmetric) cryptography [75, 152], nodes should be able to verify packets using only static, node-local knowledge (**C2, C3**). This requirement is further reinforced by the low data rates (**C4**), high latency (**C5**), limits on reachability (**C6**), and lack of advanced features (**C7**) common to low-power wireless networks.

[75] *Power Consumption and Calculation Requirement Analysis of AES for WSN IoT*, Hung and Hsu (2018)

[152] *How low energy is bluetooth low energy? Comparative measurements with ZigBee/802.15.4*, Siekkinen et al. (2012)

There are two general methods that can be used to allow nodes to verify if a packet is part of a whitelisted flow: (i) access lists or (ii) capabilities. Because using access lists would involve disseminating large amounts of information to each node of the network—which is impractical in constrained networks (**C1, C3–C7**)—Hopper follows a capability-based approach.

Past work on deny-by-default networking has used capabilities in the form of authenticated source routes [36]. However, two characteristics of industrial IoT products render such an approach unsuitable. First, roaming nodes (e.g., a sensor mounted on an automated guided vehicle (AGV)), duty cycling nodes,

[36] *SANE: A Protection Architecture for Enterprise Networks.*, Casado et al. (2006)

and fading wireless links lead to dynamic and unpredictable network paths (**C9**). Second, for reliability reasons, some protocols (e.g., Ethernet TSN) include mechanisms to replicate packets in-network and to send these packets over different paths (**C8**), complicating source routing.

Instead, a capability scheme that allows each node in the network to verify the permissions of each packet in the network should be used. Moreover, cryptographically binding together capability tokens with their packets allows Hopper's authentication goal (**HOPPER-3**) to be satisfied. However, limited computational power severely restricts the use of asymmetric cryptography (**C2, C3**).

Considering these requirements, we observe many similarities with multicast-authentication, in which many listeners must be able to verify that a packet was generated by an authorized sender [34, 142], and we use past results from this space as a basis for Hopper.

[34] *Multicast security: a taxonomy and some efficient constructions*, Canetti et al. (1999)

[142] *New results on multi-receiver authentication codes*, Safavi-Naini and Wang (1998)

Based on the discussion above, we design Hopper to authenticate each packet by adding a cryptographic tag to it. This tag consists of a concatenation of multiple MACs, which are generated using keys from a hierarchical key space. By linking the keys in this key space to the flow identifiers carried in the packets, a capability scheme is constructed.

In order to be able to verify that packets are part of an authorized flow, each forwarding element is provided with the state needed to partially verify any Hopper tag. Hiding which forwarding element verifies which part of the tag forces the adversary to consider each forwarding element to verify the entire Hopper tag. Moreover, using a key space with multiple levels of hierarchy allows receiving hosts to fully verify the Hopper tags of all incoming packets.

Given the expected prevalence of UDP/IP at the lowest levels of industrial networks, we focus our design of Hopper on this protocol stack. Although TCP is a more popular transport at higher levels off the automation pyramid, the trend in both IoT and industrial standards is towards UDP in constrained environments. For example, whereas OPC UA uses TCP on higher network levels, a UDP mapping was specified for OPC UA over

TSN [127]. Nonetheless, Hopper's design can be easily modified for operation on different layers or protocol. For example, in the evaluation (Section 7.7), we also implement and evaluate Hopper for L2 Ethernet.

[127] *OPC UA Online Reference, Section 7.3.2: OPC UA UDP*, OPC Foundation (2021)

7.4.2 *Constructing Capability Tokens*

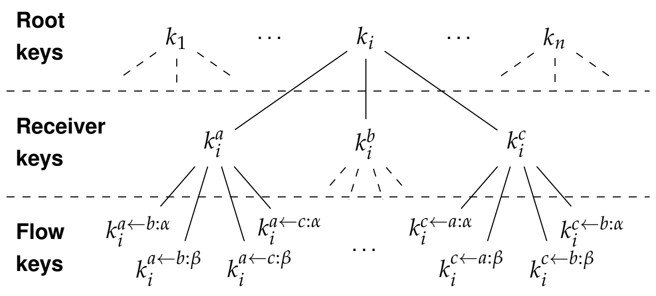

Root keys
k_1
$\dots$
k_i
$\dots$
k_n
Receiver keys
k_i^a
k_i^b
k_i^c
Flow keys
$k_i^{a \leftarrow b:\alpha}$
$k_i^{a \leftarrow b:\beta}$
$k_i^{a \leftarrow c:\alpha}$
$k_i^{a \leftarrow c:\beta}$
$\dots$
$k_i^{c \leftarrow a:\alpha}$
$k_i^{c \leftarrow a:\beta}$
$k_i^{c \leftarrow b:\alpha}$
$k_i^{c \leftarrow b:\beta}$

Figure 7.2: Excerpt from a Hopper key forest for three hosts (a, b, and c) and two ports (α and β).

The hierarchical key space is constructed as a forest consisting of n trees, as shown in Fig. 7.2, and where n is a configuration parameter. The keys in the i-th tree are used to generate and verify the i-th MAC in the Hopper tag. The root of each tree i is a random secret, denoted k_i. The set of all roots

$$\mathcal{K} = \{k_i \mid i \in [n]\} \text{ with } [n] := \{1, \dots, n\}$$

is called the root key set and is only fully known to the network controller, with subsets being distributed to all forwarding elements. Further, for each receiver r, a receiver set

$$\mathcal{K}^r = \left\{k_i^r = \mathrm{PRF}_{k_i}(r) \mid i \in [n]\right\}$$

of receiver keys is derived from $\mathcal{K}$. PRF_{k_i} is a pseudorandom function keyed with k_i.[4] The receivers store these keys locally and use them to verify incoming packets.

[4] In practice, a block cipher encryption operation can be used to implement the PRF.

If the network controller wants to issue a capability token for host s to communicate with host r on UDP port p, it generates the flow set

$$\mathcal{K}^{r \leftarrow s:p} = \left\{k_i^{r \leftarrow s:p} = \mathrm{PRF}_{k_i^r}(s \| p) \;\middle|\; i \in [n]\right\}$$

and provides it to host s. "$\|$" represents string concatenation.

7.4.3 *Sending and Verifying Packets*

Once a host receives a flow set, it can send on the corresponding flow. To send a packet, the host calculates a tag τ consisting of one MAC over the packet payload for each key in the flow key set, i.e.,

$$\tau = \text{MAC}(k_1^{r \leftarrow s:p}, \text{payload}) \| \ldots \| \text{MAC}(k_n^{r \leftarrow s:p}, \text{payload})$$

and adds it to the packet. Because the cryptographic strength of τ is dependent on its full length, the individual MACs can be short [34].

[34] *Multicast security: a taxonomy and some efficient constructions*, Canetti et al. (1999)

While the packet traverses the network, each forwarding element it encounters verifies the MACs for which it can derive the flow key using the packet's header info; i.e., the MACs at the positions for which it has the root secrets. If any of these MACs are incorrect, the packet is dropped and reported to the network controller, signalizing the presence of an attacker. The receiver always verifies the entire HOPPER tag by deriving the flow key set from its receiver key set using the information in the packet header.

7.4.4 *Distributing Root Keys*

By provisioning forwarding elements with subsets of the root key set, HOPPER allows each forwarding element to partially verify each HOPPER tag without them being able to forge valid tags. However, properly distributing the root keys is a complex problem, as a good distribution scheme must satisfy two contradicting goals: (i) as much of the HOPPER tag as possible must be verified as early in the network as possible, and (ii) when an attacker compromises one or more forwarding elements, he should not learn a sufficient number of root keys to forge HOPPER tags.

The first of these goals ensures that packets are dropped early, which minimizes their effect on the network. Intuitively this can be achieved by provisioning each forwarding element with as many keys as possible, thus ensuring that it can verify a large part of the HOPPER tag. Yet, this conflicts with the second goal, as it makes it easier for an attacker to learn the full root key set. In addition, a good key-distribution scheme should ensure that even when the attacker has compromised some keying material, forged packets will still be dropped as close to the sender as possible.

Inspired by multicast security, we propose three key distribution schemes: cover-free families [142], random, and manual distribution.

[142] *New results on multi-receiver authentication codes,* Safavi-Naini and Wang (1998)

Cover-free families are combinatorial constructs on sets: An (m, n, t)-cover-free family is a family of m subsets on a ground set of size n, so that none of the sets in the family is a subset of the union of t other sets in the family. Considering a network with m forwarding elements and a root key set of size n, we can assign each forwarding element the keys from a set from a (m, n, t)-cover-free family over the root key set. This guarantees that even when t forwarding elements are compromised, each forwarding element will still verify at least one MAC for which the adversary did not obtain the key. Unfortunately constructing cover-free families with large m and t is considered a hard problem [142].

Random distribution of the root keys avoids the construction problems of cover-free families. Consider again a root key set of size n, of which each key is now assigned to each forwarding element with probability p. This ensures that, in expectation, each forwarding element verifies $n \cdot p$ MACs per packet. When t nodes are compromised, the adversary will have obtained the full root key set with

$$P_{\text{compromise}}(t) = \left(1 - (1 - p)^t\right)^n \ .$$

The expected number of compromised keys is $n \cdot \left(1 - (1 - p)^t\right)$.

For example, setting $n = 10$ and $p = 0.2$, results in each forwarding element verifying 2 MACs on average. However, even if the adversary compromises $t = 5$ forwarding elements, it will have obtained the full root set with probability of only $P_{\text{compromise}}(5) = 0.018$ %, and is expected to know only 6.7 keys. Fig. 7.3 shows how $P_{\text{compromise}}(t)$ varies for various values of n, p, and t. Interestingly, for a given value of t, $P_{\text{compromise}}(t)$ is independent of the total number of forwarding elements in the deployment.

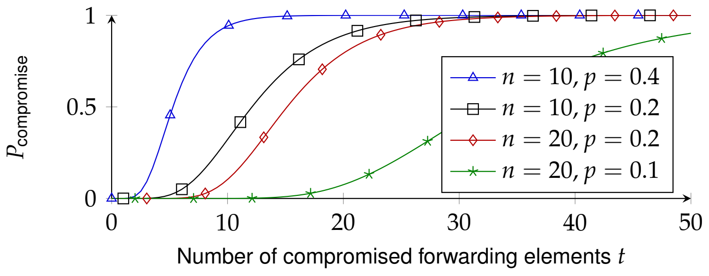


Figure 7.3: Probability that an attacker learns the full root key set when compromising t forwarding elements, assuming random root key distribution.

Manual distribution of the root keys can be useful for small deployments. However, as it is hard to perform a systematic evaluation of this strategy, this work focuses on random distribution.

Regardless of the key distribution scheme, a compromised forwarding element will leak part of the root key set. Such leakage reduces the effective strength of the Hopper tag, as the adversary needs to guess fewer tag bits. Moreover, when random root key distribution is used, the adversary might be able to compute all MACs verified at a specific forwarding element. Nonetheless, even when the attacker lacks a single root key, it is highly likely that the adversary's presence on the network will be detected. This is because (i) the receiving host always verifies the full tag, (ii) tags are different for each packet, and (iii) generating a single bad tag will lead to the tag verification failing (either at a forwarding element or at the receiver) and the network controller being notified.

7.4.5 *Security Equivalence & Network Dichotomy*

Both cover-free families and random distribution schemes are effective to make Hopper resistant against the compromise of a small number of forwarding elements. However, IoT deployments can be highly homogeneous and can consist of thousands of nodes. Moreover, in industrial networks the same type of sensor or actuator is often reused throughout a network or facility. In such settings it is likely that when one device is compromised, many will be. This especially holds if the initial node was compromised using a network attack, as the marginal cost to compromise additional nodes will be close to zero. After all, these nodes are likely to run similar software in similar configurations.

While this may seem troublesome at first, we show how the introduction of *security equivalence classes* can mitigate the negative effects that homogeneity has on HOPPER's resilience. Moreover, we highlight how the dichotomy found in most infrastructure-based networks further increases resilience.

Security equivalence Typically, the robustness of key distribution schemes is expressed in the number of devices that can be compromised before security breaks. However, as discussed above, in IoT deployments it is likely that if a single device is compromised, many devices will be. We formalize this notion by partitioning forwarding elements in *security equivalence classes*. Concretely, we assume that either all devices in a security equivalence class are compromised, or that none are. We observe that under this assumption, there is no security benefit in provisioning the forwarding elements within one equivalence class with different root keys. Specifically, when an equivalence class is compromised, all the keys present in the class will be known to the adversary regardless of how they are distributed. Similarly, when the class is not compromised, provisioning all root keys known to the class to each of its members ensures that as much of the HOPPER tag as possible is verified by each element.

Distributing root keys to security equivalence classes rather than to individual devices significantly increases the scalability of HOPPER: instead of tolerating the compromise of t devices, HOPPER now tolerates the compromise of up to t device *classes*. As for hybrid host-forwarding element devices there is no security benefit in having devices share receiver keys, we only apply this strategy to root keys. Doing so ensures that when a security equivalence class would nonetheless be only partially compromised, HOPPER's end-to-end properties still hold within that class.

How to divide an IoT network into security equivalence classes is an exercise that must be made individually for each network. More broadly speaking, finding optimal partition schemes for a given network opens up an interesting avenue for future research. For example, when defending against network-based attacks, the primary device properties to consider are the logical network layout, device types, and configurations. Conversely, when physical attacks are the primary concern, all devices in the same physical access zone can be combined in a security equivalence class.

Network dichotomy Most infrastructure-based IoT networks, much like traditional computer networks, display a clear split between hosts, and forwarding elements. We refer to this as *network dichotomy*. As in such networks the hosts do not forward packets, they do not hold root keys. Hence, no number of compromised hosts will lead to the exposure of the root keys. Additionally, the forwarding elements, which do hold root keys, will be fewer in numbers and are typically more hardened then hosts connected to them.[5] This further increases HOPPER's resilience to node compromise. We note that a similar observation was made in the multicast authentication setting by Canetti et al. [34].

