Fog Mediated Security in the Internet of Things

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Engineering and Public Policy

Emmanuel K Owusu

B.S., Computer Engineering, Iowa State University of Science and Technology

Carnegie Mellon University
Pittsburgh, PA

December, 2019

Acknowledgements

This research was supported in part by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 from the Army Research Office, by a gift from Robert Bosch LLC, by participation in the Northrop Grumman Cybersecurity Research Consortium, and by an award from the Cisco IoT Security Grand Challenge. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ARO, Robert Bosch LLC, Northrop Grumman, Cisco, CMU, CyLab, or the U.S. Government or any of its agencies.

Many people have lent a helping hand in this project, both directly and indirectly, and to each of them, I would like to acknowledge my debt. In recognition of being a consistent source of insight and encouragement — and an exceptional role model of what it means to be a teacher that goes above and beyond by taking a personal interest in the aspirations of all his students — I am grateful to my doctoral committee chair, Patrick Tague. In recognition of their excellent support and valuable suggestions for completing this work, I am grateful to my committee members: Douglas Sicker, Nicolas Christin, and Jorge Guajardo.

Additionally, many people have read earlier iterations of the manuscript for this book and have offered helpful suggestions for improving its substance and style. I am very thankful to my teachers and colleagues who have made the work I report in this thesis possible: Adrian Perrig, Amit Vasudevan, Jim Newsome, Jonathan McCune, Zongwei Zhou, and Yuan Tian. I am thankful to Olatunji Ruwase, Sauvik Das, Chen Chen, Yanlin Li, Siddhartha Chhabra, and Jun Han for their insightful discussions and for making valuable suggestions for completing this work. I would also like to thank the anonymous reviewers for their detailed comments and valuable feedback. Any errors contained in this book are, of course, my own responsibility so wherever there may be any mistakes, I ask your forgiveness.

Finally, in recognition of their indelible and unwavering support, I dedicate this book in gratitude to my father, Francis Yaw Owusu, and to my mother, Theresa Abena Korsah, who taught me, from an early age, the importance of education.

Abstract

Extending cloud computing applications to fog computers provides a rich hybrid cloud computing platform for liberating mobile and embedded devices from severe resource constraints. However, the security techniques that are well-suited to the traditional cloud computing model do not adequately protect against the increased cyber-physical and privacy risk element of the Internet of Things (IoT). Off-premise code execution in the hybrid cloud computing model must be resilient against both malicious software and an adversary with physical access. Moreover, in the absence of user-controlled and remotely verifiable data protection mechanisms, individuals must implicitly rely upon application service providers — including the full software stack responsible for the deployment, management, and monitoring of cloud workloads — for the handling of personal data.

This work develops the concept of *fog mediation* — a fog computing systems design pattern, derived exclusively from the physical microstructure of commodity CPUs, for generating user-controlled cryptographic key material and, thereupon, mediating the usability, security, and privacy requirements of security-sensitive Internet of Things applications. The tamper-evident key generation properties of an isolated execution environment enable security-sensitive application logic and data to move freely between on-premise and off-premise computing resources. Fog mediation maps the variegated device capabilities of the IoT to a manageable set of mandatory protections and discretionary controls that accord with the application requirements and the data handling preferences of individuals.

How we regulate the use of data in the context of IoT has significant implications for individual rights with regards to personal data and notions of privacy and cybersecurity in an increasingly connected society. The policy analysis focuses on laws, standards, and regulations relevant to the use of IoT in the United States. In particular, this work investigates policy measures that could be adopted through the joint efforts of U.S. federal and state regulation, private sector self-regulation, transnational technical standards organizations, and consumer-oriented non-governmental organizations.

Thesis Statement: *This dissertation advances the state-of-the-art of hardware root of trust, remotely verifiable code execution, and privacy-preserving collaborative learning models as an aid to data protection policy and edge device security in hybrid cloud computing.*

General Terms: Security, Privacy, Design

Keywords: Isolated Execution, Fog Mediation, Privacy Partitioning, Edge Computing, Hybrid Cloud, Technology Adoption

Contents

List of Figures

List of Tables

List of Algorithms

Part I

Research Overview

Chapter 1

Introduction

Today, the global prevalence of the Internet, the ubiquity of connected devices, and the ever-increasing societal reliance on computing services highlight the importance of designing computing systems with security in mind. The same computing infrastructure that increases the reach of transformative technologies that promote public well-being is routinely compromised by the unwholesome elements of human psychology and society.

It is at the intersection of computing and society where the persistence of old problems and the constant emergence of new problems seem to resist our most well-intentioned attempts to extend the attributes of computing systems (e.g., reliability and security) to the individuals and groups that make use of them. This incongruence seems to indicate that the properties achieved at the level of computing systems do not always translate in full measure to the level of services and individuals. We build properties like reliability and security into computing processes, but ensuring those properties translate into human experience presents another challenge. However, it is also at the intersection of computing and society where we observe many compelling opportunities for making tractable some of the most pressing social, political, and economic problems.

This dissertation advances the state-of-the-art of hardware root of trust, remotely verifiable code execution, and privacy-preserving collaborative learning models as an aid to data

protection policy and edge device security in hybrid cloud computing and the Internet of Things (IoT). The research problem of how to provide edge device security and user data protection in the IoT is particularly important because of the increased cyber-physical and privacy risk element of the IoT. This problem has not been solved before due to the challenges of protecting security-sensitive code execution on commodity computer processors (i.e., central processing units or CPUs) in the presence of adversaries with physical access to the hardware. This work proposes minimal architectural changes that achieve an isolated execution environment completely contained inside a modern CPU.[1] The proposed CPU instruction set extension does not require on-chip non-volatile memory to store secrets and has resilience against several classes of hardware attacks. We model and evaluate the performance of our implementation in a series of research studies demonstrating it's applicability to IoT security and privacy. We consider the laws, standards, and regulations relevant to the use of IoT in the United States. We also consider the policy implications of the technical proposals presented in this research in light of U.S. policies on IoT cybersecurity and data protection. Based on this analysis, we recommend IoT cybersecurity and data protection policy measures that could be implemented through collaboration between consumer advocacy groups, technical standardization organizations, federal and state regulatory bodies, and the private sector.

The policy conclusions we seek to inform are broad-based whereas the technical proposals examined in this work must provide enough specificity to demonstrate feasibility under competitive assumptions for cost and performance (i.e., a reasonable account of the major product drivers for the adoption of security features). The technical proposals presented here address specific use case requirements, nonetheless, the motivating design choices (see Sections 3.2.2, 4.2.2, and 5.2.2) provide actionable agency to the primary beneficiaries of policy interventions without unduly encumbering the implementers of policy interventions — a benefit-cost balance that aims to bridge the disconnect we observe between platform-level

[1]These modifications are minimal in the sense that they make use of the semiconductor processes that are already used in the production of modern CPUs.

capabilities, service-level security guarantees, and IoT policy. Concrete implementations also provide sufficient food for thought to those most interested in the potential trade-offs between various design choices or building and deploying such systems.

The general approach followed in our line of inquiry borrows both from computer security and policy analysis. We lead into the technical proposals by first investigating the existing engineering and policy landscape using the lens of the policy analyst. Our primary motivation here is to better understand the practical factors that enable policy-based security and privacy requirements to be readily adopted as product design specifications. Conversely, we investigate industry trends and technology-based mechanisms that inform and suggest perhaps more advantageous directions for policy efforts. It is with this understanding in mind that we turn to the historical context of contemporary developments in trusted computing in Chapter 2. In particular, we explore developments related to critical computing infrastructure and the capabilities they give rise to. We then present several systems architecture designs using the lens of the security researcher (Chapter 3, Chapter 4, and Chapter 5). These proposals, leveraging trusted computing principles, provide a technical means for transferring system-level trust, and its attendant risks and uncertainties, to the domain where individuals are most directly impacted by computing systems — the domain of actionable regulation and industry standardization in the form of developer-facing programming interfaces and user-facing apps. Finally, we switch back to the policy analyst lens and ask what implications do the challenges and solutions addressed in this work have for the Internet of Things security and privacy policy landscape (Chapter 6).

1.1 Research Questions

These research questions have guided the studies in this dissertation:

- *What do we need to add to a modern CPU to achieve a highly efficient isolated execution environment with remote attestation properties?*

- *What do we need to add to fog and edge computing deployments to extend the security guarantees of the proposed isolated execution environment to the diverse set of mobile and embedded devices comprising the Internet of Things?*

- *What do we need to add to the optimization functionality of deep learning architectures to learn accurate models that integrate isolated execution and remote attestation properties with user-controlled data use models?*

- *What are the gaps and limitations in the legal and regulatory landscape to realizing recommendations for user-verifiable code execution and data sharing models that utilize hybrid cloud computing?*

- *What technical strategies provide a means for decision-makers to expedite the adoption of user-verifiable code execution and data sharing models in economic markets that are supportive of user-verifiable data protection in hybrid cloud computing?*

- *What technical strategies provide alternatives for individuals in economic markets that are not supportive of user-verifiable data protection in hybrid cloud computing?*

1.2 Security Challenges in the Cloud

Evaluating the trusted computing base (TCB) of an application in today's cloud computing infrastructure is a challenge. From the perspective of the user, the cloud trust model is vast, inscrutable, and generally not designed with user-controlled data protection mechanisms. The problem of securing the commercial cloud for cloud customers is a far from trivial task and this fact is not lost on the major commercial cloud vendors. For example, Amazon created a separate air-gapped cloud computing installation for the CIA.[2]

The challenge begins with the building blocks themselves — the hardware components comprising the infrastructure. In the context of a globally distributed hardware supply chain,

[2] Announcing the New AWS Secret Region, Nov. 20th 2017.

consideration must be made, not only for assessing how well a device protects its root of trust during use, but also for how susceptible the root of trust is to compromise during the manufacturing, system integration, and distribution phases. The production and trafficking of counterfeit semiconductor products pose a major security risk to critical services that rely on electronic systems. Semiconductor counterfeiting can occur both outside and inside the authorized supply chain. For example, "e-waste" products that are re-marked to indicate a newer or higher-performing unit than the original are an example of counterfeit occurring outside an authorized supply chain. The case where a manufacturing plant produces undocumented units for sale to unauthorized distributors is an example of counterfeit occurring inside an authorized supply chain. The significant numbers of documented counterfeit semiconductor products in a small portion of the commercial and military semiconductor markets suggest that there is a recurring production of counterfeit products in the total semiconductor market [1, 2].

Counterfeit operations are not subject to the best-practice industry controls that ensure the quality, reliability, and security of semiconductor products. This results in the real possibility of compromised components being introduced into the market. A security property that would lessen the risk potential of hardware production is a root of trust based on a key generation mechanism that does not require on-chip non-volatile memory to store secrets. This security property frees distributors and manufacturers from the liability of having to protect platform secrets on behalf of the end-user. It also provides users with unique cryptographic keys that are independent of any processes external to the requesting application — resulting in a TCB that is independent of supply chain factors.

Even if the key derivation mechanism results in secrets that are verifiably independent of any processes external to the requesting code or preceding run time, accounting for the security of the remaining components in the TCB of the traditional cloud model is a difficult proposition for an end-user. The most reliable strategy for securing large complex systems is to minimize the complexity of its security-critical components — to minimize the TCB.

From the perspective of the developer, it would simplify matters to reduce the TCB to the processor and the executing code. This enables the security of a given software service to rest on the correctness of its implementation. However, the current best practices for protecting secrets in the cloud result in a TCB that contains many components not necessary for running the security-sensitive applications.

As a result, cloud users must effectively include the service provider's code, the full software stack responsible for managing and monitoring cloud workloads, as well as any privileged code (e.g., the firmware and hypervisor) in the trusted computing base. Additionally, cloud users implicitly trust the employees of the cloud service provider including systems administrators as well as anyone with physical access to the hardware (e.g., security staff, facilities staff, cleaning staff, and visitors). Lastly, as highlighted by reports regarding the Utah Data Center and the Snowden leak [3, 4], cloud users implicitly trust the law enforcement agencies in any of the areas their data may be stored or duplicated. An externally verifiable isolated execution environment that consists solely of the CPU would reduce the TCB security-sensitive applications to the processor and the executing code.

In addition to the challenges of providing a strong hardware root of trust and minimizing the size of TCB, the current cloud security model is generally designed to protect the cloud infrastructure from untrusted cloud users and not vice versa — i.e., sandboxing is commonly employed to protect privileged platform code (of the cloud provider) from untrusted code (user data and services residing in a virtual machine/container). Whereas isolated execution restricts access to user data exclusively to the authorized service (and the trusted computing base needed to establish the integrity and secrecy of the execution environment). This security model is a much more tenable prospect for the application developer and facilitates communicating privacy and security to the user. From the perspective of the user, the cloud infrastructure hosting their service operates transparently when the design objective is isolated execution with remote attestation properties — enabling cloud users to assess the security and privacy risk of their service primarily in terms of the application code.

Utilizing a hardware root of trust and minimal TCB isolated execution environment — to support integrity, secrecy, and remote attestation properties — enables user-verifiable implementations of security-sensitive application logic to run unencumbered alongside cloud task management functionality. Bolstering commodity CPUs with these properties also provides resilience against a number of the additional security challenges of extending cloud computing to fog computing and hybrid cloud computing. The term *hybrid cloud* refers broadly to cloud services that rely on multiple cloud service providers. This could be used in reference to a single cloud service that is ported to several cloud vendors to avoid lock-in. It may also be used in reference to a cloud service that is an integration of multiple cloud services. Hybrid cloud is often used to describe cloud services that have both private cloud components and public cloud components [5]. We use this special case of hybrid cloud computing (i.e., private/public or local/remote) in reference to the technical proposal presented in this work for edge computing security services deployed to local fog computing and remote cloud computing.

1.3 Security Challenges at the Edge

The Internet of Things extends the Internet from a network comprised of a well-defined set of traditional device categories (e.g., client/server or host[3]/router) to a network that includes everyday physical objects outfitted with software, sensors, and radios. The convergence of software services and networked matter is an exciting prospect. However, this prospect includes new security and privacy challenges and an increase in the cyber-physical risk element of cyber attacks [6].

The argument has been made that there is no such thing as an "Internet of Things" — that it is simply our good old Internet with many devices connected to it. It is true that the Internet has evolved as a medium, from an interconnection of workstations, routers, and mainframes, to an interconnection of billions of devices, some of which have very little

[3]Host refers to a computing system or device that can host applications (i.e., a network endpoint).

computing abilities or non-standard functionality [7]. In this sense, the IoT is an outgrowth and development of how the Internet is used. However, this outgrowth has led to distinctions in the way computing systems interface with one another and interact with the world — distinctions with significant implications for how we address cybersecurity.

The notion of a domain in the IoT context is more closely aligned with physical proximity than with the logical relationship of components. For example, a smart space may contain embedded devices, network elements, displays, computers, sensors, and actuators all operated by different users, serving different purposes, and managed by different service providers but nonetheless falling under the same domain due to their proximity to the users and user devices that leverage them. Similarly, the type of messages exchanged in IoT domains are highly spatiotemporally dependent (e.g., contextual information such as system state, observations about the environment, and control signals). The physical proximity and contextual information flow characteristics of many IoT use cases may allow us to rule out certain attack vectors (e.g., man-in-the-middle may become impractical), but may also present some trade-offs.

1.4 Grounds for Mediation

We've discussed the challenges that end-users face in accounting for and reducing the trusted computing base of cloud-based applications. We've discussed the security challenges introduced by the IoT. We've also suggested that minimal modifications to commodity CPUs and additional design considerations are keys to addressing the unique challenges of IoT security. We now consider the role fog computing can play in addressing these challenges. Just as "fog" is a cloud formation that is near to the surface of the earth, the idea of fog computing is to bring cloud computing resources closer to the edge of the network. Fog computing refers to computation, storage, and network resources residing between end devices and cloud computing data centers [8, 9]. Fog computing installations often serve as a complement to cloud

computing.

Crisp and responsive interaction remains a critical design constraint for software with high quality of service requirements (e.g., real-time analytics, streaming, online planning and actuating, and augmented reality). The physical nearness and single-hop network latency of fog computers — with respect to the edge of the network and end devices — enables both functional and performance improvements to applications [8]. When applied to the Internet of Things, fog computing enables responsive resource-intensive services to be augmented by resource-poor embedded devices and consumed by resource-constrained user devices. The common characteristics of fog computing applications include latency-sensitivity, location-awareness, wide geographical distribution[4], increased wireless access, increased mobility support[5], ubiquity[6], and heterogeneity[7] [9].

We observe that resource-rich fog computing nodes provide not only a scalable platform for liberating mobile and embedded devices from severe resource constraints but are well-positioned as a grounds for mediating trust between the cloud and the edge of the network. It is the goal of this research to examine ways in which security extensions to fog computing architectures may be applied to the challenge of providing strong security assurance in the Internet of Things. Additionally, this research analyzes U.S. policy with relevance to the IoT and considers its implications for the individual right to personal data protection.

The proposals in this body of work consider an adversary in three contexts. This first context is that of protecting the integrity and secrecy of code execution on commodity CPUs deployed to the cloud. This second context is that of protecting code execution

[4]This is in contrast to applications relying on remote centralized cloud deployments. This geographical distribution enables greater utilization of high-bandwidth wireless LAN and plays an active role in supporting quality-of-service guarantees.

[5]For example, a common technique needed for fog computing mobility support is to enable mobile devices to communicate with an application in a manner that decouples host identity from location-based and time-based identity.

[6]Ubiquity refers to the integration and orchestration of very large numbers of network nodes and end-points.

[7]The resource capacity, form factor, and deployment environments of fog computing units vary widely. Fog apps make use of virtualization support and transient customization to support these heterogeneous and dynamic deployment contexts.

on commodity CPUs residing in fog computing deployments. This third context is that of protecting individual privacy during the use of machine learning applications running on hybrid cloud deployments. The considered hybrid cloud consists of a fog computing component (local domain) and a cloud computing component (remote domain). Please note that the detailed definition for each security problem is stated at the beginning of each study. We present the adversary models and desired security properties in brief here.

The first adversary model is with regard to providing minimal TCB code execution in the cloud (isolated execution). We consider a sophisticated adversary with physical access to the computing platform. The adversary can introduce malware into the computing platform, has access to the external ports of the platform to physically attach malicious peripherals. Additionally, the adversary can probe and tamper with low-speed and high-speed buses, inject code, and modify data. With regards to this class of adversary, we'd like to achieve an isolated execution environment with a TCB consisting solely of the CPU packages — providing strong root of trust, integrity, secrecy, and remote attestation properties to end-users leasing hardware deployed to the cloud.

The second adversary model is with regard to enabling fog computing architectures to provide minimal TCB code execution and mediation services to IoT applications (fog mediation). With regard to this class of adversary, we'd like to provide integrity, secrecy, and remote attestation for security-sensitive code running on fog computing equipment in environments that are physically accessible to any interested passerby. We'd also like to support authentication and data access rules based on co-presence to the fog mediation node(s).

The third adversary model is with regards to enabling privacy-preserving data sharing within machine learning applications deployed across user devices, fog computers, and cloud computers (privacy partitioning). We consider an adversary with access to the hidden layer activations of a deep neural network during the model inference stage. This adversary is interested in carrying out an input inference attack (i.e., discovery of the original user input) and a private attribute inference attack (i.e., unauthorized extraction of a private user

attribute). With regard to this class of adversary, we'd like to lessen the applicability of intermediate network states to unauthorized learning tasks without diminishing the classification accuracy of authorized learning tasks.

1.5 Contributions

This work presents systems security design patterns for securely interfacing security-sensitive apps, IoT devices, and user data with cloud computing and fog computing infrastructure: *isolated execution*, *fog mediation*, and *privacy partitioning*. Isolated execution (OASIS) establishes a verifiable root of trust for measuring code execution environments utilizing the modern computer processor (the topic of Chapter 3). Fog mediation (MEDIA) extends the processor-level security guarantees of an isolated execution environment to support security-sensitive fog computing applications (the topic of Chapter 4). Privacy partitioning (DATUM) is a deep network optimization framework for protecting personal data originating from a fog mediation node (the topic of Chapter 5). The policy analysis focuses on identifying the policy implications of realizing user-verifiable code execution and data sharing models in the IoT. Additionally, the policy analysis focuses on identifying the gaps and limitations in the United States IoT policy landscape concerning cybersecurity and the individual right to data protection (the topic of Chapter 6). In particular, this work makes the following contributions:

OASIS Contributions.

- We present an instruction set extension for remotely verifiable, efficient code execution requiring a minimal TCB (Section 3.4).
- We propose an application programming interface where the CPU provides unique cryptographic keys to security-sensitive applications (Section 3.3).
- Our system is designed for deployment on existing commodity CPUs with minimal

modifications (Section 3.2).

- Our deployment model precludes the need for a distributor or manufacturer to protect platform secrets on behalf of the end-user or their customers (Section 3.5).
- Contrary to prior approaches, our solution does not require on-chip non-volatile memory to store secrets. Thus, in addition to avoiding the strong assumption of secure non-volatile memory, our solution is cheaper to implement in practice as it leverages semiconductor processes already used in modern CPUs (Section 3.7).

MEDIA Contributions.

- We present a security architecture for isolated execution in fog computing applications (Section 4.2).
- We examine the applicability of hardware-based root of trust and isolated execution environments to the challenges of providing strong security assurance in the Internet of Things (Section 4.3).
- We introduce the concept of fog mediated computing — a systems security design pattern that leverages edge computing infrastructure to provide a common root of trust between co-located users, mobile devices, embedded devices, and services (Section 4.4).
- We evaluate the performance of MEDIA using commodity computing hardware and public cloud resources (Section 4.5).

DATUM Contributions.

- We propose an optimization technique for learning accurate deep networks that are resilient against input inference and private attribute inference attacks (Section 5.3).
- Unlike related differential privacy research that protects statistical databases (aggregate data) during the model inference phase, we propose a solution that protects individual queries (Section 5.2).

- We experimentally demonstrate the effectiveness of the proposed optimization technique in a series of experiments. The results indicate that privacy partitioning can significantly reduce the privacy risk potential of deep network activation states (Section 5.4).

Policy Contributions.

- We identify gaps and limitations in the regulations and standards governing the United States' internal use policy on IoT (Section 6.2).
- We analyze federal and state paths to a comprehensive U.S. cybersecurity and U.S. data protection policy frameworks.
- We make the case for joint public and private interventions that address the global dimension of domestic cybersecurity policy effectiveness by drawing lessons from the policy efforts to improve labor rights and environmental standards in the global supply chain for electronics production (Section 6.1).
- We present a series of policy recommendations for IoT security and privacy based on technical findings presented in this work (Section 6.3).

1.6 Research Components

We now highlight how the individual projects fit together to support fog computing applications. Isolated execution achieves a TCB consisting of the CPU package for use by security-sensitive pieces of application logic. Some examples of security-sensitive pieces of application logic include code segments within an application that are tasked with handling cryptographic keys, carrying out cryptographic operations, or processing confidential user data. These modifications do not presume a secure manufacturing process, are remotely verifiable and enable commodity CPUs to manage cryptographic secrets and complete security-sensitive code routines — in a manner that is resilient to malware compromises as well as adversaries with physical access to the computing platform.

In particular, we consider the application of these processor extensions to cloud computing and edge infrastructure to support service integration between the cloud provider, the fog provider, and the service provider — to support fog mediation. The operations of a fog mediation node include the launch and measurement of the isolated execution environment, secure channel establishment with the remote verifier, the establishment of the proximal domain (where a proximal domain refers to the set of connected devices and fog nodes managed by a fog mediation cluster), the provisioning of services to the proximal domain (where VM-based micro-services are deployed to proximate fog computing infrastructure), the provisioning of secrets to the proximal domain (where user-provisioned secrets and hardware-generated secrets may be securely managed across any isolated execution environment instance belonging the application), code execution, saving the session state, and clean up of the isolated execution environment after each session.

Each fog mediation node contains CPUs capable of isolated execution. The programming interface for isolated execution enables proximal domains to be formed. Fog mediation nodes manage device messaging within the domain as well as mediate the interactions between users and apps. With respect to mediating co-present devices, fog mediation enables the app to be transiently customized specifically for the space defined by the proximal domain. For example, the capabilities of the deployed service can be modified based on the types of IoT resources available within the proximal domain and the security and privacy preferences set by the individuals present within the domain. With respect to mediating external resources, privacy partitioning enables the data generated by a fog mediated proximal domain to be processed remotely in a deep neural network without leaking the private attributes of the proximal domain.

Part II

Modern Processors: The Standard Unit of Trust

Chapter 2

Technology, Trust, & Policy

2.1 Trusted Computing Risks & Benefits

Concerns regarding trust have had a significant impact on the development of technology and policy [10, 11, 12]. The security of nations and the proper functioning of economies rely on communications, transportation, energy distribution, health care, and financial systems — all of which rely heavily on the proper functioning of networked computing systems. It is this reliance on the uninterrupted proper functioning of a computing system that we denote here as *trust*. Thus, a computing system is *trustworthy* to the extent that it can preserve its security properties with high probability even while under attack.

The initial research and development of what was to become the Internet as it known today began as a proof-of-concept network for evaluating the feasibility of using a packet-switched network architecture to support interactive applications like e-mail [13]. The presumption of trust in this context was appropriate — this group of early adopters consisted of a close-knit group of specialists and enthusiasts working for the United States government on private research networks. Thus, the initial protocol specifications for the Internet focused primarily on network connectivity to the omission of specific protections against misuse and mal-use [10].

During the rise of the commercial Internet, the rapid shift from boxed software products with release cycles spanning multiple years to the rise of web browsers and mobile apps with agile development cadences on the order of weeks left established technology companies scrambling to adjust to the new security challenges of a web-first world [11]. *Trustworthy Computing* is the imperative to develop computing systems that provide security, reliability, and availability by design. The term trustworthy computing came into popular use during this time of rapid growth in the commercial application of the Internet that characterized the technological paradigm shift of the 1990s [12].

At the same time, concerns regarding personal privacy have increased alongside concerns regarding the security of computing systems. In addition to privacy concerns stemming from security breaches and criminal activity, there is a crystallizing milieu of concern regarding the sheer amount of personally-identifiable data that is generated during the normal course of online use — and how this digital footprint is aggregated, analyzed, and disseminated. Notwithstanding any damping effects due to concerns over security and privacy, it is safe to assume that online activity will continue to grow. The apparent trade-off between utility and privacy takes on greater dimensions as we consider the proliferation of high-value digital assets occurring alongside increased global connectivity.

The push for trustworthy computing was in part an acknowledgment that technology must provide certain core properties if it is to be fully integrated into daily life. The majority of the works presented here focus on systems security approaches. These methodologies provide one such alternative to ceding the technological factors that support connectivity to the manifest factors that undermine trust. In particular, this work focuses on the agency of individual end-users via tools and policy prescriptions designed for software developers. This approach provides a practical means to establish end-to-end trust in highly responsive data access models, allowing individuals to securely interact with their data at any location, on any device, using any service — reconciling many of the cross-cutting concerns for both app developers and app users. Furthermore, we explore proposals for end-to-end trust and

auditing capabilities that enable developers to make operative the pronouncements on which stakeholder requirements are based (e.g., privacy-enhanced consumer protection policies). These flexible tools equip developers to build secure services that provide transparent grounds for trust even in the cases where the motivations of service providers may be at odds with the expectations of individuals or consumer advocates.

2.2 User-Centered Computer Security Design

We now discuss the top-level design goals for the technical proposals. The initial security standards for trusted computing were motivated by efforts in the U.S. Department of Defense (DoD) to classify computing systems in terms of the security controls they provide [14]. Although these standards were designed specifically for the latticed command and control organizational structure used in the military context, they were adopted and adapted by many businesses that sought to build secure computing systems.

Trusted computing, as a heuristic for providing robust solutions for a given security specification, can be factored into two aspects: (1) minimizing the trusted computing base and (2) providing a mechanism for measuring the integrity of the trusted computing base [15]. In contrast to trusted computing implementations that develop these two aspects primarily for use by hardware vendors and services providers, this work emphasizes the implementation of the first aspect (minimal TCB) primarily for the benefit of app developers and the second aspect (auditing capabilities) primarily for the benefit of end-users. Throughout this work, we provide evidence that this adjustment in stakeholder prioritization (1) encourages greater levels of end-to-end trust for all stakeholders and (2) makes tractable previously unavailable policy options.

It is accepted that there are still serious security vulnerabilities facing modern processors [16, 17, 18]. Micro-architectural performance optimizations that preserve the correctness of the architecture may not preserve system security [19]. The computer architecture

community is working towards developing methods that are formally sound for hardware security, and the work presented here is a step towards making it prohibitively difficult to compromise secret keys on modern processors. In the mobile device ecosystem, the market offers many promising avenues for hardware-based security protections, but these capabilities have only recently become accessible to apps and individuals [20]. In the broader embedded device market, platform security is often low in priority in cases where security is not the marketable value proposition for the product due to competition and costs.

In terms of cloud computing, there are other works that prioritize protecting end-users from an untrusted cloud (isolated execution) but in general, the security tools for commercial public cloud infrastructures are developed to protect the cloud service stack from potentially malicious tenets (sandboxing) [21]. In some instances, even the cloud service providers, some of the biggest purchasers of semiconductors, are themselves locked out of access to critical platform security management components, and thus, cannot fully guarantee sandboxing for the cloud stack or end-to-end isolated execution for cloud tenants.[1] Dedicated security modules like the TPM are promising in some respects but have shortcomings that make alternatives such as the CPU-based isolated execution environment proposed here more appealing (discussed in greater detail in Section 3.8).

Table 2.1: Design Goals by Stakeholder

Stakeholder Context	**Information Security Policy**
Military & Enterprise	• data secrecy
Commercial	• data integrity
Developers	• launch point integrity
	• remote attestation
Users	• data protection
	• service availability

[1] "MINIX — The most popular OS in the world, thanks to Intel." Network World. Bryan Lunduke, Nov. 2nd, 2017.

Table 2.1 contains a list of top-level security design goals in terms of stakeholders. The military and enterprise context can be expanded to refer to any organization that must maintain controlled access to assets and thus needs a security policy for preventing unauthorized disclosure (data secrecy). The commercial context is shorthand for an organization engaged in commercial exchanges. In the commercial context proper accounting (i.e., preventing unauthorized modification or data integrity) is paramount although data secrecy is still an important security property. Developers refer to the deployment side of software services whereas users refer to the consumption side of software services.

Table 2.2: Design Goals by Platform Contexts

Platform Context	**Platform Security Property**
Manufacturing	• strong root of trust
Code Execution	• minimal trusted computing base
	• isolated execution
Fog Mediation	• user authorization
	• device authorization
Proximal Domains	• privacy controls
	• service mobility
Internet of Things	• interoperability
	• remote device management

Table 2.2 contains a list of top-level security design goals in terms of the platform contexts considered our studies. In the manufacturing platform context, a strong root of trust is the key security design goal.[2] In terms of the code execution platform context, the key security design goals are minimizing the TCB while providing isolated execution. In the fog mediation platform context, entity authentication is the paramount security design goal. In the proximal domains platform context, privacy controls and mobility are paramount. In

[2]Please note that minimizing production and performance costs are top-level goals for each context and often a key decision criterion for design choices.

the IoT platform context, interoperability and remote device management (including patch support) are paramount.

2.3 Integrity & Secrecy Defined

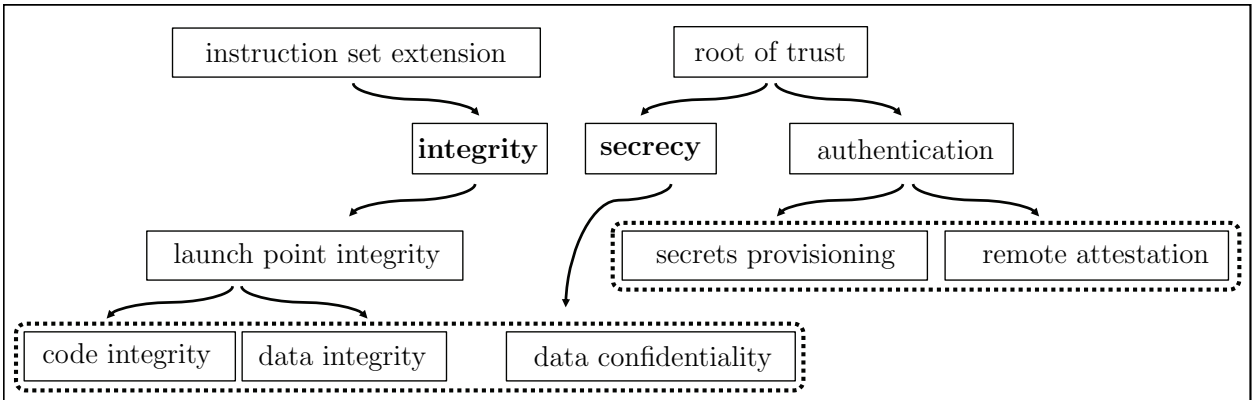


Figure 2.1: Isolated Execution Security Proprieties. The hardware basis for integrity and secrecy in the proposed architecture are the processor instruction set extensions and the processor root of trust, respectively. The code integrity, data integrity, and data confidentiality properties provide platform protections for the executing code and its associated state. The secrets provisioning and remote attestation properties enable the platform to reliably relay this information to a remote verifier.

The primary security objectives in this research are *integrity* and *secrecy*. Integrity is the assurance that code will run as written. Secrecy is the capability of the owner of a secret to control its release. Figure 2.1 shows the properties provided by the isolated execution environment proposed in this work in relation to the primary platform security objectives of integrity and secrecy. The primary purpose of the instruction set extension is to provide integrity. The property of launch point integrity provides the functional implementation of isolated execution as long as it includes the assurance that the memory region belonging to the executing code runs unaltered by any processes external to it and the integrity of the executing state is maintained in between successive context switches. Isolated execution enables strong assurances for code integrity and data integrity.

The primary purpose of the hardware root of trust is to provide strong secrets. When those secrets are used as an identifier they can be used to bootstrap authentication protocols. Similarly, when those secrets are used as inputs to pseudorandom functions (i.e., key derivation functions) they may be used for the generation of cryptographic key material. Further, when a secret is supplied in conjunction with a non-secret input (such as a user-provided code or entropy from a random number generator function) to a key derivation function, many cryptographic keys may be generated from this common secret (key diversification). Thus a strong platform secret (i.e., a strong root of trust) provides authentication and secrecy (e.g., when derivative cryptographic key material is used as an encryption key to provide data confidentiality). Authentication supports both entity authentication (i.e., the ability to make verifiable assertions of platform authenticity) and data authentication (i.e., the ability to make verifiable assertions that a given message came from the claimed platform).