[5] For example WiFi access points and backbone forwarding elements are likely to be more hardened than the low-cost IoT devices connected to them.

[34] *Multicast security: a taxonomy and some efficient constructions*, Canetti et al. (1999)

7.5 Practical Considerations

7.5.1 Rekeying and Revocation

As part of standard security practices, HOPPER networks should be periodically rekeyed. Maintaining a symmetric key for each controller-host pair allows the network controller to securely communicate the updated HOPPER keys with each network device, meaning that rekeying can be automated. By using a hierarchical key system similar to a tree in a HOPPER forest, the secure storage requirements on the network controller can be minimized. In the case the network was compromised and part of the root key set was exposed, a rekeying of the network is also required.

Periodic rekeying also prevents hosts from accumulating capabilities. That is, HOPPER requires each node to discard capabilities (i.e., flow key sets) upon receiving a revocation notice. Because uncompromised devices can generally be expected to behave as specified, in most circumstances this is sufficient. Still, periodic rekeying ensures that also the network permissions of (dormant) malicious hosts are periodically reset. By (i) using keys with overlapping validity periods, and (ii) performing rekeying together with production reconfiguration events, the impact of rekeying events on the physical process can be minimized. Coinciding rekeying events with a production reconfiguration further makes sense, as only during such events new flow keys, the accumulation of which we wish to prevent, are issued.

7.5.2 Handling Network Diversity

Hopper has a number of parameters that can be adjusted depending on the needs of the network or zone it is deployed in. For example, when deployed in a mesh-like network, Hopper tags should consist of possibly many (n is large), but short (l is small) MACs. Conversely, when deployed on a network with only a single forwarding element (e.g., a network with a single gateway and a star topology), tags could consist of a single (n is small), but long (l is large) tag. This flexibility allows a single, well-understood, protocol (i.e., Hopper) to be used in a wide range of settings rather than requiring the design of one-off protocols for each new setting.

Table 7.1 summarizes how different parameters influence Hopper's performance. We note that the number of forwarding elements in the network does not influence the data-path overhead introduced by Hopper.

Besides modifying Hopper's basic parameter, it is also possible to adapt Hopper for operation on different protocols or layers. This is done by adapting Hopper's key space, for example by deriving Hopper keys based on Ethernet instead of IP addresses. For other protocols the required changes can be more significant. For instance, in multi-receiver protocols the flow keys must be distributed between the receivers in a similar manner as the root keys are distributed to forwarding elements.

Parameter		Sender workload	Receiver workload	Forw. el. workload	Tag forging success prob.	Resiliance to bad forw. el.	Bandwidth overhead
n	Size of root key set	△	△	–	▼	▲	△
m	# forwarding elements	–	–	–	–	–	–
p	Key sampling prob.	–	–	△	▼[†]	▽	–
l	Individual MAC length	–	–	–	▼	–	△

Triangles indicate the properties' direction of change if the corresponding parameter increases. ▼/▲ indicates a desirable direction of change, ▽/△ a deterioration. [†]Forged tags more likely to be detected early in network.

Table 7.1: Effect of various parameters on Hopper performance, assuming random root key distribution.

7.5.3 *Connecting Networks*

Hopper's centralized control gives rise to a natural notion of Hopper *domains*: regions of the network sharing a set of Hopper root keys and parameters. When packets cross the borders of a Hopper domain, Hopper tags must be removed from, or added to, the packet. This is done by Hopper gateways, an extended version of the Hopper delegates introduced in Section 7.5.5.

For outgoing traffic the gateway verifies the Hopper tag, removes it from the packets, and transmits the packet over its outgoing interface. For incoming traffic, the gateway functions similarly to a firewall in drop-by-default mode: the gateway verifies if the traffic is part of a whitelisted flow, and if so attaches the corresponding Hopper tag to the packet before forwarding it to the domain-internal destination. If the traffic is not whitelisted, it is dropped. Gateways must have access to the flow keys for each flow that traverses them. As gateways are typically part of the network infrastructure and do not suffer from the constraints listed in Section 7.2, this is a reasonable requirement.

7.5.4 *Management Overhead*

As Hopper was designed to require explicit whitelisting for every flow, it naturally introduces administrator overhead to the network. Specifically, overhead is created (i) when new devices are added to the network, and (ii) when a new workload is deployed. We proceed to quantify the overhead introduced by each of those events.

New devices When a new device is added to the network, it must be provisioned with a minimum of keying material in order to be able to start communicating. When a key provisioning server and bootstrapping mechanism similar to the ones described in Section 7.5.7 are used, each new device needs to be provisioned with only a single set of flow keys. This creates an administration overhead that is comparable to provisioning devices with WPA-Enterprise credentials.

New workloads When a new workload is configured, flow keys must be created and distributed for each required flow. Also the overhead of this operation can be significantly reduced through the use of a key provisioning server as described in Section 7.5.7. Concretely, when using a key provisioning server, the administrator overhead required to authorize a new flow is comparable to overhead in the (hypothetical) case were all devices must communicate through a centrally-managed, drop-by-default firewall.

Although the introduced overhead can appear high at first, it is worth noting that beside the security aspects discussed before, HOPPER's whitelist-based approach also provides another valuable property: transparency of network assets. From our interactions with industry practitioners, we have learned that a lack of insight into the devices and flows present in industrial deployments is a real problem that is often encountered when performing a security assessment of industrial networks. HOPPER alleviates this problem by having a security-by-design approach that requires all devices and flows to be listed (and therefore inventoried) before they can participate in the network.

Further, we observe that the set of flows that need to be whitelisted is a direct function of the workload placed on an industrial deployment. This opens opportunities to integrate HOPPER whitelist management into production management, which could lead to a significant reduction of management overhead.

7.5.5 *Legacy Compatibility*

Industrial installations often have lifetimes that can reach in to the tens of years, during which they are incrementally updated. This means that it is important for new industrial systems and protocols to be brownfield compatible, i.e., be able to operate in coexistence with legacy systems. When naively deploying HOPPER alongside non-HOPPER aware protocols and devices, security issues may arise. For example, an adversary may be able to mount a downgrade attack by removing the HOPPER header from a packet.

In this section, we discuss how HOPPER can be securely deployed alongside legacy systems. We split this discussion into two orthogonal parts: interoperability with legacy networking nodes (i.e., hosts and forwarding elements), and interoperability with legacy networking protocols.

Legacy Nodes We propose three brownfield strategies that allow HOPPER to be deployed on a network with legacy nodes: (i) HOPPER delegates, (ii) zone-based deployment, and (iii) an overlay strategy.

As its name suggests, the first strategy introduces the concept of a HOPPER delegate, which is used to connect hosts that are not HOPPER-aware to the network. All traffic from/to the HOPPER-unaware host is routed over its delegate before entering/leaving the HOPPER domain. The delegate holds the HOPPER keys of the host and adds, checks and removes HOPPER tags on behalf of the host. By implementing delegatee functionality in access switches, this can be accomplished with minimal management overhead and while maintaining all of HOPPER's core properties.

When using the zone-based strategy, HOPPER-aware network nodes are deployed grouped in zones. These zones are then connected to the non-HOPPER aware parts of the network using gateways which add and remove the HOPPER headers as needed. The concept of a HOPPER gateway is discussed in more detail in Section 7.5.3. While this strategy maintains HOPPER's properties within individual zones, inter-zone traffic travels between its source and destination zones without HOPPER's protections.

The overlay strategy is similar to the zone-based strategy, but rather than removing the HOPPER headers at the edge of a HOPPER zone, traffic is tunneled between HOPPER zones to create virtual HOPPER-aware links. This preserves the HOPPER headers and allows multiple zones to be operated as a single HOPPER domain, meaning that HOPPER's traffic properties are maintained end-to-end. However, as the virtual HOPPER links may share their underlying physical links with legacy traffic, they can be susceptible to denial-of-service attacks.

Regardless of the brownfield strategy, using the extension options provided by existing protocols (e.g., "next protocol" header fields or extensions), allows HOPPER to be implemented transparently for HOPPER-unaware forwarding elements. This results in a network (zone) where HOPPER-aware forwarding elements verify the HOPPER header, while HOPPER-unaware forwarding elements simply forward the packets without checks. This is similar to how VLAN-unaware routers are able to forward VLAN-tagged traffic and should work out-of-the-box, reserving interference caused by network middleboxes.

Legacy protocols Because traditional network protocols are designed to make communicating as easy as possible, some of these protocols are at odds with HOPPER's design principles. Specifically, decentralized auto-configuration and auto-discovery protocols, such as Address Resolution Protocol (ARP), do not fit the deny-by-default paradigm. That is, these protocols (i) are explicitly designed to facilitate unplanned communication, and (ii) typically require broadcast communication, allowing each node to contact each other node. This stands in contrasts to HOPPER's design goals that limit network permissions to the absolute minimum.

When deploying HOPPER, a threat analysis of these legacy protocols must be made. If the attack surface created by the protocol is sufficiently small, no changes are needed. However, if the protocol exposes a large attack surface, it must be eliminated. This can either be accomplished by statically configuring the normally auto-configured state (e.g., pre-populating ARP tables), or by replacing the decentralized protocol by a centralized one. We provide an example of the latter approach in Section 7.5.7.

7.5.6 *Encrypting Packets*

Hopper is not designed to provide confidentiality. However, the key distribution mechanism used by Hopper can be modified to support per-flow, end-to-end encryption. An encryption key for a flow can be established either by cryptographically combining all flow keys into a single encryption key, or by adding a dedicated encryption tree to the Hopper key forest. The receiver and flow keys from this tree are distributed as normal, but the root key is not distributed to any forwarding elements. Both methods guarantee that only the sender and receiver of a flow (and the network controller) can calculate the encryption key.

7.5.7 *Bootstrapping and Provisioning*

Hopper does not specify a specific bootstrapping or key provisioning method. Nonetheless, in this section we explore how a practical key distribution system can be accomplished. Concretely, we do so by extending the functionality of the network controller to distribute Hopper keys through the network.

As Hopper is a strict capability-based network architecture, all hosts require a minimum of state before they can start communicating, even with the controller. Moreover, in order to securely exchange keys between the controller and a host, a confidential and authentic channel between them must be established.

The first requirement can be achieved by pre-provisioning each device with flow keys for the flow from the device to the controller. For example, through physical contact following a resurrecting duckling model [159]. To meet the second requirement, we observe that at any point in time, only three entities can generate all valid flow keys for a given flow: (i) *the sender*, which owns the flow keys; (ii) *the receiver*, which can derive them from his receiver keys; and (iii) *the controller*, which can derive any key from the root keys.

[159] *The resurrecting duckling: Security issues for ad-hoc wireless networks*, Stajano and Anderson (1999)

Because during communication with the controller the receiver and the controller are the same entity, the controller can uniquely authenticate any device from the packet's HOPPER tag. Thus, the flow keys can be used as shared keying material to bootstrap an authenticated and encrypted channel between any device and the controller. Once this channel is established the device can send a key request to the controller, which verifies the request against an internal policy file and sends the appropriate receiver and flow keys to the device. At this point, the device and controller can also agree on additional keying material to be used to for network recovery when the root key set is compromised.

In the case of hybrid host–forwarding element devices, the controller can also supply the root keys for the forwarding elements in this manner. This allows the network to be cold-started in a hop-by-hop fashion. Moreover, as the controller has a full view of the network, it can also distribute otherwise dynamically discovered information, such as link-layer addresses, together with flow keys.

7.6 Security Analysis

As introduced in Section 7.3, a HOPPER network consists of four sets of elements: (i) network links, (ii) forwarding elements, (iii) hosts, and (iv) the network controller. Motivated by ownership and orchestration centrality, HOPPER leverages the administrator, and by extension the network controller, as the trust root of the system. Hence, the network controller is uncompromised by assumption. This section analyses the consequences of attacks against each remaining network element. Devices which both act as host and forwarding element are susceptible to attacks relating to both devices classes. Moreover, as both hosts and forwarding elements have link access, link-related attacks are also applicable to them.

Network links An adversary with access to a network link, either wired or wireless, can observe, drop[6], duplicate, replay, or inject packets on that link. We now discuss the effect of each of these actions on the three security goals listed in Section 7.3.

[6] Or jam, in the case of wireless links.

Observing packets does not violate any of HOPPER's security goals. Moreover, as each HOPPER MAC uses full-length keys, key extraction—which would facilitate injection attacks—is not feasible.

Dropping packets can be used to mount a DoS attack, but only against the compromised link, therefore it does not violate the isolation (**HOPPER-2**) or other security goals.

Duplicating or replaying packets can also be used to mount DoS attacks, but can only create traffic towards the destination hosts of flows that traverse a compromised link, and hence does not violate the isolation (**HOPPER-2**) or other security goals. Moreover, implementing in-network duplicate suppression can further limit the scope of duplication attacks. Many IoT oriented wireless protocols already include explicit replay protection mechanisms (e.g., Bluetooth Mesh [26], the IETF's IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [3], and IEEE 802.15.4 [143]), and the TSN task group has also specified duplicate elimination for Ethernet networks in IEEE 802.1CB-2017 [79].[7]

[26] *Mesh Profile Specification 1.0.1*, Bluetooth Special Interest Group (2019)

[3] *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, Alexander et al. (2012)

[143] *Security Considerations for IEEE 802.15.4 Networks*, Sastry and Wagner (2004)

[79] *IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability*, IEEE Computer Society (2017)

[7] The duplicate frame elimination specified IEEE 802.1CB-2017 is not explicitly designed as a security mechanism, but rather as part of a reliability mechanism.

If injecting forged packets on a link is possible, each forwarding element will detect and drop each forged packet with probability $1 - 2^{-l|S|}$, where $|l|$ is the length of each MAC in bits, and $|S|$ the number of root keys known to the forwarding element. When using random distribution, $E[|S|] = pn$, with p the key sampling probability, and n the size of the root key set. When using cover-free families, $|S|$ is the size of sets in the family. After injecting c packets, the in-network detection probability of the adversary is

$$p_{\text{detection, network}} = 1 - 2^{-l \sum_{i=1}^{c} \{|S_i|\}}$$

where $|S_i|$ is the size of the union of keys known by the forwarding elements traversed by the i-th packet. When using random distribution and traversing h elements,

$$E[|S_i|] = n \cdot \left(1 - (1-p)^h\right)$$

The probability $p_{\text{detection, network}}$ quickly tends to 1. The isolation (**HOPPER-2**) goals is thus probabilistically satisfied. Further, forged packets will be detected with probability $1 - 2^{-ln} \approx 1$ by each end host. For c packets, the detection probability becomes

$$p_{\text{detection, host}} = 1 - 2^{-lnc}$$

which also quickly tends to 1, satisfying the least privilege (**HOPPER-1**) and authentication (**HOPPER-3**) goals.

Forwarding elements When an adversary compromises one or more forwarding elements, he learns part of the root key, leading to the modified detection probabilities

$$p'_{\text{detection, network}} = 1 - 2^{-l \sum_{i=1}^{c} \{|S'_i|\}}$$

and

$$p'_{\text{detection, host}} = 1 - 2^{-ln'c}$$

where n' is the number of uncompromised root keys. When random root key distribution is used, the expected value of n' can be calculated using the equations in Section 7.4.4. When using a (m,n,t)-cover-free family, $n' \geq 1$ as long as less than t forwarding elements have been compromised. $|S'_i|$ is the number of uncompromised root keys known to any forwarding element on the path of packet i. For random root key distribution

$$E[|S'_i|] = n' \cdot \left(1 - (1-p)^h\right)$$

When using a cover-free family, $|S'| \geq 1$ as long as less than t forwarding elements have been compromised. As both $p'_{\text{detection, network}}$ and $p'_{\text{detection, host}}$ tend to 1 as long as not all root keys are compromised, least privilege (**HOPPER-1**), isolation (**HOPPER-2**), and authentication (**HOPPER-3**) remain probabilistically satisfied.

Hosts If an adversary compromises a host, he obtains access to the receiver and flow keys stored on that device. The flow keys can be used to send arbitrary packets towards their corresponding receivers, but not towards other hosts, thus not violating the least privilege (**HOPPER-1**) and isolation (**HOPPER-2**) goals.

When an adversary has compromised multiple hosts, and can communicate between them, he can exchange the compromised receiver and flow keys between these hosts. The receiver keys allow the compromised hosts to generate and send traffic towards each other. The flow keys allow each host to generate traffic on the logical flows corresponding to the compromised flow keys, and to send that traffic towards the receiver of these flows. Neither of these attacks violates the least privilege (**HOPPER-1**), isolation (**HOPPER-2**), or authentication (**HOPPER-3**) goals.

7.6.1 Mitigation of Today's Common IoT Threats

Deploying Hopper on a network can mitigate common threats against IoT devices seen today. We discuss three examples.

Botnets Since the Mirai botnet demonstrated the destructive power of large-scale IoT attacks, botnets have been a major IoT security concern [130, 9]. Because IoT botnets are usually constructed by finding inadvertently vulnerable devices through network scans, Hopper's whitelist-based approach provides a strong defense against device compromise. Moreover, because the number of destinations a Hopper-enabled host can send traffic to is severely limited, Hopper devices are not attractive botnet members.

[130] *2020 Unit 42 IoT Threat Report*, Paloalto Networks (2020)

[9] *Understanding the Mirai Botnet*, Antonakakis et al. (2017)

Worms Whereas the focus of IoT attackers used to be on constructing botnets to perform DoS attacks, their focus has shifted towards IoT worms, such as cryptolockers [130, 54, 53]. Because Hopper's nano segmentation minimizes the number of potential victims an infected host can reach, the spreading of worms is severely restricted. This stands in contrast to to traditional segmentation methods where worms can spread freely within each segment.

[130] *2020 Unit 42 IoT Threat Report*, Paloalto Networks (2020)

[54] *TRISIS: Analyzing Safety System Targeting Malware*, Dragos (2017)

[53] *EKANS Ransomware and ICS Operations*, Dragos (2020)

Lateral movement In many attacks against IoT devices, the compromised devices are not the primary attack target, but are used as entry points for further lateral movement in the network [130]. Similarly to how Hopper defends against worms, Hopper's nano segmentation significantly reduces the number of new attack vectors available to an attacker after compromising a device.

[130] *2020 Unit 42 IoT Threat Report*, Paloalto Networks (2020)

7.7 *Implementation and Evaluation*

We split the evaluation of Hopper into four subsections. First, we perform a scalability analysis of the protocol. Second, we evaluate the performance of a Hopper-enabled end host. In order to verify that Hopper is suitable for constrained environments, we perform this evaluation on IoT-class hardware. Third, we evaluate a Hopper forwarding element by implementing Hopper forwarding logic on a low-end network appliance and benchmarking this appliance by physically connecting it to an emulated network. Finally, we briefly report on our implementation of a Hopper controller and confirm interoperability between Hopper's network elements.

7.7.1 *Scalability*

We evaluate the scalability of Hopper on the three active elements of a Hopper network: (i) hosts, (ii) forwarding elements, and (iii) the network controller. We also discuss bandwidth overhead.

Hosts Hosts need to store one set of receiving keys and one set of flow keys for each outgoing flow. More formally, hosts need to store $n \cdot (f+1) \cdot |k|$ bytes, where n is the size of the root key set, f the number of outgoing flows of that host, the constant "1" represents the receiver key set, and $|k|$ is the size of an individual key. As shown in Table 7.1, n is not a function of the size of the deployment, but rather of the number of compromised forwarding elements that can be tolerated. The computational overhead per packet is constant in the size of the deployment: for each incoming or outgoing packet n MACs must be calculated, and for incoming packets an additional n key derivations must be performed.

Forwarding elements Assuming random root key distribution, forwarding elements need to store $p \cdot n \cdot |k|$ bytes, where p is the key sampling probability. Also here computational overhead per packet is constant in the size of the deployment: each forwarded packet requires $2 \cdot p \cdot n$ key derivations and $p \cdot n$ MACs calculations.

Network controller For each rekeying event, the network controller must distribute new root keys to the forwarding elements, receiver keys to the receiving hosts, and flow keys to the sending hosts. However, rekeying events are rare and take place on the control rather than data plane. Therefore the performance of the network controller is not critical to HOPPER's overall performance.

Bandwidth overhead Each packet must carry a tag of length $n \cdot l$ bytes, where l is the length of an individual MAC. As discussed in Section 7.5.2, n and l are constant in the size of the deployment.

7.7.2 End Host Performance

We evaluate HOPPER's end host performance using two ST Nucleo-F439ZI development boards. These boards carry a STM32F439ZI MCU which runs at 180 MHz, representing midrange IoT devices. The MCU provides hardware cryptography acceleration, a common feature on IoT MCUs to support link-layer encryption. The boards also have on-board 10/100 Mbps Ethernet, facilitating evaluation in an isolated and reproducible environment.

We implement two versions of HOPPER: one for UDP/IP, as described in Section 7.4, and one for Ethernet, where we use the source address, destination address and L3 protocol as flow identifier.

Both HOPPER versions extend the open-source lwIP (lightweight IP) protocol stack [55] to support HOPPER tag generation and verification. Our HOPPER implementation for Ethernet defines a new EtherType and places tags directly behind the Ethernet header. The implementation for UDP/IP defines a new IP protocol number and places tags between the IP and UDP headers.

[55] *Design and Implementation of the lwIP TCP/IP Stack*, Dunkels (2001)

To perform the evaluation, the two boards are directly connected using an Ethernet cable. During each experiment one board continuously generates UDP datagrams (or Ethernet frames) with a dummy payload. The other board receives these datagrams (or frames) and processes their headers. No higher layer processing is performed.

We evaluate each Hopper implementation using both hardware-accelerated and software crypto. In the hardware setting we use AES128 as block cipher and SHA256 as hash algorithm. In the software setting we use the Speck software cipher [24] with 64-bit blocks, and use BLAKE3 [123] as hash algorithm. PRFs are implemented using the block cipher in ECB mode, keys are always 128 bits long.

[24] *The SIMON and SPECK Families of Lightweight Block Ciphers*, Beaulieu et al. (2013)

[123] *BLAKE3: one function, fast everywhere*, O'Connor et al. (2020)

The hardware ciphers were chosen because our evaluation platform provides hardware acceleration for them. Different software ciphers where selected for performance reasons: using Speck instead of an AES software implementation [88] resulted in a throughput improvement of roughly 5 to 100 %, depending on the setting. Similarly, using BLAKE3 instead of software SHA256 [114] resulted in throughput gains of 35 to 100 %.

[88] *Tiny AES*, kokke (2021)

[114] *SHA-2 implementation*, Mosnier (2019)

Further, we evaluate each Hopper implementation in each cryptography setting in two tag composition settings. The first tag setting uses tags consisting of ten 16-bit MACs calculated using CBC-MAC over a hash of the packet payload.[8] We note that hash-then-MAC is a provably secure construct [27]. The second tag setting uses tags consisting of a single 128-bit MAC for hardware crypto and a single 64-bit MAC[9] for software crypto.

[8] We use CBC-MAC, as the hash algorithms ensure constant input sizes.

[27] *A Graduate Course in Applied Cryptography*, Boneh and Shoup (2020)

[9] 64 bits corresponds to native Speck block size on a 32-bit architecture. Brute forcing a 64-bit MAC using minimal Hopper packets over a gigabit link would, in expectation, take on the order of 10^5 years.

Because the results for Hopper on Ethernet and on UDP/IP are near identical, we only discuss the latter in this section. The results for Ethernet can be found in the appendix of our ASIACCS paper [49].

[49] *Hopper: Per-Device Nano Segmentation for the Industrial IoT*, De Vaere, Tulimiero, and Perrig (2022)

Hardware cryptography We see in Fig. 7.4 that when using hardware crypto and one MAC, Hopper packets can be generated (Tx, 94 Mbps) and processed (Rx, 89 Mbps) at 99 % and 93 % the rate of plain UDP/IP (96 Mbps), respectively. The header space required by Hopper leads to a slight reduction in maximum payload compared to plain UDP/IP. When using ten MACs, we observe packet generation (67 Mbps) and processing (49 Mbps) rates of 70 % and 52 % the rate of plain UDP/IP, respectively. As can be seen from the load profiles in Fig. 7.5, the discrepancy between the results for incoming and outgoing packet is caused by the flow-key derivations required on the receiving endpoint. Key caching or probabilistic MAC evaluation could mitigate this performance gap, though the latter would lead to a reduction in security. Fig. 7.5 also shows that the majority of HOPPER's workload consists of cryptographic operations. It is worth noting that when using hardware crypto acceleration, these operations are executed by the MCU's cryptography peripheral, leaving the core free to perform other tasks.

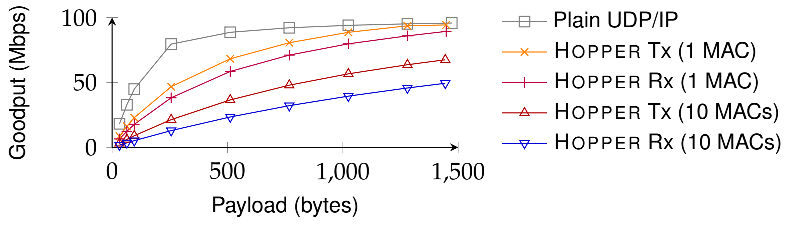


Figure 7.4: Plain UDP/IP vs. HOPPER on UDP/IP end host goodput. Using hardware-supported AES128 and SHA256.

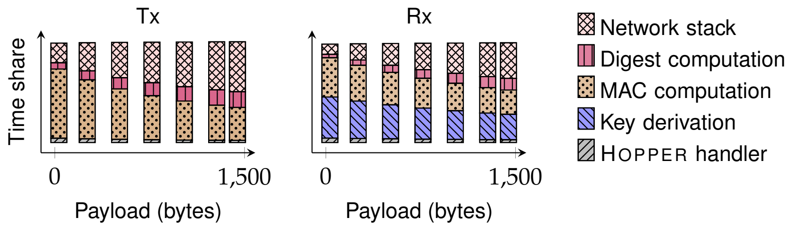


Figure 7.5: Simplified end host load profile for HOPPER on UDP/IP using 10 MACs per tag. Using hardware-supported AES128 and SHA256.

Software cryptography As shown in Fig. 7.6, when using software cryptography, throughputs between 28 Mbps and 19 Mbps are achieved. This corresponds to 29 and 20 % the rate of plain UDP/IP. The dip in performance at 1024 bytes is inherited from BLAKE3. As shown in Fig. 7.7 the performance is dominated by the hash function.

Given the low data-rates typically found in IoT and industrial applications[10], we find the obtained performance results to be

[10] For example, the Ethernet Advanced Physical Layer (Ethernet-APL), a recently developed physical layer for Ethernet which is targeted specifically at industrial applications, operates at only 10 Mbit/s [78].

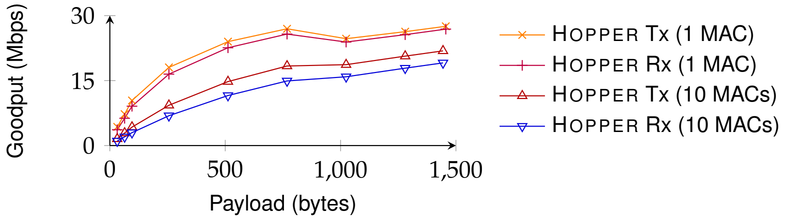


Figure 7.6: Plain UDP/IP vs. HOPPER on UDP/IP end host goodput. Using software Speck and BLAKE3.