Integrity and secrecy encapsulate the high-level systems design security goal for many isolated execution proposals [21, 22, 23, 24, 25, 26] — including the "essential trusted computing base" microprocessor architecture proposed here (Chapter 3). This is in part because integrity and secrecy provide a useful shorthand for what is meant by "secure" in the context of remotely verifiable code execution and thus provide a concise yet meaningful expository label for discussing systems security architectures without delving too far into implementation-specific properties or application-specific security objectives.

If the systems architecture provides strong assurances for code integrity then the operative security challenge shifts to assisting developers write safer software with, for example, tools that improve code correctness (such as formal verification) or tools that make is more difficult to write bad code (like type-safe programming languages). If the systems architecture provides strong tamper-resistant assurances for data secrecy then there is a strong basis, strong root of trust, for the cryptographic operations based on those secrets such as authentication or confidentiality protocols. Generally speaking, integrity and secrecy provide a foundation for any further security assurances. Without strong integrity and secrecy

assurances, the functioning of security-critical code is unpredictable due to the prevalence of vulnerabilities in modern OSes [27, 28].

Additionally, many of the application-level security properties either reduce to these two properties or are made feasible if the security architecture provides strong assurances for them. Potential applications for the hardware-based isolated execution environment proposed here include: remotely attestable computation [29, 30], secure multiparty computation [31, 32, 33], secure boot [34], secure key provisioning [35], digital rights management [36], cryptographic key generation [23, 37], auditable billing and accounting [38], privacy-preserving energy metering [39], privacy-preserving location services [40], privacy-preserving deep learning [41], trusted mobile computing [42, 43], user authentication [44], rule-based access control [45], remote policy enforcement [46], encryption at rest and encryption in transit as-a-service [47], memory encryption [48, 49], hardware wallet for cryptographic assets [50], password wallet [51], trusted path for I/O devices [52], and secure virtual machine migration [44].

2.4 Composite Security

Composite security — i.e., building secure systems from smaller components — is a core requirement for securely designing products and services that leverage the growing number of connected devices comprising the IoT. Security-preserving composition allows for modular design; enabling complex yet secure services to be built from simple building blocks.

Composite security is particularly challenging in the IoT context where security-critical services are constructed from collections of connected-devices that may each be supported by different providers, operated by different users, and serving different purposes. This research points to the importance of composite security in addressing the challenges of developing secure IoT products and services. Further, this research examines the potential for composite security interventions that address the cross-organizational nature of IoT services. Whether

composite security at the product level is achieved through industry-established best practices and design standards or is mandated by legislation, composite security is essential for securing IoT products and services.

Composite security may be further categorized as nondestructive or constructive. Nondestructive composition refers to the combination of components in a manner that ensures that the implementation of one subsystem does not compromise the security properties of another. Ensuring nondestructive composition is particularly challenging for composite systems that rely on shared state to integrate subsystems. Constructive composition refers to the combination of components in a manner that aggregates the security features contributed by each subsystem. The challenge with constructive composition is that there are many ways to combine subsystems that appear reasonable, even to domain experts, but do not, however, extend the security properties of the subsystem components to the composite system. For example, TLS is generally not secure (i.e., does not provide authenticated encryption) because of the particular way the protocol combines its MAC and encryption whereas the IPSec protocol does provide authenticated encryption [53].

At the protocol level, the cryptography community has accepted theories for composite security due in large part to the work of Canetti[54]. However, at the systems level, notions of composite security as applied to products and services leveraging connected devices and third-party apps are not well understood. Even in cases where each connected device is secure in isolation, it's a non-trivial task to ensure that the end product achieves the specified security requirements.

As an illustrative example, imagine an industrial facility with a collection of connected sensing and actuating devices produced by several businesses. Furthermore, each device is secure–i.e., conforms to the security policies established by the producer and the security administrator of the facility. However, a critical security vulnerability is introduced due to the composite insecurity of an authorized third-party software service leveraging the connected-device. How can we minimize the potential for this class of vulnerabilities without limiting

the versatility of IoT products and services? And in the case of imposed liability, who should be held accountable for a vulnerability resulting from interactions (e.g., should the system integrator be held accountable or the producer of the compromised device)?

Composite security is a fundamental computer security challenge for protecting consumer privacy and building secure Internet of Things applications. Policy recommendations aimed at enhancing security and consumer privacy that center on applying security measures to individual connected devices do not adequately address this fundamental challenge.

This challenge of extending our protocol-level understanding of composite security to product design is evidenced by real-world attacks in a wide variety of settings where the properties of one subsystem are used to undermine the security properties of another. Policy measures are perhaps better suited to addressing composite security requirements than exclusively technical interventions due to the cross-organizational coordination composite security at the product level entails.

Chapter 3

Isolated Execution

This chapter is largely a reproduction of the paper *OASIS: On Achieving a Sanctuary for Integrity and Secrecy on Untrusted Platforms* co-authored with Jorge Guajardo, Jonathan McCune, Jim Newsome, Adrian Perrig, and Amit Vasudevan [23]. This research was supported in part by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 from the Army Research Office, and by a gift from Robert Bosch LLC.

On Achieving a Sanctuary for Integrity and Secrecy (OASIS). This chapter presents OASIS, a CPU instruction set extension (ISE) for externally verifiable initiation, execution, and termination of an isolated execution environment (IEE) with a trusted computing base (TCB) consisting solely of the CPU [23]. OASIS leverages the hardware components available on commodity CPUs to achieve a low-cost, low-overhead design.

3.1 Overview

The Ken Thompson adage, "You can't trust code that you did not totally create yourself," applies as much to software as it does to the entire execution stack running the software [55]. A code execution platform that is robust against malicious software *and* to an adversary with physical access is a difficult yet important assurance for many security-sensitive applications. In particular, the increasing integration of cyber-physical systems across the

infrastructural and industrial Internet of Things only heightens the scope and import of this security challenge.

A minimal trusted computing base contained within the processor package elevates the ubiquitous CPU from a base unit of computation to a base unit for mediating trust. This work explores how strong root of trust properties derived from minimal extensions to modern CPUs may be leveraged in support of strong isolated execution environments. Additionally, the methods described here provide a powerful design abstraction for mitigating many of the risks associated with dynamically offloading security-sensitive tasks to resource-rich but untrusted compute nodes — i.e., many of the risks associated with mediating trust across mobile, cloud, and fog computing.

For developers, strong root of trust and minimal TCB properties may be leveraged in support of isolated execution environments that simplify the task of delivering secure software from reasoning about the state of an entire system to solely reasoning about the correctness and usefulness of the software in development. For individuals, the isolated execution environment presented here simplifies the task of reasoning about the security of their devices to attesting to the launch point integrity of the processor.

Despite numerous attacks against a wide spectrum of organizations [56, 57], secure execution environments protected by TCG technology have not seen widespread application — even in cloud computing, where customers want to verify execution [58, 59]. Perhaps this lack of application is due, in part, to the lack of end-to-end application software that benefits from TCG properties, lack of trust in the TPM vendors, lack of protection against local adversaries, and concerns over poor performance.

Many designs for an isolated execution environment have been proposed, but an interesting question remains: What minimal additions do we need to add to a modern CPU to achieve a highly-efficient isolated execution environment with remote attestation properties? This work investigates what minimal architectural changes are required to obtain the essential TCB — an isolated execution environment completely contained inside a modern CPU

— providing resilience against several classes of hardware attacks. In addition, we design this architecture such that minimal changes to a modern commodity CPU are required for deployment. In keeping with minimalist design, we provide a simple programming interface consisting of few instructions.

Contributions.

- We present an ISE for remotely verifiable, efficient code execution requiring a minimal TCB.
- We propose an API where the CPU provides unique cryptographic keys to security-sensitive applications.
- Our deployment model precludes the need for a distributor or manufacturer to protect platform secrets on behalf of the end-user or their customers.
- Our system is designed for deployment on existing commodity CPUs with minimal modifications.
- Contrary to prior approaches, our solution does not require on-chip non-volatile memory to store secrets. Thus, in addition to avoiding the strong assumption of *secure* non-volatile memory, our solution is cheaper to implement in practice as it leverages semiconductor processes already used in modern CPUs.

Organization. Section 3.2 describes the problem space including the threat model and design assumptions. Section 3.3 provides background information on the hardware building blocks used in the construction of the OASIS architecture. Section 3.4 describes the OASIS instruction set. Section 3.5 specifies a protocol for using the OASIS instruction set to implement a security-sensitive application. Section 3.6 describes several important design considerations not described else where in this chapter. Section 3.7 presents the experiment test bed configuration and performance evaluation. Section 3.8 and Section 3.9 discuss the related literature and the conclusions of the study, respectively.

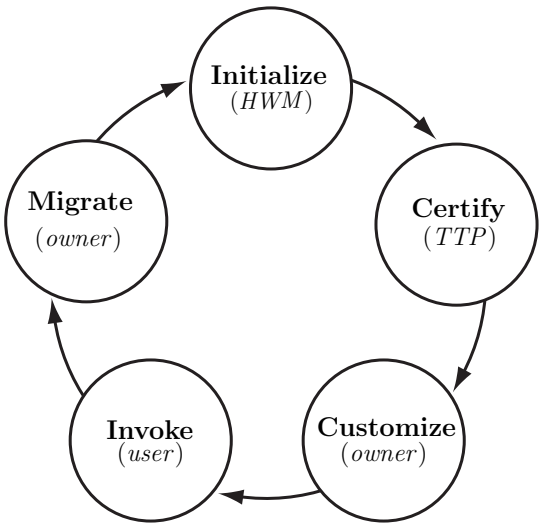


Figure 3.1: Device Life-cycle. This figure shows the stages during an OASIS-enabled device's life-cycle where each stakeholder party comes into play. The three key stakeholders involved in varying degrees of bootstrapping platform-based trust and mediating application-specific trust decisions during the life-cycle of the processor include the hardware manufacturer, the device owner, and the user.

3.2 Problem Definition

3.2.1 Model & Assumptions

Deployment Model. Our use case defines outsourced computation in the sense advocated by public cloud computing. Thus, we identify three key parties (see Figure 3.1); and their different roles and levels of trust as a device moves from production to use:

(i) The processor ***hardware manufacturer*** (HWM). The HWM is trusted to manufacture the CPU to initialize a cryptographic device key with a Physically Unclonable Function.

(ii) The ***service provider*** (or device owner) that offers the device as a platform to customers who wish to lease them for a certain amount of time or computation.

Finally, (iii) the ***user*** (or cloud customer) who wishes to lease computing resources. Users are interested in verifying the trustworthiness of devices leased to them, guaranteeing the integrity and confidentiality of their computations and data.

The hardware-based root of trust model proposed here is vendor-neutral, enables end-users (i.e., device owners and cloud customers) to derive secrets that are independent of the

production process, and does not require costly secure manufacturing processes. Although, the manufacturer and/or system integrator provides platform certification that attests to quality assurance guarantees, no stage of the production and integration process requires the injection of secret key material and thus precludes the need for any distributor to protect platform secrets on behalf of the end-user or their customers.

In the remainder of this chapter, we refer to the service provider's device simply as the *platform* or P, and to the user's device as the *verifier* or V.

Adversary Model. We assume a sophisticated adversary with physical access to the computing platform. In particular, the adversary can introduce malware into the computing platform (e.g., to compromise an application, device drivers, the OS, or firmware), and has access to the external ports of the platform to physically attach malicious peripherals to P. Similarly, the adversary can probe and tamper with low-speed and high-speed buses (e.g., to eavesdrop on a memory or PCI bus), and/or inject code and/or modify data. However, the adversary cannot perform attacks that require complete unscrutinized access to the CPU for extended periods of time. In particular, this implies that the service provider has organizational procedures in place to prevent attacks, but cannot guarantee the absence of a small set of rogue employees.[1] We consider denial-of-service, side-channel, and fault injection attacks beyond the scope of this work.

Assumptions. With respect to the service provider, we assume that the CPU on the untrusted platform P is not malicious and meets the vendor certification guarantees ascribed to it (i.e., we trust the processor). We assume that this CPU contains a Physically Unclonable Function that can only be accessed through the specified APIs. We assume that the CPU has a true random number generator. Additionally, we assume that the CPU is tamper-resistant — thus, physical security is not a requirement. Lastly, we assume that the verifier V has the correct public key of the provider's platform P.

[1] For example, a cloud service provider may unintentionally grant data center access to malicious [60] or negligent [61] employees.

3.2.2 Desired Properties

The following list contains the desired properties for OASIS.

*P*1 **Secure.** We would like the following security objectives to be satisfied:

*P*1.1 *Externally Verifiable.* Attestable code execution that guarantees platform integrity, code integrity, launch point integrity, and unmodified code execution on the untrusted platform.

*P*1.2 *Key Code Binding.* Ensure that a unique cryptographic key is available to each distinct code module that executes in the isolated environment.

*P*1.3 *Program State Binding.* The ability to bind data to code.

*P*1.4 *Device Transferability.* The ability to transfer ownership of a chip without exposing the secrets of the previous owner.

*P*1.5 *Limited Trust.* The HWM should not have access to any device secrets.

*P*2 **Economical.** We would like the following economic objectives to be satisfied:

*P*2.1 *Low-cost.* No substantial increase of manufacturing cost or complexity (e.g., by requiring non-volatile memory within the CPU).

*P*2.2 *Self-contained.* No requirement for additional hardware support such as secure co-processors or TPMs.

*P*3 **Essential.** We aim for a balanced and simple design:

*P*3.1 *Minimal TCB.* On-die isolated execution environment with trustworthy computing primitives entirely within the CPU package.

*P*3.2 *Minimal Interface.* Minimal interface with minimal controls, which presents a usable programming abstraction.

*P*3.3 *Minimal Setup.* Efficient environment setup where expensive operations are bypassed during repeated invocation.

3.3 Hardware Building Blocks

3.3.1 PUFs, Fuzzy Extractors, & TRNGs

Pappu et al. introduce the concept of *Physical Unclonable Functions* (PUFs), which are functions where the relationship between input (or *challenge*) C and output (or *response*) p_e is defined via a physical system [62, 63]. The physical system has the additional properties of being random and unclonable. The system's unclonability originates from random variations in a device's manufacturing process, which even the manufacturer cannot control. In their most general form, PUFs can accept a large number of challenge-response pairs. Examples of PUF constructions include: optical PUFs [63], silicon PUFs [62, 64], coating PUFs [65], SRAM PUFs [66, 67], reconfigurable PUFs [68], and Flash memory-based PUFs [69].

Because of PUF variability across different environmental conditions (voltage, temperature, humidity, etc.), when a PUF is challenged with C_i, a response p'_e (a noisy version of p_e) is obtained. In applications where the PUF response is used as a cryptographic key this noisy response p'_e is not acceptable. To solve this problem, algorithms known as *fuzzy extractors* leverage non-secret helper data to work around the noisy nature of physical measurements typical of PUF applications [70, 71, 72]. We assume that the fuzzy extractor is implemented in a silicon block and is accessible as a function that is used (in combination with the PUF interface) to realize our instructions.

While stability is fundamental for PUFs, variation in unstable bits can be leveraged for random number generation [67, 73, 69]. For the purposes of this paper, we focus on PUFs based on memory arrays, such as SRAM commonly used in CPU caches. SRAM memory can be used as the raw source for a PUF as well as the entropy source for a *True Random Number Generator* (TRNG).[2]

[2]The Intel random number generator is based on the instability of a couple of cross-coupled inverters, which are the basic building block of an SRAM cell [73].

3.3.2 Cache-as-RAM Mode

Cache memory is ubiquitous across CPU architectures. Traditionally, SRAM is used to implement a cache. Modern CPUs often include several megabytes of memory on-die which can be leveraged to create a *Cache-as-RAM* (CAR) execution environment [74]. Typically, CAR mode is used to perform system boot-up tasks while DRAM (external to the CPU) is initialized. Prior work has demonstrated that the CPU cache subsystem can be repurposed as a general-purpose memory area for isolated code execution and data read/write operations [24]. The CPU CAR environment offers an isolated execution environment using exclusively on-die hardware.

3.4 Instruction Set Extension

This section provides a high-level overview of the design — describing the requirements, execution model, and implementation rationale for the instruction set extension proposed in this paper. The notation used in the remainder of the chapter is summarized in Table 3.1.

3.4.1 Processor Requirements

OASIS is a set of new CPU instructions that aim to enable an isolated execution environment contained entirely on-chip by leveraging CAR mode execution and by creating a secret key available only to the CPU. OASIS is designed for ease of adoption and deployment with respect to existing computing systems.

A critical support for the level of security achievable by OASIS-enabled software is the PUF-derived secret key K_p. This secret key material is only briefly established during the short-lived moment needed to generate derivative platform keys, is only accessible to the CPU by construction, and functions as the root of trust of the whole isolated execution environment.

OASIS is based on SRAM-PUFs [66, 67]. This has several advantages: (i) SRAM is

Table 3.1: Notation Used in Instruction Set and Protocol

Notation	
`hw_inst[]`	hardware instructions that make up the OASIS programming interface are denoted using a fixed-width font
`f_hw_func[]`	hardware functions are only accessible by OASIS hardware instructions and are denoted using a fixed-width font identifier starting with the letter `f`
$y \leftarrow x$	the value of x is assigned to variable y
$\perp$	this symbol is used to denote a failed platform operation
$x \| \| y$	concatenation of x and y
$x.param$	returns parameter $param$ of variable x
$x.*$	data element formed by concatenating all parameters of variable x
$A \rightarrow B : \langle m \rangle$	A sends message $\langle m \rangle$ to B
$r \xleftarrow{R} \{0,1\}^{\ell}$	assigns a random integer of ℓ bits to r
K_X	party X's symmetric key
K_X^+, K_X^{-1}	party X's public and private asymmetric key pair
$\{P\}_K$	the resulting ciphertext of plaintext P encrypted using key K
$H(x)$	cryptographic hash function with input x
$\mathsf{Enc}_K(P)$	encrypt plaintext P using key K
$\mathsf{Dec}_K(C)$	decrypt ciphertext C using key K
$\mathrm{KDF}_K(x)$	key derivation function of key K and non-secret parameter x
$\mathrm{MAC}_K(x)$	message authentication code of x under key K
$\mathtt{Sign}_{K_X^{-1}}(m)$	sign message m with party X's private key K_X^-
$\mathtt{Verify}_{K_X^+}(m, \sigma)$	verify signature σ on message m using party X's public key K_X^+
$\mathtt{Cert}_y(x, K_X^+)$	certificate issued by y that binds the identity x to the public key K_X^+

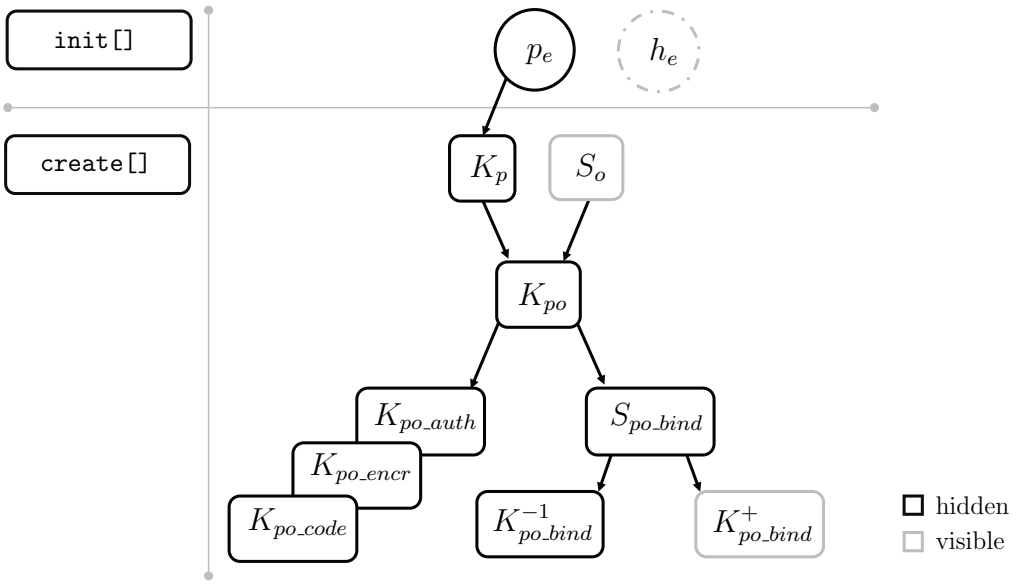


Figure 3.2: Key Generation Hierarchy. These keys are generated as the `init[]` and `create[]` instructions are called. Hidden key material refers to platform secrets only accessible by the processor. Visible key material refers to non-secret key material. Key material that is unique to the PUF element, the platform, or the owner are denoted with subscripts e, p, or o, respectively.

already available on modern CPUs in the form of the cache, (ii) SRAM PUFs need to be powered to create the secret key material, thus, they cannot be read offline making them resistant against scanning electron microscope-based attacks, (iii) because of their properties, PUFs are tamper-evident (and in some cases tamper-resistant), a property that other technologies do not offer [66], and (iv) SRAM is manufactured using the standard semiconductor process, which leads to decreased costs when compared to non-volatile memory.

OASIS assumes the availability of external non-secure non-volatile memory. This memory is used to store public helper data as well as program state. External storage is plentiful and does not further complicate the OASIS design since no special security guarantees are assumed. In particular, alterations to the public helper data can be easily detected through the use of robust fuzzy extractors [75, 76].

Table 3.2: Variables Used in Instruction Set and Protocol

Hidden Variables: values accessible by processor	
p_e	Raw PUF response
K_p	Root key generated from PUF
S_o	Secret seed value set by platform owner
p^*,q^*	Primes corresponding to an RSA private key
$CR.K_{po}$	Master platform secret for a specific owner seed
$CR.K_{po\_auth}$	Platform key for authenticating data from untrusted storage
$CR.K_{po\_encr}$	Platform key for encrypting data before transfer to untrusted storage
$CR.K_{po\_code}$	Platform key used to derive code specific keys
$CR.S_{po\_bind}$	Platform binding secret used to derive asymmetric binding keys
$CR.K^{-1}_{po\_bind}$	Platform private binding key, derived deterministically from $CR.S_{po\_bind}$
$CR.PCR$	Platform configuration registers
$CR.K_C$	Unique cryptographic key for code $\mathbf{C}'$
Visible Variables: values accessible by software	
$K^+_{po\_bind}$	Platform public binding key, derived deterministically from $CR.S_{po\_bind}$
h_e	Helper data used for noise reduction of p_e
h_{PK}	Helper data used for retrieving asymmetric keys

3.4.2 Root of Trust Instantiation

The SRAM-PUF response, p_e, serves as a unique cryptographic secret which is used to bootstrap a unique device identity, per-application encryption and authentication keys, and random number generation. The resulting key material (Figure 3.2) is unique not just per physical device, but per device owner. The SRAM-PUF response is used to derive the secret root key, K_p, which never leaves the processor and is never directly accessible by any party (including any software running on the processor).

The PUF-derived secret root key, K_p, enables the derivation of a key hierarchy as follows. The device owner derives a key (K_{po}) unique to themselves and the device via a key derivation function (KDF), which accepts as inputs an owner supplied seed, S_o, and the PUF-derived secret root key, K_p.

This master processor secret, K_{po}, can then be used, in turn, to derive symmetric keys for bulk encryption, authentication, and asymmetric operations. The details for the key derivation are discussed in terms of the invoked functions and instructions in Section 3.4.4 and Section 3.4.5.

All keys are stored inside the CPU in a set of special-purpose cache registers ($CR.*$) which are only available within the OASIS environment and only accessible by the OASIS instructions. Table 3.2 lists the keys stored in $CR.*$. Observe that the root key, K_p, is only used for the derivation of the master processor secret. More importantly, the entire key hierarchy is based on an owner seed (S_o), enabling personalization and device transferability.

3.4.3 Instruction Set Overview

So far we have described how the CAR environment has been augmented with a secret key (via the PUF) to be used as the root of trust in a transparent and low-cost manner. Next, we describe how the PUF-based root of trust is used to enable the desired security objectives of Section 3.2.2 by defining five new instructions: `init[]`, `create[]`, `launch[]`, `unbind[]`, and `bind[]` (Figure 3.3).

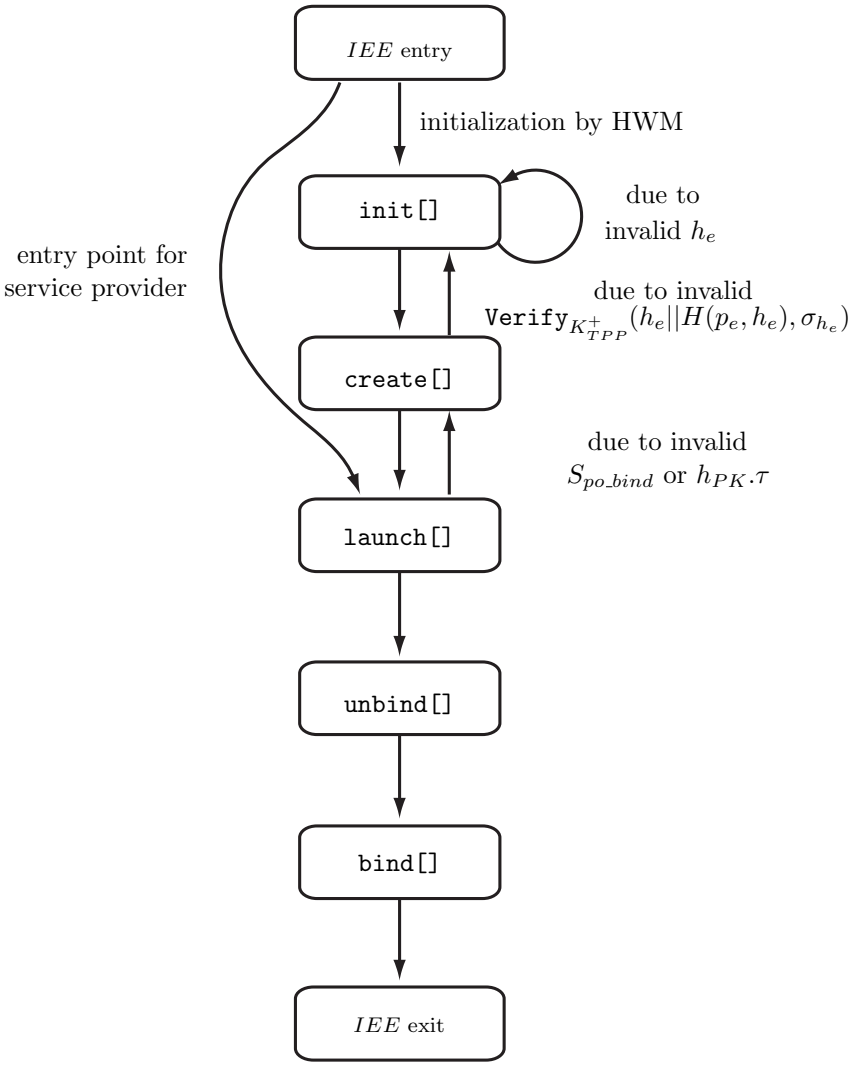


Figure 3.3: Invocation Life-cycle. This figure shows the call order for the OASIS instruction set where the labelled lines indicate special case instruction flows.

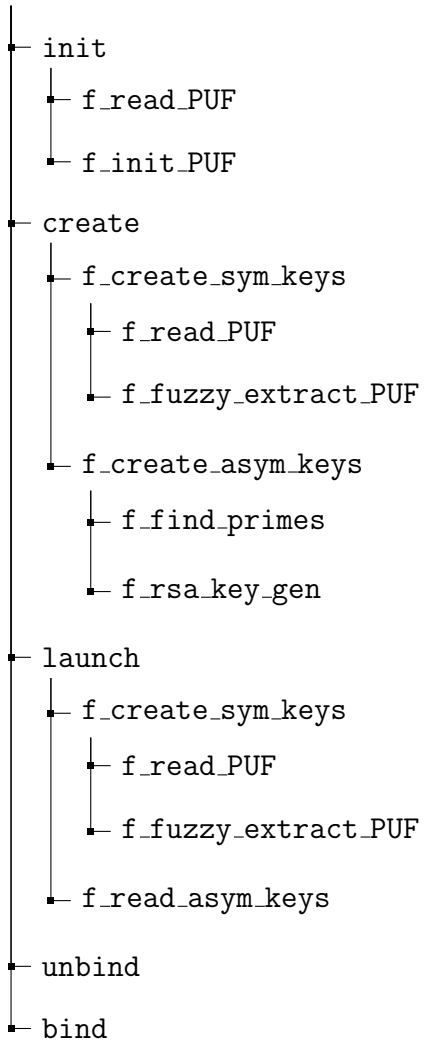


Figure 3.4: Function Tree. This figure shows the function call stack for the OASIS instruction set. Instructions are *externally available* for call by executing software whereas functions are *internally available* to OASIS instructions.

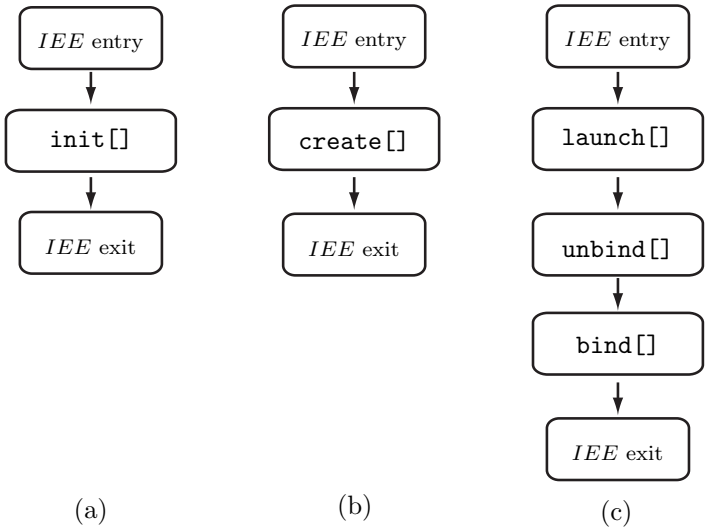


Figure 3.5: Invocation by Session. This figure shows the OASIS session during (a) initialization by the manufacturer, (b) setup by the device owner, and (c) code execution by the user.

Initialization. We distinguish between three stages in the life cycle of the CPU. The first stage is performed by the hardware manufacturer (Figure 3.5(a)). After manufacture, the HWM initializes the master processor key K_p by calling `init[]`. The output of this operation is helper data h_e and a hash $H(p_e, h_e)$, which is published and available to anyone using the device.

We assume that the instruction `init[]` can be called only once or a limited number of times to prevent attacks that exploit repeated invocations of the generator function `f_init_PUF[]` to learn p_e.[3] In addition to preventing these types of attacks, this constraint allows for a trusted third party to certify the helper data via a standard public-key signature $\sigma_{h_e} \leftarrow \texttt{Sign}_{K_{TTP}^{-1}}(h_e||H(p_e, h_e))$, which can be verified in a straight forward manner with the public key K_{TTP}^{+} of the TTP. This verification step need only be performed once per owner during the initial call to `create[]` (Figure 3.3).

Given that the HWM does not have control of the PUF response p_e (or by extension K_p

[3]The security of robust fuzzy extractors is directly related to the number of queries performed during an attack [76]. It has been shown that, for non-robust fuzzy extractors, given a sufficiently large number of invocations to the generator function `f_init_PUF[]`, the resulting public outputs h_{e_1},..., h_{e_n} contain enough information to leak the secret input p_e with high probability [77].

as it is derived from p_e), the `init[]` instruction enables the limited trust ($P1.5$), low cost ($P2.1$), self contained ($P2.2$), and the minimal TCB ($P3.1$) properties of Section 3.2.2.

Configuration. The second stage is performed by the device owner (Figure 3.5(b)) and corresponds to the generation of the key hierarchy (Figure 3.2). This is accomplished by calling the `create[]` instruction, which has as its primary function the derivation of symmetric and asymmetric keys. During the call to `create[]`, the platform owner generates keys that are specific both with respect to the owner and the device so long as the secrecy of input seed S_o is maintained.

These keys are used to exchange confidential and authenticated messages between the prover (device owner) and the verifier (user) and to guarantee external verifiability ($P1.1$). The main output of the instruction is a public key, which has been derived from the PUF-based root key K_p and a seed S_o known only to the device owner. This allows for transferability of the platform ($P1.4$) as a new device owner can create his or her own public/private keypair $(K^{+}_{po\_bind}, K^{-1}_{po\_bind})$ by choosing a different seed S'_o. Furthermore, even though the device owner initiates the generation of the public/private keypair, only the CPU can access the private key and thus decrypt messages encrypted with the public key.

Code Execution. The third stage corresponds to the execution of code on the device by the user (Figure 3.5(c)). The user launches the code to be executed by issuing the `launch[]` instruction. This instruction populates the $CR.*$ registers with the symmetric keys derived from the PUF helper data h_e, the device owner's seed S_o, and the public key information generated using `create[]` in the previous stage. Then, the `unbind[]` instruction can be called to check the input's integrity with respect to a code-specific key and decrypt any input whose confidentiality is preserved by the verifier. The instruction provides two options, one using public-key and one using symmetric-key primitives. The asymmetric option is used the first time the application is called to transmit a secret symmetric key, K_{VP}, only known to the verifier (user) of the platform P. After this initial set-up, the verifier can use fast symmetric-key operations to verify the integrity and confidentiality of its data ($P3.3$). At this point the

code C can be executed in the isolated execution environment, state is saved (and encrypted if desired), and integrity information is computed on the state using `bind[]` ($P1.3$). Finally, all OASIS memory and internal registers are cleared out, and control is returned to the OS. Observe that any program can in principle be executed in a secure environment using these last three instructions, providing for a minimal and simple programming interface ($P3.2$). Furthermore, the `bind[]` and `unbind[]` instructions, together with the key hierarchy derived with the help of `create[]`, enable external verifiability ($P1.1$) and program state binding ($P1.2$), not only to a particular program but also to a specific CPU, a property unique to OASIS.

3.4.4 Functions

We now describe the functions (Section 3.4.4) and instructions (Section 3.4.5) used in the design of OASIS. We make a distinction between functions (which are only *internally available* to OASIS instructions) and instructions (which are *externally available* for call by executing software).[4] In practice, functions and instructions might be implemented as digital logic, integrity-checked firmware, microcode, or another process-specific mechanism. Refer to Figure 3.4 for an overview of the organization of functions and instructions.

We have omitted explicit pseudocode definitions for several functions where the specific implementation is left to the hardware manufacturer. Table 3.3 lists these functions. The functionality of `f_read_PUF[], f_init_PUF[]` and `f_fuzzy_extract_PUF[]` are briefly discussed next. Implementation details for the `f_find_primes[]` [5] and `f_rsa_key_gen[]` are discussed in the section on function `f_create_asym_keys[]` (Function $3.2a$) which is invoked by the `create[]` instruction (Instruction 2).