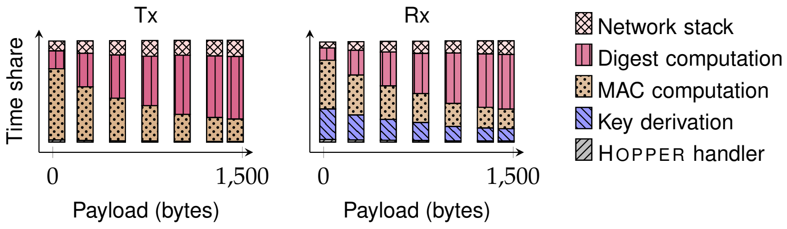


Figure 7.7: Simplified end host load profile for HOPPER on UDP/IP using 10 MACs per tag. Using software Speck and BLAKE3.

satisfactory. Although the use of software cryptography results in a significant performance penalty, most MCUs targeted at IoT applications already provide hardware cryptography acceleration to support link-layer encryption. Hence, we expect software cryptography to only be used in the most constrained settings, which typically feature low throughput requirements.

Binary size The MCU binaries for the tests without HOPPER, with hardware crypto, and with software crypto were 75, 100, and 113 kB in size respectively. These binaries were compiled for optimal performance. When compiling for an optimized binary footprint, the sizes reduce to 61, 78, and 89 kB, respectively. Doing so results in a throughput reduction of 0 to 15 %, depending on the setting.

7.7.3 *Forwarding Element Performance*

We implement a HOPPER forwarding element using a PC Engines APU2D4 system board. The APU2 is a popular platform for low-end network appliances, and with a quad-core AMD GX-412TC CPU running at 1 GHz and 4 GB DDR3-1333 memory, its performance is roughly comparable to a Raspberry Pi 4. The board also has 3 Gigabit Ethernet interfaces.

We implement the Hopper forwarding logic for UDP/IP using DPDK [168] and OpenSSL [128]. For each packet the forwarding element receives, it verifies the Hopper tag using the root keys with which it was provisioned. If successful, the packet is forwarded without further processing. If the verification fails, the packet is dropped. The forwarding element uses AES128 as block cipher and SHA256 as hash algorithm. PRFs are implemented using AES128 in ECB mode. We use Hopper tags with 10 MACs of 16 bits each.

[168] *Data Plane Development Kit*, The Linux Foundation (2021)

[128] *OpenSSL*, OpenSSL Software Foundation (2021)

Test traffic generation We create a gopacket [65] implementation of Hopper. For each experiment, we first generate a pool of randomized 5-tuples. We then generate Hopper packets by randomly sampling (with replacement) 5-tuples from this pool, and adding a random payload. The generated packets are then stored in a pcap file to be replayed using tcpreplay during the experiment.

[65] *GoPacket*, Google (2021)

Throughput To measure the forwarding element's throughput, we physically connect it to two hosts that each emulate a network, as illustrated in Fig. 7.8. We then let the source host transmit 10^6 packets at line rate and measure the incoming traffic volume at the receiver host. We repeat this experiment multiple times varying the number of root keys the forwarding element is provisioned with. We also perform a baseline measurement in which the forwarding element immediately forwards all packets without parsing them.

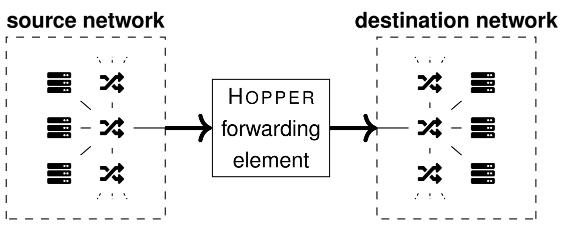


Figure 7.8: Measurement setup used to benchmark the Hopper forwarding element. Bold arrows indicate physical cables.

We see in Fig. 7.9 that for small packets, or when verifying only one MAC, using Hopper does not meaningfully reduce the forwarding element's throughput. When verifying 2 MACs, performance is reduced by approximately 10 % to 890 Mbps. Verifying all 10 MACs results in a throughput of 565 Mbps.

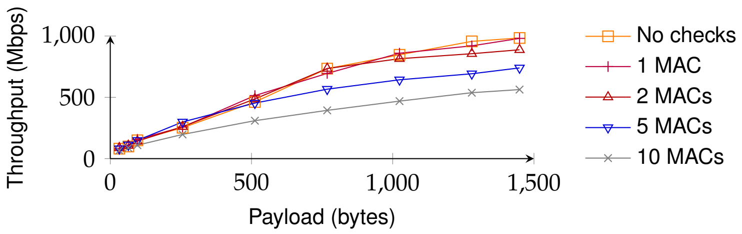


Figure 7.9: Hopper forwarding element performance for different numbers of MACs checked.

Scalability In Section 7.7.1, we showed that scaling the size of a Hopper deployment does not influence the performance of the forwarding elements. We now verify this result by varying the size of the 5-tuple pool (which effectively corresponds to changing the size of the emulated network), and rerunning the throughput experiments. For these experiments we configure the forwarding element to verify 2 MACs per packet.

Fig. 7.10 shows that, somewhat counterintuitively, increasing the number of network flows *increases* the throughput of the forwarding element. This is a side effect of the hash-based receiver side scaling used by DPDK: the larger the set of flows, the more uniformly the workload is spread across the four processor cores. We verified this hypothesis by running the same experiment using only a single processor core, and we observe that doing so results in identical throughput regardless of the traffic mix (not shown).

Latency We evaluate the latency added to the forwarding element by the Hopper checks. We again use the test setup shown in Fig. 7.8, but now emulate both the source and destination network on the same host, using different host interfaces. In each test setting, we transmit 10^3 packets, one at a time, and we capture the kernel-generated send and receive timestamps of each

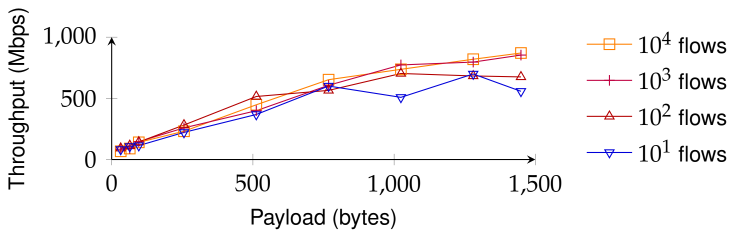


Figure 7.10: Hopper forwarding element performance for different emulated network sizes. Two MACs are verified.

packet using tcpdump. We also measure two baselines: one using a loopback cable which bridges the source and destination interface on the emulation host, and one in which we configured DPDK to directly forward all packets without performing any checks or parsing.

The results of the latency measurements are shown in Fig. 7.11. We see that verifying two Hopper MACs per packet results in a median latency increase of 20 to 40 μs, depending on the packet size. The increase in latency overhead with increasing packet sizes is a direct consequence of SHA256 hash that is calculated over the packet payload. Verifying 10 MACs instead of 2, adds an additional 20 μs to the median latency, regardless of the packet size. Further, we observe only a minor effect of Hopper on latency jitter, measuring an average standard deviation of 17, 17, 20, and 22 μs for the loopback, plain DPDK, 2 MACs, and 10 MACs settings, respectively.

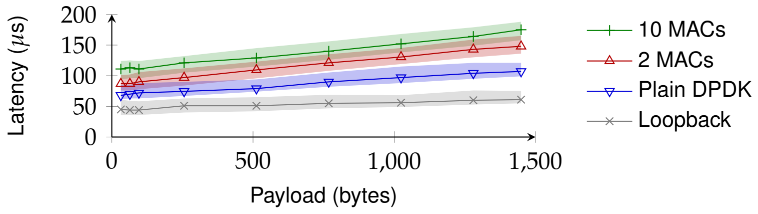


Figure 7.11: Median Hopper forwarding element latency. *Loopback* shows the base latency measured without forwarding element, *Plain DPDK* uses a forwarding element that forwards all traffic without any checks or packet parsing. The shaded areas show the 25th and 75th percentiles.

7.7.4 Controller Implementation

We implemented a proof of concept Hopper controller that allows an administrator to specify which flows are whitelisted using a policy file and dynamically distributes flow keys on request. We also extended our Hopper implementations to automatically request missing flow keys from the controller and performed a functional evaluation of interoperability and the bootstrapping mechanism described in Section 7.5.7. Because the speed of the initial key distribution is not critical, we did not benchmark the controller.

7.8 Relationship with Tableau *and IEC 62443*

In Chapter 6, we introduced Tableau, a modern zoning approach for automation networks. Then, in Section 6.4, we discussed how Tableau can be used to divide networks into zones and conduits in accordance with the IEC 62443 standard [80].

[80] *IEC 62443 Standard Series: Industrial communication networks - IT security for networks and systems*, International Electrotechnical Commission (2020)

Similar to Tableau, Hopper is a network segmentation protocol. However, whereas Tableau uses a perimeter-based approach, Hopper provides in-fabric and per-device segmentation. These two approaches are complementary and are to be used in tandem. Concretely, Tableau is used to group devices into zones at a macro level, and to set macro-level flow policies between these groups. Each Tableau zone can simultaneously be Hopper domain, providing low-level protection uniformly across the zone's network fabric. The Tableau transition points serve simultaneously as Hopper gateways (see Section 7.5.3), adding and removing hopper headers as packets enter or leave a Tableau zone, respectively. From an IEC 62443 perspective, Tableau is used to divide the networks into zones and conduits, whereas Hopper assists towards increasing the security level (SL) within each zone.

Combining Hopper with Tableau as described above has as advantage that it decouples the multiple Hopper domains. This means that (i) compromised keys in one domain do not affect other domains; (ii) Hopper parameters (e.g., number of MACs) can be set on a per-zone basis, and (iii) Hopper tags are always fully verified before a packet leaves a network zone, ensuring that even when the root key set of one zone is partially compromised, no forged packets can leave the local network zone. Additionally, no number of compromised devices[11] allow an adversary to send packets between zones that are not legitimately

[11] excluding the Tableau transition points or controller

intended to interact.

7.9 *Summary*

New trends in industrial automation are challenging the ways in which industrial networks are secured. In order to address these challenges, this work presents Hopper, a nano segmentation scheme for industrial IoT deployments. By deploying capability-based and deny-by-default networking, Hopper extends the protections typically only provided at the edge of the network to the entire network fabric, realizing per-device network segmentation. Moreover, Hopper was designed to be compatible with the diverse constraints encountered in IoT deployments, allowing the same protocol to be deployed across a wide range of settings.

Contrary to the ossification seen in traditional networks, the IoT networking stack is still in flux. This represents a unique, but ephemeral, opportunity to embed security-by-design into the core of the IoT stack. Hopper accomplishes this by extending the broadly accepted principle of least privilege across the network. Moreover, as the IoT's momentum is ever increasing, the need for the strong but flexible defenses that Hopper provides has never been so critical.

8 Related Work

8.1 Sensors and Actuators

Remote attestation The use of remote attestation techniques to secure IoT devices has been widely studied. In their most essential form, attestation techniques provide integrity guarantees about binaries. Such *static* techniques are not only widely used in research (e.g., in the Sancus security architecture for IoT devices [119]), but have also matured enough to see applications in industry, most commonly in combination with secure boot [15, 7].

[119] Noorman et al. (2017)

[15] ARM Ltd. (2009)

[7] Analog Devices (n.d.)

However, static attestation cannot provide guarantees about control-flow integrity. C-FLAT [1] and LO-FAT [50] address this issue by designing IoT-compatible control-flow attestation schemes. C-FLAT does so by relying on TrustZone for Cortex-A, while LO-FAT introduces custom hardware. SIMPLE [6] continues this line of work by proposing a software-only remote attestation mechanism that implicitly guarantees control flow integrity. In contrast, OAT [164] maintains a dependency on TrustZone, but adds data integrity guarantees.

[1] Abera et al. (2016)

[50] Dessouky et al. (2017)

[6] Ammar, Crispo, and Tsudik (2020)

[164] Sun et al. (2020)

Although all of these results provide strong guarantees about the code being executed on an IoT device, they cannot prevent a compromised device from accessing its interaction resources. Therefore they provide only limited privacy guarantees. In summary, device attestation is an orthogonal research direction to ours, and we discuss in Sections 3.9.3 and 4.8.1 how device attestation, SA^{4}P, and Kimya can complement each other.

IoT recovery mechanisms A number of prior efforts studied how IoT devices can be efficiently recovered after compromise. For example, FIRE [149]) sends code update requests and blacklists devices when a successful recovery cannot be confirmed. Cider [186] presents a mechanism that guarantees that updates are installed within a bounded time frame. Lazarus [74] provides similar properties as Cider, though it targets more constrained devices and uses TrustZone on Cortex-M to eliminate some of Cider's hardware requirements. Verify&Revive [5] presents a pure-software-based device healing scheme, albeit at the cost of weaker availability guarantees.

[149] Seshadri et al. (2004)

[186] Xu et al. (2019)

[74] Huber et al. (2020)

[5] Ammar and Crispo (2020)

These works are similar to SA^{4}P in that they can be used to prevent an adversary from sampling sensors after a breach has been detected. However, they cannot be used to regulate sensor access during normal deployment operations. Although these works could be used to harden voice assistants, they would not be able to protect against unknown vulnerabilities in the voice assistant codebase.

Managing sensor access Brasser et al. use TrustZone on Cortex-A and remote memory writes to restrict peripheral access [29]. Unlike SA^{4}P, it targets feature-rich devices (e.g., smartphones or laptops) and does not provide fine-grained temporal control. Instead, the peripheral access policy can only be updated during check-in and check-out events. SeCloak [98] targets similar devices as Brasser et al. and also relies on TrustZone. However, it is designed to provide on-device control, i.e., SeCloak does not delegate access policy management to a remote server. AWare [132] is targeted towards mobile devices, and provides an operating-level service that binds user interactions with specific user interface elements to sensor access rights. EnTrust [133] further generalizes this to other types of input events, and to co-operating applications. Neither allows policy enforcement by a remote entity.

[29] Brasser et al. (2016)

[98] Lentz et al. (2018)

[132] Petracca et al. (2017)

[133] Petracca et al. (2019)

Although they provide strong guarantees, none of these works is designed to capture the semantics of always-standby sensors. Therefore, sensor access must be permanently granted for event detection to work. Moreover, these works were not designed for constrained environments.

Work on trusted I/O paths (e.g., SGXIO [180] and Wimpy kernels [195]) can be used to provide secure I/O access to TEEs. It is conceivable, but not verified, that these works can facilitate TEE-based $\mathcal{P}$EG implementations. Although TEEs provide isolation, they do no provide amnesia. It might be possible to implement Kimya-like logic using multiple TEEs (e.g., multiple SGX enclaves) and message passing channels, but future work would be needed to confirm this.

[180] Weiser and Werner (2017)

[195] Zhou, Yu, and Gligor (2014)

VERSA [121] uses a modified MCU to ensure that only attested and explicitly authorized routines can access sensors, based on remotely-issued authorization tokens. Although VERSA requires the remote verifier to issue a new token for each invocation of the sampling routine, it does not provide the same bounded-time guarantees as our work. Moreover, VERSA requires custom hardware and its reliance on attestation requires significantly closer integration between the remote verifier and the VERSA-enabled device.

[121] Nunes et al. (2022)

Auditing of sensor access Viola [112] provides guarantees that sensor notifications (e.g., LED indicators) are active when (and only when) a sensor is accessed. Ditio [111] securely logs sensor access for later auditing. Neither of these mechanisms is designed for always-standby sensors, and will mark an always-standby sensor as being continuously accessed. 6thSense [154] analyses sensor access patterns to detect malicious activity. However, it does not support always-standby sensors. Depending on the concrete sensor type, 6thSense would mark an always-standby sensor as always accessed, or assign it an access state that is independent from the event detection. Kimya can be used to augment 6thSense as discussed in Section 4.8.2. Neither Viola, Ditio, nor 6thSense allows for remote policy enforcement.

[112] Mirzamohammadi and Sani (2018)

[111] Mirzamohammadi et al. (2017)

[154] Sikder, Aksu, and Uluagac (2020)

Camera privacy Much research has focused on camera privacy, typically by performing video anonymization by obfuscating sensitive video regions [183]. Contributions in this area can be categorized based on when the obfuscation is performed: before [193, 135, 176], during [182]), or after [39, 184] capture

[183] Winkler and Rinner (2014)

[193] Zhang et al. (2014)

[135] Pittaluga and Koppal (2017)

[176] Wang et al. (2022)

[182] Winkler, Erdelyi, and Rinner (2014)

[39] Chattopadhyay and Boult (2007)

[184] Winkler and Rinner (2010)

time. The first two of these categories are of special interest, as, similar to our work, they perform access regulation before the CPU. Zhang et al. [193] and Pittaluga et al. [135] propose to place dynamic, privacy-preserving optics in front of image sensors. CamShield [176] takes a different approach by fully obscuring the view of the main image sensor, and instead expose it to a pre-anonymized digital video stream. TurstEYE.M4 [182] proposes a sensor unit with an integrated privacy filter, resulting in a pre-filtered video stream being received by the CPU.

[193] Zhang et al. (2014)
[135] Pittaluga and Koppal (2017)
[176] Wang et al. (2022)
[182] Winkler, Erdelyi, and Rinner (2014)

Contrary to our work, all of these works focus on image privacy and none of them have provisions for remote policy enforcement or event detection. CamShield [176] does perform region-of-interest detection, but requires all detection code to be part of the TCB.

Node-level IoT access management Many results consider device-level IoT access control. This includes both research papers (e.g., [59, 124, 56, 48, 117, 85]) and industry standards (e.g., [148, 43]). Unlike the present work, these techniques operate at the device or agent level. However, some of them, e.g., AoT's [117] full-lifecycle key management mechanism, could be used to instantiate a $\mathcal{P}$EG pairing module.

[59] Fedrecheski et al. (2022)
[124] Oh, Kim, and Cho (2019)
[56] Echeverria et al. (2019)
[48] De Vaere and Perrig (2019)
[117] Neto et al. (2016)
[85] Kim et al. (2017)
[148] Seitz et al. (2022)
[43] Connectivity Standards Alliance (2022)

Information flow tracking There is a large body of work on information-flow or *taint* tracking for mobile devices [57, 66, 73, 20]. FlowFence [60] provides taint tracking for IoT cloud platforms. These works rely either on static or dynamic code analysis. In the former case, they must be combined with software attestation or similar mechanisms. In the latter case, they are challenging to apply to constrained environments. Moreover, these mechanisms were not designed to support always-standby semantics, meaning that they do not provide guarantees on which historic data can be accessed when a trigger event occurs.

[117] Neto et al. (2016)
[57] Enck et al. (2014)
[66] Gordon et al. (2015)
[73] Hornyack et al. (2011)
[20] Arzt et al. (2014)
[60] Fernandes et al. (2016)

Skill behavior A number of works [189, 69, 151] analyze the behavior of third-party voice assistant *skills*. This is comparable to analyzing the behavior of Android or smart-home apps, and focuses on individual functionality add-ons, rather than on the

[189] Young et al. (2022)
[69] Guo et al. (2020)
[151] Shezan et al. (2020)

underlying system. Therefore, we believe these works to be synergetic to ours.

8.2 Automation Networks

Future-oriented network architectures and models The most visible proposal for a future-proof OT architecture is the NAMUR Open Architecture (NOA) [115]. NOA places a secondary *monitoring and optimization* network in parallel to the existing *core* automation infrastructure. Data is fed from the core network into the secondary network through data diodes, where it can be analyzed. Control commands from the secondary network are transferred back to the core network through a request verification gateway. Although NOA has the advantage that it leaves the existing automation network largely untouched, the functionality of the secondary network stays limited to a supporting role. This means that NOA does not address how to handle changes to the core of the automation architecture, e.g., the introduction of virtual automation functions or the increasing prevalence of highly-autonomous remotely controlled facilities. In fact, the NOA approach is largely complementary to Tableau, as Tableau can be used to structure and secure the monitoring and optimization network of NOA deployment.

[115] NAMUR (2020)

Another prominent standardization effort is the Reference Architectural Model for Industrie 4.0 (RAMI 4.0) [51], which was developed to support Industry 4.0 initiatives. However, RAMI 4.0 focuses on the representation and management of assets, and does not discuss network topologies.

[51] Deutsches Institut für Normung (2016)

IoT gateways A common method to secure IoT networks is by deploying a specialized IoT firewall, often called a *gateway*. Past proposals include DeadBolt [86], IoT Sentinel [109], IoTSec [190], and a design by Simpson et al. [156]. Concretely, Simpson et al. explore how to protect and isolate vulnerable devices and how to securely apply patches; IoT Sentinel and IoTSec focus on automated identification of compromised devices; and DeadBolt aims to enforce the application of good security practices on IoT devices while mediating traffic to non-complying devices.

[86] Ko and Mickens (2018)
[109] Miettinen et al. (2017)
[190] Yu et al. (2015)
[156] Simpson, Roesner, and Kohno (2017)

These proposals all consist of a security proxy that all network traffic must be routed through. While IoTSec envisions security proxies to be instantiated at various points in the network with overlay routing forcing traffic through them, the others effectively require a physical or overlay network with a star topology.

Apple recently started including a basic version of an IoT firewall in their *HomeKit* platform [11]. The vendors of HomeKit devices must supply a manifest file stating which connections their product is supposed to establish, and by default all other communication flows are blocked. Closely related to this, RFC 8520 [97] standardizes the description of the expected network behavior of devices, facilitating deny-by-default policies such as the one implemented by *HomeKit*.

[11] Apple (2020)

[97] Lear, Droms, and Romascanu (2019)

While suitable in some environments, approaches as discussed above are not suitable for industrial networks. That is, indirect routing introduces a single point of failure to the data plane, increased latencies, and increased network overhead. Moreover, aggregating and forcing network traffic along an indirect path directly counteracts the properties achieved by TSN. The proposals listed above also do not achieve the source-authentication properties of HOPPER.

Capability-based and Deny-by-Default networking HOPPER deploys capability-based networking in order to achieve per-device nano segmentation. While our work is (to the best of our knowledge) the first to apply this concept to the IoT setting, capability-based networking, and more generally, deny-by-default networking, have been considered in other settings.

Concretely, a first set of past work considers the use of capabilities for DoS protection on the Internet. Proposals in this space include a design by Anderson et al. [8], and SIFF [187]. As these proposals are designed to provide DoS protection, they only prevent volumetric attacks, but still allow low volume traffic flows to reach hosts. Hence, they do not prevent lateral movement.

[8] Anderson, Roscoe, and Wetherall (2004)
[187] Yaar, Perrig, and Song (2004)

Around the same time, Ballani et al. [21] proposed to propagate whitelists through the Internet using a mechanism similar to BGP, dropping all non-whitelisted traffic. While this scheme does prevent all unwanted traffic from reaching an end host,

[21] Ballani et al. (2005)

it still assumes an inter-domain setting, which is inapplicable to IoT networks. For example, it requires large amounts of reachability information to be transferred between forwarding elements.

A third set of work considers enterprise networks, most notably SANE [36] and Ethane [35]. SANE uses capabilities in the form of authenticated source routes, which are checked at each switch. However, because routes in (wireless) IoT networks can be non-predictable (see Section 7.2), it is not a suitable mechanism for IoT settings. Ethane uses a complementary approach: each switch stores a whitelist of permitted traffic. High control overhead and strong assumptions on the network architecture render this mechanism impractical in (constrained) IoT settings. Further, neither SANE nor Ethane provide packet authentication.

[36] Casado et al. (2006)
[35] Casado et al. (2007)

Fieldbus authentication There are a number of proposals that add authentication to industrial fieldbuses [140, 37, 169, 170]. However, these proposals do not provide least privilege (**HOPPER-1**) or isolation (**HOPPER-2**) and requires device to be pre-provisioned with per-flow keys.

[140] Radu and Garcia (2016)
[37] Castellanos et al. (2017)
[169] Tsang and Smith (2008)
[170] Van Herrewege, Singelee, and Verbauwhede (2011)

9 Conclusion

9.1 Summary

Today, automation devices have largely unrestricted access to their physical environment; they can actuate and sense at will. Hence, when an adversary manages to compromise an automation device, they can also compromise the physical world. To mitigate such attacks, we introduce SA^{4}P, an architectural framework to enforce access control policies between a device's software $\mathcal{R}$untime and its sensors and actuators. To achieve this, we designed the $\mathcal{P}$EG, a small, trusted, building block that physically guards sensing and actuation peripherals. The $\mathcal{P}$EG is explicitly designed to have a small footprint. This ensures (i) compatibility with a large number of device classes, and (ii) that the $\mathcal{P}$EG design is easy to verify. Each $\mathcal{P}$EG communicates with the centralized deployment manager, which can leverage its system-level view to make well-informed access decisions.

For cases in which the physical world must be continuously monitored, we introduced Kimya. Similarly to SA^{4}P, Kimya places access enforcement between a device's main software runtime, and its sensor peripherals. However, rather than delegating access decisions to a centralized entity, Kimya provides local applications with an isolated and amnestic execution environment dedicated to always-standby event detection. Code running inside the Kimya container has full access to the protected sensor at all times. Moreover, it is free to grant that same privilege to the entire software runtime at any point in time. However, whenever this happens, a user-auditable notification is generated, thus making the device's actions externally visible. Users, audit these notifications, and the device can be held accountable for its actions. Audits can be both formal or informal in character.

Comparing SA4P and KIMYA, both regulate access to a device's local sensors. Additionally, SA4P can also regulate access to actuators. However, whereas SA4P allows a remote, trusted entity to make access decisions, KIMYA allows access decisions to be made by a local, untrusted-but-audited entity. Moreover, KIMYA allows for sensor readings from the protected sensors to be used as a basis for access decisions.

On the network front, we reviewed how current defenses are coarse-grained and based on assumptions that are rapidly being invalidated. To demonstrate the feasibility of alternative approaches, we present TABLEAU. Concretely, TABLEAU shows how the use of modern technologies such as Mondrian and automated network verification allows for networks to be secured in a way that is more scalable than the current approaches without sacrificing security.

Finally, we present HOPPER, a nano-segmentation protocol for industrial networks. Contrary to TABLEAU and other perimeter-based approaches, HOPPER applies its protections uniformly throughout the network fabric. The result is that each device is placed in its own virtual network segment, consisting of only those network resources (forwarding elements and end-hosts) that it is legitimately supposed to interact with. HOPPER's nano-segments strictly limit both the exposure of individual devices, and an attacker's ability for lateral movement within network zones after an initial device has been compromised.

SA4P and KIMYA are primarily targeted towards consumer devices; TABLEAU and HOPPER were designed with an industrial setting in mind. Nonetheless, all contributions in this work can be applied to both the industrial and consumer settings. Moreover, all of the contributions of this thesis are composable: an always-standby device can be hardened using KIMYA and simultaneously be enrolled in a SA4P deployment. Its network can use TABLEAU macro zoning with HOPPER providing additional hardening within the zones.

9.2 Future Work

SA^{4}P *policies* SA^{4}P provides a distributed architecture for a centralized deployment monitor to enforce access restrictions between remote CPUs and their peripherals. However, this thesis does not discuss how such policies would look like. Although $\mathcal{P}$EG messages contain little context for access requests, the deployment manager's centralized nature means that it can have a rich, system-wide view of a SA^{4}P deployment. However, how the deployment manager should collect information from the deployment, and how policies should be specified remains an open question. We expect that adapting existing schemes for different settings (e.g., ContextIot [82] and CPS [106]) will prove to be efficient.

[82] *ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms*, Jia et al. (2017)
[106] *CPS: Stateful Policy Enforcement for Control System Device Usage*, McLaughlin (2013)

$\mathcal{P}$EG *enforcement modules* We presented three basic $\mathcal{P}$EG enforcement modules. We see opportunities for additional, purpose-optimized designs. Advanced enforcement modules could selectively reduce sensor resolution or limit access rates. Similarly, additional work to explore how to best protect complex, permanently-driven, actuators is needed.