The function `f_read_PUF[]` does not accept any inputs; it simply outputs the raw PUF response p_e. We provide two functions to interact with a (robust) fuzzy extractor [75, 76] as

[4] Instructions and functions are denoted using a fixed-width identifier. Functions begin with '`f_`'.

[5] Including an alternate higher performance implementation of `f_find_primes[]`: $p, q, \delta_p, \delta_q \leftarrow$ `f_find_primes[`S_{po_bind}`, RSAParam.size]` and $p, q \leftarrow$ `f_retrieve_primes[`$S_{po\_bind}, \delta_p, \delta_q$`]`

Table 3.3: Hardware Manufacturer Implemented Functions

p_e	$\leftarrow$ `f_read_PUF[]`
$h_e, H(p_e, h_e)$	$\leftarrow$ `f_init_PUF[`$p_e, rand$`]`
K_p	$\leftarrow$ `f_fuzzy_extract_PUF[`$p_e, h_e, H(p_e, h_e)$`]`
p, q	$\leftarrow$ `f_find_primes[`S_{po_bind}`, RSAParam.size]`
$K^{+}_{po\_bind}$, $K^{-1}_{po\_bind}$	$\leftarrow$ `f_rsa_key_gen[`$p, q,\ e$`]`

is common in the literature: (1) `f_init_PUF[]` and (2) `f_fuzzy_extract_PUF[]`.

The function `f_init_PUF[`$p_e, rand$`]`, which is called during the execution of the `init[]` instruction (Instruction 1), accepts a raw PUF response p_e and a random value $rand$ and outputs helper data h_e and a hash $H(p_e, h_e)$. The helper data h_e can be used to reconstruct a uniformly random value K_p from a noisy raw PUF response p'_e. The hash is used to guarantee that only values of K_p constructed with the original helper data h_e are used for further processing in OASIS.

The function `fuzzy_extract_PUF` implements a (robust) fuzzy extractor [75, 76], whose output is K_p. Called during the execution of the `create[]` instruction (Instruction 2), `f_fuzzy_extract_PUF`$[p'_e, h_e, H(p_e, h_e)]$ accepts a (noisy) raw PUF response p'_e and helper data h_e and outputs a uniformly random value K_p which can be used as a cryptographic key.[6] We assume the use of existing hardware-supported fuzzy extractor implementations [78, 79]. Alternatively, this can be combined with a controlled PUF (CPUF), which are PUFs that can only be accessed through an algorithm that is physically bound to the PUF [64]. Such an interface would further limit the information learned by an attacker who can probe the challenge-response behavior of the PUF. The function `f_fuzzy_extract_PUF[]` checks for correctness in the value of $H(p_e, h_e)$ and outputs a special symbol $\perp$ if the input does not correspond to the computed value. If the output is $\perp$, the instruction calling `f_fuzzy_`

[6]The original definition of a robust fuzzy extractor requires a label as input to the fuzzy extractor, which is used to define a member of a family of universal hash functions, which is used to create a uniform K_p [76]. Here we have abstracted out this detail for ease of exposition.

`extract_PUF[]` should take appropriate action. In the case of OASIS, we clear all key registers and abort execution.

Function 3.1: `f_create_sym_keys[]`

INPUT: $S_o, h_e, H(p_e, h_e)$

OUTPUT: S_{po_bind}

1: $p'_e \leftarrow$ `f_read_PUF[]`
2: $K_p \leftarrow$ `f_fuzzy_extract_PUF[`$p'_e, h_e, H(p_e, h_e)$`]`
3: Clear p'_e
4: $CR.K_{po} \leftarrow KDF_{K_p}(S_o)$
5: $S_{po\_bind} \leftarrow KDF_{CR.K_{po}}$("bind")
6: $CR.K_{po\_auth} \leftarrow KDF_{CR.K_{po}}$("auth")
7: $CR.K_{po\_encr} \leftarrow KDF_{CR.K_{po}}$("encr")
8: $CR.K_{po\_code} \leftarrow KDF_{CR.K_{po}}$("code")
9: **if** $K_p = \perp$ **then**
10: Clear $CR.*$
11: $S_{po\_bind} \leftarrow \perp$
12: Clear K_p
13: **return** S_{po_bind}

Function 3.1. This function loads the helper parameter h_e and the hash $H(p_e, h_e)$ into memory.[7] Next, the PUF is read and the fuzzy extractor is invoked to generate the platform symmetric secret key, K_p. Internally, the fuzzy extractor checks whether the inputs $H(p_e, h_e)$ and h_e correspond to the reconstituted PUF response p'_e. A special symbol $\perp$ is output should the values be different.

[7] A small portion of on-chip SRAM can be reserved for loading helper data as needed by the instructions. Typically, helper data for an SRAM PUF requires no more than 10 K bytes of memory [78, 79].

The key K_p and the (device owner) supplied seed S_o are used to derive the master processor secret, $CR.K_{po}$. The seed value S_o allows the device owner to personalize the processor keys. The symmetric key $CR.K_{po}$ is used for the derivation of four symmetric platform keys: (i) $CR.S_{po\_bind}$, the platform binding secret, (ii) $CR.K_{po\_auth}$, the platform key used for authenticating data residing in untrusted storage from prior invocations (iii) $CR.K_{po\_encr}$, the platform key used for encrypting data and (iv) $CR.K_{po\_code}$, the platform key used to derive code-specific keys. In all cases, keys are derived via a KDF, which in turn may use pseudo-random functions (e.g., HMAC, CMAC) as building blocks. Constructions of key derivation functions accepting secret and public parameters are well-known [80, 81]. At the end of the process, the function checks if the fuzzy extractor returned the special symbol $\perp$, which would indicate that either the PUF response was too noisy and therefore it was not possible to reconstruct K_p or $H(p_e, h_e) \neq H(p'_e, h'_e)$.[8] In either case, all OASIS registers are cleared and the function returns the special symbol $\perp$ indicating failure. After the check, K_p is cleared and S_{po_bind} is returned.

[8]The function `f_create_sym_keys` is written so as to always perform the same number of operations regardless of whether the fuzzy extractor returns $\perp$ or not. This helps in preventing leaking information via timing attacks. In practice, this would require dummy operations erasing dummy memory at the end of the **if** statement but this has not been included for ease of exposition.

Function 3.2*a*: `f_create_asym_keys[]` (using RSA key generation)

INPUT: S_{po_bind}

OUTPUT: h_{PK}

1: $p, q, \leftarrow$ `f_find_primes[`S_{po_bind}, `RSAParam.size]`

2: $\left\{ \begin{array}{c} K^{+}_{po\_bind}, \\ CR.K^{-1}_{po\_bind} \end{array} \right\} \leftarrow$ `f_rsa_key_gen[`p, q, e`]`

3: $\{K^{-1}_{po\_bind}\}_{CR.K_{po\_encr}} \leftarrow \mathsf{Enc}_{CR.K_{po\_encr}}(CR.K^{-1}_{po\_bind})$

4: $\tau \leftarrow \mathrm{MAC}_{CR.K_{po\_auth}}(\{K^{-1}_{po\_bind}\}_{CR.K_{po\_encr}}, K^{+}_{po\_bind})$

5: $h_{PK} \leftarrow \left\{\{K^{-1}_{po\_bind}\}_{CR.K_{po\_encr}}, K^{+}_{po\_bind}, \tau\right\}$

6: **if** $S_{po\_bind} = \perp$ **then**

7: Clear $CR.*$

8: Clear h_{PK}

9: **return** h_{PK}

Function 3.2*a*. This function generates the processor asymmetric keys. The `f_find_primes[]` function picks a random seed value of size `RSAParam.size` and begins search until the first prime is found. The process is repeated for the second prime using a new seed value. `f_find_primes[]` returns secret primes, p and q. The function `f_rsa_key_gen[]` takes the primes and a public exponent as inputs and generates the keypair $K^{+}_{po\_bind}, K^{-1}_{po\_bind}$. Notice that the RSA private key $K^{-1}_{po\_bind}$ is composed of p, q, and the inverse of the RSA public exponent $\bmod \phi(N)$, where $N = p \cdot q$. Methodologies to generate primes are well-understood and standardized [82]. The RSA private key $K^{-1}_{po\_bind}$ is encrypted using $CR.K_{po\_encr}$, and a message authentication code τ is computed over this value and the corresponding public key $K^{+}_{po\_bind}$. Finally, a data store h_{PK}, containing the asymmetric keys and τ, is returned.

Extended Implementation. Functions $p, q, \delta_p, \delta_q \leftarrow$ `f_find_primes[`S_{po_bind}`, RSAParam.size]` and $p, q \leftarrow$ `f_retrieve_primes[`$S_{po\_bind}, \delta_p, \delta_q$`]` describe a methodology which circumvents

the prime search and, thus, it is much more efficient than the $p, q \leftarrow$ `f_find_primes[`S_{po_bind}`, RSAParam.size]` used in function 3.2*a*. We remark that we are not aware of it being described in other works and thus, it might be of independent interest:

This function generates the processor asymmetric keys. The `f_find_primes[]` function picks a random seed value of size `RSAParam.size` and begins search until the first prime is found. The process is repeated for the second prime using a new seed value. The `f_find_primes[]` returns secret primes, p and q, and their offsets from the seed values, δ_p and δ_q. The function `f_rsa_key_gen[]` takes the primes and a public exponent as inputs and generates the keypair $K^{+}_{po\_bind}, K^{-1}_{po\_bind}$.

The RSA helper parameters, δ_p and δ_q, are stored to avoid the expensive search for primes in subsequent calls. The parameters are non-secret, however, they need to be checked for authenticity before future use. Therefore, we construct a MAC of δ_p, δ_q, and $K^{+}_{po\_bind}$ under platform data authentication key K_{po_auth}. The resulting asymmetric keys $(K^{+}_{po\_bind}, K^{-1}_{po\_bind})$ are stored to the register bank CR and the MAC, the offsets, and the public key are returned.

Methodologies to generate primes are well-understood and standardized [82].[9] Function 3.2*a*, however, has a few additional features, which are worth highlighting. The first time the RSA primes are generated, the `f_find_primes[]` subroutine proceeds as follows:

1. Generate a random value s_p from S_{po_bind} of size `RSAParam.size` bits.
2. Ensure that s' is odd: $s' = s_p|1$.[10]
3. Check s' for primality. If s' is prime:
   (a) set $p = s'$, store $\delta_p = p - s_p$.
   (b) go to step 1, using random value s_q to generate q and δ_q.
4. Else $s' = s' + 2$, go to step 3.

[9] We have sketched roughly how standard prime generation works. This methodology can be easily modified to accommodate the requirement of *strong* primes.

[10] Where "|" denotes the bitwise OR operator.

Once a key pair is generated, one could go through the process of searching and generating primes again when the key pair needs to be used. However, this is unnecessarily time-consuming. Given the offsets, the RSA primes are generated using the `f_retrieve_primes[]` subroutine as follows:

1. Generate the random seed value s_p from S_{po_bind}.
2. Set $p = s_p + \delta_p$.
3. Generate second random value s_q.
4. Set $q = s_q + \delta_q$.
5. Call `f_rsa_key_gen`$[p, q,\text{e}]$ to generate $K^{+}_{po\_bind}, K^{-1}_{po\_bind}$.

Observe that the expensive prime search and prime testing procedures required during the original key generation are no longer needed thanks to the pointers δ_p, δ_q. In particular, Riemann's Prime Number Theorem holds that the density of prime numbers is $n/\ln(n)$ for the range 1 to n. Therefore, if we ignore even numbers in our search, the probability of randomly selecting a prime number within the range 1 to n is $2/\ln(n)$. Assume we use a 2048 bit RSA modulus; the bit-length required to provide data protection guarantees into the year 2022 [83]. The probability that the k-th number is a prime follows a Geometric distribution. Therefore, we would need to search roughly 350 numbers on average before finding primes of 1024 bits or less ($p(X^* = k) = (1-p)^{k-1} * p$ where $p = 2/\ln(2^{1024})$ and $E[X^*] = 1/p$). We would like to avoid the performance hit associated with searching and testing for RSA primes every time this instruction is called. Thus, we propose to store the plain pointers δ_p, δ_q to the specific primes in untrusted memory. To prevent tampering, a MAC is computed on the plain pointers and the RSA (public) modulus. The alternative is to incur a performance hit by searching and testing the integers until the desired primes are found, which may be undesirable.

Regarding the secrecy of RSA primes p and q, one could encrypt the pointers δ_p and δ_q under key K_{po}. However, the following argument provides evidence that this is not necessary. Formally, what we would like to show is that given difference pointer δ_p, the amount

of information leaked about p (or equivalently about s_p) is negligible. The seed s_p is chosen uniformly at random from an interval $(2^k, 2^{k+1} - 1)$, thus, s_p is uniformly distributed in $[1, 2^{-k}]$. For large k and small (relative to s_p) δ_p, the distribution of $s_p + \delta_p$ can be shown to be *statistically* indistinguishable from random with statistical distance $< 1/2^{k-\log_2 \delta_p}$. To show that in general δ_p is small, we resort to a conjecture by Cramér [84] and subsequent improvements [85], which estimate the maximal gap between a prime p and the next successive prime to be $\ll \ln^{2+\epsilon} p$. In practice, the largest gaps found are less than 2^{16} for very large primes (see [86] for extensive lists), thus, the amount of information leaked by the pointers can be considered to be negligible.

Function 3.2*b*: `f_create_asym_keys[]` (using ECC key generation)

INPUT: S_{po_bind}

OUTPUT: h_{PK}

1: $\left\{ \begin{matrix} K^{+}_{po\_bind}, \\ CR.K^{-1}_{po\_bind} \end{matrix} \right\} \leftarrow$ `f_ecc_key_gen[`S_{po_bind}`, ECCParam]`

2: $\{K^{-1}_{po\_bind}\}_{CR.K_{po\_encr}} \leftarrow \mathsf{Enc}_{CR.K_{po\_encr}}(CR.K^{-1}_{po\_bind})$

3: $\tau \leftarrow \mathrm{MAC}_{CR.K_{po\_auth}}(\{K^{-1}_{po\_bind}\}_{CR.K_{po\_encr}}, K^{+}_{po\_bind})$

4: $h_{PK} \leftarrow \left\{\{K^{-1}_{po\_bind}\}_{CR.K_{po\_encr}}, K^{+}_{po\_bind}, \tau\right\}$

5: **if** $S_{po\_bind} = \perp$ **then**

6: Clear $CR.*$

7: Clear h_{PK}

8: **return** h_{PK}

Function 3.2*b*. We describe an alternative implementation of the `f_create_asym_keys[]` (Function 3.2*a*) using elliptic curves in Function 3.2*b*. The implementation of this function is analogous but much more efficient than its RSA counterpart, since there is no prime search step. Key generation is a single elliptic curve multiplication, which in general is efficient. In

addition, this version has the advantage of small area overhead, if support for asymmetric operations is implemented at the hardware level. These advantages come at the cost of a significant increase in the time required to perform a signature verification operation (when compared to RSA). It is up to the HWM to decide which implementation is more appropriate based on its own requirements and constraints.

Function 3.3: `f_read_asym_keys[]`

INPUT: h_{PK}

OUTPUT: $K^{+}_{po\_bind}$

1: $\tau' \leftarrow \text{MAC}_{CR.K_{po\_auth}}\left(h_{PK}.\left\{\{K^{-1}_{po\_bind}\}_{CR.K_{po\_encr}}, K^{+}_{po\_bind}\right\}\right)$
2: **if** $h_{PK}.\tau \neq \tau'$ **then**
3: Clear $CR.*$
4: Clear $h_{PK}.K^{+}_{po\_bind}$
5: **else**
6: $CR.K^{-1}_{po\_bind} \leftarrow \text{Dec}_{CR.K_{po\_encr}}(h_{PK}.\{K^{-1}_{po\_bind}\}_{CR.K_{po\_encr}})$
7: **return** $h_{PK}.K^{+}_{po\_bind}$

Function 3.3. This function is very efficient as it only requires symmetric cryptographic operations. In particular, `f_read_asym_keys[]` checks tag $h_{PK}.\tau$ to ensure that input data has not been tampered with. If this verification passes, the function decrypts the private binding key to $CR.K^{-1}_{po\_bind}$, using the symmetric key $CR.K_{po\_encr}$. Note that the corresponding read functions, for create functions 3.2*a* and 3.2*b*, are the same except for the sizes of the operands, outputs, and registers required to store private and public keys.

3.4.5 Instructions

Instruction 1: `init[]`

INPUT:

OUTPUT: $\{h_e, H(p_e, h_e)\}$

1: $p_e \leftarrow$ `f_read_PUF[]`
2: $rand \xleftarrow{R} \{0,1\}^{\ell}$
3: $\{h_e, H(p_e, h_e)\} \leftarrow$ `f_init_PUF[`$p_e, rand$`]`
4: Clear $p_e, rand$
5: **return** $\{h_e, H(p_e, h_e)\}$

Instruction 1. This instruction initializes the helper data h_e used to de-noise the raw SRAM PUF value p_e. The functions `f_read_PUF[]` and `f_init_PUF[]` read the raw PUF value and instantiate the helper data, as described in Section 3.4.4. The hash value $H(p_e, h_e)$ will be used by later instructions to prevent modified helper data from being used in attempts to learn information about the PUF. Observe that a hardware-generated random number, *rand*, is used to introduce entropy in the resulting helper data's value.

The variable *rand* needs to remain secret and exposed only inside the processor. It is also assumed that h_e can only be set once (or a limited number of times) to prevent exposing the output of the fuzzy extractor. This can be achieved during the initialization, which is performed by the HWM. Because of our use of *robust* fuzzy extractors [75, 76], we do not require *any secure* non-volatile memory. All data is stored outside the chip, either locally or externally published on a website. An additional step, not shown and not performed as part of Instruction 1 is the signing of $h_e||H(p_e, h_e)$ by the HWM or a TTP with output $\sigma_{h_e} \leftarrow \texttt{Sign}_{K_{TTP}^{-1}}(h_e||H(p_e, h_e))$. This guarantees to any third party (users, system integrators, device owners, etc.) that the helper data was created by the HWM and not some other (untrusted) party. Notice this is done only once during the lifetime of the device.

Instruction 2: `create[]`

INPUT: $S_o, h_e, H(p_e, h_e), \sigma_{h_e}, K^+_{TTP}$

OUTPUT: h_{PK}

1: **if** $\texttt{Verify}_{K^+_{TTP}}(h_e||H(p_e, h_e), \sigma_{h_e}) = \texttt{accept}$ **then**
2: $S_{po\_bind} \leftarrow$ `f_create_sym_keys[`$S_o, h_e, H(p_e, h_e)$`]`
3: $h_{PK} \leftarrow$ `f_create_asym_keys[`S_{po_bind}`]`
4: Clear S_{po_bind}
5: **return** h_{PK}
6: **else**
7: ABORT

Instruction 2. This instruction generates a hierarchy of cryptographic keys from the raw PUF response p_e. Symmetric and asymmetric keys are generated by `f_create_sym_keys[]` (Function 3.1) and `f_create_asym_keys[]` (Function 3.2*a* or 3.2*b*), respectively.

The h_{PK} variable is assigned the $\{K^+_{po\_bind}, K^{-1}_{po\_bind}\}$ keypair generated by `f_create_asym_keys[]`. Observe that h_{PK} is encrypted and contains authentication information, which is verified internally by OASIS using a key derived from the internal PUF key and the seed S_o. Lastly, note that verification of the signature σ_{h_e} is most efficient if the signature algorithm is based on RSA using a small exponent (e.g., $3, 17$, or $2^{16} + 1$). Regardless of the latency due to signature verification, we expect that this step is performed rarely – e.g., whenever the device changes ownership or if a user desires to set up the environment for future use.

Instruction 3: `launch[]`

INPUT: $\mathbf{C}, \mathbf{C}.inputs, S_o, h_e, H(p_e, h_e), h_{PK}$

OUTPUT:

1: Configure CPU into CAR Mode
2: Load **C** into the CPU cache
3: $S_{po\_bind} \leftarrow$ `f_create_sym_keys[`$S_o, h_e, H(p_e, h_e)$`]`
4: $K^{+}_{po\_bind} \leftarrow$ `f_read_asym_keys[`h_{PK}`]`
5: $CR.PCR \leftarrow H(\mathbf{C})$
6: $CR.K_C \leftarrow KDF_{CR.K_{po\_code}}(H(\mathbf{C}))$
7: **if** $(S_{po\_bind} = \perp)$ **then**
8: Clear $CR.*$
9: ABORT
10: Clear S_{po_bind}
11: Transfer control to **C**'s entry point

Instruction 3. The `launch[]` instruction is designed to set up the OASIS environment for code **C** and populate all necessary registers. It begins by setting up a clean-slate CAR environment, including disabling interrupts and hardware debugging access. It then reads and loads $CR.*$ registers with cryptographic key material for further processing by other instructions.[11]

To avoid the expensive operations performed in `create[]` for asymmetric key generation (e.g., prime generation), an encrypted data store h_{PK} is returned by `f_create_asym_keys[]` and `f_read_asym_keys[`h_{PK}`]` is used on subsequent invocations. This function's overhead is equivalent to a few efficient *symmetric-key* operations.

[11] A possible optimization is to conditionally invoke `f_create_sym_keys[]` and `f_read_asym_keys[]`. For example, `launch[]` can be modified to only invoke `f_create_sym_keys[]` once after the processor reboots and maintain the resulting keys in CR.* during successive OASIS sessions. This optimization must be carefully considered and constructed by the implementer to manage the security trade-off (PUF-derived secrets persisting between invocations). Additionally, the call to `f_read_asym_keys[]` may be skipped for sessions that only require symmetric keys.

Observe that if we want to make the public binding key available outside the environment, Instruction 2 must be called first. Also note that Instruction 2 verifies the signature σ_{h_e} every time it executes, whereas Instruction 3 does not. We expect that signature verification will be performed at most once *per session*, where each session might call the `launch[]` instruction multiple times. Notice that even if the signature verification function is performed every time, the overhead should be minimal, assuming RSA signatures. Refer to Figure 3.5 for details on when instructions are called.

Next, `launch[]` stores a hash of the target code **C** to the platform configuration register $CR.PCR$. Finally, a symmetric key K_C is generated using a key derivation function based on $CR.K_{po\_code}$ and a hash of target code **C**. K_C is used for encrypting and authenticating the executing code's *state* for local storage to untrusted memory.

At the end of `launch[]`, the following registers have been populated: $CR.K_{po}$, $CR.K_{po\_auth}$, $CR.K_{po\_encr}$, $CR.K_{po\_code}$, $CR.K^{-1}_{po\_bind}$, $CR.PCR$, and $CR.K_C$.

Instruction 4: `unbind[]`

INPUT: $\{X_1, PCR\_ver\}_{K^+_{po\_bind}}, \{X_2, PCR\_ver\}_{K_C}$

OUTPUT: X

1: **if** $\{X_1, PCR\_ver\}_{K^+_{po\_bind}} \neq NULL$ **then**
2: $\quad X, PCR\_ver \leftarrow \mathsf{Dec}_{CR.K^{-1}_{po\_bind}}(\{X_1, PCR\_ver\}_{K^+_{po\_bind}})$
3: **else if** $\{X_2, PCR\_ver\}_{K_C} \neq NULL$ **then**
4: $\quad X, PCR\_ver \leftarrow \mathsf{AuthDec}_{CR.K_C}(\{X_2, PCR\_ver\}_{K_C})$
5: **else**
6: $\quad X \leftarrow \perp$
7:
8: **if** $CR.PCR \neq PCR\_ver$ **then**
9: $\quad X \leftarrow \perp$
10: **return** X

Instruction 4. Inputs X_1 and X_2 contain data values that should only be released to the code that generated the data. The unbind instruction provides assurance to the verifier that the inputs will only be released to the code with measurement PCR_ver. Note that `unbind[]` can decrypt data encrypted under either of the binding key $K^+_{po\_bind}$ or the application secret key K_C.

In the protocol described in Section 3.5, included in X_2 is a symmetric key K_{VP}, which is generated by the verifier V for bulk encryption of data to be transferred between V and the platform P. Notice that we do not suggest using the public binding key, $K^+_{po\_bind}$, for bulk encryption. Instead, symmetric keys should be used for bulk encryption operations and the public binding key for storing bulk encryption keys. This is a common practice used to avoid the performance cost of public key cryptography.

In choosing the asymmetric encryption scheme, some care must be taken to prevent an attacker from using the ciphertext $\{X, PCR\_ver\}_{K^+_{po\_bind}}$, which is intended to be decrypted only by the code with measurement PCR_ver, to generate a related ciphertext $\{X, PCR\_ver'\}_{K^+_{po\_bind}}$, which the device would be willing to decrypt for different code with measurement PCR_ver'. To prevent this, the encryption scheme must be *non-malleable* – i.e., an attacker cannot use one ciphertext to generate a second ciphertext that decrypts to a plaintext related to the original plaintext. The formal definition of non-malleable is known as Chosen Ciphertex Attack of type 2 or CCA2. Examples of CCA2 (non-malleable) asymmetric encryption schemes include RSA-OAEP and RSA-OAEP+ [87].[12] An alternative strategy to using a non-malleable public-key encryption scheme is to use the secret encrypted with the asymmetric primitive to derive two keys: an encryption key and a MAC key. The MAC key should be used to compute a MAC over the bulk-encrypted ciphertext,

[12]Note that it is possible for an encryption scheme to be semantically secure while still being malleable [88]. For example, in a hybrid scheme where RSA is used to encrypt a symmetric key, which is in turn used in a block cipher to encrypt the bulk data, then clearly the last block of the bulk-encrypted data can be modified without changing the decryption of the preceding plaintext blocks. This could allow the attacker to change the specified PCR if it appears at the end of bulk encrypted data. Even if the authorized PCR is at the beginning, the attacker would still be able to modify the end of the bulk data without changing the value of the preceding ciphertext.

and the receiver should reject ciphertexts with an inconsistent MAC. This is the strategy used in the Integrated Encryption Scheme [89]. In this work, we simply assume that we are using a CCA2 public-key encryption scheme regardless of its particular implementation.

Instruction 5: `bind[]`

INPUT: *out*

OUTPUT: $K_{VP}, stateOS, hashInputs, resultV, update$

1: **if** $update \neq NULL$ **then**
2: $\quad \mathbf{C}' \leftarrow \mathsf{AuthDec}_{K_{VP}}(update)$
3: $\quad$ **if** $\mathbf{C}' \neq \perp$ **then**
4: $\quad\quad CR.PCR \leftarrow H(\mathbf{C}')$
5: $\quad\quad CR.K_C \leftarrow KDF_{CR.K_{po\_code}}(CR.PCR)$
6: $out.OS \leftarrow \mathsf{AuthEnc}_{CR.K_C}(stateOS, CR.PCR)$
7: $V.hosstate \leftarrow H(stateOS)$
8: $V.hinp \leftarrow hashInputs$
9: $V.encK \leftarrow \mathsf{AuthEnc}_{CR.K_C}(K_{VP}, CR.PCR)$
10: $V.res \leftarrow resultV$
11: $out.V \leftarrow \mathsf{AuthEnc}_{K_{VP}}(V)$
12: Clear $CR.*$
13: Clear all state
14: **return** *out*

Instruction 5. The `bind[]` instruction prepares data for transfer to the untrusted code. This instruction should be called by the executing code right before returning. Inputs to this instruction include a shared secret K_{VP}, the application state $stateOS$, a hash of application input $hashInputs$, and the application results $results$. The variables $out.OS$ and $out.V$ are ciphertext to be stored in local memory and sent to the verifier, respectively. Please note that $out.OS$ and $V.encK$ bind $stateOS$ and K_{VP} to the launch point measurement of executing

code C. Finally, observe that `bind[]` enables program code **C** updates. This is enabled by checking whether the update has been encrypted and authenticated with the shared secret K_{VP} and upon successful verification, updating $CR.PCR$ and $CR.K_C$, accordingly.

3.5 Remote Execution Protocol

The secure remote execution protocol (Figures 3.6-3.10) shows the interaction between the verifier (V) and the untrusted system (OS) during the initial invocation (*setup*) and repeated invocations (*compute*) of code **foo()** within the isolated execution environment (IEE).

We assume that the remote verifier V has a copy of the public platform binding key, $K^{+}_{po\_bind}$. Similarly, the verifier can keep a certificate that is used to confirm the authenticity of the public key it receives from the platform. We also assume that the verifier has access to the plaintext code.

Setup Session

1. V : $V.inp.cmd \leftarrow$ `command`
: $V.inp.pubdata \leftarrow pubInputs$
: **if** ($V.inp.cmd =$ "setup")
: $K_{VP} \xleftarrow{R} \{0,1\}^{\ell}$
: $V.inp.privdata \leftarrow \mathsf{AuthEnc}_{K_{VP}}(privInputs)$
: $V.inp.encsym \leftarrow \mathsf{Enc}_{K^{+}_{po\_bind}}(\{K_{VP}, H(\mathbf{foo}())\})$
: **else if** ($V.inp.cmd =$ "compute")
: $V.inp.privdata \leftarrow$
: $\mathsf{AuthEnc}_{K_{VP}}(privInputs, outV.hosstate)$
: $V.inp.encsym \leftarrow outV.encK$

$V \rightarrow OS$: $\langle \mathbf{foo}(), V.inp \rangle$

: **else** /* other functionality */...

Figure 3.6: Protocol Step 1. Setup Session

3.5.1 Setup Session

In Step 1, the verifier V initiates an isolated execution session with the platform. During the initial invocation, the verifier V uses the public platform key $K^{+}_{po\_bind}$ to establish shared secret K_{VP} which is used for repeat invocations. V generates an encryption key K_{VP}, and binds the hash of the code $foo()$ with K_{VP}. Bind allows the verifier to encrypt data using the public part of the platform key while ensuring that only the correct code running in a correctly set up execution environment can access the data. The inputs along with the code are sent to the platform.

Launch Code

2. OS : $OS.inp \leftarrow out.OS$
   : `launch[`**foo**$(), \{V.inp, OS.inp\}$`]`

Figure 3.7: Protocol Step 2. Launch Code

3.5.2 Launch & Execute Code

In Step 2, the OS calls the hardware instruction `launch[]` using the plaintext code $foo()$, the verifier inputs $V.inp$, and the previously stored state $OS.inp$ as inputs. Please note that $OS.inp$ is assigned $NULL$ during the first launch.

In Step 3, the isolated execution environment IEE first checks inputs received from the verifier. If a "setup" command was received from the verifier the IEE attempts to unbind the encrypted inputs from V as follows. The IEE releases shared encryption key K_{VP}, using the `unbind[]` instruction, and decrypts any private inputs, aborting execution if either operation fails. These checks prevent unauthorized code from proceeding. After the checks, the application logic is executed. For example, if the application is a secure counter, during the first iteration the counter is set to zero. In the case of an encrypted database, the first records could be stored in the database or all records could be initialized to zero.

Execute Code

3. IEE : **if** ($V.inp.cmd =$ "setup") **then**
: $ksym \leftarrow$ `unbind`$[V.inp.encsym, NULL]$
: **if** ($ksym = \perp$) **then** ABORT
: $data_1 \leftarrow V.inp.pubdata$
: **if** ($V.inp.privdata \neq NULL$) **then**
: $data_2 \leftarrow \mathsf{AuthDec}_{ksym}(V.inp.privdata)$
: **if** ($data_2 = \perp$) **then** ABORT
: **else**
: $data_2 \leftarrow NULL$
: $state \leftarrow$ **doWork1**$(data_1, data_2)$
: $out \leftarrow$ `bind`$[ksym, state, H(V.inp), NULL]$

: **else if** ($V.inp.cmd =$ "compute")**then**
: $ksym \leftarrow$ `unbind`$[NULL, V.inp.encsym]$
: **if** ($ksym = \perp$) **then** ABORT
: $data_1 \leftarrow V.inp.pubdata$
: **if** ($V.inp.privdata \neq NULL$) **then**
: $data_2 \leftarrow \mathsf{AuthDec}_{ksym}(V.inp.privdata)$
: **if** ($data_2 = \perp$) **then** ABORT
: $state_{old} \leftarrow$ `unbind`$[NULL, OS.inp]$
: **if** ($data_2.Vhosstate \neq H(state_{old})$) **then** ABORT
: $\{state_{new}, res\} \leftarrow$ **doWork2** $(state_{old},$
: $data_1, data_2)$
: $out \leftarrow$ `bind`$[ksym, state_{new}, H(V.inp), res]$

: **else** /* other functionality */ . . .

Figure 3.8: Protocol Step 3. Execute Code

Save State

4. $IEE \rightarrow OS$: $\langle out.OS, out.V \rangle$
OS : store $\langle$**foo**$(), out.OS\rangle$

Figure 3.9: Protocol Step 4. Save State

Verify Execution

5. $OS \rightarrow V$: $\langle out.V \rangle$
   V : $outV \leftarrow \mathsf{AuthDec}_{K_{VP}}(out.V)$
   : **if** $(outV = \perp)$ **then**
   : reject: invalid computation
   : **if** $(outV.hinp \neq H(V.inp))$ **then**
   : reject: invalid inputs

Figure 3.10: Protocol Step 5. Verify Execution

3.5.3 Save State & Verify Execution

Steps 4 and 5 show the parameters returned to the OS and the verifier, respectively. Step 5 is critical as it provides evidence to the verifier that the computation has been performed on the correct inputs and, in particular, that the inputs have not been manipulated prior to entering OASIS.

3.6 Discussion

3.6.1 Linkable Code Blocks

So far, we have presented how an application C that is fully contained within the CPU cache is executed in OASIS. Recall that the `unbind[]` instruction guarantees that only C can access its protected state during future invocations by verifying that the loaded application has measurement $H(C)$ before decrypting.

Now we consider the case of an application that has size greater than the cache (e.g., application $C = C_0|C_1|\ldots|C_n$ where C_i refers to the i^{th} application code block). Execution of more complex applications is achieved by computing a Merkle hash tree over the entire program, and binding the resulting tree's root value to the application state. The loaded code block C_i is accepted if and only if the hash tree validation succeeds.

The hash tree construction provides several nice properties. First, it extends state protec-

tion and load-time integrity checking to applications of arbitrary size. Second, it maintains a small TCB. Third, it enables efficient execution because code block C_i may be safely executed before the entire application C has been hashed.

3.6.2 Rollback Prevention

A *rollback attack* occurs when on old state is presented to the isolated execution environment. Since the stale state is cryptographically consistent, an isolated execution environment implemented without rollback prevention will incorrectly accept it – potentially bypassing stateful protection mechanisms to, for example, undo the append-only property of an audit log. Thus, rollback resistance is needed to guarantee state continuity of the executing application.