KIMYA *on advanced architectures* We have presented a KIMYA implementation for a single core ARM Cortex-M platform. We see promise in running the KIMYA container on a separate processor core, thus decoupling the timing of the container from that of the main application. We believe that such an architecture would be more efficient for multi-stage detection pipelines. Additionally, it should be explored if KIMYA can be extended to include distributed or cloud-supported event detection. Potential research avenues include using secure multi-party computation and using Intel SGX enclaves as an extension of the on-device KIMYA container.

HOPPER *policy generation & TABLEAU policy blueprints* HOPPER requires each network flow to be explicitly whitelisted. To avoid high management overhead, systems to automatically synthesize HOPPER configurations should be developed. Ideally, such a system would be integrated together with a TABLEAU configuration tool. Further, for efficient TABLEAU policy configuration, a set of policy blueprints and design patterns should be developed.

TABLEAU *interaction with NOA* The NAMUR Open Architecture (NOA) is an alternative new architecture for industrial automation networks (see Section 8.2). We believe that NOA and TABLEAU are complementary technologies, as TABLEAU can be used to further segment both the NOA core and optimization networks. However, further work is needed to define which form interactions between the two NOA networks would take under TABLEAU's presence.

9.3 Closing Remarks

No matter whether we consider consumer IoT devices or industrial controllers, we observe that today's automation designs consider sensing, actuation, and (local) network access to be commodities: cheap and readily available. Access control is a nuisance and an afterthought. This work encourages designers to more closely consider the implications of sensing, actuation, and communication behavior; we emphasize that the mere ability to perform a particular function does not justify its implementation.

Bibliography

[1] Tigist Abera, N. Asokan, Lucas Davi, Jan-Erik Ekberg, Thomas Nyman, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. "C-FLAT: Control-Flow Attestation for Embedded Systems Software". In: *Proceedings of the ACM Conference on Computer and communications security (CCS)*. 2016. DOI: `10.1145/2976749.2978358`.

[2] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. "Peek-a-boo: I see your smart home activities, even encrypted!" In: *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. 2020. DOI: `10.1145/3395351.3399421`.

[3] R. Alexander, A. Brandt, J.P. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. RFC 6550. 2012. DOI: `10.17487/rfc6550`.

[4] Amazon.com. *Alexa Voice Service (AVS) Security Requirements*. 2022. URL: `https://developer.amazon.com/en-US/docs/alexa/alexa-voice-service/avs-security-reqs.html`.

[5] Mahmoud Ammar and Bruno Crispo. "Verify&Revive: Secure Detection and Recovery of Compromised Low-end Embedded Devices". In: *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*. 2020. DOI: `10.1145/3427228.3427253`.

[6] Mahmoud Ammar, Bruno Crispo, and Gene Tsudik. "SIMPLE: A Remote Attestation Approach for Resource-constrained IoT devices". In: *Proceedings of the ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*. 2020. DOI: 10.1109/iccps48487.2020.00036.

[7] Analog Devices. *The Fundamentals of Secure Boot and Secure Download: How to Protect Firmware and Data within Embedded Devices*. n.d. URL: https://www.analog.com/en/technical-articles/the-fundamentals-of-secure-boot-and-secure-download.html.

[8] T. Anderson, T. Roscoe, and D. Wetherall. "Preventing Internet denial-of-service with capabilities". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2004). DOI: 10.1145/972374.972382.

[9] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. "Understanding the Mirai Botnet". In: *Proceedings of the USENIX Security Symposium (USENIX Security)*. 2017.

[10] Apple. *Apple advances its privacy leadership with iOS 15, iPadOS 15, macOS Monterey, and watchOS 8*. 2021. URL: https://www.apple.com/newsroom/2021/06/apple-advances-its-privacy-leadership-with-ios-15-ipados-15-macos-monterey-and-watchos-8/.

[11] Apple. *HT210544: Use routers secured with HomeKit*. 2020.

[12] Noah Apthorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, and Nick Feamster. *Spying on the Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic*. 2017. arXiv: 1708.05044 [cs.CR].

[13] Ionut Arghire. *Nuki Smart Lock Vulnerabilities Allow Hackers to Open Doors*. SecurityWeek. 2022. URL: https://www.securityweek.com/nuki-smart-lock-vulnerabilities-allow-hackers-open-doors/.

[14] ARM Ltd. *ARM Cortex-M33 Devices Generic User Guide*. Version r1p0. 2020. URL: `https://developer.arm.com/documentation/100235/0100`.

[15] ARM Ltd. *ARM Security Technology: Building a Secure System using TrustZone Technology*. 2009.

[16] ARM Ltd. *ARM v8-M Security Extensions: Requirements on Development Tools*. Version 1.1. Nov. 2019. URL: `https://developer.arm.com/documentation/ecm0359818/11/`.

[17] ARM Ltd. *Armv8-M Architecture Reference Manual*. Version B.r. Dec. 2021. URL: `https://developer.arm.com/documentation/ddi0553/br`.

[18] ARM Ltd. *CMSIS DSP Software Library*. 2021. URL: `https://github.com/ARM-software/CMSIS_5/tree/develop/CMSIS/DSP`.

[19] ARM Ltd. *Introduction to the ARMv8-M architecture*. Version 1.0. Feb. 2017. URL: `https://developer.arm.com/documentation/100688/0200`.

[20] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. "FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Appls". In: *Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI)*. 2014. DOI: `10.1145/2594291.2594299`.

[21] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. "Off by Default!" In: *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*. 2005.

[22] Tony J. Bates and Yakov Rekhter. *Scalable Support for Multi-homed Multi-provider Connectivity*. RFC 2260. 1998. DOI: `10.17487/RFC2260`.

[23] BBC News. *Smart speaker recordings reviewed by humans*. 2019. URL: `https://www.bbc.com/news/technology-47893082`.

[24] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. *The SIMON and SPECK Families of Lightweight Block Ciphers*. Cryptology ePrint Archive, Report 2013/404. 2013. URL: https://eprint.iacr.org/2013/404.

[25] Frank Bentley, Chris Luvogt, Max Silverman, Rushani Wirasinghe, Brooke White, and Danielle Lottridge. "Understanding the Long-Term Use of Smart Speaker Assistants". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2018). DOI: 10.1145/3264901.

[26] Bluetooth Special Interest Group. *Mesh Profile Specification 1.0.1*. 2019.

[27] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. 2020. URL: http://toc.cryptobook.us/.

[28] Carsten Bormann, Mehmet Ersue, and Ari Keränen. *Terminology for Constrained-Node Networks*. RFC 7228. 2014. DOI: 10.17487/RFC7228.

[29] 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 ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2016. DOI: 10.1145/2906388.2906390.

[30] Fabian Bräunlein and Luise Frerichs. *Smart Spies: Alexa and Google Home expose users to vishing and eavesdropping*. Security Research Labs. 2019. URL: https://www.srlabs.de/bites/smart-spies.

[31] Manuel Bronstein. *Bringing you the next-generation Google Assistant*. Google. 2019. URL: https://blog.google/products/assistant/next-generation-google-assistant-io/.

[32] Bureau Veritas. *Construction photos of EUT H2C*. Aug. 2019. URL: https://fcc.report/FCC-ID/A4R-H2C/4401997.

[33] Cadence Design Systems. *Xtensa Instruction Set Architecture (ISA) Summary*. Apr. 2022. URL: `https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/ip/tensilica-ip/isa-summary.pdf`.

[34] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. "Multicast security: a taxonomy and some efficient constructions". In: *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*. 1999. DOI: `10.1109/infcom.1999.751457`.

[35] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. "Ethane: Taking Control of the Enterprise". In: *ACM SIGCOMM Computer Communication Review (CCR)* (2007). DOI: `10.1145/1282427.1282382`.

[36] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker. "SANE: A Protection Architecture for Enterprise Networks." In: *Proceedings of the USENIX Security Symposium (USENIX Security)*. 2006.

[37] J. H. Castellanos, D. Antonioli, N. O. Tippenhauer, and M. Ochoa. "Legacy-compliant data authentication for industrial control system traffic". In: *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS)*. 2017. DOI: `10.1007/978-3-319-61204-1_33`.

[38] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. "SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems". In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 2020. DOI: `10.1109/SP40000.2020.00061`.

[39] Ankur Chattopadhyay and T.E. Boult. "PrivacyCam: a Privacy Preserving Camera Using uCLinux on the Blackfin DSP". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2007. DOI: `10.1109/cvpr.2007.383413`.

[40] Laurent Chuat, Markus Legner, David Basin, David Hausheer, Samuel Hitz, Peter Müller, and Adrian Perrig. *The Complete Guide to SCION. From Design Principles to Formal Verification*. Springer International Publishing AG, 2022. DOI: 10.1007/978-3-031-05288-0.

[41] CISCO. *IT/OT Convergence. Moving Digital Manufacturing Forward*. 2018. URL: https://www.cisco.com/c/dam/en_us/solutions/industries/manufacturing/ITOT-convergence-whitepaper.pdf.

[42] Cisco Systems and Rockwell Automation. *Ethernet-to-the-Factory 1.2 Design and Implementation Guide*. 2008.

[43] Connectivity Standards Alliance. *Matter Specification Version 1.0*. 2022.

[44] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. "A Large-Scale Analysis of the Security of Embedded Firmwares". In: *Proceedings of the USENIX Security Symposium (USENIX Security)*. 2014.

[45] Anupam Das, Martin Degeling, Daniel Smullen, and Norman Sadeh. "Personalized Privacy Assistants for the Internet of Things: Providing Users with Notice and Choice". In: *IEEE Pervasive Computing* (2018). DOI: 10.1109/mprv.2018.03367733.

[46] Piet De Vaere, Claude Hähni, Franco Monti, and Adrian Perrig. "Tableau: Future-Proof Zoning for OT Networks". In: *Proceedings of the Internation Conference Critical Information Infrastructures Security (CRITIS)*. 2021. DOI: 10.1007/978-3-030-93200-8_12.

[47] Piet De Vaere and Adrian Perrig. "Hey Kimya, Is My Smart Speaker Spying on Me? Taking Control of Sensor Privacy Through Isolation and Amnesia". In: *Proceedings of the USENIX Security Symposium (USENIX Security)*. USENIX Association, 2023. URL: /publications/papers/devaere2023kimya.pdf.

[48] Piet De Vaere and Adrian Perrig. “Liam: An Architectural Framework for Decentralized IoT Networks”. In: *Proceedings of the IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. 2019. DOI: `10.1109/mass.2019.00056`.

[49] Piet De Vaere, Andrea Tulimiero, and Adrian Perrig. “Hopper: Per-Device Nano Segmentation for the Industrial IoT”. In: *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIACCS)*. 2022. DOI: `10.1145/3488932.3501277`.

[50] Ghada Dessouky, Shaza Zeitouni, Thomas Nyman, Andrew Paverd, Lucas Davi, Patrick Koeberl, N. Asokan, and Ahmad-Reza Sadeghi. “LO-FAT: Low-Overhead Control Flow ATtestation in Hardware”. In: *Proceedings of the ACM Annual Design Automation Conference (DAC)*. 2017. DOI: `10.1145/3061639.3062276`.

[51] Deutsches Institut für Normung. *DIN SPEC 91345:2016-04: Reference Architecture Model Industrie 4.0 (RAMI4.0)*. Technical Standard. 2016.

[52] Brian Dorey. *Echo Dot 3rd Gen Digging Deeper*. Amazon. 2019. URL: `https://www.briandorey.com/post/echo-dot-3rd-gen-digging-deeper`.

[53] Dragos. *EKANS Ransomware and ICS Operations*. 2020.

[54] Dragos. *TRISIS: Analyzing Safety System Targeting Malware*. 2017.

[55] A. Dunkels. *Design and Implementation of the lwIP TCP/IP Stack*. Tech. rep. Swedish Institute of Computer Science, 2001.

[56] Sebastian Echeverria, Grace A. Lewis, Dan Klinedinst, and Ludwig Seitz. “Authentication and Authorization for IoT Devices in Disadvantaged Environments”. In: *Proceedings of the World Forum on Internet of Things (WF-IoT)*. 2019. DOI: `10.1109/wf-iot.2019.8767192`.

[57] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, 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: *ACM Transactions on Computer Systems* (2014). DOI: 10.1145/2619091.

[58] Haytham M. Fayek. *Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between*. 2016. URL: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html.

[59] Geovane Fedrecheski, Laisa Caroline Costa De Biase, Pablo C. Calcina-Ccori, Roseli de Deus Lopes, and Marcelo Knorich Zuffo. "SmartABAC: Enabling Constrained IoT Devices to Make Complex Policy-Based Access Control Decisions". In: *IEEE Internet of Things Journal* (2022). DOI: 10.1109/jiot.2021.3110142.

[60] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. "FlowFence: Practical Data Protection for Emerging IoT Application Frameworks". In: *Proceedings of the USENIX Security Symposium (USENIX Security)*. 2016.

[61] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. "Efficient network flooding and time synchronization with Glossy". In: *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 2011.

[62] Jean-Marie Flaus. *Cybersecurity of Industrial Systems*. Wiley, 2019. ISBN: 9781786304216.

[63] Dennis Giese and Guevara Noubir. "Amazon echo dot or the reverberating secrets of IoT devices". In: *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. 2021. DOI: 10.1145/3448300.3467820.

[64] E. Gilman and D. Barth. *Zero Trust Networks*. O'Reilly Media, 2017. ISBN: 9781491962190.

[65] Google. *GoPacket*. 2021. URL: `https://github.com/google/gopacket`.

[66] Michael I. Gordon, Deokhwan Kim, Jeff Perkins, Limei Gilham, Nguyen Nguyen, and Martin Rinard. "Information-Flow Analysis of Android Applications in DroidSafe". In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2015. DOI: `10.14722/ndss.2015.23089`.

[67] David Greenfield. *Is the Purdue Model Still Relevant?* Automation World. 2020. URL: `https://www.automationworld.com/factory/iiot/article/21132891/is-the-purdue-model-still-relevant`.

[68] Eileen Guo. *Roomba testers feel misled after intimate images ended up on Facebook*. MIT Technology Review. 2023. URL: `https://www.technologyreview.com/2023/01/10/1066500/`.

[69] Zhixiu Guo, Zijin Lin, Pan Li, and Kai Chen. "SkillExplorer: Understanding the Behavior of Skills in Large Scale". In: *Proceedings of the USENIX Security Symposium (USENIX Security)*. 2020.

[70] Jun Han, Abhishek Jain, Mark Luk, and Adrian Perrig. "Don't Sweat Your Privacy: Using Humidity to Detect Human Presence". In: *Proceedings of the International Workshop on Privacy in UbiComp (UbiPriv)*. 2007.

[71] Brad Hegrat, Joel Langill, and Dale Peterson. *S4x19 Panel Discussion: Is The Purdue Model Dead?* 2019. URL: `https://s4xevents.com/past-events-2/s4x19/`.

[72] Richard Leslie Hills. *Power from Wind*. Cambridge University Press, 1996. ISBN: 9780521566865.

[73] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. "These aren't the droids you're looking for". In: *Proceedings of the ACM Conference on Computer and communications security (CCS)*. 2011. DOI: `10.1145/2046707.2046780`.

[74] Manuel Huber, Stefan Hristozov, Simon Ott, Vasil Sarafov, and Marcus Peinado. "The Lazarus Effect: Healing Compromised Devices in the Internet of Small Things". In: *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIACCS)*. 2020. DOI: 10.1145/3320269.3384723.

[75] C. Hung and W. Hsu. "Power Consumption and Calculation Requirement Analysis of AES for WSN IoT". In: *Sensors* (2018). DOI: 10.3390/s18061675.

[76] IEEE 802.1. *Time-Sensitive Networking (TSN) Task Group*. 2020. URL: https://1.ieee802.org/tsn/.

[77] IEEE 802.1 and IEC SC65C/WG18. *IEC/IEEE 60802 TSN Profile for Industrial Automation (Draft D1.2)*. 2020. URL: https://1.ieee802.org/tsn/iec-ieee-60802/.

[78] IEEE Computer Society. *IEEE Standard for Ethernet - Amendment 5: Physical Layer Specifications and Management Parameters for 10 Mb/s Operation and Associated Power Delivery over a Single Balanced Pair of Conductors*. IEEE Std 802.3cg-2019. 2020. DOI: 10.1109/IEEESTD.2020.8982251.

[79] IEEE Computer Society. *IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability*. IEEE 802.1CB-2017. 2017. DOI: 10.1109/IEEESTD.2017.8091139.

[80] International Electrotechnical Commission. *IEC 62443 Standard Series: Industrial communication networks - IT security for networks and systems*. Technical Standard. 2020.

[81] International Electrotechnical Commission. *IEC 62443-3-2:2020 Security for industrial automation and control systems - Part 3-2: Security risk assessment for system design*. Technical Standard. 2020.

[82] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Z. Morley Mao, and Atul Prakash. "ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms". In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2017. DOI: 10.14722/ndss.2017.23051.

[83] Ted Karczewski. *Cloud-Based Wake Word Verification Improves "Alexa" Wake Word Accuracy on Your AVS Products*. Amazon. 2017. URL: https://developer.amazon.com/blogs/alexa/post/b136b3e7-0ba8-4589-aaf9-2a037fc4e9c9/cloud-based-wake-word-verification-improves-alexa-wake-word-accuracy-on-your-avs-products.

[84] Jim Karki. *Application Report: Understanding Operational Amplifier Specifications*. 2021.

[85] Jun Young Kim, Wen Hu, Dilip Sarkar, and Sanjay Jha. "ESIoT: enabling secure management of the internet of things". In: *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. 2017. DOI: 10.1145/3098243.3098252.

[86] R. Ko and J. Mickens. "DeadBolt: Securing IoT Deployments". In: *Proceedings of the ACM/IRTF Applied Networking Research Workshop (ANRW)*. 2018. DOI: 10.1145/3232755.3232774.

[87] Sinclair Koelemij. *The Purdue Reference Model outdated or up-to-date?* 2020. URL: https://otcybersecurity.blog/2020/06/08/the-purdue-reference-model-outdated-or-up-to-date/.

[88] kokke. *Tiny AES*. 2021. URL: https://github.com/kokke/tiny-AES-c.

[89] M. Krotofil, A. A. Cárdenas, B. Manning, and J. Larsen. "CPS: Driving Cyber-Physical Systems to Unsafe Operating Conditions by Timing DoS Attacks on Sensor Signals". In: *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*. 2014. DOI: 10.1145/2664243.2664290.

[90] Deepak Kumar, Riccardo Paccagnella, Paul Murley, Eric Hennenfent, Joshua Mason, Adam Bates, and Michael Bailey. "Skill Squatting Attacks on Amazon Alexa". In: *Proceedings of the USENIX Security Symposium (USENIX Security)*. 2018.

[91] Rajath Kumar, Mike Rodehorst, Joe Wang, Jiacheng Gu, and Brian Kulis. “Building a Robust Word-Level Wake-word Verification Network”. In: *Proceedings of Interspeech*. 2020. DOI: 10.21437/Interspeech.2020-2018.

[92] Jonghoon Kwon, Claude Hähni, Patrick Bamert, and Adrian Perrig. “Mondrian: Comprehensive Inter-domain Network Zoning Architecture”. In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2021. DOI: 10.14722/ndss.2021.24378.

[93] Davy Landman. *compact25519: A compact portable X25519 + Ed25519 implementation*. Feb. 2022. URL: https://github.com/DavyLandman/compact25519.

[94] Marc Langheinrich. “A Privacy Awareness System for Ubiquitous Computing Environments”. In: *Proceedings of the ACM Conference on Ubiquitous Computing (UbiComp)*. 2002. DOI: 10.1007/3-540-45809-3_19.

[95] Ralph Langner. “Stuxnet: Dissecting a Cyberwarfare Weapon”. In: *IEEE Security & Privacy* (2011). DOI: 10.1109/msp.2011.67.

[96] Josephine Lau, Benjamin Zimmerman, and Florian Schaub. “Alexa, Are You Listening? Privacy Perceptions, Concerns and Privacy-seeking Behavious with Smart Speakers”. In: *Proceedings of the ACM on Human-Computer Interaction* (2018). DOI: 10.1145/3274371.

[97] E. Lear, R. Droms, and D. Romascanu. *Manufacturer Usage Description Specification*. RFC 8520. 2019. DOI: 10.17487/rfc8520.

[98] Matthew Lentz, Rijurekha Sen, Peter Druschel, and Bobby Bhattacharjee. “SeCloak: ARM trustzone-based mobile peripheral control”. In: *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2018. DOI: 10.1145/3210240.3210334.

[99] Yahui Li, Xia Yin, Zhiliang Wang, Jiangyuan Yao, Xingang Shi, Jianping Wu, Han Zhang, and Qing Wang. “A Survey on Network Verification and Testing With Formal

Methods: Approaches and Challenges". In: *IEEE Communications Surveys & Tutorials* (2019). DOI: 10.1109/comst.2018.2868050.

[100] L. Lo Bello and W. Steiner. "A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems". In: *Proceedings of the IEEE* (2019). DOI: 10.1109/jproc.2019.2905334.

[101] Lan Luo, Yue Zhang, Clayton White, Brandon Keating, Bryan Pearson, Xinhui Shao, Zhen Ling, Haofei Yu, Cliff Zou, and Xinwen Fu. "On Security of TrustZone-M-Based IoT Systems". In: *IEEE Internet of Things Journal* (2022). DOI: 10.1109/JIOT.2022.3144405.

[102] Dorian Lynskey. *'Alexa, are you invading my privacy?' – the dark side of our voice assistants*. The Guardian. 2019. URL: https://www.theguardian.com/technology/2019/oct/09/alexa-are-you-invading-my-privacy-the-dark-side-of-our-voice-assistants.

[103] Zongheng Ma, Saeed Mirzamohammadi, and Ardalan Amiri Sani. "Understanding Sensor Notifications on Mobile Devices". In: *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 2017. DOI: 10.1145/3032970.3032978.

[104] Lydia Manikonda, Aditya Deotale, and Subbarao Kambhampati. "What's up with Privacy? User Preferences and Privacy Concerns in Intelligent Personal Assistants". In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society (AEIS)*. 2018. DOI: 10.1145/3278721.3278773.

[105] A. Matsumoto, T. Fujisaki, R. Hiromi, and K. Kanayama. *Problem Statement for Default Address Selection in Multi-Prefix Environments: Operational Issues of RFC 3484 Default Rules*. RFC 5220. 2008. DOI: 10.17487/RFC5220.

[106] Stephen McLaughlin. "CPS: Stateful Policy Enforcement for Control System Device Usage". In: *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*. 2013.

[107] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. "The TAMARIN Prover for the Symbolic Analysis of Security Protocols". In: *Computer Aided Verification*. 2013. DOI: 10.1007/978-3-642-39799-8_48.

[108] Nicole Meng, Dilara Keküllüoğlu, and Kami Vaniea. "Owning and Sharing: Privacy Perceptions of Smart Speaker Users". In: *Proceedings of the ACM on Human-Computer Interaction* (2021). DOI: 10.1145/3449119.

[109] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi, and S. Tarkoma. "IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT". In: *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*. 2017. DOI: 10.1109/icdcs.2017.283.

[110] Dan Miklovic. *IIoT Will Change Our View of CIM; The Purdue Model Is Becoming Dated*. LNS Research. 2015. URL: https://blog.lnsresearch.com/iiot-will-change-our-view-of-cim-the-purdue-model-is-becoming-dated.

[111] Saeed Mirzamohammadi, Justin A. Chen, Ardalan Amiri Sani, Sharad Mehrotra, and Gene Tsudik. "Ditio: Trustworthy Auditing of Sensor Activities in Mobile & IoT Devices". In: *Proceedings of the ACM Conference on Embedded Network Sensor Systems (SenSys)*. 2017. DOI: 10.1145/3131672.3131688.

[112] Saeed Mirzamohammadi and Ardalan Amiri Sani. "Viola: Trustworthy Sensor Notifications for Enhanced Privacy on Mobile Systems". In: *IEEE Transactions on Mobile Computing* (2018). DOI: 10.1109/tmc.2018.2812706.

[113] Mission Secure. *Is the Purdue Model Relevant in a World of Industrial Internet of Things (IIoT) and Cloud Services?* Mission Secure. 2021. URL: https://www.missionsecure.com/blog/purdue-model-relevance-in-industrial-internet-of-things-iiot-cloud.

[114] A. Mosnier. *SHA-2 implementation*. 2019. URL: https://github.com/amosnier/sha-2.

[115] NAMUR. *NAMUR Recommendation NE 175: NAMUR Open Architecture – NOA Concept*. Technical Standard. 2020.

[116] Sashank Narain, Triet D. Vo-Huu, Kenneth Block, and Guevara Noubir. "Inferring User Routes and Locations Using Zero-Permission Mobile Sensors". In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 2016. DOI: 10.1109/sp.2016.31.

[117] Antonio L. Maia Neto, Artur L. F. Souza, Italo Cunha, Michele Nogueira, Ivan Oliveira Nunes, Leonardo Cotta, Nicolas Gentille, Antonio A. F. Loureiro, Diego F. Aranha, Harsh Kupwade Patil, and Leonardo B. Oliveira. "AoT: Authentication and Access Control for the Entire IoT Device Life-Cycle". In: *Proceedings of the Conference on Embedded Network Sensor Systems (CD-ROM)*. 2016. DOI: 10.1145/2994551.2994555.

[118] Jakob Nielsen. *Usability Engineering*. Academic Press, 1993. ISBN: 9780125184069.

[119] Job Noorman, Jo Van Bulck, Jan Tobias Mühlberg, Frank Piessens, Pieter Maene, Bart Preneel, Ingrid Verbauwhede, Johannes Götzfried, Tilo Müller, and Felix Freiling. "Sancus 2.0: A Low-Cost Security Architecture for IoT Devices". In: *ACM Transactions on Privacy and Security* (2017). DOI: 10.1145/3079763.

[120] NSA. *Defense in depth: A practical strategy for achieving Information Assurance in today's highly networked environments*. 2012.

[121] Ivan De Oliveira Nunes, Seoyeon Hwang, Sashidhar Jakkamsetti, and Gene Tsudik. "Privacy-from-Birth: Protecting Sensed Data from Malicious Sensors with VERSA". In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 2022. DOI: 10.1109/sp46214.2022.9833737.

[122] NXP Semiconductors. *UM10204: I2C-bus specification and user manual*. 2021.

[123] J. O'Connor, J.P. Aumasson, S. Neves, and Z. Wilcox-O'Hearn. *BLAKE3: one function, fast everywhere*. 2020. URL: `https://blake3.io`.

[124] Se-Ra Oh, Young-Gab Kim, and Sanghyun Cho. "An Interoperable Access Control Framework for Diverse IoT Platforms Based on OAuth and Role". In: *MPDI Sensors* (2019). DOI: `10.3390/s19081884`.

[125] ON Semiconductor. *2N7000G: Small Signal MOSFET 200 mApms, 60 Volts*. ON Semiconductor, 2011.

[126] ON Semiconductor. *MC74VHC541: Octal Bus Buffer*. 2014.

[127] OPC Foundation. *OPC UA Online Reference, Section 7.3.2: OPC UA UDP*. 2021. URL: `https://reference.opcfoundation.org/v104/Core/docs/Part14/7.3.2/`.

[128] OpenSSL Software Foundation. *OpenSSL*. 2021. URL: `https://www.openssl.org/`.

[129] George Orwell. *Nineteen Eighty-Four*. Secker and Warburg, 1949. ISBN: 9780451524935.