One technique for ensuring state continuity is to include a protected monotonic counter as part of the state [90]. Another technique for rollback prevention is to keep a trusted summary (e.g., a hash) of the expected state. Parno et al. include a summary of the state *history* to permit reverting to a safe state in the case of an unexpected crash [91]. These methods can be achieved by using protected non-volatile memory for persistent storage of data describing the expected state. However, we seek a rollback prevention mechanism that enables OASIS to remain stateless between invocations. Additionally, we rule out using a trusted third party for state management.

What follows is a description of how the verifier can confirm state continuity using the OASIS instruction set. During the execution protocol, the `unbind[]` instruction is invoked to decrypt any state belonging to code C (Figure 3.8). After executing code C, the `bind[]` instruction is invoked to protect state destined for the OS as well as output destined for the verifier. Included in the output for the verifier is a summary of the current state, $H(stateOS)$. The verifier output is encrypted under key K_{VP} before transferring control to untrusted OS code for delivery to the verifier. The verifier includes this state summary as an input during the next invocation. If the state presented by the untrusted OS matches

the expected state, the code executes and the new state summary is communicated to the verifier as acknowledgment. Otherwise, the protocol aborts. In this fashion, we achieve rollback prevention without requiring a persistent application state in the OASIS TCB.

3.6.3 Distributed Deployment

We have presented cryptographic techniques for data secrecy, authenticity, and freshness. Still, the rollback prevention mechanism described thus far is insufficient if we consider the distributed deployment model where multiple verifiers collaborate through a remote service provider. In this asynchronous context, even if cryptographic techniques prevent forged responses and data snooping, a compromised OS can launch *forking attacks* by concealing the operations of one verifier from another. For example, a compromised server may simply omit the current state and replay an old state to the other verifiers.

Fork consistency ensures that all verifiers see the same operations log before an omission but no verifier can see any other verifier's operations after an omission fault (fork). Furthermore, the fork consistency condition enables the verifiers to detect a misbehaving service provider after a single omission.

Li et al. present a protocol for achieving fork consistency where each verifier maintains a signed version structure list [92]. Each verifier signs increasing version numbers and appends these to their respective lists, allowing them to compare lists and detect a fork attack.

3.6.4 Version Updating

To support version updating (i.e., updating code $\mathbf{C}$ to legitimate new code $\mathbf{C}'$), the application must implement an update command which calls `bind[]` with parameter *update* set (where the *update* parameter contains the new code version $\mathbf{C}'$ encrypted under key K_{VP}). The `bind[]` instruction first checks parameter *update* for authenticity and then updates $CR.PCR$ and $CR.K_C$ using the new code version $\mathbf{C}'$ (refer to Table 3.2 for definitions of variables and Instruction 5 for details on `bind[]`). In this way the application state of the

current software version $\mathbf{C}$ is bound to the new software version $\mathbf{C}'$. Accordingly, the next invocation of `unbind[]` will release the application state to $\mathbf{C}'$.

The decryption and authentication operations prove to OASIS that the software originated from the verifier V as she is the only one in possession of the key K_{VP}. It is possible to design an alternative update mechanism, based on asymmetric operations, which has the advantage that an entity different from V can provide an update $\mathbf{C}'$, thus granting it access to the current OASIS state. However, this comes at the cost of requiring certification which would add complexity and computational overhead.

3.6.5 Device Transferability

Recall that the device owner selects seed value S_o during key generation (refer to Function 3.1 for details). The seed value S_o enables the derivation of owner-specific processor keys. Customization, via the owner-generated seed S_o, precludes previous device owners, including the manufacturer, from generating the same platform secret as the current owner. Thus, the device can be safely transferred. This protects the owners of new devices by limiting the ability of malicious parties (e.g., along the supply chain) to learn the platform secrets of the end-user. This allows, for example, a device to be repurposed at a new business unit or sold to a new owner.

Please note that the owner-generated seed S_o effectively disassociates any resulting key material from the device manufacturer. Nevertheless, the owner needs a mechanism to prove the authenticity of their processor to a third party. A default seed value that is fixed for the life of the device may be included to support secure device transfer while still providing a mechanism for proving the authenticity of the executing platform. We refer to this default seed value as the *identity* seed value or ${S_o}^*$. Next, a master signing key is derived from root secret K_p and identity seed ${S_o}^*$. Certification can be handled by a third party for further unlinkability. In this way, secrets linked to the hardware are derived from the fixed identity seed ${S_o}^*$ whereas secrets exclusive to the owner are derived from the custom owner seed S_o.

Allowing the owner to choose any S_o as often as they like may allow an attacker to leak the root platform key K_p through cryptanalysis. This can be mitigated by rate-limiting requests for a fresh S_o. Upon request, the device generates a fresh seed value S_o and computes a MAC over it using a key derived from the root secret K_p and the identity seed value $S_o{}^*$. This ensures that chosen values of S_o cannot be correlated with a response, during device initialization, to learn the root platform key K_p.

3.7 Performance Evaluation

3.7.1 System Configuration

We model our proposed processor instruction set using Simics, a full-system simulator [93]. We build a prototype system by adding our new instructions to the x86-hammer model.[13] We model a 2 GHz processor with non-unified L1 cache (64 KB data and instruction caches). We use a modified Linux 2.6.32 kernel as our target operating system.

3.7.2 Microbenchmark Results

To evaluate micro- and macro-level benchmarks, we measure the performance of our implementation against TCG-style implementations of common security-sensitive code operations. We use a pessimistic benchmark for the OASIS isolated execution environment and compare it to an optimistic benchmark for TCG 1.2. See Table 3.4 for a list of the platform primitives and their associated costs. See Table 3.5 for a comparison of performance overheads for OASIS and DRTM-based implementations.[14]

We base the median performance costs associated with the cryptographic primitives by leveraging open source libraries LibTomCrypt and OpenSSL.[15] It is likely that these functions

[13] *x86-hammer* is a hardware model representing a generic 64-bit AMD Operteron processor sans on-chip devices [94].

[14] We have based performance overheads in Table 3.5 on TPM benchmarks from [95] where the reference DRTM implementation does not provide performance numbers for 2048-bit RSA operations.

[15] LibTom: `www.libtom.org`. OpenSSL: `www.openssl.org`.

further increase in performance with a hardware implementation.

Table 3.4: Performance Overheads for Platform Operations

	avg (of 2^{10} executions)		
	cycles		*time*(*ms*)
Platform Support			
$rand \xleftarrow{R} \{0,1\}^{\ell}$	1.6	K	$7.91 \cdot 10^{-4}$
`f_read_PUF`	~		$2.55 \cdot 10^{-5}$
`f_init_PUF`	~		$2.40 \cdot 10^{-5}$
`f_fuzzy_extract_PUF`	~		$3.30 \cdot 10^{-5}$
Crypto			
$H(p_e)$	4.9	K	$2.49 \cdot 10^{-3}$
$KDF_{CR.K_{po}}$	20.9	K	$1.04 \cdot 10^{-2}$
`f_sym_encrypt`	1.2	K	$6.02 \cdot 10^{-4}$
`f_sym_decrypt`	1.2	K	$6.12 \cdot 10^{-4}$
`f_rsa_key_gen`	3.2	B	$1.61 \cdot 10^{+3}$
`f_rsa_encrypt`	3.08	M	$1.54 \cdot 10^{+0}$
`f_rsa_decrypt`	65.7	M	$3.29 \cdot 10^{+1}$
$\texttt{Sign}_{K_X^{-1}}(m)$	65.9	M	$3.30 \cdot 10^{+1}$
$\texttt{Verify}_{K_X^{+}}(m, \sigma)$	3.1	M	$1.53 \cdot 10^{+0}$
OASIS Functions			
`f_create_sym_keys`	104	K	$5.21 \cdot 10^{-2}$
`f_create_asym_keys`	3.7	B	$1.84 \cdot 10^{+3}$
`f_read_asym_keys`	18.5	K	$9.26 \cdot 10^{-3}$
OASIS Instructions			
`init`	7.2	K	$3.58 \cdot 10^{-3}$
`create`	4.3	B	$2.16 \cdot 10^{+3}$
`launch`	137	K	$6.84 \cdot 10^{-2}$
`unbind` with asym	68.1	M	$3.40 \cdot 10^{+1}$
`unbind` with sym	17.9	M	$8.95 \cdot 10^{+0}$
`bind`	3.12	M	$1.56 \cdot 10^{+0}$

Performance overheads for platform operations used to instrument the OASIS isolated execution environment hardware simulation. Times are based on a 2 GHz processor clock.

Table 3.5: Comparison of Performance Overheads by Invocation Scenario

Scenario	**OASIS**		**DRTM**		
	operation(s)	*time*	*operation(s)*	*time*	*ref.*
One Time	`init[]`	3.6 μsec	NV Write, TPM 2048 Root Key Generation	$>$ 25 sec	[95]
One Time per Owner	`create[]`	2.6 sec	TrustVisor-modeled AIK Generation: TPM and μTPM 2048 AIK Generation	$>$ 25 sec	[95]
Per Module Launch (First Time)	`launch[]` and `unbind[]` with $K^{+}_{po\_bind}$ encrypted input	34.1 msec	TrustVisor-modeled DRTM: Transfer SLB over LPC, Unseal μTPM keys, Quote SLB, μTPM HV_Quote of PAL	$>$ 1.8 sec	[96]
Per Module Launch (Repeated Invocation)	`launch[]` and `unbind[]` with K_C encrypted input	9.0 msec	TrustVisor-modeled DRTM: Set-up and HV_Quote of PAL	22 msec	[96]

3.7.3 Performance Advantages

We now present the performance advantages of our architecture as compared to a TPM implementation.

In terms of processor speed, cryptographic applications benefit from running on a processor core instead of a TPM. For example, the Infineon TPM co-processor operates at 33 MHz, which pales in comparison to even mid and low-end commodity processor speeds.

In terms of communication overhead, we avoid costly communication overheads by implementing cryptographic functions on-chip instead of on a co-processor. For example, the TPM interfaces using the Low Pin Count (LPC) bus. The LPC is used to connect low-bandwidth devices to the CPU (4-bit-bus on a 33 MHz clock).

3.8 Related Work

Architecture Extensions. Hardware-based security mechanisms have been proposed and implemented by both commercial and academic groups. In terms of commercial hardware-based IEE technologies, the main components are the Trusted Execution Environment (TEE) which provides capabilities for isolated execution and ensuring software is in a known good state before launch, and the Trusted Platform Module (TPM) which provides remote attestation, binding, and sealing capabilities. Popular TEE implementations include ARM Trust

Zone [97], and Intel TXT [98]. More recently, Intel has improved on the TXT architecture with the development of Intel SGX [99]. These techniques can be combined with the OASIS API. For example, Enclaves from SGX would replace CAR mode based memory isolation to support applications of much larger size.

Similar to our work, Defrawy et al. propose SMART, an architecture for establishing a dynamic root of trust in remote devices [100]. SMART focuses on remote embedded devices (in particular, low-end microcontroller units (MCUs)) whereas we are applicable to high-end processors. Additionally, SMART investigates the usage of secret key material to establish a root of trust, assuming the existence of *secure* non-volatile memory to store the secret. In contrast, OASIS is based on the use of SRAM memory-based PUFs [66, 67].

Previous work has explored hardware extensions designed for an adversary model where software and physical attacks are possible. Lie et al. present XOM, a hardware implementation of eXecute-only-memory [101]. Similar to our adversary model, XOM assumes a completely untrusted OS. Unlike OASIS, XOM assumes a secure manufacturing process, allows secure XOM applications to access the platform secret, and requires secure non-volatile memory. Lee et al. present SP, a processor architecture for isolated execution [102, 103]. Similar to OASIS, SP does not require a secure manufacturing process; however, SP includes no immutable device secret which makes it a challenge to prove the authenticity of the executing platform to a third party.

Memory cloaking provides secrecy and integrity of application data while allowing the OS to carry on most of its memory management tasks by limiting the OS's data access to ciphertext. More recently, Williams et al. (Secure Executables [104]) and Chhabra et al. (SecureMe [25]) propose an isolated execution environment using hardware-based memory cloaking. Secure Executables uses CPU-protected memory regions to store the register set (e.g., while a Secure Executable is suspended during a system call). This solution has the advantage of avoiding cryptographic operations; however, direct memory attacks may be possible (e.g., by a DMA-enabled hardware component). The root of trust in Secure Executables

is based on a public/private keypair that is installed in the CPU during manufacturing. In our design, the manufacturer and the device owner (or system integrator) both contribute to initializing a root of trust. This reduces the possibility of any large-scale data breaches and also facilities repurposing the device for new owners. SecureMe improves upon previous cloaking methods by ensuring that the entire address space of the application remains protected at all times. OASIS differs from SecureMe in its usage model. Unlike SecureMe, OASIS enforces isolation in the strictest sense by suspending the OS for the duration of its sessions.

PUF-Based Secrets. In 2002, Gassend introduced the notion of the Physically Unclonable Functions (or a PUF) as a mechanism for storing secrets[105]. The PUFs unclonability is due to random variations in a device's manufacturing process, which even the manufacturer cannot control. As a result, greater levels of physical security can be provided at a lower cost than with previous approaches for storing digital secrets.

Guajardo et al. analyzes SRAM-based Physically Unclonable Functions (PUFs). The results of this analysis indicate that PUFs based on SRAM cells provide excellent properties for secure key storage[106].

Suh et al. propose a secure processing architecture, AEGIS, that makes use of Physical Unclonable Functions for creating and protecting secrets [107]. AEGIS consigns security-sensitive OS functionality (e.g., context switching and virtual memory management) to a security kernel. However, this approach faces the same problem as the trusted OS model – the resulting TCB can be quite large.

Similar to our work, Suh et al. propose a secure processing architecture, AEGIS, that makes use of Physical Unclonable Functions for creating and protecting secrets [108]. AEGIS consigns security-sensitive OS functionality (e.g., context switching and virtual memory management) to a security kernel. However, this approach faces the same problem as the trusted OS model – the resulting TCB can be quite large.

Alternate Deployment Models. Our ISE is inspired by the recommendations of McCune et al. [109] but in contrast to previous approaches that use a TPM as the root of trust, we use a PUF-derived key, integrated within the processor. This integration increases performance and diminishes the possibility of attacks on the buses connecting the platform to the TPM.

We use hardware instructions to ensure strong isolation properties during the execution of self-contained security-sensitive code. Another alternative is to use a special-purpose hypervisor instead of additional hardware instructions. The hypervisor provides a less expensive alternative to hardware instruction set extensions and is significantly smaller than a full OS. Nonetheless, a disadvantage of this approach is that the hypervisor is trusted to enforce memory isolation and DMA protection for executing code and, accordingly, must be included in the TCB.

An alternative to extending functionality to the CPU is to use a secure co-processor [110]. A dedicated TPM is the approach endorsed by the TCG. In terms of manufacturing, this approach has the advantage of decoupling system security from the production of traditional processors. A drawback of using co-processors, however, is a reduction of physical security due to the exposed bus. Additionally, the performance hit due to communicating over the bus is not suitable for minimal TCB execution where sessions are repeatedly set up and torn down.

Alternatively, a co-processor could be included as an IP on an SoC which would provide speed, tighter control, and enhanced security. The motivation for extending the processor ISA rather than an SoC TPM implementation is cost savings.

3.9 Conclusion

Currently, TPM-based solutions have not reached widespread application in security-sensitive contexts, perhaps because TCG solutions lack protection against a more resourceful adversary, lack sufficient properties for end-to-end application protection, lack architectural safe-

guards against supply-chain compromises, or concerns over poor performance. OASIS offers a stronger degree of protection through highly efficient isolated execution with no hardware dependencies outside the CPU.

We have explored the extent to which minimal modifications to commodity CPUs can support isolated code execution. The ISA extensions explored in this research enable compute service providers and application developers to provide high-security assurance at low cost in terms of platform and software complexity.

Part III

Micro Clouds: Securing the Edge Through the Fog

Chapter 4

Fog Mediation

This chapter is largely a reproduction of the paper *MEDIA: Mediating Edge Device-based Integrated Access* co-authored with Faysal Shezan, Yuan Tian, Jorge Guajardo, and Patrick Tague [111]. This research was supported in part by participation in the Northrop Grumman Cybersecurity Research Consortium and by an award from the Cisco IoT Security Grand Challenge.

Mediating Edge Device-based Integrated Access (MEDIA). This chapter develops the concept of *fog mediation* as a method for extending the security guarantees of an isolated execution environment to the diverse set of resource-constrained embedded devices comprising the Internet of Things [111]. MEDIA leverages hardware-based security primitives integrated into fog computing elements to provide end-to-end security guarantees, improved performance, and mediate trust decisions between the cloud and co-located mobile devices. Fog mediation enables policy related to IoT data protection practices within a fog computing deployment to be independently verified by end-users.

4.1 Overview

In the context of the Internet of Things (IoT) and burgeoning data streams, the centralized cloud data processing model has several drawbacks including (i) unstable connections and un-

certain response times between the cloud and service endpoints, (ii) poor usability for latency-sensitive applications due to the natural limits of message propagation speeds [112, 113] (iii) increased network congestion due to a lack of local data processing, (iv) scalability challenges due to an increasing number of Internet-accessible sensors and actuators [114] (v) and legal and privacy issues related to writing sensitive data to geographically remote data centers. Additionally, the security techniques well-suited to the traditional cloud computing model are insufficient as we consider a significant shift towards more security-critical operations performed at the edge of the network — e.g., autonomous cars and industrial equipment which were previously offline or siloed in private networks can now receive remote actuating signals — increasing the cyber-physical risk element of cyberattacks. The motivation for delegating some computation to the edge (fog computing) rather than doing everything in the core of the network (cloud computing) is to resolve these drawbacks (see Table 4.1).

Table 4.1: Comparison of the Cloud Computing and Fog Mediation Models

	Cloud Computing	**Fog Mediation**
Adversary Type	`AL2` [24]: Sophisticated remote attacks (e.g., malware in app, OS, or cloud management software)	`AL5` [24]: Sophisticated remote and local attacks (e.g., attaching malicious peripherals or bus tampering)
Security Objective	Primarily system sandboxing	Primarily application isolation
Privacy Policy	Data confidentiality defined by service agreement	Privacy preservation defined by data provenance
Platform Ownership	Centralized ownership by enterprise cloud computing entities	Decentralized ownership by a collective of individuals and institutions
Platform Deployment	Climate-controlled data center managed by specialists	Service endpoint managed by non-specialists
Platform Usage	Hundreds of tenants at any given time	Tens of tenants at any given time
Network Speeds	WAN bandwidth & latency (> 30ms [112])	LAN bandwidth & latency (< 30ms)
Application State	Primarily persistent	Primarily transient

The first three items (adversary type, security objective, and privacy policy) are directly pertinent to the fog mediation security model whereas the remaining items (platform ownership, platform deployment, platform usage, network speeds, and application state) apply broadly to fog computing.

We argue that a purely cloud-centric computing model is not adequate for IoT security and privacy. For one, the cloud simply cannot support all application use cases. The growth in the rate of data generation and an increasingly variegated landscape of connected devices has been accompanied by an increased demand for computation, storage, and network resources co-located with those data streams and connected devices — placing practical limits on the applicability computing model [115, 7, 116]. The initial motivation for introducing fog computing was, in part, to supplement remote data centers and bolster resource-poor connected devices in edge computing use cases with high-performance requirements [8, 9].

Similarly, security techniques designed with traditional device categories in mind (e.g., smartphones and personal computers) do not address the unique challenges presented by IoT device categories. Connected devices, or "things," are generally not managed by IT professionals or they may operate under deployment constraints that make it costly and time-consuming to apply software patches as needed (e.g., battery-bound sensor networks monitoring remote regions). Additionally, connected-devices are generally designed to meet the functional requirements at the lowest cost possible and, as such, do not feature general-purpose security functionality. In all these situations, fog mediation provides a means for reducing the complexity of individual IoT devices in addition to providing a means of achieving high-performance networking and computing.

We observe that resource-rich fog computing nodes provide not only a scalable platform for liberating mobile and embedded devices from severe resource constraints but are well-positioned to act as a grounds for mediating trust between the cloud and the edge of the network — a concept we call *fog mediated computing*. It is the goal of this research to examine the extent to which fog mediation can be applied to the task of providing strong security assurance in the IoT. To that end, this work explores *fog mediation* — i.e., establishing isolated execution environments on fog computing nodes. Fog mediation extends the security guarantees of an isolated execution environment to the diverse set of resource-constrained embedded devices comprising the IoT. We evaluate our model in terms of performance

and security and compare the results for real-world application traces against benchmark implementations.

Contributions. This research explores minimal hardware and software extensions to fog computing platforms that realize strong isolated execution properties using commodity processors. Further, we expand these extensions to provide security to a wide range of IoT devices. In particular, this work makes the following contributions:

- We observe that resource-rich fog computing nodes are well-positioned as a platform for mediating trust between the apps, devices, and stakeholders spanning the cloud, mobile, and ubiquitous computing models. We propose an intrinsic trust architecture specification for isolated execution in the fog (Section 4.2).

- We examine the applicability of hardware-based root of trust and isolated execution environments to the challenges of providing strong security assurance in the Internet of Things (Section 4.3).

- We introduce the concept of *fog mediated computing* — a systems security design pattern that leverages hardware-based security primitives instrumented as part of edge computing infrastructure to provide local points of trust between co-located mobile/embedded devices and cloud-based services (Section 4.4).

- We evaluate the performance of MEDIA using off-the-shelf hardware and public cloud (Section 4.5).

Organization. This work posits that effective security measures for interfacing IoT devices with the global Internet must adequately support *isolated execution* and *contextual information flow*. Isolated execution establishes a verifiable basis of trust at the processor level (the topic of Chapter 2 and Chapter 3). Contextual information flow maps the variegated device capabilities and data use requirements of the IoT to a manageable set of mandatory protections and discretionary controls that accord with the privacy expectations of individuals (a topic further developed in Chapter 5).

At the nexus of these two top-level design goals is the need for a mechanism that extends the hardware-based platform-level security properties to the user-centered app-level requirements and, conversely, provides assurance that the app-level requirements are founded on the reliable ground of platform-level root of trust (the topic of the current chapter, Chapter 4).

In this study, we present, *MEDIA*, a systems design for mediating usability, security, and privacy requirements, for a given set of Internet of Things users and devices, using a network of hardware-hardened fog computing nodes. Section 4.2 describes the problem space including the threat model, the proposed deployment model, design assumptions, and desired properties. In particular, the problem space treated here focuses primarily on the challenge of providing security and privacy in the context of the Internet of Things. Section 4.3 provides descriptions of the key hardware building blocks used in the construction of the MEDIA architecture (refer to Chapter 3 for design details on the CPU instruction set, OASIS, that forms the basis of the isolated execution environment used in MEDIA). Section 4.4 presents the MEDIA architecture which develops fog mediation as a set of platform primitives for supporting performance and security requirements. Section 4.5 presents the experiment testbed configuration and performance evaluation. Section 4.6 discusses the conclusions of the study.

4.2 Problem Definition

4.2.1 Model & Assumptions

Deployment Model. Our use case defines outsourced computation in the sense advocated by the hybrid public cloud and edge computing deployment models.[1] Accordingly, we identify the following five key stakeholders and their primary roles:

(i) The ***hardware manufacturer*** (or HWM) is trusted to manufacture the fog node according to published vendor certification guarantees including the capability of the CPU

[1]The deployment model presented in this section builds upon the description presented in Section 3.2.1.

to initialize cryptographic device keys with a Physically Unclonable Function (PUF) and support the cryptographic instruction set extension (ISE) discussed in the section on the MEDIA hardware building blocks.

(ii) The ***cloud provider*** (data center) offers remote computation, storage, and network resources as-a-service to cloud customers who wish to lease them for a certain amount of time or computation.

(iii) The ***fog provider*** (domain admin) offers computation, storage, and network resources as-a-platform for use by the network of co-located users and embedded devices. Domain admins are interested in ensuring that all services deployed within their network follow the domain's security requirements and privacy policies.

(iv) The ***service provider*** (app developer) wishes to lease both fog computing and cloud computing resources. Service providers are interested in verifying the trustworthiness the devices leased to them, guaranteeing the integrity and confidentiality of their computations and data.

Finally, (v) the ***end user*** (user device) wishes to use fog mediated services — i.e., applications that leverage protected persistent application states sourced from the cloud and co-located data streams emanating from the collection of embedded devices comprising the domain. Users are interested in verifying the trustworthiness of proximate fog infrastructure, guaranteeing the integrity and confidentiality of their computations and data.

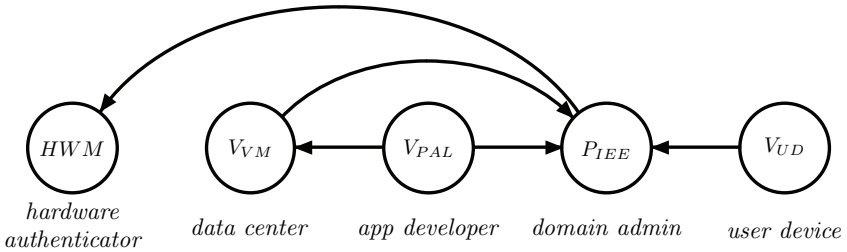


Figure 4.1: Stakeholder Interaction Model.

The interactions between stakeholders in the hybrid public cloud and edge computing

deployment model considered here are depicted in Figure 4.1. For ease of exposition, we refer to the fog mediation node (i.e., the class of devices managed by domain admins) simply as the *untrusted platform* or P. Similarly, we refer to the class of devices used to interface with the fog mediation node simply as the *verifier* or V. We term each security-sensitive block of app code running on P as a piece of application logic (PAL) [117, 96, 118, 24, 109]. We refer to the hardware-protected software instances on P where the PAL is run as isolated execution environments (P_{IEE}).

Please note that the class of devices belonging to V includes devices managed by the cloud provider (V_{VM}), service provider (V_{PAL}), and domain user (V_{UD}) stakeholder groups as these stakeholders are interested in attesting to the launch-point integrity of isolated execution environment P_{IEE} on platform P via virtual machine/container, a piece of application logic, or user device, respectively. Also note that although, in practice, the service provider, V_{PAL}, is interested in verifying the trustworthiness of the leased computation, storage, and network resources residing both within remote data centers *and* fog mediated domains ($P_{VM} \longleftarrow V_{PAL} \longrightarrow P_{IEE}$), the scope of this study is on the latter ($V_{PAL} \longrightarrow P_{IEE}$).

Although we treat each stakeholder group as a distinct party for the remainder of this study, in practice a party may take on a combination of stakeholder responsibilities as the use case dictates. In particular, the domain admins responsibilities may overlap with that of the service provider or end-user depending on the type of domain. For example, the domain admin would also take on the role of services provider in the case of a software product team that manages remotely deployed embedded sensing devices. Similarly, the domain admin would also take on the role of the end-user in the case of small-business owners managing a private local area network. In other contexts, the domain admin may refer to a different set of individuals (e.g., as in the case of a dedicated systems security admin team for a university network).

Adversary Model. In contrast to large scale public cloud data centers, which have organizational processes in place to limit physical access to computing resources, fog computers

must be deployed in proximity to end-users. Some fog computing environments have similar resources in place for building security (e.g., like a large industrial facility that leverages IoT and fog computing equipment for managing operations). Still, many high-value use cases benefit from deploying equipment in environments that are readily accessible to any interested passerby.

We assume an adversary that can introduce malware into the computing platform (e.g., by compromising an application, the operating system, or firmware). We assume an adversary with physical access to the computing platform. In particular, the adversary can probe and tamper with low-speed and high-speed buses or physically attach malicious peripherals to the external ports of the platform to (e.g., to eavesdrop on a memory or PCI bus, inject code, or modify data).

Additionally, we assume that all protocols used in platform P are discoverable by the adversary such that an adversary can compute response $R_{it} = P(C_i, S_t)$ if challenge C_i and state S_t are known (Kerckhoffs' principle [119]).

Assumptions. With respect to the domain admin, we assume that the CPU on untrusted platform P is not malicious and meets the vendor certification guarantees ascribed to it (i.e., we trust the processor contained within the fog mediating node). With respect to the verifying parties, we assume that V has the correct public key of the domain administrator's platform P.

We assume that the CPU contains a True Random Number Generator (TRNG) and a Physically Unclonable Function (PUF) that is only accessible through the specified APIs (see Section 4.3.1). We assume that the memory cells used as building blocks in cryptographic protocols are not accessible for general use (e.g., memory cells with stable and unstable power-states states used for authentication or as an entropy source, respectively). We assume that the construction of the processor-based cryptographic primitives has tamper-resistance properties.

Lastly, we consider denial-of-service, side-channel [120], and fault injection style attacks

beyond the scope of this work.

4.2.2 Desired Properties

The following list contains the desired properties for MEDIA.

*P*1 **Trustworthy Root.** We would like the following intrinsic platform trust objectives to be satisfied:

*P*1.1 *Accessible.* Assets entrusted to the isolated execution environment should be prohibitively expensive to compromise (i.e., leak or alter) even in the presence of malware or an adversary with physical access to the platform.

*P*1.2 *Self-generating.* The root of trust for the isolated execution environment should be ownerless for the entire duration of the platform life-cycle.

*P*1.3 *Self-contained.* The isolated execution environment should not depend on additional hardware support such as secure co-processors or trusted devices.

*P*1.4 *Non-aligned.* The secrecy of processor-generated key material should be independent of management processes external to the platform including processes for certification of platform authenticity.

*P*2 **Trustworthy Mediation.** We would like the following fog mediation security objectives to be satisfied:

*P*2.1 *Integrity.* Data integrity guarantees and replay prevention for data loaded from encryption-protected main memory regions to the processor.

*P*2.2 *Secrecy.* Data confidentiality guarantees for isolated on-die memory regions and encryption-protected off-die memory regions.

*P*2.3 *Verification.* Remotely verifiable code execution that guarantees platform authenticity, code integrity, and launch point integrity.

*P*2.4 *State-bindable.* The ability to generate unique cryptographic keys for each distinct isolated execution environment and to bind program state to hardware, code, and/or verifier group.

4.3 Hardware Building Blocks

The high-level aim of this collection of hardware building blocks is to provide integrity and secrecy for a PAL executing on an untrusted host — i.e., isolated execution [23, 22, 26, 24, 20, 25, 21] on a fog node. For ease of exposition, we use the term *isolated execution environment* to refer to the attendant properties required to provide a remote user with an end-to-end assurance of isolated execution without having to trust the platform provider. Namely, the (i) root of trust, (ii) remote attestation, (iii) secrets provisioning, and (iv) isolated execution properties.

4.3.1 Bootstrapping Trust

The MEDIA architecture provides isolated execution via a suite of hardware security building blocks built directly into the CPU package [108, 24, 23]. These building blocks are necessary to establish strong root of trust (RoT) guarantees that, in turn, enable the generation of platform-derived secret key material for use in cryptographic techniques that provide launch point measurement and code integrity (i.e., proof of unmodified code execution even in the context of an untrusted platform with potentially compromised application, firmware, or OS) as well as data integrity and secrecy for all fog-mediated PALs.

Nourishing the Roots

Physically Unclonable Functions (PUFs) refer to structures that react in unpredictable but repeatable ways to physical stimuli. PUFs are typically constructed from integrated circuits and derive their uncontrollable, yet reliable, response characteristic from the naturally

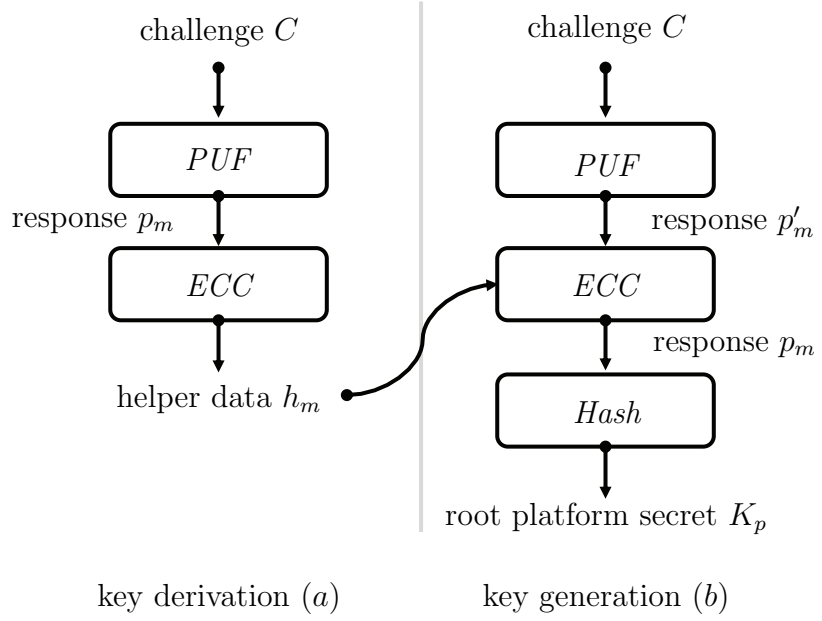


Figure 4.2: PUF-based Key Derivation and Key Generation. This diagram shows the process for the derivation (a) and generation (b) of secrets suitable for use in cryptographic protocols. The resulting bit sequence K_p may be used as an input to a key derivation function. During the key generation sequence, public helper data h_m is used in conjunction with an error-correcting code and a potentially noisy PUF response p'_m to recover PUF response p_m.

occurring variations in semiconductor microstructures that are introduced during the manufacturing process. These variations function as a *"silicon biometric"* that makes it possible to distinguish between any two otherwise structurally identical semiconductors.

In their function as building blocks in secure architectures, they provide a high-entropy source of tamper-resistant bits suitable for applications like IP protection [66, 121, 122, 123], authentication [124, 125], and the generation of cryptographic key material [37]. PUFs based on volatile memory elements (e.g., latches, flip-flops, or SRAM) have been proposed as a cost-effective alternative to secure non-volatile memory as they leverage the semiconductor processes already used in modern CPUs and do not incur the costs and risks of having injecting platform secrets.

For suitability as a cryptographic building block, a PUF construction should be highly

reliable (i.e., exhibit consistent challenge-response behavior under varying voltage, temperature, and aging effects). In practice, due to the complexity of physical microstructures and variations in ambient conditions during challenge-response, error-correcting functionality, called a *fuzzy extractor*, is required to produce bit-accurate data suitable for cryptography from a noisy response (Figure 4.2). In the case of PUF-based cryptographic key generators, it is also useful to have a "strong" PUF (i.e., a PUF construction that supports a very large number of challenge-response pairs) [37].

Leveraging Processors

Cryptographic building blocks that leverage SRAM have significant benefits in terms of a low barrier to adoption (SRAM is manufactured using the standard semiconductor process and already available on modern CPUs), lower costs (avoids the costly requirement of using secure non-volatile memory). SRAM PUFs are of particular interest in the context of hardware-hardening fog computing nodes because (i) SRAM PUFs are unpredictable, reliable, and compact i.e., an excellent approximation of an ideal PUF, (ii) resilient to side-channel and fault injection attacks, (iii) and need to be powered to create the secret key material and thus cannot be read offline [126, 127]. Taken together, these attributes satisfy the accessible ($P1.1$), self-generated ($P1.2$), and self-contained ($P1.3$) properties. In particular, the self-generating property lessens the cost, complexity, and security risk potential of the CPU package and the manufacturing process by removing the need for non-volatile memory and a secure secrets injection process.

A study by Bhargava et al. compares the reliability characteristics of several classes of PUF constructions [126]. They find that SRAM PUFs outperform the Arbiter PUFs, Ring Oscillator PUFs, and Sense Amplifier PUFs in terms of reliability characteristics under variable voltage and temperature conditions. A study on the use of SRAM based FPGAs for IP Protection by Guajardo et al. finds SRAM provides an excellent source of entropy (with approximately 0.95 bits entropy for every SRAM cell [66]).

Several attacks highlight some of the potential risks and trade-offs of using SRAM-PUFs for secure storage. Helfmeier et al. present a proof-of-concept procedure for cloning SRAM PUFs [128]. The general procedure is to first characterize the response of a PUF and then use Focused Ion Beam circuit edit to create replica PUF with the same response. Although the procedure demonstrates the plausibility of creating a physical clone, in the context of fog mediation, the procedure is practical. The circuit characterization and circuit editing tools needed to carry out procedure are prohibitive to all but the most dedicated and sophisticated adversary — costing around $100K to $200K in 2012 USDs in equipment, suitable lab space, takes 20 plus hours to produce the physical clones, and the same procedure has to be repeated for each new PUF response. Nonetheless, this attack points out a potential attack vector for counterfeiting or compromising platform secrets during the supply chain.

Yossef Oren et al. presents a side-channel attack based on remanence decay in volatile memory [129, 130]. However, this side-channel assumes that the memory used for the PUF may also be used for normal program operations. Thus, the remanence decay side-channel can be prevented by reserving a range of cells for exclusive use in PUF-based key generation.

Sampling Randomness

As mentioned, in the case of key extractor PUFs instability in the power-up state of memory-cells is undesirable. However, this instability can be leveraged as a source for introducing entropy and as as a basis for strong or true random number generators (TRNGs). Thus, the power-up SRAM state can be used both as a fixed fingerprint, in authentication and key generation, and as an entropy source for random number generation [131].

4.3.2 Computing in the Fog

Fog computers are resource-rich well-connected computer clusters available *nearby*. The concept of "fog computing" was originally introduced as "cloudlets" and formalized as a VM technology by Satyanarayanan et al. [8]. In its most general definition, the notion of

fog computing refers both to integrating cloud technologies into the core network (e.g., as general-purpose enhancements to network infrastructure to better support edge computing) and to deploying cloud computing resources at network endpoints (e.g., to better support mobile computing).

Before the availability of public cloud (i.e., "computing as a utility" [132]), mainframe computing (a precursor to cloud and fog computing) was used by enterprises for business-critical applications. Moving business-critical applications to the public cloud allowed businesses to avoid the costly overhead of buying and maintaining proprietary computing architectures in-house. The tech sector as a whole benefited from the economies of scale resulting from the outsourcing and standardization of information technology. On the other hand, the users of cloud services were now entrusting data security to cloud service providers. During this same time, the significant scale-up in connectivity (e.g., 4G to 5G), interoperability (e.g., HTML5), and demand for mobile computing have re-established the business case for on-premise computation infrastructure.

The vision of fog computing extends both mobile and cloud computing paradigms to better support responsive performance for resource-intensive and latency-sensitive services (e.g., real-time planning and actuating, machine learning, computer vision, natural language processing, and augmented reality) leveraging resource-poor embedded devices (e.g., IoT devices) and resource-constrained mobile clients (e.g., smartphones). Fog computing augments cloud-backed persistent app states with transient local states, significantly reducing the need to route traffic across the global Internet and the signal load of the core network (significant obstacles to a high quality of service and security guarantees as we consider half a billion Internet-addressable devices outfitted with high-resolution sensors). Additionally, deploying general-purpose high-performance networking elements to fog nodes offers a scalable point for cross-layer network state exchange for adaptive wireless network elements [133].

In practice, fog computing elements may be deployed as standalone cloudlets or as extensions to existing infrastructure such as cellular towers, networking controllers, content

delivery networks, road-side traffic management equipment, in-vehicle communications networks, home energy management systems, home heating systems[134], routers, modems, televisions, set-top boxes, video game consoles, etc. The alternative terms for fog computing found in other contexts are useful for emphasizing specific aspects such as virtual machine provisioning (cloudlets), network architecture components (mobile edge computing), or cloud computing capabilities (micro cloud) [135]. In this work, we use the terms cloudlet and fog computing interchangeably to refer to these technologies that enable distributed computing environments to process information within proximity to cellular subscribers and WiFi hotspots.

Table 4.2: Perception and Latency

	Response Time (ms)			
Task Type	***low***	***median***	***high***	***ref.***
Website Load Time	100 ms	250 ms	500 ms	[136, 137, 138]
Video Frame Rate	17 ms	33 ms	42 ms	[139]
Screen Refresh Rate	4 ms	7 ms	17 ms	[140]
PC Input Latency	15 ms	30 ms	150 ms	[141]
VR/AR Input Latency	11 ms	20 ms	30 ms	[142, 143, 144]
Stylus Input Latency	8 ms	17 ms	17 ms	[145]

Responsiveness thresholds during common human-computer interactive tasks for *low* latency (i.e., the threshold for optimal usability where improvements are generally imperceptible), *median* latency (i.e., the threshold for target usability where the user interface is experienced as snappy and responsive), and *high* latency (i.e., the threshold for satisfactory usability where lag is quite noticeable).

Human reaction time and delay tolerance is a complex process spanning contact (i.e., the awareness of a sense object), perception (i.e., the labels of meaning applied to sensed objects), and fabrication (i.e., mental proliferation such as discursive thinking or decision making based on the sensed object). Additionally, there are physiological factors and context-dependent queues that significantly impact human perception (e.g., the heightened reaction time of a trained fighter pilot flying a jet vs. the relaxed reaction time of someone waiting in line at the grocery store). Although delay tolerance depends on these subjective perceptual factors, human factors professionals agree that responsiveness is a key usability requirement for in-

teraction design [136], faster is generally better [137, 138], and the threshold for "seamless" interaction is more or less fixed for a given task. In Table 4.2 we limit the discussion of suitable input response times for human-computer interactive systems to the sensory perception stage and make the case for the sub-30-millisecond threshold for seamless interaction (compare this result to the cloud latency times found in Table 4.6).

Bringing computing power closer to the devices that interact with cloud services (i.e., cloudlets [8] and fog computing [146]) enables more seamless user experiences that are less susceptible to latency jitter and unstable network conditions while reducing overall congestion through better support for localized online processing — essentially allowing resource-constrained mobile and embedded devices to act as thin-clients. Additionally, the availability of co-located computation power and a federated app deployment model better supports user-centered privacy controls based on data provenance and spatial-temporal filters [147].

4.4 Fog Mediation Architecture

4.4.1 Service Integration

This section describes the MEDIA architecture components pertaining to the application including service integration between the cloud provider, the fog provider, and the service provider.

Provisioning Domains

This section describes the process for establishing a proximal domain (Figure 4.3 (a)). The notion of a *domain* in the IoT context is more closely aligned with physical proximity than with the logical relationship of components. For example, a smart space may contain embedded devices, network elements, displays, computers, sensors, and actuators all operated by different users, serving different purposes, and managed by different service providers but falling under the same domain due to their proximity to the user devices and the end-users

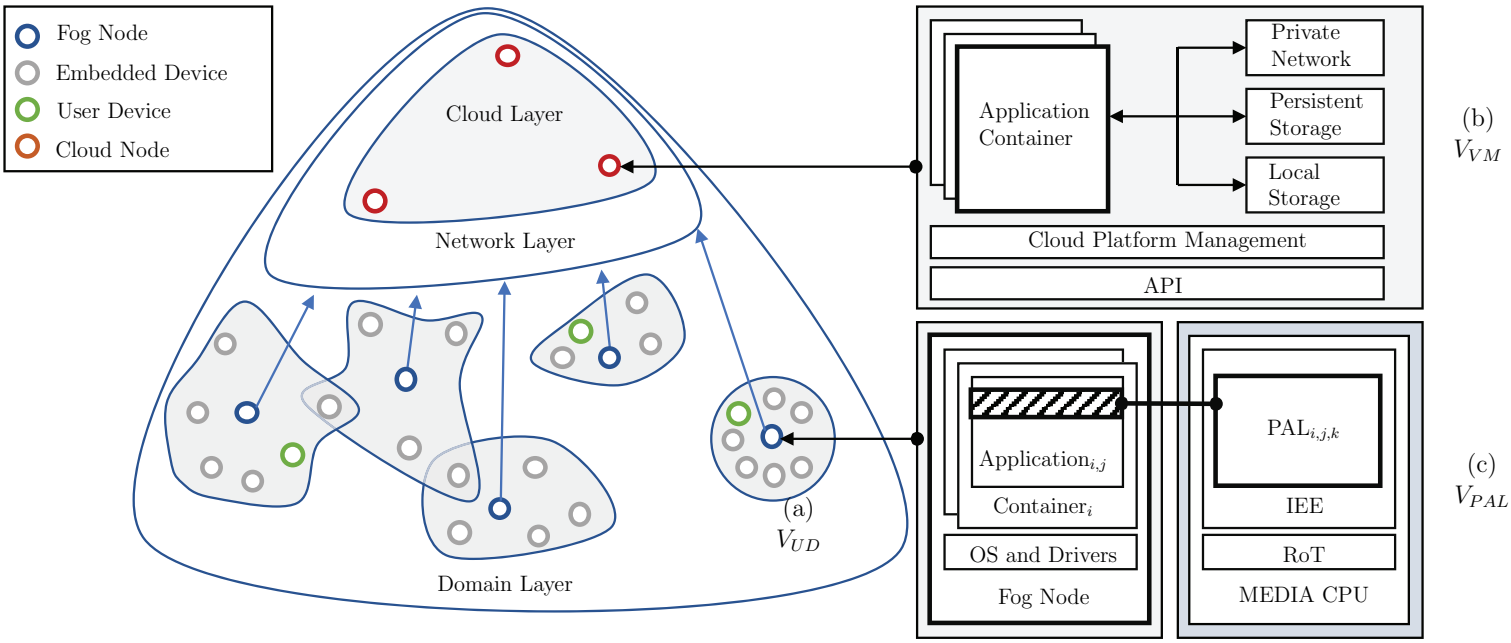


Figure 4.3: Fog Mediation Architecture. This network diagram is organized into three abstraction layers: public cloud (cloud layer), public telecommunications infrastructure (network layer), and proximal domains (domain layer). The interactions of verifier group V_{UD} (user devices and co-located connected devices), verifier group V_{PAL} (application service providers), and verifier group V_{VM} (cloud service providers) are mediated by MEDIA-enabled processors residing on fog nodes within the domain layer. The primary interaction for V_{UD}, V_{VM}, and V_{PAL} occur during (a) proximal domain establishment (Section 4.4.1), (b) service provisioning (Section 4.4.1), and (c) secrets provisioning (Section 4.4.1), respectively.

that leverage them.

We also note that the type of messages that are exchanged in IoT domains are highly spatiotemporally dependent (e.g., contextual information such as system state, observations about the environment, and control signals). Additionally, there is a dynamic and transient coupling between IoT devices and the services and users that employ them. As a result, approaches to trust establishment and access control that assumes a client-server model does not adequately support the joint security and privacy requirements of the stakeholders in the IoT use case.

Thus we introduce the notion of a *proximal domain* as a practical programming abstraction for deploying IoT applications. A proximal domain is the set of N connected devices and

MEDIA-enabled platforms managed by a fog mediation cluster. The fog mediation cluster refers to the set of MEDIA-enabled co-located fog computers. Connected devices include user attended connected devices (i.e., user devices or UDs) or unattended connected devices (i.e., embedded devices or EDs).

In general, our interaction model assumes that the responsibility of managing access control to the proximal domain by an application belongs to the domain admin V_{UD}. Whereas the responsibility of managing access control to the application by the connected devices belongs to the service provider V_{PAL}. The individuals consuming services from within a proximal domain (i.e., end-users), as well as connected devices, can independently interface with a proximal domain. The mode of interaction between a given connected device and an isolated execution environment instance P_{IEE} depends in part on the capabilities of the connected device and application-specific requirements. The goal then of the fog mediation cluster is to provide integrity and secrecy assurances to each running PAL while mediating stakeholder interactions and connected device capabilities.

Provisioning Services

This section describes the process for provisioning services to MEDIA-enabled fog computing nodes (Figure 4.3 (b)). We utilize a hardware virtualization technology to efficiently deploy custom fog computing software stacks. Deployable virtual machines and containers support two general types of operations. The first operation is mobile offloading where a resource-poor device can offload services to the resource-rich fog nodes in its local network. The second operation is cloudlet formation where the service provider V_{VM} retrieves and re-deploys cloud-based micro-services on proximate fog computing infrastructure to improve performance, usability, and network quality of service guarantees. These virtual containers may then be orchestrated using inter-container connections both to local fog-based containers and to remote cloud-based containers.

The resulting virtual environments are transiently constructed to support the user de-

vices, embedded devices, and services comprising the proximal domain. Further, the virtual environments may be suspended and migrated from one proximal domain to another. For example, when a user moves from location A to location B, the services registered to the user can seamlessly re-authenticate and integrate with the new fog mediated connected device context. Once the container's filesystem has been loaded on to platform P, executing services can switch from the normal execution contexts to hardware-protected isolated execution contexts by defining PALs that invoke P_{IEE} instances.

The sequence of steps beginning with a clean slate fog node and culminating in the provisioning of a service to platform P are as follows: (i) lease cloud-based services, (ii) construct custom container, and (iii) service provisioning. Throughout this process, the verifiers complete remote attestation independently using the same process for remote attestation to P_{IEE} that is described in Section 3.5 and Section 4.4.1.

In step (i) the service provider V_{PAL} and cloud provider V_{VM} negotiate cloud-based micro-services for security-sensitive application PAL. These services act as the server-side component of the fog application. In step (ii) we utilize the dynamic VM-synthesis for efficiently constructing custom containers on P [8]. Cloud provider V_{VM} pre-loads the base container image designated by V_{PAL} onto P. V_{VM} attests that the base image is in a known safe state by requesting a measurement from P_{IEE} and publishes the result to the verifier group. Next, the cloud provider V_{VM} retrieves the private container overlay designated by mobile user V_{UD} onto P. V_{UD} attests that the private container overlay is in an expected state by requesting a measurement from P_{IEE} and publishes the result to the verifier group. The base image and private overlay are then dynamically merged on P before launch. Leveraging hardware-to-container bindings [148], V_{PAL} measures the resulting security-sensitive PAL(s) associated with the constructed container image, and publishes the result to the verifier group.

In construction presented here, the launching of security-sensitive PALs is contingent on the successful measurement of components such as the base image and private overlay

container construction. Alternately, the PAL and its associated program state can be bound to specific hardware and verifier group member or application-specific conditions such as the presence of specific users or devices.

Remotely Attesting & Provisioning Secrets

Whereas Section 4.4.2 describes processor-derived secrets, this section describes the process for provisioning verifier-generated secrets to an isolated execution environment (Figure 4.3 (c)). Unlike the processor-generated secrets, verifier-provisioned secrets are directly accessible by the executing code. Nonetheless, secrets provisioned to an IEE instance benefit from the same hardware protections as all IEE-protected states. Utilizing both derived secrets and provisioned secrets is useful for decoupling systems security programming logic (e.g., launch-point integrity checks, remote attestation, isolated execution, and encryption-protected main memory access) from application-specific programming logic (e.g., loading a user's private key and the accompanying code for generating signatures into the IEE). In this way, processor-derived secrets are opaque to the application code while verifier-provisioned secrets may be securely managed across any IEE instance belonging the application. Additionally, reliance on provisioned secrets is desirable for use cases where application code would like to make use of MEDIA-enabled CPUs if available but should still degrade gracefully to support differing application programming interfaces.

The sequence of steps culminating in remote attestation ($P2.3$) and the provisioning of secrets to the IEE instance are as follows: (i) launch state measurement, (ii) secure channel establishment, (iii) platform authentication, (iv) remote attestation, and (v) secrets provisioning. In step (i) platform P measures the safety of the launch state and the contents of the IEE prior to executing app code PAL. In step (ii) platform P establishes a secure connection with the remote verifier V_{PAL}. In step (iii) platform P sends the measurement evidence and proof of the platform authenticity to verifier V_{PAL}. In step (iv), given valid measurement evidence (i.e., corresponding to a known safe execution state) and valid platform authenti-

cation evidence, verifier V_{PAL} acknowledges the successful remote attestation of the launch state on P. In step (v), verifier V_{PAL} provisions private inputs the PAL executing in the IEE residing on platform P. The description here focuses on the interaction between the service provider V_{PAL} and a MEDIA-enabled platform P. A similar procedure may be used by the other verifier classes, including cloud service providers V_{VM} and domain users V_{UD}, to complete remote attestation and secrets provisioning.

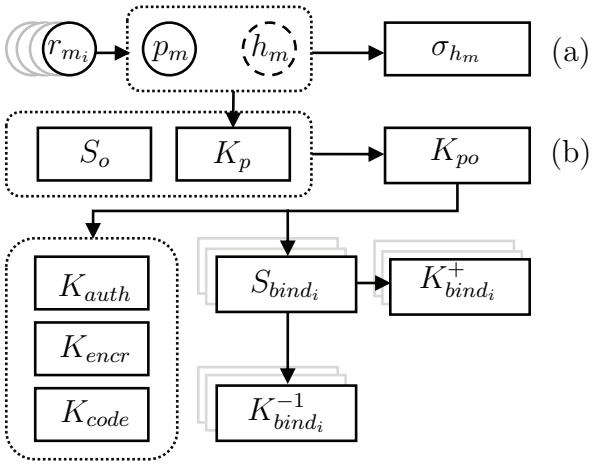


Figure 4.4: Platform Authentication and Key Generation. This diagram shows the processor-based key generation hierarchy belonging to a single session of the isolated execution environment. The secret materials and derived keys in this key generation hierarchy include hardware-generated random number r_m, PUF response p_m, helper data h_m, owner-provided seed value S_o, root platform secret K_p, master platform secret K_{po}, symmetric sessions keys K_{auth}, K_{encr}, and K_{code}, platform binding secret S_{bind}, and asymmetric session keys K^{-1}_{bind}, and K^{+}_{bind}. Response p_m may serve as either the cryptographic basis for platform-derived authenticity evidence σ_{h_m} (a) or as the root of trust for platform-derived key materials beginning with master platform secret K_{po} (b). Random numbers r_{m_i} and asymmetric key tuple $[S_{bind_i} \rightarrow K^{-1}_{bind_i}, K^{+}_{bind_i}]$ are depicted as stack elements to indicate that multiple instances of processor-derived symmetric and asymmetric keys may be generated as needed during a single session.

4.4.2 Processor Integration

This section describes the MEDIA architecture components of the CPUs residing in a fog mediation node.

Deriving Secrets

A central feature of the isolated execution environment is a hardware-derived root of trust (Section 4.3.1) that is used to support the desired security objectives of fog mediation — i.e., the integrity property *P*2.1, the secrecy property *P*2.2, the verification property *P*2.3, and the state-bindable property *P*2.4 — by generating unique cryptographic key material for use by security-sensitive PALs.

The root of trust instantiation (Figure 4.2) and the resulting hierarchy of key material are shown in Figure 4.4. PUF response, p_m, may serve both as as the root response for cryptographic key generation and as a digital fingerprint for device authentication (Section 4.4.2). Hardware-generated random numbers r_{m_i} provide an entropy source for use in the construction of the key hierarchy (to introduce entropy to helper data h_m) and for general purpose use within the IEE (e.g., by a PAL requesting a cryptographic nonce). The function that initializes p_m accepts a random value r_{m_0} and outputs helper data h_m and hash $H(p_m, h_m)$. Public helper data h_m is used by a robust fuzzy extractor function to retrieve p_m given a noisy response p'_m (as it is commonly done in the literature [76]) and reconstruct the root platform secret, the uniformly random value K_p.

The keys are derived using a key derivation function (KDF) based on pseudo-random functions and accept secret parameters and public parameters [80, 81]. The root platform secret K_p and the owner seed S_o are inputs to a key derivation function, resulting in the master platform key K_{po}. The owner-supplied seed value S_o enables the domain admin to customize all resulting platform secrets. The use of an SRAM PUF response avoids the costly requirements of on-chip non-volatile memory and a secure key injection process. Together, these two mechanisms enable device transfers without the need for any other party to protect platform secrets on behalf of the current device owner, satisfying the non-aligned property (*P*1.4).

The master platform key K_{po} is then used to derive symmetric (K_{auth}, K_{encr}, K_{code}, and S_{bind}) and asymmetric (K^{-1}_{bind},and K^{+}_{bind}) session keys. Random numbers r_{m_i} and asymmetric

key tuple $[S_{bind_i} \rightarrow K^{-1}_{bind_i}, K^{+}_{bind_i}]$ are depicted as stack elements in Figure 4.4 to indicate that multiple instances of processor-derived symmetric and asymmetric keys may be generated as needed during a single session.

All key material derived by the processor are short-lived — i.e., not manifest on P beyond the completion of the IEE session — and managed internally — i.e., only accessible to specialized processor instructions and written to dedicated memory cells — satisfying the accessible property ($P1.1$). Since all of the platform key materials are PUF-based ($P1.2$) and initialized by end-users, as opposed to burned into the processor's fuses or stored off-die in a trusted device ($P1.3$), this avoids the security risk potential related to the existence a database of cryptographic keys. Additionally, since all of the platform key materials required for attesting to the launch-point integrity of an IEE session on platform P are internally generated this avoids the security risks related to requiring cryptographic keys to be obtained over the Internet to use the platform ($P1.4$) [149].

Declaring Authenticity

During platform initialization the manufacturer certifies helper data h_m using a standard public-key signature $\sigma_{h_m} \leftarrow \texttt{Sign}_{K^{-1}_{HWM}}(h_m||H(p_m, h_m))$. This provides a basis for platform-generated authenticity assertions. The resulting signature σ_{h_m} may then be verified using a certified public key K^{+}_{HWM} of the manufacturer to assert the authenticity of helper data h_m and, by extension, the authenticity of the MEDIA-enabled platform P it is bound to. Alternately, a trusted third party (such as a system integrator) or an impartial witness (such as a regulatory body tasked with consumer protection) may complete this platform certification step, fulfilling the role of the signing party for σ_{h_m}. Additionally, centralized or decentralized trust models for certification authorities — such as public key infrastructure or web of trust — may be used to manage and distribute platform certificates of authenticity.

Special care should be taken to protect the secrecy of the platform-derived root of trust (Figure 4.4 (b)) by ensuring its independence from platform-derived authenticity assertions

(Figure 4.4 (a))($P1.4$). Although the same process used for bootstrapping key generation may be used for bootstrapping platform authentication, saving chip space, the additional cost of a relatively small portion of reserved chip space is a small cost to pay for less exposure to critical compromises. Thus, the cell arrays used for key generation should be different than the cell arrays used for platform authentication to avoid the security risk potential of compromising either process by correlating data intended for use by the other process.

Similarly, special care should be taken to protect the privacy of verifier interactions. During an interaction between an IEE session and a remote verifier, the verifier would like a trustworthy response to the query "is this an authentic MEDIA-enabled environment?" without disclosing any other usage information to the manufacturer or certifying authority. Once the verifier is able to establish the authenticity of the platform and retrieve the associated public platform binding key $K^{+}_{bind_0}$, a secure channel between both parties may be established before for confidential operations such as remote attestation and secrets provisioning (refer to Section 3.5 for details on the remote execution protocol). Depending on the construction of the platform authentication process and the nature and duration of the usage information retained it is plausible for the certificate authority to learn when a particular verifier interacts with a specific piece of hardware. In this construction, platform authentication need only occur one time per verifier-platform pair — limiting the potential privacy and anti-trust risks related to placing the HWM at the center of each interaction between every verifier and platform pair ($P1.4$) [149].

Extending the Instruction Set Architecture

A MEDIA-enabled platform achieves isolated execution using exclusively on-die components and a programming interface that enables security-sensitive PALs running on platform P to invoke a set of CPU instructions for transitioning from the normal computing environment to an isolated execution environment P_{IEE}. What follows is a brief recapitulation of the ISE presented in Chapter 3:

The programming interface supported by the ISE includes the *init*[], *create*[], *launch*[], *unbind*[], and *bind* instructions: The **init**[] instructions is used by the HWM to initialize the platform and extract platform-derived authenticity evidence. The **create**[] instruction creates a hierarchy of key material from the raw PUF response. The **launch**[] instruction is responsible for properly entering and exiting the dedicated IEE of the PAL. When invoked, *launch*[] sets up a clean-slate IEE by disabling instructions that require system interrupts, changes to a higher privilege setting, hardware debugging access, or I/O transitions. The *launch*[] instruction also stores a hash of the PAL in the platform configuration register for use during remote attestation and writes generated key material to dedicated IEE registers. At the end of the execution session, *launch*[] prepares the IEE for transfer back to the systems. The **unbind**[] instruction confirms that the hash of PAL matches the expected measurement before decrypting app states retrieved from external non-volatile disk storage and checks the integrity of any inputs received from the verifier using keys specific to the PAL. The **bind**[] instruction prepares any outbound app state and messages for a verifier using as a key derived from the measurement of the current state of the PAL. Additionally, *unbind*[] and *bind*[] provide a mechanism for authorized software updates.

4.4.3 Operating System Integration

This section describes the MEDIA architecture components of the OS.

Hardening Execution

The rich execution environment (REE) is the normal execution context where the OS and applications run. The REE balances system-level security requirements and process-level security requirements while optimizing for overall CPU utilization. Even though modern CPUs implement some form of hierarchical memory protection and privilege separation to increase the OS's control over applications (e.g., x86 protection rings), the REE is generally too large for formal verification and too complex for a meaningful assessment of security

based solely on integrity checks. Thus the OS is considered untrusted from the perspective of the remote verifier in our adversary model.

As a building block for secure systems, TEEs provide trust anchors from which trusted components are used to assess the integrity of the components comprising the execution context. Minimizing the size and complexity of trust anchors is an effective way to reduce the attack surface available to would-be attackers. Mechanisms for dynamically establishing a trusted execution environment (TEE) provide strong security assurance for security-sensitive PALs, a practical separation-of-concerns framework for developers, and enable a remote verifier to reliably infer the security of services using integrity checks. Hardware-based TEEs are widely considered an indispensable building block for meeting stakeholder requirements in many contexts including personal computing [90], cloud computing [23], mobile computing [43], and as we recommend here, in the ubiquitous computing contexts of the IoT.

In this work, we develop a two-tier execution framework comprised of a rich execution environment (REE) and an isolated execution environment (IEE) where the IEE is a hardware-based TEE containing the building blocks described in the preceding sections. Please note that throughout this work we use the terms secure world vs. normal world, protected vs. unprotected, and trusted vs. untrusted interchangeable to refer to these two modes of running PALs.

Isolating Execution

Modern CPUs store copies of frequently used data in cache memory to minimize the average time and energy costs of accessing the main memory. Additionally, modern CPUs utilize processor cache for system boot-up tasks until the main memory is initialized — a technique known as *Cache-as-RAM* (CAR) [150, 74] — since cache memory can be initialized in a few instructions. Whereas dynamic RAM is widely used for main memory functions due to its low-cost and high-capacity properties, static RAM is generally used to implement cache memory subsystems in modern processors due to its high-performance and structurally-

independent properties.

Prior work has explored the security applications of CAR-mode execution [24, 23, 43]. In particular, the low initialization costs of cache memory enables short-lived IEEs to run seamlessly in conjunction with the REE provided by the OS. Vasudevan et al. present CARMA, an x86 architecture that leverages CAR-mode to provide a general-purpose secure execution environment with a trusted computing base (TCB) consisting of the processor (for isolated execution) and a trusted device (for root of trust initialization) [24]. The motivating challenge that CARMA addresses — i.e., defending against a sophisticated adversary with physical access to the platform — is one that is quite relevant to our challenge of establishing trust in co-located critical infrastructure that is physically accessible to any passerby.

Owusu et al. present OASIS, an instruction set extension that leverages exclusively on-die structures for processor-rooted trust establishment, integrated support for cryptographic primitives, and on-demand isolated execution environments [23]. The OASIS architecture uses SRAM cells with stability and instability stimuli-response characteristics as the cryptographic basis for authentication and randomness, respectively, and the CARMA isolated execution model for an efficient isolated execution environment with a TCB consisting solely of the CPU.

Leveraging processor-derived isolation, MEDIA offers an efficient isolated execution environment (*P*1.3). The following instructions are disallowed from within the MEDIA IEEs:

- hardware debugging access
- inter-process communication
- system calls (i.e., requesting kernel services)
- I/O instructions
- privilege escalation
- reading memory residing outside processor package

The resulting IEE provide strong integrity, secrecy, and freshness guarantees to any executing security-sensitive PALs (properties *P*2.1 and *P*2.2). In general, the IEE is suitable for use cases where the primary functionality of the PAL is to invoke the processor via an IEE to manage secrets as well as carry out cryptographic operations based on those secrets on its behalf. In contrast to SEEs, the strict run time isolation of IEEs necessarily places restrictions on the types of services that can be executed. However, these restrictions avoid the potential of providing system access to hardware isolated malware. So there is a trade-off between usability and the risk potential for protecting malware.

Sandboxing the Guest

The sandboxing capability protects privileged platform code from potentially malicious user code (the primary concern for platform owners like cloud provider and domain admin stakeholders). Even though the primary security objective in the context of fog mediation (i.e., from the perspective of the service provider and the end-user) is isolated execution, in terms of overall system security, it is beneficial to have a protection mechanisms that integrates both isolated execution and the inverse property, sandboxing — e.g., Li et al. present MiniBox an x86 architecture providing two-way sandboxing [117] and Baumann et al. present Haven [21] is a memory shielding architecture leveraging Intel SGX [151, 22] for isolation and Drawbridge Library OS [152] for sandboxing. It's also worth mentioning that the isolated execution mechanism presented here reduces the security risks faced by both platform owners (e.g., physical attacks like memory probing) and platform users (e.g., large, privileged, and potentially malicious system code).

Baumann et al. introduce the notion of *shielded execution* and define it as essentially the inverse of sandboxing — i.e., the ability to achieve confidential code execution for unmodified application binaries with complex OS interactions [21, 148, 153]. Shielded execution environments (SEEs) build on isolated execution environments to extend the hardware-based integrity and secrecy guarantees of cache-resident IEEs to general-purpose apps — address-

ing the challenges of using isolated execution for code that requires REE-level capabilities such as access to system services, fault and exception handling, support for dynamically allocated memory, and interactions with untrusted libraries [154].

In terms of memory protection, an IEEs trust perimeter ends at the boundary of the CPU package. The task of safely accessing application states residing in untrusted memory DRAM is achieved through integrating a memory encryption service [48]. In the remainder of the chapter, we use the term isolated execution to refer to TEE instances utilizing both cache memory and integrity-checked and encryption-protected access to main memory. Please note that this memory protection architecture does not specify protections for an adversary conducting traffic analysis style side-channels on encrypted data as it is loaded into on-die memory or transferred to main memory (i.e., oblivious RAM is beyond the scope of this work).

4.5 Performance Evaluation

This section discusses the experiments used to evaluate the fog mediation architecture (Section 4.5.1). We evaluate the performance both in terms of the platform-level overheads of an individual fog mediation computing node as it interacts locally with a security-sensitive PAL (Section 4.5.2) and in terms of end-to-end application-level run-times for service deployed to a MEDIA testbed as it mediates the interaction between co-located devices and remote cloud resources (Section 4.5.3). We measure performance times during the execution of benchmark machine learning and computer vision tasks to evaluate end-to-end application performance under the client-server deployment model (i.e., traditional cloud) and the proposed client-fog-cloud deployment model (i.e., fog mediation).

4.5.1 Implementing MEDIA on SGX

Reference Hardware

The proposed fog mediation architecture is evaluated on a commodity microarchitecture to capture instruction-level performance overheads. We use the 4 CPU Intel Core i7-6600U x64-based microarchitecture with a 2.60 GHz processor base frequency (3189 MHz measured frequency), 64 KB L1 instruction-cache, 64 KB L1 data-cache, 512 KB L2 cache, and 4 MB L3 cache on a computing system with 20 GB main memory and 512 GB SSD disk memory as the reference hardware for each fog node [155].

Performance Instrumentation

Our performance evaluation does not attempt to evaluate security guarantees (the topic of Section 3.7). Instead, this evaluation aims to leverage SGX as an isolated execution hardware security basis for the proposed MEDIA implementation (MEDIA-SGX) and to obtain performance estimates for benchmark applications. We implement MEDIA-SGX in C++ running on 64-bit Microsoft Windows. Each job is deployed as a native 64-bit executable (*.exe) containing the application. The MEDIA-SGX application contains the IEE code and a static library containing the normal world (PAL^+) and secure world (PAL^-) enclave partitions. The MEDIA-SGX implements an instruction handler for all SGX instructions used by the IEE code including `OCALL`, `ECALL`, `EREPORT`, `RDRAND`, and `EGETKEY` [151]. We leverage the mechanism described in [151] to access model-specific registers (MSRs) for execution tracing and performance monitoring. We use the windows kernel function `SwapContext` as described in [156] to preserve the same register context between subsequent enclave invocations.

The enclave page cache (EPC) refers to the 4 KB aligned protected memory regions used to store pages belonging to an executing enclave [151]. An executing PAL^- may read and write contents to the EPC via specialized CPU instructions reserved for managing enclaves. The SGX-enabled processor uses a data structure called the enclave page cache

map (EPMC) for address-translation to the EPC where each page table entry tracks the contents of an EPC page. The EPC and EPMC are processor assets (i.e., implemented in SRAM) providing an isolated execution environment for the executing enclave. A memory encryption engine (MEE) may be incorporated to support a shielded execution environment for enclave instances that store the EPC in main memory (in DRAM) [48]. Please note that the EPC specification for SGX is processor specific [151].