[130] Paloalto Networks. *2020 Unit 42 IoT Threat Report*. Unit 42. 2020. URL: `https://unit42.paloaltonetworks.com/iot-threat-report-2020/`.

[131] Trevor Perrin. *The Noise Protocol Framework*. Revision 34. July 2018. URL: `https://noiseprotocol.org/noise.pdf`.

[132] Giuseppe Petracca, Ahmad-Atamli Reineh, Yuqiong Sun, Jens Grossklags, and Trent Jaeger. "AWare: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings". In: *Proceedings of the USENIX Security Symposium (USENIX Security)*. 2017.

[133] Giuseppe Petracca, Yuqiong Sun, Ahmad-Atamli Reineh, Patrick McDaniel, Jens Grossklags, and Trent Jaeger. "EnTrust: Regulating Sensor Access by Cooperating Programs via Delegation Graphs". In: *Proceedings of the USENIX Security Symposium (USENIX Security)*. 2019.

[134] Sandro Pinto and Nuno Santos. "Demystifying Arm TrustZone: A Comprehensive Survey". In: *ACM Computing Surveys* (2019). DOI: `10.1145/3291047`.

[135] Francesco Pittaluga and Sanjeev Jagannatha Koppal. "Pre-Capture Privacy for Small Vision Sensors". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017). DOI: `10.1109/tpami.2016.2637354`.

[136] Rebecca S. Portnoff, Linda N. Lee, Serge Egelman, Pratyush Mishra, Derek Leung, and David Wagner. "Somebody's Watching Me?: Assessing the Effectiveness of Webcam Indicator Lights". In: *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. 2015. DOI: `10.1145/2702123.2702164`.

[137] Jing Qiu, Zhihong Tian, Chunlai Du, Qi Zuo, Shen Su, and Binxing Fang. "A Survey on Access Control in the Age of Internet of Things". In: *IEEE Internet of Things Journal* (2020). DOI: `10.1109/jiot.2020.2969326`.

[138] Qualcomm Technologies. *Snapdragon 8 Gen 1 Mobile Platform*. 2021. URL: `https://www.qualcomm.com/products/snapdragon-8-gen-1-mobile-platform`.

[139] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. M. Zanchettin, and S. Zanero. "An Experimental Security Analysis of an Industrial Robot Controller". In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 2017. DOI: `10.1109/sp.2017.20`.

[140] A. Radu and F.D. Garcia. "LeiA: A lightweight authentication protocol for CAN". In: *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*. 2016. DOI: `10.1007/978-3-319-45741-3_15`.

[141] E. Ronen, A. Shamir, A. Weingarten, and C. O'Flynn. "IoT Goes Nuclear: Creating a ZigBee Chain Reaction". In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 2017. DOI: `10.1109/sp.2017.14`.

[142] R. Safavi-Naini and H. Wang. "New results on multi-receiver authentication codes". In: *Lecture Notes in Computer Science*. 1998.

[143] N. Sastry and D. Wagner. “Security Considerations for IEEE 802.15.4 Networks”. In: *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. 2004. DOI: 10.1145/1023646.1023654.

[144] T. Sauter, S. Soucek, W. Kastner, and D. Dietrich. “The Evolution of Factory and Building Automation”. In: *IEEE Industrial Electronics Magazine* (2011). DOI: 10.1109/mie.2011.942175.

[145] Florian Schaub, Rebecca Balebako, and Lorrie Faith Cranor. “Designing Effective Privacy Notices and Controls”. In: *IEEE Internet Computing* (2017). DOI: 10.1109/mic.2017.265102930.

[146] Florian Schaub, Rebecca Balebako, Adam L. Durity, and Lorrie Faith Cranor. “A Design Space for Effective Privacy Notices”. In: *Proceedings of the USENIX Security Symposium (USENIX Security)*. 2015.

[147] Sandra Scott-Hayward, Sriram Natarajan, and Sakir Sezer. “A Survey of Security in Software Defined Networks”. In: *IEEE Communications Surveys & Tutorials* (2016). DOI: 10.1109/comst.2015.2453114.

[148] L. Seitz, G. Selander, E. Wahlstroem, S. Erdtman, and H. Tschofenig. *Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)*. RFC 9200. 2022. DOI: 10.17487/rfc9200.

[149] Arvid Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. *Using FIRE and ICE for Detectin and Recovering Compromised Nodes in Sensor Networks*. Tech. rep. Carnegie Mellon University, 2004.

[150] Ehab S Al-Shaer and Hazem H Hamed. “Modeling and Management of Firewall Policies”. In: *IEEE Transactions on Network and Service Management* (2004). DOI: 10.1109/tnsm.2004.4623689.

[151] Faysal Hossain Shezan, Hang Hu, Gang Wang, and Yuan Tian. “VerHealth”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2020). DOI: 10.1145/3432233.

[152] Matti Siekkinen, Markus Hiienkari, Jukka K. Nurminen, and Johanna Nieminen. "How low energy is bluetooth low energy? Comparative measurements with ZigBee/802.15.4". In: *Proceedings of the IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. 2012. DOI: 10.1109/WCNCW.2012.6215496.

[153] Amit Kumar Sikder, Hidayet Aksu, and A. Selcuk Uluagac. "6thSense: A Context-aware Sensor-based Attack Detector for Smart Devices". In: *Proceedings of the USENIX Security Symposium (USENIX Security)*. 2017.

[154] Amit Kumar Sikder, Hidayet Aksu, and A. Selcuk Uluagac. "A Context-Aware Framework for Detecting Sensor-Based Threats on Smart Devices". In: *IEEE Transactions on Mobile Computing* (2020). DOI: 10.1109/TMC.2019.2893253.

[155] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, and A. Selcuk Uluagac. "A Survey on Sensor-Based Threats and Attacks to Smart Devices and Applications". In: *IEEE Communications Surveys & Tutorials* (2021). DOI: 10.1109/COMST.2021.3064507.

[156] Anna Kornfeld Simpson, Franziska Roesner, and Tadayoshi Kohno. "Securing vulnerable home IoT devices with an in-hub security manager". In: *Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 2017. DOI: 10.1109/percomw.2017.7917622.

[157] Siri Team. *Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant*. Apple. 2017. URL: https://machinelearning.apple.com/research/hey-siri.

[158] Dimitrios Slamaris. *Embedded Systems Security and TrustZone*. 2022. URL: https://embeddedsecurity.io.

[159] Frank Stajano and Ross Anderson. "The resurrecting duckling: Security issues for ad-hoc wireless networks". In: *Proceedings of the International workshop on security protocols*. 1999. DOI: 10.1007/10720107_24.

[160] S. S. Stevens, J. Volkmann, and E. B. Newman. "A Scale for the Measurement of the Psychological Magnitude Pitch". In: *The Journal of the Acoustical Society of America* (1937). DOI: 10.1121/1.1915893.

[161] STMicroelectronics. *DB3788: X-CUBE-AI Databrief*. Sept. 2021. URL: https://www.st.com/en/embedded-software/x-cube-ai.html.

[162] STMicroelectronics. *Reference manual: STM32L552xx and STM32L562xx advanced ARM-based 32-bit MCUs*. RM0438 Rev 6. May 2020.

[163] STMicroelectronics. *Ultra-low-power ARM Cortex-M33 32-bit MCU with TrustZone*. DS12737 Rev 6. Oct. 2020.

[164] Zhichuang Sun, Bo Feng, Long Lu, and Somesh Jha. "OAT: Attesting Operation Integrity of Embedded Devices". In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 2020. DOI: 10.1109/sp40000.2020.00042.

[165] D. Temple-Raston. *A 'Worst Nightmare' Cyberattack: The Untold Story Of The SolarWinds Hack*. National Public Radio. 2021. URL: https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack.

[166] Texas Instruments. *LMV3xx Low-Voltage Rail-to-Rail Output Operational Amplifier*. 1999.

[167] Texas Instruments. *TLV320ADC3101 Low-Power Stereo ADC With Embedded miniDSP for Wireless Handsets and Portable Audio*. Aug. 2015.

[168] The Linux Foundation. *Data Plane Development Kit*. 2021. URL: https://dpdk.org/.

[169] P. P. Tsang and S. W. Smith. "YASIR: A low-latency, high-integrity security retrofit for legacy SCADA systems". In: *Proceedings of the IFIP International Information Security Conference*. 2008.

[170] A. Van Herrewege, D. Singelee, and I. Verbauwhede. “CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus”. In: *Procedings of the ECRYPT Workshop on Lightweight Cryptography (ECRYPT)*. 2011.

[171] Various Authors. *Google Nest Mini (H2C)*. 2022. URL: `https://wikidevi.wi-cat.ru/Google_Nest_Mini_(H2C)`.

[172] VDI/VDE Gesellschaft Mess- und Automatisierugnstechnik. *Cypber-Physical Systems: Chancen und Nutzen aus Sicht der Automation*. 2013.

[173] Voicebot Research. *U.S. Smart Speaker Consumer Adoption Report 2021 (Executive Summary)*. Jan. 2021. URL: `https://research.voicebot.ai/report-list/smart-speaker-consumer-adoption-report-2021/`.

[174] Yinxin Wan, Kuai Xu, Feng Wang, and Guoliang Xue. “IoTAthena: Unveiling IoT Device Activities From Network Traffic”. In: *IEEE Transactions on Wireless Communications* (2022). DOI: `10.1109/twc.2021.3098608`.

[175] Anran Wang, Dan Nguyen, Arun R. Sridhar, and Shyamnath Gollakota. “Using smart speakers to contactlessly monitor heart rhythms”. In: *Communications Biology* (2021). DOI: `10.1038/s42003-021-01824-9`.

[176] Zhiwei Wang, Yihui Yan, Yueli Yan, Huangxun Chen, and Zhice Yang. “CamShield: Securing Smart Cameras through Physical Replication and Isolation”. In: *Proceedings of the USENIX Security Symposium (USENIX Security)*. 2022.

[177] Pete Warden. *Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition*. 2018. arXiv: `1804.03209 [cs.CL]`.

[178] Andrew Waterman, Krste Asanović, and Joan Hauser. *The RISC-V Instruction Set Manual: Volume II: Privileged Architecture*. 2021.

[179] Stacy Wegner, Daniel Yang, and Kim Waterman. *Apple HomePod Teardown and Cost Comparison*. TechInsights. 2018. URL: `https://www.techinsights.com/blog/apple-homepod-teardown-and-cost-comparison`.

[180] Samuel Weiser and Mario Werner. "SGXIO: Generic Trusted I/O Path for Intel SGX". In: *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY)*. 2017. DOI: `10.1145/3029806.3029822`.

[181] Theodore J. Williams. *A Reference Model for Computer Integrated Manufacturing (CIM)*. Instrument Society of America, 1989. ISBN: 1556172257.

[182] Thomas Winkler, Adam Erdelyi, and Bernhard Rinner. "TrustEYE.M4: Protecting the sensor — Not the camera". In: *Proceedings of the IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. 2014. DOI: `10.1109/avss.2014.6918661`.

[183] Thomas Winkler and Bernhard Rinner. "Security and Privacy Protection in Visual Sensor Networks". In: *ACM Computing Surveys* (2014). DOI: `10.1145/2545883`.

[184] Thomas Winkler and Bernhard Rinner. "TrustCAM: Security and Privacy-Protection for an Embedded Smart Camera Based on Trusted Computing". In: *Proceedings of the IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. 2010. DOI: `10.1109/avss.2010.38`.

[185] M. Wollschlaeger, T. Sauter, and J. Jasperneite. "The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0". In: *IEEE Industrial Electronics Magazine* (2017). DOI: `10.1109/mie.2017.2649104`.

[186] Meng Xu, Manuel Huber, Zhichuang Sun, Paul England, Marcus Peinado, Sangho Lee, Andrey Marochko, Dennis Mattoon, Rob Spiger, and Stefan Thom. "Dominance as a New Trusted Computing Primitive for the Internet of Things". In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 2019. DOI: `10.1109/sp.2019.00084`.

[187] A. Yaar, A. Perrig, and D. Song. "SIFF: a stateless internet flow filter to mitigate DDoS flooding attacks". In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 2004. DOI: 10.1109/secpri.2004.1301320.

[188] Joseph Yiu. *White Paper: Introduction to the ARM Cortex-M55 Processor*. Feb. 2020.

[189] Jeffrey Young, Song Liao, Long Cheng, Hongxin Hu, and Huixing Deng. "SkillDetective: Automated Policy-Violation Detection of Voice Assistant Applications in the Wild". In: *Proceedings of the USENIX Security Symposium (USENIX Security)*. 2022.

[190] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu. "Handling a Trillion (Unfixable) Flaws on a Billion Devices: Rethinking Network Security for the Internet-of-Things". In: *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*. 2015. DOI: 10.1145/2834050.2834095.

[191] Kim Zetter. *Inside the Cunning, Unprecedented Hack of Ukraine's Power Grid*. Wired. 2016. URL: https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/.

[192] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. *Hello Edge: Keyword Spotting on Microcontrollers*. 2018. arXiv: 1711.07128 [cs.SD].

[193] Yupeng Zhang, Yuheng Lu, Hajime Nagahara, and Rin-ichiro Taniguchi. "Anonymous Camera for Privacy Protection". In: *Proceedings on the International Conference on Pattern Recognition (ICPR)*. 2014. DOI: 10.1109/icpr.2014.715.

[194] Binbin Zhao, Shouling Ji, Jiacheng Xu, Yuan Tian, Qiuyang Wei, Qinying Wang, Chenyang Lyu, Xuhong Zhang, Changting Lin, Jingzheng Wu, and Raheem Beyah. *One Bad Apple Spoils the Barrel: Understanding the Security Risks Introduced by Third-Party Components in IoT Firmware*. 2022. arXiv: 2212.13716 [cs.CR].

[195] Zongwei Zhou, Miao Yu, and Virgil D. Gligor. "Dancing with Giants: Wimpy Kernels for On-Demand Isolated I/O". In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 2014. DOI: 10.1109/SP.2014.27.