4.5.2 Study 1: Target Platform Micro-Benchmarks

We evaluate the performance for the target platform by running a series of computation and memory-intensive tasks called `nbench` (Native Mode Benchmarks) [157] on the same hardware with the same operating system running under two execution models (REE and TEE). We utilize the reference hardware described in Section 4.5.1 running on Linux with the following software configuration:

- OS: Linux 4.13.0-38-generic
- C compiler: gcc version 5.4.0
- libc: libc-2.23.so

The `nbench` test suite is a series of ten benchmarks that are designed to measure the performance capabilities of a target platform's CPU including the capabilities of its floating-point unit and memory controller:

1. *Numeric Sort* - sorting of an array of long integers.
2. *String Sort* - sorting of an array of strings of varying lengths.
3. *Bitfield* - computes a set of bit manipulation functions.
4. *Emulated Floating-Point* - a software floating-point package.

5. *Fourier Coefficients* - a numerical analysis routine for calculating series approximations of waveforms.

6. *Assignment Algorithm* - a task allocation algorithm.

7. *Huffman Compression* - a text and graphics compression algorithm.

8. *IDEA Encryption* - a block cipher.

9. *Neural Net* - a back-propagation network simulator.

10. *LU Decomposition* - an algorithm for solving linear equations.

Table 4.3: Target Platform Micro-Benchmark

	`nbench` [157]		`sgx-nbench` [158]		
Benchmark	***iter.* / sec.**	**σ (RSD)**	***iter.* / sec.**	**σ (RSD)**	***sgx overhead* % +\-**
Numeric Sort	$1.62 \cdot 10^3$	1.02 %	$1.49 \cdot 10^3$	1.83 %	+9.9 %
String Sort	$1.67 \cdot 10^3$	1.27 %	$8.15 \cdot 10^1$	1.38 %	+95.1 %
Bitfield	$7.78 \cdot 10^8$	0.29 %	$6.95 \cdot 10^8$	0.34 %	+14.5 %
Emulated Floating-Point	$7.48 \cdot 10^2$	0.23 %	$4.69 \cdot 10^2$	0.88 %	+37.2 %
Fourier Coefficients	$5.04 \cdot 10^4$	0.81 %	$1.09 \cdot 10^5$	0.25 %	-116.3 %
Assignment Algorithm	$6.62 \cdot 10^2$	1.26 %	$6.61 \cdot 10^2$	1.18 %	+0.2 %
IDEA encryption	$1.30 \cdot 10^4$	0.17 %	$1.27 \cdot 10^4$	1.55 %	+2.3 %
Huffman Compression	$6.22 \cdot 10^3$	0.32 %	$6.43 \cdot 10^3$	0.76 %	-3.3 %
Neural Net	$1.26 \cdot 10^2$	0.66 %	$1.05 \cdot 10^2$	0.56 %	+16.7 %
LU Decomposition	$2.94 \cdot 10^3$	1.24 %	$2.88 \cdot 10^3$	0.58 %	+2.04 %

Performance evaluation (measured in terms of algorithmic iterations per second) for the same benchmark, running on the same hardware and operating system, executing in two modes (REE and TEE). We measure the performance differences between the two execution modes using the reference hardware described in Section 4.5.1. The `nbench` [157] test suite provides a baseline performance measurement for the target platform processing tasks using the REE. The `sgx-nbench` [158] test suite—an SGX port of the `nbench` test suite—provides a measurement of the performance costs of executing tasks within the TEE of the target platform. The values in the iterations per second columns are an average of 20 test runs performed for each benchmark. The values in the RSD columns are the relative standard deviations (i.e., the coefficients of variation) of each series of test runs.

The `nbench` [157] test suite provides a baseline performance measurement for the target platform processing tasks using the REE. The `sgx-nbench` [158] test suite—an SGX port of the `nbench` test suite—provides a measurement of the performance costs of executing tasks within the target platform's TEE. We find that isolating code to the TEE adds a %1-%30 performance overhead depending on the type of computation (see Table 4.3). Two notable exceptions to this range are the String Sort tasks — where `sgx-nbench` runs twice as slow as the baseline, indicating that this type of computation is relatively more expensive to run in isolation — and Fourier Coefficients task — where `sgx-nbench` runs twice as fast as the baseline, indicating that certain computations benefit from running in isolation perhaps due to optimizations designed to offset average performance penalties.

4.5.3 Study 2: Fog Mediation App-Benchmark

Face recognition is a widely used application. To successfully figure out whether a particular person present in an image or not, it is useful to first detect all of the human faces in an image. Therefore, the less complex task of locating human faces inside an image, or face detection, became a popular research topic in the computer vision field. Similarly, it is useful to be able to distinguish between the varies human facial features such as the boundary of eyes, mouth, and nose — facial landmark detection. For testing the app benchmark, we selected real-time facial landmark detection using OpenCV [159]. OpenCV's facial landmark detection has three different implementations: FacemarkKazemi [160], FacemarkAAM [161], FacemarkLBF [162]. All of them follow a similar pattern. We selected FacemarkLBF for detecting facial landmarks.

We use AES for the encryption and decryption of input image files. We use the WolfSSL framework with Intel SGX support for performing cryptographic operations inside the SGX enclave [163].

1. AES Key Size = 256 bit

2. No. of Images = 1000

3. Mode = Simulation (32 bit)

4. Platform = Microsoft Visual Studio 2015

We use 256 bit AES key and run the experiments with 1000 images of different people where each image size is approximately 7-14 KB [164]. In this experiment, we use SGX enclave in simulation mode and Microsoft Visual Studio 2015.

Table 4.4: Baseline Analysis of Fog Mediation

Functionality	***Single Image***	***Multiple Images* (1000 Images)**
Plain Image(s) Size	7725 bytes	14,231,214 bytes
Encrypted Image(s) Size	7728 bytes	14,238,736 bytes
Encryption (TEE)	1 ms	2.167 s
Decryption (TEE)	1 ms	1.463 s
Facial Landmark Detection (REE)	6.633 s	115.284 s
Total Computation Time	6.635 s	118.914 s

We provision the shared secret key along with the encrypted data from the untrusted zone to the trusted enclave. Inside the enclave, we decrypt all the image files using the AES algorithm. After decryption, all images are sent back to the untrusted zone for further image processing computation. When all of the images are decrypted by the enclave, the application completes facial landmark detection on all the images inside the untrusted zone. The measured computational time are listed in Table 4.4.

We further investigate fog mediation operations along with some of the basic operations of SGX enclave to get a clear view of the computational time required for each operation. We investigate the required time for enclave creation, encryption, and decryption of 1000 images and facial landmark detection time on those 1000 images. We run our experiments using the same settings over 100 times and list the mean running times and standard deviations of those experiments in Table 4.5.

Table 4.5: Measurement of Fog Mediation Operations

Functionality	***Mean (ms)***	***Std Dev (ms)***
Create Enclave	10.63	1.75
Encryption of 1000 images (TEE)	1523.67	129.16
Decryption of 1000 images (TEE)	1758.47	268.02
Facial Landmark Detection (REE)	119850.12	6844.59
Destroy Enclave	1.21	0.19

4.6 Conclusion

This research develops the concept of fog mediation as a method for extending the security guarantees of an isolated execution environment to the IoT. Fog mediation leverages hardware-based security primitives integrated into fog computing elements to provide end-to-end security guarantees, improved performance, and mediate trust decisions between the cloud and co-located mobile devices. We evaluate the performance of fog mediation in terms of the platform-level overheads and end-to-end application-level run-times using Intel SGX reference hardware.

Future work will explore extensions to the fog mediation architecture that provides additional security assurances. In particular, this work will explore novel notions of personal privacy and data ownership in the context of fog mediated computing as well as propose networking and software programming abstractions leveraging fog mediation.

Table 4.6: Cloud Latency by Region

Data Center		Response Time (ms)			
Platform	**Location**	***low***	***median***	***high***	σ
GCE (asia-northeast1-b)	Tokyo, Japan	179 ms	180 ms	189 ms	2.65
GCE (us-central1-c)	Iowa, USA	51 ms	51.5 ms	55 ms	1.32
GCE (europe-west4-b)	Netherlands	105 ms	108.5 ms	114 ms	2.34
GCE (us-west1-b)	Oregon, USA	88 ms	91 ms	93 ms	1.42
GCE (us-east1-c)	S. Carolina, USA	33 ms	35 ms	38 ms	1.26
GCE (australia-southeast1-a)	Sydney, Australia	220 ms	230.5 ms	395 ms	77.62
GCE (southamerica-east1-a)	São Paulo, Brazil	160 ms	162 ms	170 ms	2.63
GCE (europe-west2-a)	London, UK	96 ms	99 ms	106 ms	2.47
GCE (asia-east1-b)	Changhua, Taiwan	203 ms	205.5 ms	333 ms	35.34
GCE (asia-southeast1-b)	Jurong, Singapore	238 ms	242 ms	411 ms	67.53
GCE (us-east4-a)	Virginia, USA	21 ms	23 ms	25 ms	1.11
GCE (europe-west1-c)	St. Ghislain, Belgium	105 ms	106 ms	112 ms	1.85
GCE (asia-south1-a)	Mumbai, India	299 ms	357.67 ms	421 ms	53.96
GCE (europe-west3-c)	Frankfurt, Germany	109 ms	111 ms	119 ms	2.77
GCE (northamerica-...1-a)	Montréal, Canada	35 ms	37 ms	45 ms	2.58
AWS (us-east-2)	Ohio, USA	80 ms	83.5 ms	89 ms	2.8
AWS (us-west-2)	Oregon, USA	216 ms	221 ms	226 ms	2.75
AWS (sa-east-1)	São Paulo, Brazil	302 ms	309 ms	320 ms	5.85
AWS (us-east-1)	N. Virginia, USA	55 ms	65.5 ms	72 ms	4.89
AWS (us-west-1)	California, USA	192 ms	199 ms	204 ms	3.2
AWS (ap-northeast-2)	Seoul, Korea	235 ms	451.5 ms	614 ms	147.67
AWS (eu-west-1)	Ireland	211 ms	215.5 ms	224 ms	3.43
AWS (ap-northeast-1)	Tokyo, Japan	213 ms	517.5 ms	615 ms	145.66
AWS (ap-southeast-1)	Singapore	282 ms	481.5 ms	699 ms	133.35
AWS (eu-central-1)	Frankfurt, Germany	233 ms	244 ms	252 ms	5.72
AWS (ca-central-1)	Central Canada	72 ms	81 ms	86 ms	3.48
AWS (eu-west-3)	Paris, France	210 ms	224 ms	230 ms	6.64
AWS (eu-west-2)	London, UK	196 ms	200.5 ms	205 ms	2.72
AWS (ap-southeast-2)	Sydney, Australia	238 ms	482.5 ms	614 ms	136.27
AWS (ap-south-1)	Mumbai, India	226 ms	453 ms	615 ms	134.01

Round-trip delay times (RTDs) of ICMP pings with 72 byte data frame transferred between source connection (Pennsylvania, USA) and regional Google Compute Engine (GCE) and Amazon Web Services (AWS) EC2 cloud installations. Reported response times based on 12 tests preformed for each location in February 2018.

Part IV

Smart Spaces: Contextual Privacy Controls

Chapter 5

Privacy Partitioning

This chapter is largely a reproduction of the paper *Privacy Partitioning: Protecting User Data During the Deep Learning Inference Phase* co-authored with Jianfeng Chi, Xuwang Yin, Tong Yu, Yuan Tian, and Patrick Tague [41]. This research was supported in part by participation in the Northrop Grumman Cybersecurity Research Consortium.

Data Access Through User Mediation (DATUM). We present a deep network optimization framework for protecting personal data during the inference phase of deep learning. This framework applies to client-server deployment scenarios where the client (the app tasked with handling inputs to a deep network) has a data privacy requirement. In the proposed framework, a deep learning model is split into two parts — a local partition and a remote partition. The two partitions are trained such that they jointly achieve a similar classification accuracy as the reference deep network with the added constraint that the intermediate state that is generated by the local partition and received by the remote partition is less useful for unauthorized learning tasks. We term this approach *privacy partitioning*.

We quantify the protection strength and performance cost of this framework in a series of experiments using benchmark computer vision tasks: MNIST (handwritten digit classification), LFW (face recognition), CIFAR-10 (object detection), and AgeDB (age and gender classification). The experimental results demonstrate that the performance optimization mechanism of deep networks can be extended to simultaneously protect user data. We

achieve a 0.02%-4.26% performance cost in the experiments where privacy partitioning is applied to defend against input inference (MNIST, LFW, and CIFAR-10 benchmarks). We achieve a 0.60% performance cost in the experiment where privacy partitioning is applied to defend against private attribute inference (AgeDB benchmark).

5.1 Overview

Consider the case where N users of a mobile app and each user collects and stores personal data locally on their mobile devices. The goal of collaborative deep learning is to leverage a deep neural network (DNN) that incorporates the data of all N user devices to produce relevant personalized services for each user [165, 166, 167, 168, 169].

Due to the potentially sensitive nature of personal data, we would like to achieve a high-performance deep network while simultaneously providing strong data privacy guarantees for all participants. That is, we would like to maximize the accuracy of the learned model while minimizing the number of individual attributes that can be leaked or inferred about each participant when the learned model is in use. Individual attributes include unprocessed input data, such as audio from a voice assistant device or video from a home security system, as well as personally-identifiable user attributes, such as identity, interests, habits, location, or social network.

In general, there are two architectural approaches to learn large-scale deep networks: *centralized learning* and *distributed learning*. The choice between the two (as well as the spectrum of topologies spanning them) have differing implications for performance, scalability, ease of deployment, and data privacy.

Centralized Learning. Traditionally, machine learning frameworks have assumed access to a centralized repository of data (or otherwise considered data mining as outside the scope of model learning and inference tasks). The centralized approach is developer-friendly since a single operator can manage the entire process. Access to a unified data store of user data is

an appealing option for service providers even in the case of privacy-sensitive learning tasks requiring access to personally-identifiable datasets [170].

Investigative reports regarding popular voice devices suggest that some products do stream clear audio to the cloud — perhaps for leveraging more powerful server-side computational resources — and that in some cases employees do access customer audio recordings — perhaps as a practical consideration for simplifying manual engineering tasks and honing user experience [171, 172]. Similarly, developer plug-ins required to generate ad revenue also act as data collection vehicle that communicates privacy-sensitive user data labeled with device identifiers to third party analytics providers — even in-app use cases that do not call for advanced collaborative learning (e.g., a simple health tracker) [173].

This incoming stream of user data can be collated with other datasets, annotated for utilization in supervised learning models, and directly applied to unsupervised deep learning models to improve existing services and or develop entirely new the customer experiences. In light of this, restricting the model inference phase exclusively to local processing on the client may not be a viable commercial solution. For similar reasons, centralized learning presents a significant privacy risk to users.

Cryptography-based protocols have also been proposed to protect user privacy during the model inference phase [174, 175]. However, these protocols impose significant computational overheads. Differential privacy-based protection mechanisms have largely focused on protecting training data (i.e., privacy-preserving model learning) [176, 177]. However, deep learning services may pose more of a privacy risk during the model inference phase where potentially more user data is processed. Researchers have also proposed applying differential privacy during the model inference phase [178]. However, these approaches only apply to collaborative learning tasks that produce aggregate results (e.g., crowd-sourced private statistical database) and not to the problem addressed here — the common task of processing individual data items.

Distributed Learning. Researchers have proposed a variety of decentralized architectures

as solutions to both performance and privacy. Decentralized deep learning (also known as distributed or federated deep learning) includes proposals for building and updating a unified model without the need to store individual data in the cloud and proposals that combine both a personalized individual model managed locally on user devices and a shared model constructed from anonymity averaged data [176, 177].

In principle, distributed learning enables participants to enjoy the full benefits of rich shared models without the need to centrally store data. This benefit often comes at the expense of increased complexity due to asynchronous operations and an assorted computing and data management context. Additionally, fully distributed learning implementations may complicate attempts to protect the trade secrets and intellectual property of the DNNs.

Partitioned Learning. We present partitioned learning as an alternative to centralized and distributed learning architectures. Partitioned learning is a variant of centralized learning since all input data is collected by a single administrative domain. This variant provides the ease of software development benefits of centralized learning while realizing the user-controlled privacy benefits of distributed learning.

In a deep network, each successive intermediate layer performs a transformation operation using the output of the preceding layer. Prior work has demonstrated that the output of intermediate layers (hidden layer activations) can be used to successfully launch model inversion style attacks including input inference and private attribute inference [179, 180]. On the other hand, if a deep network generates hidden layer activations that cannot be used to infer private attributes or discover input data then the network itself may serve as a basis for user privacy. Further, if the privacy-enhanced hidden layer activations can be utilized in subsequent iterations of update the model then the solution is not an obstacle to improving customer experience like the local processing only approach. We demonstrate that these user privacy and data utility properties are jointly achievable experimentally.

In a partitioned learning architecture, deep network Θ is partitioned into two parts, resulting in bipartite network $\{\Theta_l, \Theta_r\}$ where access to the network's inputs are restricted to

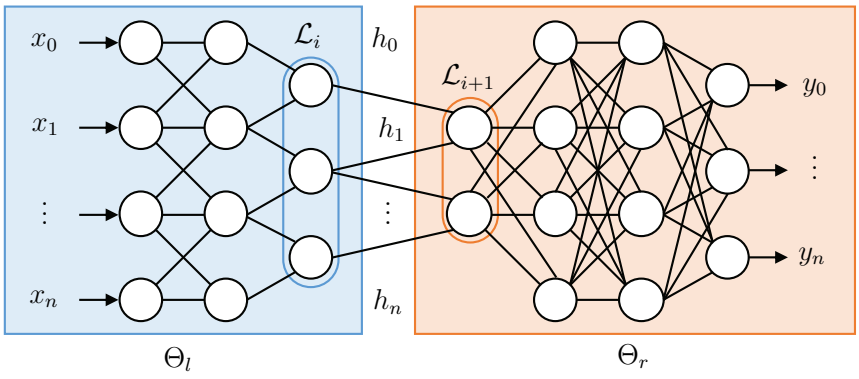


Figure 5.1: Partitioned Deep Network Topology. This figure depicts privacy partitioning applied to deep convolutional neural network Θ. Partition Θ_l is hosted by the *local* computing context. Partition Θ_r is hosted by the *remote* computing context. Hidden layer activation $\mathcal{H}$ is the output of the last local layer $\mathcal{L}_i$. Given deep network Θ, this framework generates partitioned network $\{\Theta_l, \Theta_r\}$ that learns model $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ while attenuating the potential for adversarial networks to learn function $f_{\theta_a} : \mathcal{H} \rightarrow \mathcal{Z}$ where $\mathcal{Z}$ is a vector of private attributes and learning $\mathcal{Z} = \mathcal{X}$ is equivalent to launching an input recovery attack.

the local partition Θ_l (Figure 5.1). We consider the case where an adversary has complete access to the remote partition Θ_r including its input $\mathcal{H}$. The optimization goal the entire network is high accuracy whereas the optimization goal of the local portion of the network is to reduce the extent to which such an adversary can discover input data or infer additional user characteristics (see Figure 5.1).

We demonstrate that the deep network optimization functionality can be augmented to reduce the extent to which such an adversary can discover input data or infer additional user characteristics given access to intermediate states generated by a portion of the deep network. In the case of privacy partitioned age and gender estimation service, for example, the users of the service have the assurance that other inference tasks such as subject identification cannot be easily learned given access to the output of the local layers $\mathcal{H}$ the remote layers in Θ_r.

The bipartite topology is applicable to the many app use cases featuring a private local data handling and public remote data processing deployment model. This includes many

client-server use cases, private edge computing and public cloud computing integration, and use cases where resource-constrained mobile devices must offload to privacy-sensitive user data to resource-rich proximate infrastructure for computationally-expensive analysis.

Contributions. In this study we present privacy partitioning, a privacy protection framework for user-controlled model inference and model learning. In summary, this work makes the following contributions:

- We propose a framework for learning accurate deep networks that are resilient against input inference and private attribute inference attacks.
- Unlike related differential privacy research that protects statistical databases (aggregate data) during the model inference phase, we propose a solution that protects the individual queries.
- We experimentally demonstrate the effectiveness of our approach. In all experiments, the results indicate that the privacy partitioning framework can significantly reduce the privacy risk potential of deep network activation states.

Organization. The topic of the next section, Section 5.2, is the problem definition including the deployment model, adversary model, assumptions, and desired properties. Section 5.3 presents the proposed framework. Section 5.4 presents experimental results for the privacy protection mechanism proposed here using three computers vision classifier networks. Section 5.6 discusses the related literature, and Section 5.7 presents a summary of conclusions.

5.2 Problem Definition

In general, the ideal problems for utilizing the privacy partitioning framework have some or all of the following aspects: (i) the inputs to the deep network are privacy-sensitive, (ii) it is required or advantageous to implement user-controlled restrictions to remote data

usage, (iii) it is required or advantageous to decouple data handling processes from model learning and model inference processes, or (iv) it is advantageous to support deep network optimization-based privacy controls.

In this section, we introduce the deployment model, threat model, assumptions, and design goals for the proposed framework. For ease of presentation, we present the privacy partitioning framework using the case study of a deep learning IoT service deployed to edge computing infrastructure. In this case study, the bipartite deep network is deployed to a private edge computing network (local computing context) and a public cloud computing service (remote computing context).

Case Study: Edge Computing. Consider the case where admins of the local computing context manage M mobile and embedded devices for N users (e.g., an enterprise IoT deployment). Collectively, the devices comprising the local computing context collect and store contextual data regarding the users and activities within the local domain. The admins are responsible for maintaining the confidentiality of user and sensor data generated within the local computing context and for initiating requests to deep learning IoT services. The remote computing context is responsible for completing queries received from the local computing context.

5.2.1 Model & Assumptions

In this study, we evaluate the privacy implications of deep network hidden layer activations during the model inference stage. In the case of centralized learning, the entirety of deep network Θ is deployed to a computing context managed by a single administrative domain. The threat model defined here involves a deep network service deployed to two administrative domains: the data custodian (local admin) and the cloud computing provider (remote admin). This deployment model addresses a commonly occurring secrecy requirement where a data custodian would like to safely leverage remote compute resources on behalf of its users and devices.

Deployment Model. Our use case defines outsourced computation in the sense advocated by the edge computing and cloud computing deployment models. Accordingly, we identify the following key stakeholders and their primary roles:

(i) The ***edge provider*** (local admin) offers edge network-based computing, storage, and network resources (i.e., the local computing context) for use by the network of co-located users and embedded devices. During the inference stage, it hosts the first few layers of the deep learning model, and sends the output of this local partition (denoted by Θ_l) to the remote cloud for the classification. The local admin is tasked with maintaining the confidentiality of data originating from its local computing context.

(ii) The ***cloud provider*** (remote admin) offers data center-based compute, storage, and network resources (i.e., the remote computing context) to cloud customers who wish to lease them for a certain amount of time. During the inference stage, it receives the output of local partition Θ_l, and host the remaining layers (denoted by Θ_r) of the deep learning model.

(iii) The ***service provider*** (app developer) wishes to lease edge computing and cloud computing resources to deploy its machine learning as-a-service (MLaaS) offerings. Its offerings must include privacy protections for the users of the edge provider's local computing context. In particular, the solutions should have resilience against the use of deep network intermediate states for input inference and private attribute inference style attacks without resulting in significant reductions to the accuracy of the service.

Additionally, the privacy protections should be able to leverage the optimization functionality of deep networks and be readily applicable to existing deep networks to make software development easier. Similarly, it's desirable for the privacy protection mechanism to be deployed alongside best practice end-to-end encryption and privacy-preserving model learning mechanisms.

Finally, (iv) the ***end user*** (user device) wishes to use authorized deep learning services to analyze user data while restricting the confidentiality of user data to the local computing

context.

Adversary Model. We consider two classes of adversary: (i) the remote attacker (e.g., an honest-but-curious service provider or unauthorized remote access in the case of a data breach) with access to the output of local partition $\mathcal{H}$, remote partition Θ_r, and the output of the remote partition $\mathcal{Y}$, and (ii) the network attacker with access to the output of the local partition $\mathcal{H}$. We consider two classes of attacks: (i) the input inference attack: discovery of the original user input, and (ii) the private attribute inference attack: unauthorized extraction of a private user attribute.

Assumptions. We consider data compromises resulting from the local computing context out of scope. Additionally, we consider side-channel attacks beyond the scope of this work. The servers constituting local and remote computing contexts are trusted to operate normally (i.e., we consider denial-of-service style attacks out of scope). The attacker may know, may partially know, or may not know the dataset that is used during the model learning phase to train and validate bipartite network $\{\Theta_l \cdot \Theta_r\}$. We evaluate the framework using attackers who do have access to the training data used to train $\{\Theta_l \cdot \Theta_r\}$.

5.2.2 Desired Properties

The following list contains the desired properties for privacy partitioning.

*P*1 **Privacy.** We would like the following data privacy objectives to be satisfied:

- *P*1.1 *Input Inference Resilience.* Protections can reduce the usefulness of deep network intermediate states to input inference style attacks.
- *P*1.2 *Input Repurposing Resilience.* Protections can reduce the usefulness of deep network intermediate states to private attribute inference style attacks.

*P*2 **Utility.** We would like the following software development utility objectives to be satisfied:

*P*2.1 *Performant.* Protections result in a negligible reduction to model accuracy.

*P*2.2 *Learning-Based.* Protections do not require processes that are outside of the deep learning tool-set such as managing cryptographic secrets.

*P*3 **Versatility.** We would like the following framework versatility objectives to be satisfied:

*P*3.1 *Compatible.* Protections can be readily applied to existing deep networks.

*P*3.2 *Complementary.* Protections can be readily deployed alongside end-to-end encryption mechanisms and privacy-preserving model learning mechanisms.

5.3 Framework

In this section, we present a deep learning optimization for learning accurate deep networks that are resilient against input inference (*P*1.1) and attribute inference attacks (*P*1.2). We term this optimization framework *privacy partitioning*.

5.3.1 Partitioned Learning

The privacy partitioning framework encapsulates a process for building privacy-enhanced deep learning models as well as a process for deploying the resulting network. An overview of these two processes (model learning and model inference) are shown in Figure 5.2. The desired outcome of the model learning phase is an accurate inference model with privacy protections in place (Section 5.3.2). The resulting deep network has two partitions with parameters that are tuned such that the output of local partition Θ_l, hidden layer activation $\mathcal{H}$, does not reveal private attribute $\mathcal{Z}$ while the output of remote partition Θ_r is highly accurate. This enables Θ_r to be deployed remotely without jeopardizing privacy attributes defined by $\mathcal{Z}$.

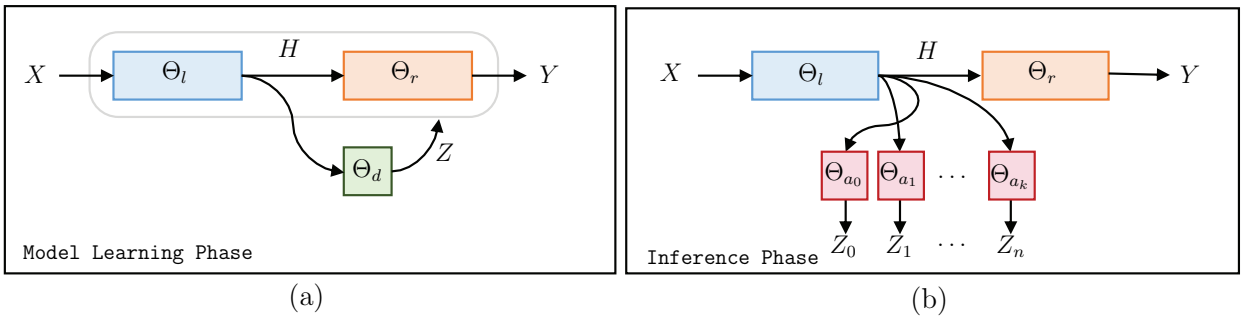


Figure 5.2: Model Learning and Model Inference. This figure illustrates the two phases of deploying privacy partitioning protections to a deep network: (a) the model learning phase and (b) the inference phase. The model learning phase encompasses several machine learning activities including model training and validation. The inference phase refers to processing inputs from end-users after the network has been deployed. During the model learning phase, deep network Θ is first partitioned into a bipartite topology $\{\Theta_l, \Theta_r\}$ and defender Θ_d, acting as an oracle for attacker behavior, is added as a feedback mechanism for minimizing the potential of recovering private attribute Z in X via hidden state H (see Equation 5.2). During the inference phase — when bipartite network $\{\Theta_l, \Theta_r\}$ has been deployed — the adversary devises counterpart networks $\{\Theta_{a_0}, \Theta_{a_1}, ...\Theta_{a_n}\}$ that each attempt to recover Z from H.

For example, assume that a deep network Θ implements an object detection service that takes an image as input and indicates whether or not an apple is present in the image (f_θ : `image` $\rightarrow$ **boolean**: `containsApple`). In the case where private attribute $\mathcal{Z}$ is set to the input vector for oranges, this framework generates partitioned deep network $\{\Theta_l, \Theta_r\}$ that detects apples with similar accuracy as reference deep network Θ with the added optimization that ensures the outputs generated by $\{\Theta_l$ are not suitable for detecting the presence of oranges in $\mathcal{X}$. In the cases where private attribute $\mathcal{Z}$ is set to input vector $\mathcal{X}$, privacy partitioning protects against image reconstruction.

In summary, to protect user's data during the inference stage, we train the machine learning model so that the remote portion that will be deployed in the cloud server carries less sensitive information about the user. For example, during the inference stage, it will be more challenging for attackers to infer the private attribute of user data or reconstructed user input. Since the privacy partitioning framework protects the privacy of user data while maintaining the utility of the data, it enables offloading a portion of the machine learning

task to remote computing resources.

Comparison with Differential Privacy. An alternative approach to privacy-preserving data analysis is differential privacy [181]. Differential privacy aims to ensure that a private statistical database (i.e., a database of statistics resulting from the analysis of confidential data) does not overly depend on any individual entry belonging to the underlying dataset the results are based on. The intuition is that the maximally private scenario for each individual is if their data was not part of the confidential dataset, to begin with. Therefore, privacy loss to an individual can be quantified in terms of how similar a private statistical database that includes their data is to that same database less their entry. If this differential is below a certain threshold for individuals, statistically, we conclude that an adversary cannot identify individual subjects by querying the statistical database.

Since differential privacy provides privacy guarantees for individuals entries used for producing an aggregate statistical result it has direct application to providing privacy to deep learning during the model learning phase. Whereas privacy partitioning provides privacy protections for each input to a deployed deep network during the model inference phase. Thus, they are complementary solutions but not directly comparable. Since they are complementary, researchers can use both the differential privacy and privacy partitioning to protect user data during both the model learning and the model inference phases. Since they are complementary, researchers can use both the differential privacy and privacy partitioning to protect user data during both the model learning (for privacy-preserving database membership) and the model inference phases (for privacy-preserving query).

5.3.2 Model Learning Phase

Intuitively, an initial motivation for computing a portion of the deep network locally is because the hidden states of the successive layers contain more transformations, resulting in less a traceable representation with regards to the attributes of the original inputs to the network.

However, this initial measure is not enough because an attacker can, in many cases, generate a fairly accurate estimation of the privacy attribute $z \in \mathcal{Z}$ of the raw input $x \in \mathcal{X}$ from the output of the local layers $h \in \mathcal{H}$ (i.e., the attacker can compute $f_{\theta_a} : \mathcal{H} \rightarrow \mathcal{Z}$).

A privacy partitioned deep network should meet two goals. (i) Bipartite network $\{\Theta_l, \Theta_r\}$ should be a functional approximation of reference deep network Θ — i.e., the resulting model should achieve good performance. (ii) It should be prohibitively difficult for an attacker to reconstruct raw input data or estimate the private attributes of the input data given the output of local layers. Therefore, when training the models, we need to optimize the objective function so that model performance is maintained as the potential for such an attack is lessened.

In effect, we would like to prevent the attacker from discovering the inputs by ensuring the local layer operations are irreversible. To achieve this, we introduce an additional component into the model learning phase: *defender* (Θ_d). The role of the defender is to simulate the attackers. That is, the defender attempts to infer the private attributes of input data or discover the input data given hidden state $h \in \mathcal{H}$. The defender network and the bipartite network are trained concurrently with the defender providing feedback regarding the efficacy of the privacy partitioning during each round and the bipartite network updating it's function $f_{\theta_d} : \mathcal{H} \rightarrow \mathcal{Z}$ based on this information (see Figure 5.2 (a)).

Suppose we would like to learn a DNN model $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ with the training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^m$ with each individual data containing an private attribute z_i (e.g., $z_i = x_i$ for input inference attack). According to our bipartite design, the deep learning model can be formulated as $f_\theta = f_{\theta_l} \circ f_{\theta_r} = f_{\theta_r}(f_{\theta_l}(\cdot))$, where $f_{\theta_l} : \mathcal{X} \rightarrow \mathcal{H}$ is the function mapping from input domain $\mathcal{X}$ to domain of the intermediate activation state $\mathcal{H}$ in local partition Θ_l and $f_{\theta_r} : \mathcal{H} \rightarrow \mathcal{Y}$ is the function mapping from $\mathcal{H}$ to output domain $\mathcal{Y}$ in remote partition Θ_r.

According to our design, the defender learns a mapping function $f_{\theta_d} : \mathcal{H} \rightarrow \mathcal{Z}$ and its objective can be formulated as:

$$\min_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \delta\big(z_i, f_{\theta_d}(f_{\theta_l}(x_i))\big) \tag{5.1}$$

where $\delta\big(\cdot,\cdot\big)$ is the privacy distance metric between the ground truth label of private attribute and the estimated private attribute by the defender. In Section 5.4 we will present some choices of $\delta\big(\cdot,\cdot\big)$ in different tasks.

When training the model, the service provider would leverage the output (guess of $\mathcal{Z}$) of defender's as feedback to better optimize its parameters. The overall objective function of a service provider can be formulated as:

$$\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} l\big(y_i, f_\theta(x_i)\big) - \lambda \cdot \delta\big(z_i, f_{\theta_d}(f_{\theta_l}(x_i))\big) \tag{5.2}$$

where $\theta = \{\theta_l, \theta_r\}$, and $l\big(\cdot,\cdot\big)$ denotes the loss function for the original task of the model. λ is the defender weight that help the service provider trades off between utility loss and privacy loss.

To train different components collectively — e.g., to train the partition and defender networks simultaneously — we use alternating update algorithms as it is commonly done in the training Generative Adversarial Nets [182]. The details are described in Algorithm 5.1 by replacing the defender suite with the single defender.

5.3.3 Model Inference Phase

After the privacy partitioned model has been properly deployed to local and remote computing contexts, the adversary would like to learn a mapping function $f_{\theta_a} : \mathcal{H} \rightarrow \mathcal{Z}$ among all possible DNN architectures. We assume the attacker owns an auxiliary dataset $\mathcal{D}' = \{\hat{x}_i, \hat{z}_i\}_{i=1}^{n}$ that helps them to learn f_{θ_a} with the objective function formulated as follows:

$$\min_{\theta_{a_i} \in \{\theta_{a_1}, \ldots, \theta_{a_k}\}} \min_{\theta_{a_i}} \frac{1}{n} \sum_{i=1}^{n} \delta\big(\hat{z}_i, f_{\theta_{a_i}}(f_{\theta_l}(\hat{x}_i))\big) \tag{5.3}$$

Note the auxiliary dataset $\mathcal{D}'$ that the attacker uses for might be different from the model's training data. Attacker's training data can be the data the attacker collected by himself/herself, part of the training data set or the whole training data in the worst case.

In Equation 5.3, the attacker first obtains the output of the local partitions by querying their auxiliary dataset. Then, they use the output as well as the corresponding private label z to train the adversary network θ_a that could help perform the attack.

5.3.4 Concealed Learning Phase

This section describes a process by which a privacy partitioned deep network may be updated online to incorporate new data. This provides input inference resilience and private attribute inference resilience for data included in an update. We denote these updates as the *online learning phase* to distinguish it from the initial learning phase described thus far.

Let $\mathcal{L}_i$ denote the ith intermediate layer in the reference deep network defined by Θ and its corresponding bipartite deep network $\{\Theta_l \cdot \Theta_r\}$. Let $\{\mathcal{L}_i, \mathcal{L}_{i+1}\}$ refer to neighboring intermediate layers at the partition point of $\{\Theta_l \cdot \Theta_r\}$ where $\mathcal{L}_i$ is the last local layer and $\mathcal{L}_{i+1}$ is the first remote layer. The process for supporting online updating of the remote partition occurs as follows:

1. initial learning phase:
    (a) select reference network Θ
    (b) select privacy partitioning point $\{\mathcal{L}_i, \mathcal{L}_{i+1}\}$
    (c) learn bipartite network $\{\Theta_l, \Theta_r\}$
2. online learning phase:
    (a) lock local partition Θ_l

(b) securely generate updates $f_{\theta_l} : \mathcal{X} \rightarrow \mathcal{H}^*$

(c) update remote partition $f_{\theta_r} : \mathcal{H}^* \rightarrow \mathcal{Y}$

(d) validate and test update $f_{\theta^*} : \mathcal{X} \rightarrow \mathcal{Y}$

(e) deploy bipartite network update $\{\Theta_l, \Theta_{r^*}\}$

The inclusion of an online learning phase, occurring after the initial model is learned, has the primary benefit of completely removing any requirement to conduct data collection and curation tasks in coordination with a remote computing context. Our construction thus far assumes an initial learning stage where a single administrative domain requires access to the entire topology and training data set. During the online learning phase and any subsequent inference phases, the protections of privacy partitioning extend to training, validating, and testing processes. Thus, there is no need for a specialized single admin learning phase after the initial learning phase used to generate the local partition. Further, a remote admin and local admin can negotiate an online update without having to grant training data access to the remote computing context or requiring local computing context to manage the entire process.

Additionally, during the online learning phase, the locked local partition may be deployed individually to each device (as opposed to a single local computing context) to implement fully distributed concealed collaborative learning. The lightweight local partition acts as a privacy filter for each device. For the user, this modifies the privacy assurance from confidentiality provided by a local admin to secrecy provided by their device both during model inference and model learning phases. Individuals can opt-in to improving the shared model by contributing only their protected hidden layer activations. These contributions can be directly applied to updating unsupervised learning models. For updates to supervised learning models, users must also provide labels (as in crowd-sourced tagging) since this information can no longer be inferred from the protected hidden layer activations.

5.3.5 Multiple Defenders

The depiction of the model learning phase in Figure 5.2 (a) shows a single defender deployed at a single deep network partition point $\{\Theta_l \cdot \Theta_r\}$. In practice, the privacy partitioning protections for a given deep learning task f_θ may be extended to include more than one defender loss function at a given partition point. In other words, the defender f_{θ_d} can be extended to *defender suite* $\mathcal{F}_D$=$\{f_{\theta_{d_0}}, f_{\theta_{d_1}}, \ldots, f_{\theta_{d_D}}\}$ at the cost of increasing computational complexity of the network and increasing the time it takes to learn f_θ. Including more defenders at the partition point provides more robust privacy protections for the associated hidden state since the model can leverage the best defender of the defender suite in the model learning phase, which can be formulated as follows:

$$\begin{aligned}\min_{\theta=(\theta_l,\theta_r)} \frac{1}{m}\Big(\sum_{i=1}^{m} l\big(y_i, f_\theta(x_i)\big) \\ - \lambda \min_{\theta_{d_i}\in\{\theta_{d_1},\ldots,\theta_{d_D}\}} \min_{\theta_{d_i}} \sum_{i=1}^{m} \delta\big(z_i, f_{\theta_{d_i}}(f_{\theta_l}(x_i))\big)\Big)\end{aligned} \tag{5.4}$$

However, solving this optimization problem is difficult due to the complex form of the optimization problem. In practice, the model provider may train multiple defenders individually using a batch of training data in each update step. Then, he/she chooses the best defender from the defender suite in each update step and updates the model parameters based on the chosen defender using SGD. The model learning process with multiple defenders is described in Algorithm 5.1.

Function 5.1: Privacy Partitioning with Multiple Defenders

INPUT: Training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{m}$

OUTPUT: Trained model f_θ

1: Initialize model θ and defender suite $\{\theta_{d1}, \ldots, \theta_{dD}\}$

2: **for** $t \in [T]$ **do**

3: **for** each mini-batch $\{x_i, y_i\}_{i=1}^{B} \in \mathcal{D}$ **do**

4: 1. Update the defender suite parameters $\theta_d \in \{\theta_{d1}, \ldots, \theta_{dD}\}$ via SGD using the gradient

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} d\big(x_i, f_{\theta_d}(f_{\theta_l}(x_i))\big)$$

5: 2. Choose the best defender via

$$\theta_d = \underset{\theta_d \in \{\theta_{d1}, \ldots, \theta_{dD}\}}{\arg\min} \frac{1}{B} \sum_{i=1}^{B} d\big(x_i, f_{\theta_d}(f_{\theta_l}(x_i))\big)$$

6: 3. Update the model parameters θ via SGD using the gradient

$$\nabla_{\theta} \frac{1}{B} \sum_{i=1}^{B} l\big(y_i, f_\theta(x_i)\big) - \lambda \cdot d\big(x_i, f_{\theta_d}(f_{\theta_l}(x_i))\big)$$

7: **return** θ

5.3.6 Multiple Partitions

Another strategy for hardening the privacy partitioning framework is to extend a bipartite network $\{\Theta_l, \Theta_r\}$) with one privacy partitioning $\{\mathcal{L}_i, \mathcal{L}_{i+1}\}$ into a partition suite ($\{\mathcal{L}_i, \mathcal{L}_{i+1}\}, \{\mathcal{L}_j, \mathcal{L}_{j+1}\}, \ldots,$ Integration can occur by training a new deep network for each partition in the suite. Simultaneously learning more than one privacy partitioning provides a more complete protection surface and more deployment configuration options at the cost of additional training over-

head. The additional configurations enable flexible controls that can be adjusted to suit changing privacy and accuracy requirements.

5.4 Experiments

In this section, we experimentally demonstrate that the proposed framework can generate accurate deep networks with input inference and private attribute inference resilience in a series of benchmark computer vision applications. Section 5.4.1 presents the methodology used for evaluating the privacy partitioning framework. Section 5.4.2 presents a proof of concept validation of the privacy partitioning framework using the MNIST dataset [183]. Section 5.4.3 presents a more comprehensive validation of the privacy partitioning framework using a selection of datasets and classifier scenarios.

5.4.1 Evaluation Methodology

The evaluation aims to measure how well privacy partitioning performs against state-of-the-art input inference and attribute inference style attacks. We quantify the protection strength and performance cost of privacy partitioning in a series of experiments using benchmark computer vision tasks: MNIST (handwritten digit classification), LFW (face recognition), CIFAR-10 (object detection), and AgeDB (age and gender classification).

To evaluate the performance of the input inference attack, we use similarity metrics that are highly consistent with human perceptual assessments to quantify the similarity of recovered inputs as compared to original inputs. Specifically, we use mean square error (MSE) and structural similarity index (SSIM) [184] in our experiments. MSE=0 or SSIM=1 score corresponds to a perfect reconstruction. We also use deep perceptual loss (DPL) [185] and reprint accuracy (the model accuracy when inputting recovered data to pre-trained DNNs) to quantify the machine perceptual loss of DNNs. All attacker networks are trained to minimize human perceptual loss in input inference attacks after the model learning phase.

For the attribute inference attack, the attacker attempts to infer the private attribute Z of X from H. All of the assumptions for the input inference attacks scenario also apply to the attribute inference attack scenario with the exception that the outputs of the attacker DNNs are guesses of Z. We evaluate the performance of attribute inference attack using accuracy scores.

The evaluation process for each benchmark is as follows. Given a classification task and reference deep network Θ, we generate bipartite network $\{\Theta_l \cdot \Theta_r\}$ using the privacy partitioning method described in Section 5.3. We then evaluate the protection strength of $\{\Theta_l \cdot \Theta_r\}$ in terms of how well it reduces the accuracy of an input inference attack by attacker DNN Θ_a.

Initially, we tested the privacy partitioning framework against the most powerful model used for input inference found in the research literature. We tested the state-of-the-art input (image) inference attack proposed by Fredrikson et al. and found that privacy partitioning is very effective against this implementation [179]. This attacker model achieves an SSIM of < 0.2 for all datasets in primary evaluation (Section 5.4.3). Next, we devised a stronger attacker DNN that (1) has access to the full training dataset and (2) is better calibrated for visually dissimilar class members.[1]

In all experiment settings, we assume that the attacker has full access to the training dataset. In practice, the attacker does not need access to the same training dataset as the target network to learn a deep network for input discovery (e.g., the attacker may use publicly available images to compromise an image classifier). However, we are interested in evaluating the effectiveness of the proposed protection mechanisms under the best-case scenario for the attacker. This assumption is the most difficult to defend against since the attacker has access to the same data distribution that was used to train the targeted model.

We focus on two types of attacks: the input inference attack and the attribute inference attack. For the input inference attack, we assume the attacker trains DNNs to discover

[1]It has been demonstrated that the input inference model presented in [179] is unlikely to succeed when class members are not all visually similar [186].

the raw input data X from the output H of Θ_l (see Equation 5.3). The set of candidate DNNs used for the attacker is selected from a combination of hyperparameter configurations including layer type, choice of activation function, and the number of neurons per layer. The selection of the strongest attacker DNNs is the result of an exhaustive search over the hyperparameter space for each benchmark.[2] We also assume the attacker DNNs are adaptive to the layer that is selected for partitioning: when a deeper layer is selected for partitioning, the attacker adapts by using a more complex attacker DNN structure for input inference. Additional details regarding the selection of DNN parameters are in Section 5.4.5.

(a) original images (b) no defender (c) has defender

Figure 5.3: Handwritten Digit Image Recovery. This figure compares the input images (a) to a `handwriting-to-digit` classifier network to the images recovered by the two-layer ReLU-based MLP from this classifier network's intermediate layer output when the privacy partition *is not* applied (b) and when the privacy partition *is* applied (c). The input to the recovery network consists of the hidden layer output passed between the local domain and the remote domain of the classifier network. The MSE for the images in (b) and (c) are ≈0.07 and ≈0.11 (SSIM ≈0.55 and ≈0.35), respectively.

5.4.2 Preliminary Evaluation

As a preliminary step, we validate the effectiveness of our method with a simple benchmark dataset. We choose one of the most commonly used image benchmark datasets MNIST to start our tests. MNIST [183] is a benchmark computer vision dataset containing 60,000 gray-scale images of handwritten digits (50,000 training and 10,000 testing). We train a

[2] We have chosen DNN structures that are based on autoencoders since autoencoders are the most powerful DNN structures for input inference [187, 188, 179].

three-layer ReLU-based MLP with 800 hidden units per layer for our task. The model achieves an accuracy of around 98.4%. We choose the output of the second hidden layer for partitioning. In this experiment, we evaluate the quality of the attack using both human perceptual loss metric (MSE and SSIM). The details of training the model and defender as well as details of the chosen attacker models can be found in Section 5.4.6.

(1) How does adjusting the defender weight impact the performance of attackers?

First, we consider the case where the model provider uses only one defender in the model learning phase. We configure the defender model to a two-layer ReLU-based MLP. For the attacker, we choose eight attacker models in total in different combinations of hyperparameters. We set the defender weight λ (see Equation 5.2) to be $0, 100, 200, 300, 400, 500$ to test the corresponding model accuracy and human perceptual metrics. The results are shown in Table 5.1.

Table 5.1: MNIST Model with Multiple Attackers Versus a Single Defender

Defender Weight		0	100	200	300	400	500
Model Accuracy		98.4%	98.2%	98.2%	98.1%	98.1%	98.1%
MSE / SSIM	Attacker ①	0.070/0.546	0.072/0.534	0.079/0.500	0.081/0.495	0.086/0.466	0.116/0.347
	Attacker ②	0.072/0.524	0.068/0.553	0.080/0.500	0.082/0.491	0.093/0.449	0.117/0.361
	Attacker ③	0.016/0.834	0.018/0.823	0.021/0.807	0.021/0.797	0.021/0.818	0.023/0.792
	Attacker ④	0.022/0.783	0.021/0.794	0.025/0.755	0.025/0.765	0.023/0.787	0.025/0.760
	Attacker ⑤	0.066/0.552	0.062/0.587	0.071/0.533	0.076/0.517	0.076/0.517	0.087/0.457
	Attacker ⑥	0.074/0.514	0.070/0.543	0.087/0.457	0.097/0.420	0.092/0.448	0.131/0.298
	Attacker ⑦	0.070/0.527	0.063/0.575	0.071/0.531	0.070/0.532	0.081/0.470	0.091/0.443
	Attacker ⑧	0.032/0.734	0.037/0.713	0.046/0.659	0.049/0.637	0.048/0.640	0.061/0.558

The overall trend is a reduction in attacker recovery accuracy as defender weights are increased.

Result Analysis: In Table 5.1, we can see that for each attacker DNNs, the attacker is less successful in discovering the inputs with the increase of the defender weight. In all cases, the model classification accuracy remains at a high level ($> 98\%$). The results demonstrate the effectiveness of our framework: adding defenders in our framework makes it harder for the

attacker to discover input images while maintaining the model inference accuracy. Figure 5.3 illustrates MNIST images recovered by the two-layer ReLU-based MLP with the defender present and without the defender present. We can see that with the defender present, the recovered images by the attacker network is harder to recognize.

(2) How does the addition of multiple defenders affect the performance of multiple attackers?

Next, we extend the experiment by adding multiple defender DNNs in the model learning phase. For comparison, we select the four different types of defender DNNs which increase the diversity of defender DNN structure. The details of the chosen defenders are in Section 5.4.6.

As for the attacker DNNs, we use the same eight attackers used in the previous experiment for comparison to see how the case of multiple defenders improves from that of a single defender. We choose the defender weight $\lambda = 200$ for comparison. Experiment results are shown in Table 5.2.

Result Analysis: In Table 5.2, we can see that the performance of each attacker DNN degrades by a large extent when there are multiple defenders present, compared to the performance of each attacker model when there is no or only one defender present. This is because by training multiple defenders in the model learning phase, the model can choose the best defender in each step that has the best recoverability to optimize its parameters. This lessens the potential for attacker models to accurately infer input images.

5.4.3 Primary Evaluation

In this section, we validate privacy partitioning in different scenarios. We use three datasets for different experimental settings. We use the LFW [189] and CIFAR-10 [190] datasets for evaluating input inference protection. We use AgeDB [191] to evaluate attribute inference protection. Next, we describe the details of these datasets and the data preprocessing steps in our experiments.

The **LFW** dataset contains 13,233 images of faces collected from the web and each image

Table 5.2: MNIST Model with Multiple Attacker Versus Multiple Defenders

Defenders Present		**No Defender**	**Single Defender**	**Multiple Defenders**
Model Accuracy		98.4%	98.2%	98.0%
MSE	Attacker ①	0.070	0.079	0.215
	Attacker ②	0.072	0.080	0.209
	Attacker ③	0.016	0.021	0.070
	Attacker ④	0.022	0.025	0.073
	Attacker ⑤	0.066	0.071	0.195
	Attacker ⑥	0.074	0.087	0.202
	Attacker ⑦	0.070	0.071	0.168
	Attacker ⑧	0.032	0.046	0.192
SSIM	Attacker ①	0.530	0.500	0.076
	Attacker ②	0.540	0.500	0.098
	Attacker ③	0.841	0.807	0.500
	Attacker ④	0.820	0.755	0.494
	Attacker ⑤	0.543	0.533	0.194
	Attacker ⑥	0.518	0.457	0.117
	Attacker ⑦	0.568	0.531	0.205
	Attacker ⑧	0.722	0.659	0.160

contains a label of the person depicted. 1,680 subjects have at least two distinct images. The size of the images is 250×250. We only retain the images of subjects that have at least 30 different pictures and re-scale the images to 64×64. After filtering, there are 2,370 images left with 34 subjects in total. We use 80% of the images for training and reserve the remaining 20% for the test set.

The **CIFAR-10** dataset is a benchmark datasets for object recognition and it contains $50,000$ training images and $10,000$ test images. We use this dataset for input inference attacks.

The **AgeDB** dataset contains 16,488 images of faces containing gender, age, and subject

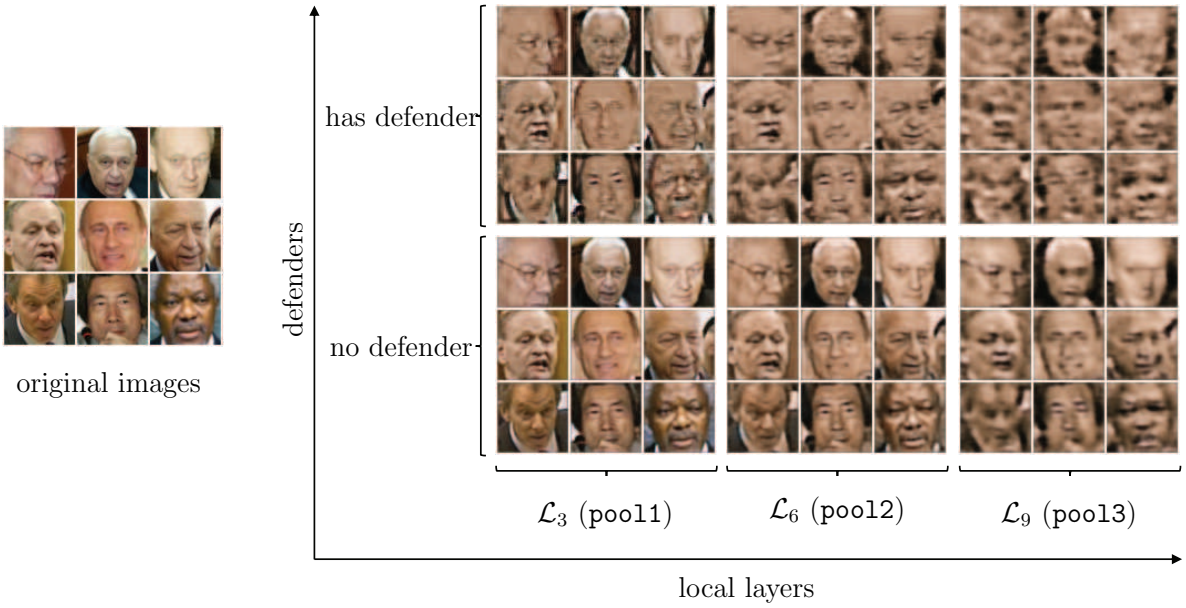


Figure 5.4: Faces Photo Recovery. This figure compares nine sample images recovered by six configurations (2 defender configurations by 3 partition configurations) in LFW dataset. Note that the recovery error both increase as a defender is applied to the privacy partition (bottom-to-top trend) and as more layers are included in the local partition Θ_l (left-to-right trend) (refer to Figure 5.5 for the full results of the `photo-to-id` experiments).

label. There are 566 subjects in total. We filter out those images where faces are detected using OpenCV library [192]. We have 14,857 images left after filtering. We then split 80% as the training set and the rest 20% as the test set. We use this dataset for attribute inference attacks: the deep network learns gender estimation while the attacker tries to learn subject labels from H.

(1) Where should we place the privacy partition?

To answer this question, we apply privacy partitioning to several candidate layers for input inference attacks for LFW and CIFAR-10. We use the CNN model for LFW dataset (see Section 5.4.7 for model architecture) and VGG-19 [193] for CIFAR-10. We choose the output of the first, second, and third pooling layer (denoted by `pool1`, `pool2`, `pool3`) in each model for privacy partitioning. We choose these layers because the outputs of

these layers represent different levels of abstraction of features in the model in the feature extraction module in the CNNs [194]. Note that when privacy partitioning goes deeper (in terms of layers) in the model, the attacker DNNs are more complex (e.g., more layers). We use 3 different types of attacker DNNs for both datasets. As for the defender, we choose the deconvolution decoder structure as the structure of the defender DNN [195, 188]. The detailed structures of the attacker DNNs and the defender DNN are in Section 5.4.7. The defender weights are set to be 0.1 and 1.0 in the LFW and CIFAR-10 experiments, respectively. We set the defender weight according to the ratio of model loss and defender loss (3 : 1) in the first few iterations in case that the defender loss becomes the dominant loss.

We measure the model accuracy, the human perceptual loss (SSIM and MSE), machine perceptual loss (DPL and reprint accuracy) in all different settings. Figure 5.5(a)-(e) shows the results of the best attacks for LFW experiments and Figure 5.5(f)-(j) for CIFAR-10 experiments. Note that we run all experiments five times and report the average.

From Figure 5.5(a)(f), we can see that privacy partitioning maintains model classification accuracy at a high level; in all layers we test, the model accuracy is greater than 88% in LFW experiments (91% in CIFAR-10 experiments). We also notice that applying privacy partitions to "deeper" layers yields less reduction in model performance compared to placing privacy partitions in the first layer `pool1`.

Figure 5.5(b)(e)(g)(j) shows that when privacy partitions are applied at any layer, it is less likely for the DNNs to distinguish the recovered image from the original images correctly (lower reprint accuracy and higher DPL). The overall trend is a decrease (increase) for reprint accuracy (DPL) as privacy partitioning are deployed to deeper layers.

In Figure 5.5(c)(d)(h)(i), we consider human perceptual loss. The defender deteriorates the quality of recovered input images in terms of perceptual metrics. The results also suggest that by deploying privacy partitions to the deeper layers, the recovered images are perceptually less discernible. Figure 5.4 shows the visualization results of LFW experiments

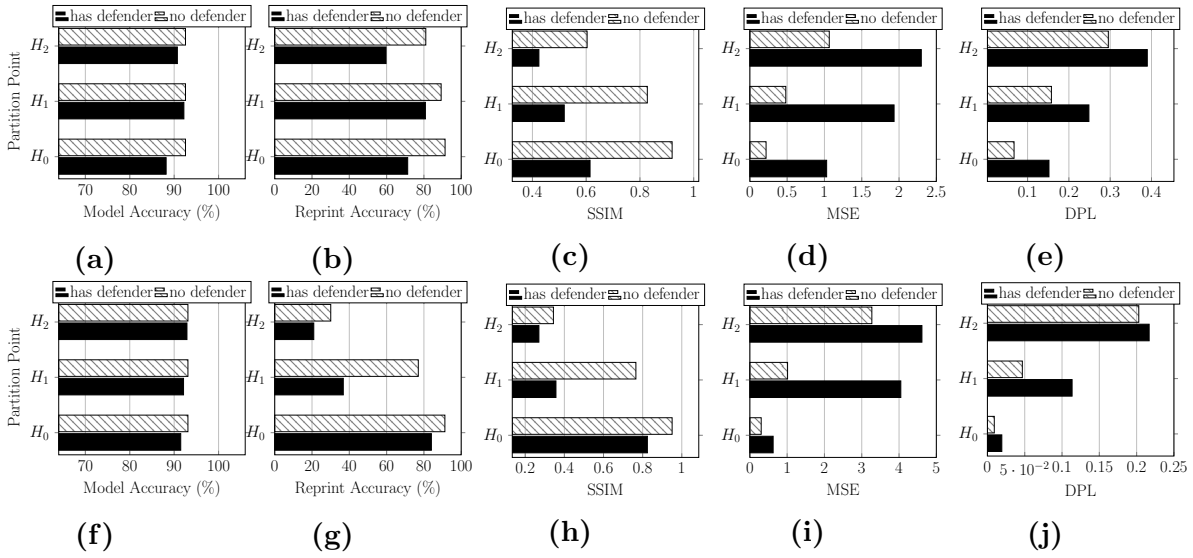


Figure 5.5: Input Recovery Attacks Results in LFW and CIFAR-10 Experiments. This figure shows the results for 6 configurations in LFW experiments (charts a-e) and CIFAR-10 experiments (charts f-j), measured in terms of five performance criteria. The configurations includes 2 defender configurations by 3 partition point selections: H_0 (`pool1` activation), H_1 (`pool2` activation), H_2 (`pool3` activation). The 5 performance criteria include the privacy partition model classification accuracy; the reprint accuracy of the recovered image; the structural similarity index (SSIM) of recovered and originals images; the mean squared error (MSE) of recovered and originals images; and the deep perceptual loss (DPL) of recovered and originals images.

Overall, since our goal is to maintain the model performance while keeping the recovery error of the attacker as high as possible, combined with all metrics we consider, deeper layers such as `pool3` are the best positions for the privacy partition. However, deploying more layers in the local domain would compromise the intellectual property of the deep learning models and increased local computational resources. The model providers and end-users should, therefore, have a trade-off.

(2) Can privacy partitioning protect other private attributes?

We now demonstrate that the privacy partition method can defend against attribute inference style attacks where an attacker attempts to learn other sensitive attributes from hidden deep network states. We use AgeDB to train a DNN for gender classification. The goal of the attacker is to identify the subject of the input images given access to the network's

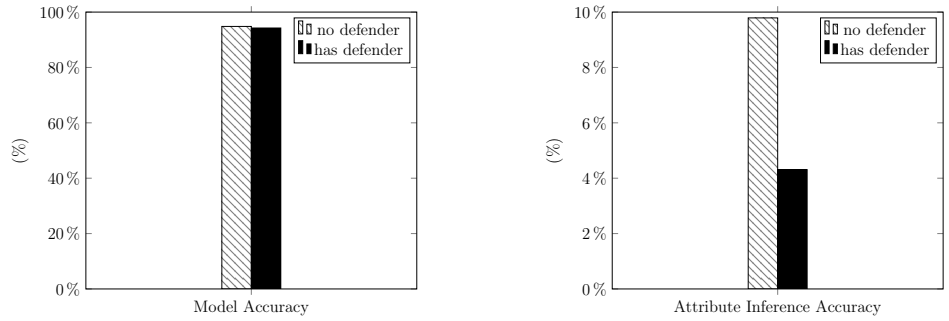


Figure 5.6: Attribute Inference Attack Results: In the highest privacy setting of the gender estimation experiment, privacy partitioning reduces the subject identification ability of attacker by half (from 9.79% to 4.31%) with negligible (≈0.60%) reduction to the model accuracy.

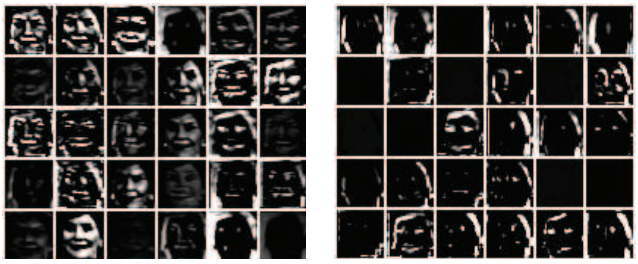
hidden state. We use VGG-19 convolutional neural network [193] for age estimation and we choose the output of feature extractor in VGG-19 for privacy partitioning. We choose this layer for privacy partitioning because this layer is relatively deep in the model (almost near the output layer of the model). Our previous experiments demonstrate that such a deployment strategy is the best setting to defend against the attackers.

The design strategy of defender DNN and attacker DNNs are the same as the previous experiment except that the output is a prediction score for each subject. We set the defender weight to be 0.3. All the DNNs architecture are shown in Section 5.4.9 and the results are shown in Figure 5.6.

In Figure 5.6, we can see that our method has a negligible impact on the original classification task (94.82% drops to 94.25%). In the meantime, we find that the attacker can successfully identify nearly 10% of the subjects (out of 566) even in the hardest setting for the attackers. Applying the defender, we find that the subject identification ability of the attacker is decreased by half (9.79% drops to 4.31%). Thus, we find that the privacy risk potential of repurposing hidden layer activations is significant and that reduces the risks of leaked private attributes.

(3) What is the insight of the defender's role when training the model?

Our experiment results demonstrate quantitatively and qualitatively the effectiveness



(a) no defender (b) has defender

Figure 5.7: Hidden Layers and Defender Strategy. This figure shows the impact of defender strategy on the hidden layer activations in LFW experiment. The hidden layer activation visualizations are outputs of pooling layer H_0 when there is no defender present (a) and when there is a defender present (b).

of our method. To have a more intuitive understanding of why the defender is effective, we compare the visualizations of the hidden layer activation we protect. We visualize the "protected" activations with those without protection in the LFW experiment. Note that we choose the activation `pool1` and `pool2` for visualization since deeper layer features are hard for a human to interpret semantically. From the visualization comparisons, we can qualitatively check how the defender helps protect the information of hidden layer features from leaking privacy-invasive characteristics. Figure 5.7 shows the visualization results for `pool1`.

We can see from Figure 5.7 the differences between hidden layer activation of `pool1` with and without the defender. In Figure 5.7(a), we can see clearer human visually-recognizable features in more filter activations, whereas in Figure 5.7(b), we can only capture the features such as the basic outlines of human faces in fewer filter activations.

The visualization results meet our intuition that privacy partitioning reduces the discoverability of private attributes while maintaining the features that support the authorized classification task. Combined with the model classification accuracy results shown in Figure 5.5(a), we can conclude that performing the privacy partitioning in the proper layer not only maintain the model performance (emphasizing key features) but also make it

harder for the attacker to steal more information from input images (deactivating sensitive features that are less relevant to the classification tasks).

(4) Can we continue improving the model accuracy in privacy partitioning?

According to Section 5.3.4, we can continue to train remote partition in the inference phase. This scheme of privacy partitioning fits well with fine-tuning concept [196] in DNNs. We can continue to fine-tune the remote layers for better performance while keeping the local layers unchanged.

We fine-tune our pre-trained privacy-partitioned model in all settings we tested before the CIFAR-10 experiments for 50 epochs. We do not add any new training data when continuing to train the remote layer. We still observe $0.2\% - 0.3\%$ increase in model accuracy for all privacy partitioning deployments in `pool1`, `pool2`, and `pool3`. Note that we already use VGG-19 (one of the state-of-the-art model for CIFAR-10 and it is hard to further improve the model accuracy) and privacy partitioning only causes $0.2\% - 1.6\%$ accuracy degradation in VGG-19. We can see that continuing to fine-tune the remote layers can even make an impact on the model accuracy more insignificant. It also demonstrates that even in the inference phase of privacy partitioning, we can continue to train remote layers to further improve the model accuracy.

(5) How does privacy partitioning compare to the method of adding differentially-private noise to inputs?

Questions arise when we add noise directly to the input image to maintain data privacy. How does it compare to privacy partitioning? Does this method provide better utility and privacy trade-off?

To answer this question, we compare privacy partitioning with differential private image pixelation [197], which extends the standard differential privacy notion to image data under a similar threat model as ours (image owners wish to share with untrusted recipients). Differential private image pixelation is proposed under the notion of m-neighborhood for image data: two images are neighbors if they have the same dimension and they differ by at

most m pixels. It has been argued that the m-differences of pixels can help to protect the presence or absence of any sensitive information in the image (e.g. object, text, or person). [197] also proposes the Differentially Private Pixelation algorithm that achieves ϵ-differential privacy (see in Section 5.4.10 for details). The effectiveness of the algorithm is validated by SSIM.

We apply differential private image pixelation algorithm for the CIFAR-10 dataset in the inference phase on the pre-trained model. We set the pixelation grid cell length to be 2 and m to be 1 to minimize the negative effects on image utility [197]. We change the value of ϵ to see how SSIM and the classification accuracy change accordingly. The results are shown in Figure 5.8.

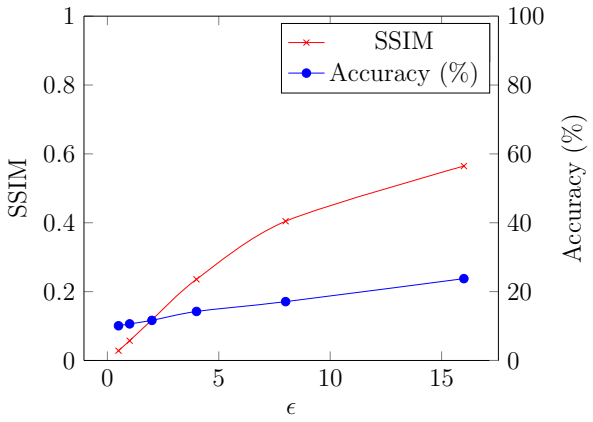


Figure 5.8: Differentially Private Image Pixelation Results. This figure shows that as the privacy parameter ϵ in differentially private image pixelation increases, both the SSIM and classification accuracy increase.

As shown in Figure 5.8, the classification accuracy drops significantly. In comparison, our method almost has no impact over the model accuracy Figure 5.5(f)-(g)). This demonstrates our method is more practical in real world scenarios than approaches that add noise directly to inputs.

In this section, we present additional details on the parameter design choices used in this study.

5.4.4 Deep Learning Background

In our work, we focus on the setting of supervised learning for simplicity: a DNN model $f_\theta : \mathcal{X} \to \mathcal{Y}$ parameterized by θ. For classification problems, $\mathcal{X}$ is a high dimensional vector space and $\mathcal{Y}$ is the space for the classes. Given a labeled dataset $\{(x_i, y_i)\}_{i=1}^{m'}$ where $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. The dataset is usually partitioned into training data of size m and test data.

To learn a good DNN model that perform well on the test data, we will try to minimize the loss function l which measures the difference between ground truth labels and the predicted labels:

$$\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} l\big(y_i, f_\theta(x_i)\big)$$

There are many optimization algorithms to solve the above problem. Stochastic gradient descent (SGD) and its variants are commonly used to train a DNN. Popular choices of f_θ in the application of deep learning include Multilayer Perceptrons (MLPs) [169], Convolutional Neural Networks (CNNs) [165], Recurrent Neural Networks (RNNs) [168] or a combination of them [167].

5.4.5 Input Inference Metrics

How best to quantify the privacy leakage of a recovered image as compared to its reference image is a challenging problem in the field of computer vision. In general, providing formal privacy guarantees is a challenging problem in deep learning theory due to the complexity of the deep learning model and the unknown data distribution [198]. In many cases, the definition of privacy depends on the application and user expectations. For example, a user may want to upload portraits to a facial recognition deep learning service to help automate tasks related to organizing and sharing photos but at the same time may not want demographic information to be collected from those photos.

Perceptual metrics are widely used in computer vision and lossy compression tasks as statistical measures of human perception and data quality [184, 199, 200, 185]. Moreover, researchers conducting user studies and statistical evaluations on a variety of image quality measurement datasets have found that perceptual metrics are highly consistent with human perceptual assessments of image quality [201, 202, 203, 204].

In the experiments that follow, we use perceptual metrics primarily as loss function scores for balancing image discoverability with classification accuracy and as an application-agnostic proxy for quantifying privacy leakage. Perceptual metrics are useful because they can be readily computed in the context of optimization and have clear physical meanings. Please note that the scoring metric can be replaced in the loss function without additional alteration to the proposed privacy partition framework (e.g., to replace general-purpose metrics with application-specific privacy metrics).

We use four types of perceptual metrics to measure the perceptual similarity between original and recovered images: (i) a per-pixel similarity measure (MSE) [184], (ii) a perceptual similarity measure (SSIM) [199], (iii) a deep-learning-based perceptual similarity measure [200, 185], and (iv) an application-specific validation metric we term reprint accuracy.

MSE: The mean squared error (MSE) measures the per-pixel l_2 Euclidean distance between two images [184]. If MSE $= 0$, two images are identical. MSE is usually used to measure the quality of image reconstruction. In the context of image or video compression, it is often used as an approximation of human perception of reconstruction quality. However, the MSE is insufficient to assess highly-structured images since it assumes pixel-wise independence. For example, blurring an image can result in small l_2 changes but large perceptual changes.

SSIM: The structural similarity index (SSIM) improves on MSE by assuming pixel-wise dependence [184, 199]. It is computed using a sliding window to capture the structural information between two images. It ranges from $[-1, 1]$, where SSIM $= 1$ indicates that

the two images are identical. SSIM is highly consistent with human perceptual assessments. For example, image obfuscation techniques such as pixelation and blurring result in smaller SSIM values.

DPL: Recently, studies have been shown that internal activations of deep convolutional networks trained on image inference tasks are surprisingly effective at capturing an objective measure of "perceptual loss" that corresponds to human visual perception especially in terms of perceptual spatial ambiguities [200, 185]. In our experiment, we use deep perceptual distance (DPL) [185] to evaluate the perceptual indistinguishability of recovered images as a complementary metric to SSIM and MSE.

Reprint accuracy: The reprint accuracy is a measure of the model classification accuracy using recovered inputs by the attacker (i.e., the classification accuracy of $f : \mathcal{H} \rightarrow \mathcal{X} \rightarrow \mathcal{Y}$). Reprint accuracy provides a measure of the utility of recovered inputs as compared to the original inputs. It can be modified to assess how well an adversarial network can complete any privacy-invasive classification task using recovered images (re-classification accuracy).

5.4.6 MNIST Hyperparameter Settings

In this section, we will describe the experimental details for MNIST dataset.

In our experiment settings for MNIST, we train the model and the defender(s) using the Adam optimization algorithm [205]. We set the learning rate of the model and the defender at 0.0001 and 0.001, respectively. We train the model and the defender for 500 epochs with batch size 32. We also use the dropout technique [206] with a drop probability of 0.1 to prevent over-fitting.

We use SSIM as the distance metric (defined in Equation 5.2). We use MSE loss function for training the attacker models, as it is typically used in computer vision tasks.

Here are all attacker DNNs used in the MNIST experiment:

- Attacker ①: `800` → `ReLU` → `800` → `Sigmoid`

- Attacker ②: `800` → `ReLU` → `dropout(0.1)` → `800` → `Sigmoid`
- Attacker ③: `800` → `Tanh` → `800` → `Sigmoid`
- Attacker ④: `800` → `Sigmoid` → `800` → `Sigmoid`
- Attacker ⑤: `512` → `ReLU` → `512` → `Sigmoid`
- Attacker ⑥: `1024` → `ReLU` → `1024` → `Sigmoid`
- Attacker ⑦: `1-D conv` → `ReLU` → `800` → `Sigmoid`
- Attacker ⑧: `784` → `Sigmoid`

Note the `512` means fully connected layer with 512 neurons, → denote the network data flow direction and other notations denote the layer name and the corresponding layer parameters.

Here are the defender DNNs used in the multiple defender training experiments:

- Defender ①: `800` → `Tanh` → `800` → `Sigmoid`
- Defender ②: `800` → `Sigmoid` → `800` → `Sigmoid`
- Defender ③: `1-D conv` → `ReLU` → `800` → `Sigmoid`
- Defender ④: `784` → `Sigmoid`

5.4.7 LFW Hyperparameter Settings

We use the CNN model in LFW experiments. The model used for LFW experiment is:

`conv2d 5×5` → `conv2d 5×5` → `maxpool 2×2 (pool1)` → `conv2d 3×3` → `conv2d 3×3` → `maxpool 2×2 (pool1)` → `conv2d 3×3` → `conv2d 3×3` → `maxpool 2×2 (pool3)` → `conv2d 3×3` → `conv2d 3×3` → `maxpool 2×2` → `512` → `dropout(0.5)` → `512` → `dropout(0.5)` → `512`

Note that each convolutional layer is followed by the ReLU activation and batch-normalization layer.

We train the model and the defender using SGD algorithm with momentum 0.9 and an initial learning rate of 0.01. The learning rate decays by a factor of 0.1 every 100 epochs, and there are 250 epochs in total. l_2 regularization is also applied to prevent over-fitting.

The design of the defender DNN is adaptive to the layer chosen for partitioning (e.g., in our experiment, we choose the outputs of first three pooling layers `pool1`, `pool2`, and `pool3` for partition). For example, if we choose `pool2` for partition, the architecture of defender network would be the "reversed" version of local layers:

`deconv2d 3×3 (stride 2)` → `conv2d 3×3` → `ReLU` → `deconv2d 5×5 (stride 2)` → `deconv2d 5×5` → `tanh`

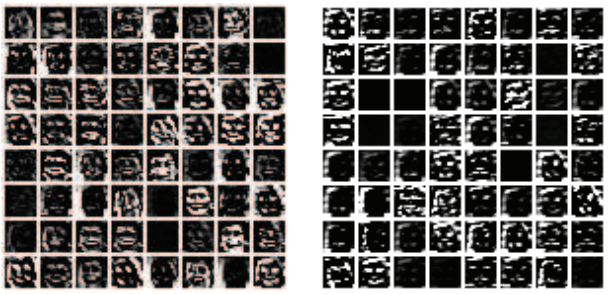
Note that this type of architecture resembles the design strategy of a convolutional autoencoder [187, 188]. The attacker DNNs also depend on which layer for partition since the input dimension might be different from layer to layer. For example, if we choose `pool2` for partition, the attacker is

- Attacker ① (`DECONV ATTACKER`): `deconv2d 3×3 (stride 2)` → `conv2d 3×3` → `ReLU` → `deconv2d 5×5 (stride 2)` → `deconv2d 5×5` → `tanh`
- Attacker ② (`FC ATTACKER`): `4096` → `ReLU` → `12288` → `Tanh`
- Attacker ③ (`SPARSE FC ATTACKER`): `4096` → `ReLU` → `dropout(0.5)` → `12288` → `Tanh`

Note that these types of attackers are chosen since they are the most commonly used decoder architecture in auto-encoder designs [187] and they cover the most commonly used operations in deep neural networks in the area of computer vision.

5.4.8 CIFAR-10 Hyperparameter Settings

We used batch-normalized VGG-19 [193] for the classification model. We train the model with random crop and random horizontal flip data augmentation techniques for 150 epochs.



(a) no defender (b) has defender

Figure 5.9: Hidden Layers and Defender Strategy II. This figure shows the impact of the defender strategy on the `pool2` activations (hidden layer H_1) in LFW experiments. Applying the defender strategy (b) results in more information loss to the intermediate layer features a compared to training normally (a).

The initial learning rate is 0.01 with a decay factor of 0.1 for every 50 epochs. We use the SGD algorithm with momentum 0.9 and l-2 penalty weight decay 0.0005.

The design strategies of defender DNN and attacker DNNs are similar in the LFW experiment. For example, if we choose the second pooling layer `pool2` for partition, the defender DNN is:

`deconv2d 3×3 (stride 2)` → `BatchNorm` → `ReLU` → `deconv2d 3×3 (stride 2)` → `BatchNorm` → `ReLU` → `deconv2d 3×3` → `BatchNorm`

And the attacker DNNs are:

- Attacker ① (`DECONV ATTACKER`): `deconv2d 3×3 (stride 2)` → `BatchNorm` → `ReLU` → `deconv2d 3×3 (stride 2)` → `BatchNorm` → `ReLU` → `deconv2d 3×3` → `BatchNorm`
- Attacker ② (`FC ATTACKER`): `1024` → `ReLU` → `3072`
- Attacker ③ (`SPARSE FC ATTACKER`): `1024` → `ReLU` → `dropout(0.5)` → `3072`

5.4.9 Hyperparamter Settings in AgeDB Experiment

We used batch-normalized VGG-16 [193] for the classification model. We train the model with random crop and random horizontal flip data augmentation techniques for 50 epochs.

The initial learning rate is 0.01 with a decay factor of 0.1 for every 20 epochs. We use the SGD algorithm with momentum 0.9 and l-2 penalty weight decay 0.0005.

The design strategies of defender DNN and attacker DNNs are similar in the previous experiments, except that the output of model is a 566-dimensional vector (We have 566 subjects in total). The defender DNN is:

`conv2d 1×1 (stride 1)` → `BatchNorm` → `1024` → `ReLU` → `dropout(0.2)` → `566`

And the attacker DNNs are:

- Attacker ① (`CONV ATTACKER`): `conv2d 1×1 (stride 1)` → `BatchNorm` → `1024` → `ReLU` → `dropout(0.2)` → `566`
- Attacker ② (`FC ATTACKER`): `1024` → `ReLU` → `dropout(0.2)` → `566`
- Attacker ③ (`Sparse FC ATTACKER`): `1024` → `ReLU` → `dropout(0.5)` → `566`

5.4.10 Differentially Private Image Pixelation

In this section, we will discuss the notion of differentially private image pixelation [197] and how to achieve it. We define "neighboring images" as the application of the notion of differential privacy to images as follows:

Definition [m-Neighborhood] Two images I_1 and I_2 are neighboring images if they have the same dimension and they differ by at most m pixels.

To achieve ϵ-differential privacy, they also propose Differentially Private Pixelation algorithm. The details of the algorithm are in Algorithm 5.2. It has been proved that Algorithm 5.2 is ϵ-differential private in [197].

Function 5.2: Differentially Private Image Pixelation

INPUT: Image I with size $M \times N$, neighbouring parameter m, pixelation grid size b, privacy parameter ϵ

OUTPUT: Differentially private image $\tilde{I}$

1: Divide I into $K = \lceil \frac{M}{b} \rceil \lceil \frac{N}{b} \rceil$ cells c_k, where $\quad k = 1, \ldots, \lceil \frac{M}{b} \rceil \lceil \frac{N}{b} \rceil$

2: Pixelate the image

$$p(I; b) = \{\frac{1}{b^2} \sum_{(x,y) \in c_1} I(x, y), \ldots, \frac{1}{b^2} \sum_{(x,y) \in c_K} I(x, y)\}$$

3: Sample Laplacian noises $\tilde{\mathbf{N}} = \{\tilde{N}_1, \tilde{N}_2, \ldots, \tilde{N}_K\}$ with means 0 and and scales $\frac{255m}{b^2\epsilon}$

4: Add noises to the pixelated image $\tilde{I} = I + \tilde{\mathbf{N}}$

5: **return** $\tilde{I}$

5.5 Discussion

In this section, we discuss the benefits and limitations of the privacy partitioning framework for concealment-enhanced hidden layer activation states.

We evaluate the protection strength and performance cost of this framework in a series of experiments using benchmark computer vision tasks (Section 5.4), demonstrating that privacy partitioning increases input inference resilience ($P1.1$) and private input inference resilience ($P1$) with negligible degradation of performance ($P2.1$).

The privacy partitioning framework is constructed using the deep learning optimization functionality ($P2.2$). The privacy partitioning framework can be applied directly to existing deep networks ($P3.1$). The resulting network can be readily deployed alongside end-to-end encryption mechanisms. Similarly, the partitioned learning privacy optimization may also be utilized by decentralized architectures and federated resource sharing models because the

defender component can be applied independently to each data resource (*P*3.2).

Since the transferred hidden layer activations are input inference and input repurposing resilient, the local computing context exerts more control over how data is used than the remote-only deployment. Since defender suites need not be disclosed by the service provider[3], to either the local admin or the remote admin, the service provider exerts more control over their intellectual property in partitioned learning than in the case of a remote-only or local-only deployment of centralized learning.

This framework is particularly beneficial when the clients in the local computing context are computationally resource-constrained devices such as mobile devices and IoT devices. Since only a portion of the deep network is deployed locally, the local computing context requires less computational resources than a remote-only deployment.

This approach does introduce additional components to the model learning phase. The additional components (primarily the defender) result in increases to the time required to learn a high-performance deep learning model. However, the scale of the defender is proportional to that of the local partition [187]. Thus, the increased model complexity and performance overhead during model learning is not a substantial performance or cost barrier during model inference when compared to the baseline cost of deploying the reference deep network. In the model inference phase, the only performance overhead is due to the communication overhead of transferring hidden layer activations.

Another limitation of the proposed framework is that the selection of the defender suite is an engineering task without the formal guarantee of a globally maximal implementation. Similarly specifying the sufficient threshold for data secrecy or user privacy is application-specific. For example, an SSIM threshold that provides a sufficient level of concealment for one data type may not be adequate for another type of data. Nonetheless, privacy partitioning enables privacy to be implemented as a learning problem where improvements

[3]The defender suite is included in the optimization functionality of the deep network during the model learning phase. However, the defender suite is not required to verify the protection strength of the resulting deep network or during model inference phase when the deep network is in use.

to the quality of the training data distribution improve both model performance and data protection properties.

5.6 Related Work

In this section, we discuss the related research in the areas of machine learning privacy threats and privacy-preserving solutions of deep learning with a comparison to our work.

5.6.1 Machine Learning Privacy Threats

Data privacy of machine learning has been an active research topic for long. In one of the newest works, Hitaj et al. [207] posit that there are fundamental limitations to the level of privacy that can be achieved using a decentralized approach to training deep learning models. Song et al. [208] consider a malicious machine learning services provider who supplies model-training code to the data-holder. Fredrikson et al. [179] explore model inversion attack: they show that model inversion could lead to unexpected privacy threats by leveraging confidence values given by machine learning models. Shokri et al. [186] study membership inference attacks: they assume black-box access to a machine learning inference model and determine whether a labeled data instance appears in the training data that is used to train the model. Other membership inference problems are studied in [209, 210, 211, 212]

Model privacy in machine learning is another important research area. Tramèr et al. [213] demonstrate the feasibility of duplicating the functionality of machine learning models such as decision tree and Logistics Regression in MLaaS system. Tramèr et al. term it as *model extraction attack*. Wang et al. [214] propose *hyperparameter stealing attacks* and demonstrate its effectiveness theoretically and empirically in machine learning models like logistics regression and support vector machine.

5.6.2 Privacy-Preserving Deep Learning

Differential Privacy. Differential privacy originates from the domain of database and it provides formal privacy guarantees for each data record in the database. Differential privacy has been applied to ensuring the privacy of training data of deep learning to protect against the case that the model provider could learn from model parameters whether the individual data is present or not [215, 216].

However, our threat model is different and it is hard to directly apply the standard differential privacy notion in our case: for the task of image publication for MLaaS, we often send image data to the model provider to request for the certain services for that particular image data and the information leakage by directly sending to the model provider is not guaranteed by differential privacy. In summary, standard differential privacy is for protecting the privacy of individual entries in a confidential database (which is well defined for training a machine learning model), whereas the privacy partitioning is for training a deep network in such a way that inputs for classification are protected. They are complementary but not directly comparable. In summary, differential privacy provides privacy/anonymity for the individual samples training database while the privacy partitioning provides privacy during model inference.

Erlingsson et al. [178] propose a Randomized Aggregated Privacy-Preserving Ordinal Response (RAPPOR) to provide strong privacy guarantees for crowdsourcing population statistics from end-users. However, a user might send a single data item for MLaaS system. Thus, RAPPOR cannot be applied to this scenario if a user might send a single data item for MLaaS. Fan et al. [197] also extend the standard notion of differential privacy to image privacy for image publication. However, their method suffers from low data utility compared with our method (Section 5.4.3).

Collaborative Training. Shokri et al. [177] propose a collaborative deep learning framework to render multiple parties to learn a deep neural network without uploading their data to the remote server in the model learning phase. Each local party has a copy of the model

that can upload and download parameters during training so that the model is trained without uploading the data to the central server party. However, the deployment of the model on the local side would increase the local computing powers and leak the model privacy. Our approach complements this framework by deploying part of the model in the local domain in the model inference phase. McMahan et al. [176] propose a federated averaging algorithm to protect the privacy of training data. Instead of uploading data directly to a remote server, the client trains the model locally and uploads updated parameters to the central model. Still, our work complements it since we aim to protect data privacy in the model inference phase.

Cryptography-based Solutions. Cryptography-based protocols have long been used in machine learning models to protect data privacy [217, 218, 219, 175]. Liu et al. [174] present cryptography-based oblivious protocols to protect data privacy in the model inference phase for deep learning models. They design oblivious protocols for linear transformations, popular activation functions and pooling operations using secret sharing and garbled circuits in the online prediction phase and perform request-independent operations using homomorphic encryption together with single instruction multiple data (SIMD) batch processing technique. Since the method requires no change in the pre-trained model, our approach is complementary to theirs. Furthermore, our method is compatible with all cryptography-based protocols in principle.

5.7 Conclusion

We develop the privacy partitioning framework in the context of a common privacy requirement: confidential local data handling and concealed remote data handling. We evaluate the effectiveness of the proposed privacy partitioning framework in a series of experiments utilizing benchmark computer vision tasks. We find that it is a promising method for significantly reducing the capacity for an adversary with access to intermediate layer activations

to perform input and private attribute inference attacks.

Future research can explore integration with complementary privacy protections, integration with software and hardware security modules used to secure a local domain, and formal guarantees under varying data distributions, learning task types, and deep network configurations.

Part V

Security & Privacy in the Internet of Everything

Chapter 6

Policy Implications of Internet of Things Security & Privacy Practices

IoT impacts daily life in many ways. A national cybersecurity strategy for IoT policy, or a lack thereof, has significant implications for national security, public safety, civil liberties, and technology adoption. Critical infrastructure such as road and air traffic control, oil pipelines, dams, electrical grids, communications networks, water systems, nuclear waste processing, ports of entry, law enforcement, financial services, emergency medical services, and manufacturing are all targets for national security and public safety threats. How we regulate data use in the context of IoT also has significant implications for individual rights with regards to personal data and notions of privacy in an increasingly connected society. Similarly, how we regulate IoT security and privacy standardization has significant implications for technology adoption. It is because IoT intimately impacts our lives in these critical areas that governments and business leaders must establish a clear cybersecurity framework and play a more proactive role in deploying solutions, setting standards, and enforcing compliance. It is also important for society at large to establish data protection as a fundamental individual right.

The analysis so far advances state-of-the-art enhancements to fog computing elements

and privacy-preserving collaborative learning models as an aid to IoT cybersecurity and data protection policy. Due to the global nature of the challenges presented by IoT security and privacy, the results of these proposals have broad policy implications. However, we focus on the policy implications for cybersecurity practices within governments and the data protection rights of individuals — with particular reference to the United States. We will also have a look at the policy landscape in other parts of the world like the European Union to inform this discussion.

This analysis focuses on laws, standards, and regulations relevant to the use of IoT in the United States. The key stakeholders considered here are state and federal policymakers (i.e., those responsible for developing cybersecurity and data protection policy frameworks) and business leaders (i.e., those responsible for guiding organizational strategies on the adoption of connected devices and cloud computing). In particular, this work recommends policy measures that could be adopted through the collective efforts of consumer-oriented non-governmental organizations, transnational technical standards organizations, private sector self-regulation, and federal and state regulation to implement IoT security and data protection policy.

6.1 Internet of Things, Individual Rights, & Comprehensive Data Protection

Data protection refers to how personal data is used by organizations such as businesses and governments [220]. A data protection policy is one approach to facilitating improvements to IoT security and privacy. This approach has the benefit of providing clear guidelines regarding individual rights over personal data. Additionally, since legislation is defined in terms of how data is used instead of in terms of specific data handling processes, this approach also has the benefit of providing a framework that is independent of — yet consequential to — innovations to information technology processes. The General Data Protection Regulation

(GDPR) of the European Union is perhaps the most comprehensive data protection policy framework [221]. We consider it here as a potential reference framework for a comprehensive U.S. data protection policy.

The objective of GDPR is to establish data protection as a fundamental right for the European Union. It sets out policy in the form of individual rights, recommendations, and obligations aimed at the legal protection of this right during the free movement of data. The impact of GDPR is global in its reach since any company with employees or customers based in the E.U. must comply to continue to do business. The scope of the regulated free movement of data pertains both to the nature of the data being handled (i.e., any personally identifiable data related to members of the E.U.) and to nature of the party handling the data (i.e., any party with dealings in the E.U.).[1]

GDPR grants individuals with certain rights regarding their personal data.[2] Under GDPR, individuals are granted rights related to transparency (e.g., Article 12: the right to concise and intelligible communication of data processing practices), access (e.g., Article 15: the right of individuals to access personally identifiable data maintained by a data controller), erasure (e.g., Article 17: the "right to be forgotten"), and decision-making (e.g., Article 21: the right to refuse a data processing request).

GDPR also contains guidelines and requirements for data handlers with any dealings in the E.U.[3] Under GDPR, the main obligations of data handlers is to process data in a manner that is lawful, fair, and transparent with respect to the data subject, to use collected data solely for pre-established legitimate reasons (purpose limitation), to only collect data that is required for the authorized purpose (data minimization), to ensure collected data is accurate and up to date, to store data in personally-identifiable form for only as long as needed (storage limitation), to ensure data is protected (integrity and confidentiality), and to demonstrate

[1]GDPR Article 2 defines four scenarios within this scope where the regulations do not apply, including in cases of purely personal data use or with regards to law enforcement in matters of public safety.

[2]Individuals are referred to as "natural persons" or "data subjects" in the GDPR text.

[3]The GDPR defines two parties involved in the handling of personal data: the controller and the processor. The data controller is the party that establishes the purpose and means by which personal data is collected and processed. The processor is the set of operations performed on personal data.

compliance (accountability) [221]. The lawful use of data rule requires prior authorization and/or a legitimate reason that is in accord with public interests and not in conflict with the rights granted to data subjects (e.g., to meet a contractual obligation, legal obligation, or protects the interests of data subjects). In cases of infringement of the regulation, the GDPR enforcement rules grant data subjects the right to lodge complaints to a public independent supervisory authority that can result in a judicial remedy and penalties such as liability compensation and administrative fines.

So what do data protection policies such as GDPR mean for IoT? The broad guidelines of the GDPR extend to IoT networks and devices in several ways. Under GDPR, businesses face greater legal requirements to gain informed and timely consent from customers regarding the purpose of collecting and processing personal data. Additionally, all businesses are tasked with supporting significant user controls — such as right to data access, right to restrict data access, right to restrict processing, right to data portability, and right to erasure — whether or not data protection is the primary value proposition of the product or at odds with the interests of the service provider. These requirements and guidelines are sensible in terms of enforcing individual data protection rights but may prove cumbersome in practice for IoT service providers to implement.

GDPR maintains that the responsibility of data handling applies across the entire product supply chain. This is a challenge for IoT service providers since many of the unresolved supply chain factors that make embedded systems particularly susceptible to compromise are exacerbated in the IoT due to safety-critical sensing and control components. Chip manufacturers (e.g., Qualcomm) and systems manufacturers (e.g., Foxconn) in the embedded systems market differentiate their products by hardware features and performance specifications leaving little economic incentive for any individual entity to take on the additional engineering cost of patching and updating board support packages once the product has shipped [222]. This leaves the potential for an IoT marketplace with costly to patch software and ambiguity regarding which entity should be held accountable for vulnerabilities stem-

ming from system integration. It also leaves the potential for customer-facing entities — that do not have the expertise and are not entirely to blame for the security risks of proprietary and open-source sub-components — facing the greater share of customer complaints and legal penalties. Governments should play a more active role in ensuring that market dynamics support after-sales software updates.

Supporting the rights granted to individuals under GDPR requires mechanisms specifying individual data protection preferences. This entails additional support for the subset of IoT form-factors that do not include screens, keypads, or other human interfaces. There is also the potential for the guidelines to stymie competition and innovation. Compliance is an added hurdle for entrants into markets where notification and consent lengthen new customer enrollment processes [223]. Additionally, it is also unclear how service providers will achieve compliance without impeding pervasive computing user experiences that benefit from nonintrusive and ad hoc authentication such as IoT deployments to smart spaces.

Challenges like those could benefit from industry-established open-source mechanisms — including the technical treatment of data protection such as those presented in this work — that provide mechanisms that enable users to manage and remove their data trail using their mobile phones [224]. Mobile and edge network operators could play a key role by supporting privacy-preserving device identity management protocols that associate IoT device identifiers and user device identifiers, providing a common mechanism for completing informed consent, access, and deletion requests. Utilizing the mobile user identity enables service providers to interact with users even in use cases where a mobile application is not the primary user client or it's otherwise challenging to complete a user authentication process. The resulting opaque mobile network identifiers enable service providers to readily coordinate with any number of stakeholders to complete access and deletion requests in a privacy-preserving and auditable manner.

Approaches that implement user-controlled compliance mechanisms on mobile and edge network elements enables individual IoT devices to be free from additional complexity while

providing a common auditing medium for all stakeholders. This is one promising application of the fog mediation proposal explored in this work. Even in cases where there are deployed devices with known vulnerabilities, such a framework could provide stakeholders with the relevant information to make decisions. For example, the notion of informed consent can be extended to include live updates about which devices within an IoT network may have unpatched components. Individuals would have the information to take informed action before proceeding with a service. Service providers would be more transparent about the security risks faced by their networks. There would be more economic incentive and regulatory pressure for supply-chain partners with the relevant expertise to participate in after-market updates.

We now discuss U.S. federal and state proposals for data protection policies. In particular, we consider how the U.S. can develop and implement a GDPR-like policy. Differences in institutional settings — including the formal structures of the state, the preferences of dominant social groups, the roles of political parties in linking social preferences with state institutions, the power of bureaucracies, and economic constraints — place limits on cross-national policy comparison [220]. Nonetheless, the U.S. federal and state organizational structure has similarities to the political and economic union of E.U. member states — which lends itself to a comprehensive data protection framework that can be further delineated by each state to address specific needs.

Presently, there is no United States counterpart to the European Union's General Data Protection Regulation. The Privacy Act of 1974 governs the collection and use of personally identifiable information maintained by federal agencies [225]. The Children's Online Privacy Protection Act of 1998 (COPPA) is a United States federal law that provides specific regulation for the privacy of children including requirements such as verifiable consent from parents or guardians and restrictions on marketing [226]. In addition to being limited in scope, these U.S. federal regulations do not provide specific language that addresses areas where notions of data privacy, informed consent, and 4th Amendment rights are challenged

by emerging technologies [171].[4] A comprehensive U.S. data protection and privacy framework would provide a national standard, with global impact, that specifically addresses the contemporary policy implications of modern information and communication technologies.

There are promising indications that point to the potential for the development and convergence of data protection policies in a bottom-up state-by-state fashion. Presently, all 50 states, the District of Columbia, Guam, Puerto Rico, and the Virgin Islands have legislation in place requiring privately owned entities and government agencies to notify individuals impacted by security breaches [227]. Similarly, all 50 states have enacted a piece of consumer privacy legislation [228]. The California Data Privacy Act of 2018 (CCPA) is the most stringent and expansive of the privacy laws in the United States [229]. CCPA — which was released shortly after the E.U.'s GDPR and goes into effect in 2020 — gives much more control over private data to consumers than any other state legislation. This legislation has resulted in a number of other states releasing similar legislation [230].

There are pros and cons to pursuing a bottom-up state-by-state approach to data protection regulation as opposed to defining a top-down comprehensive framework. State-by-state policy implementation provides an agile vehicle for consolidating best practices nation-wide. A state-by-state policy landscape enables different strategies for data protection to be evaluated concurrently. States can also delve deeper into rules that address the specific needs of their constituents than would be appropriate in federal regulation. The strategies that are found to be most effective may then serve as references for other states or future iterations of federal data protection policy. Similarly, there is a smaller geographic scope for policy remedies that meet with adverse unanticipated consequences.

A drawback of the state-driven approach is that inconsistency in the U.S. policy landscape may make the U.S. market less accessible than global markets with consistent and clearly defined regulations on data use. Differences in requirements may complicate the compliance efforts of businesses with dealings in multiple states. There is the potential of the private

[4]The Fourth Amendment to the United States Constitution grants individuals with freedom from unreasonable searches and seizures and establishes legal requirements for issuing warrants.

sector exerting greater political influence over the state policy in ways that are at odds with the interests of individuals — states may end up competing with each other for jobs and taxable revenue at the expense of individual rights as has occurred in the global market for manufacturing [231, 232]. There is also a greater potential of legal loopholes and grey areas in a nonuniform policy landscape.

Another route, perhaps complementary or antecedent to comprehensive federal regulation, is to let domain experts in industry professional associations and non-regulatory government standardization organizations develop guidelines for IoT security and privacy policy. A consortium of industry and government organizations in partnership with civil liberties groups and privacy rights advocates may be an effective option to address security risks that are rapidly changing, global in scope, and occur within a geographically heterogeneous legal landscape. Several non-governmental organizations have issued cybersecurity frameworks for fog computing and IoT [233, 234, 235]. However, there is no clear consensus on which framework to adopt. Additionally, they have no legal or regulatory authority.

Members of the United States Congress are addressing the topic of smart device security and believe that the National Institute of Standards and Technology (NIST) should play a key role in defining a clear framework in four areas: (i) secure development standards, (ii) identity management standards for smart devices, (iii) patching standards for IoT devices, and (iv) configuration management for connected devices [236]. This bill introduced by Senator Mark Warner defines the covered devices as any physical object that connects to the Internet and has the computing capacity to store, send, and receive data but is not classed as a general-purpose computing device (e.g., personal computer, smartphone, programmable logic controls, or mainframe computing systems). Notably, the bill requires IoT devices meet certain security standards to be purchased by the U.S. government which may encourage IoT security research and development more broadly.

Collaborative public and private interventions is another route to data protection policy development and convergence that can integrate all of the components of federal regulation,

state regulation, and standardization. Research into the impacts of various approaches to IoT regulation on cybersecurity and the individual right to data protection is nascent. We can draw some lessons from policy efforts to improve labor rights and environmental standards in the global supply chain where, similar to IoT policy, state interventions must address a geographically dispersed issue involving conflicts of interest between civil liberties and commercial interests. Moreover, our capacity to achieve beneficial societal outcomes with respect to the cybersecurity and data protection mechanisms of connected devices is inextricably linked with our capacity to ensure beneficial societal outcomes in the global market for the production of these electronics.

Research and policy efforts to improve labor rights and environmental standards in the global supply chain highlight both the limitations of solely private interventions as well as the opportunity for productive cooperation between civil society, state governments, and multinational actors [231, 232]. The allure of increased employment and tax revenues for host nations coupled with slim profits margins and stiff competition have led to poor working conditions and lax environmental standards in the facilities producing global brands [237, 231].

In response of these issues, researchers have studied intervention efforts including state regulation (e.g., the enforcement of national labor and environmental laws), interventions by labor-oriented non-governmental organizations (NGOs), and voluntary regulatory systems implemented by multinational corporations (e.g., codes of conduct, audit programs, certification programs). In the area of labor and environmental issues within the global supply chain, there is little evidence that suggests that private initiatives alone lead to significant improvements to labor and environmental standards [238, 239]. However, research on the intersection of governance and labor standards suggests that the combination of public policy and private interventions has led to improvements to environmental standards and the work conditions of laborers in the global electronics supply chain [232].

Research on how public and private regulations interact in practice suggests that whether

they interact as compliments or substitutes depends on the national context and the issue being addressed [232]. In particular, Locke et al. find that private interventions to enforce labor and environmental standards are significantly affected by state actors and non-governmental actors. A quantitative analysis of the Hewlett-Packard supplier responsibility program (including factory audits, interviews with buyers and supplier management, and field research) found that national context — i.e., the strength of a nation's civil society and regulatory institutions — is the key factor in determining compliance [240].

Policy Recommendations: Addressing the following items would get the U.S. closer to having a comprehensive regulatory framework that addresses the security and privacy risks of connected device ecosystems:

- Governments should play a more active role in ensuring that the embedded systems market provides after-sales security updates to board support packages.
- The federal government should release a comprehensive framework on personal data use which addresses the challenges of informed consent in IoT.
- Mobile and edge network operators should play the unifying role of providing opaque device identity management services that facilitate individual control over personal data in dynamic IoT ecosystems.
- IoT standardization efforts should prioritize protocols that address the challenges of granting individuals with greater control over personal data in the context of IoT.
- IoT standardization efforts should explore user-controlled compliance mechanisms that minimize the need for individual IoT devices to implement regulated functionality.
- The United States should release federal regulation that establishes data protection as a fundamental individual right.
- The government should assist industry cybersecurity standardization efforts by providing a clear consensus on the legal definition of data protection as a fundamental

individual right.

- The government should assist industry cybersecurity standardization efforts by providing a clear consensus on which framework to adopt as well as a regulatory authority for enforcement.
- NIST should play a unifying role in defining a clear technical framework for secure development standards, identity management standards, patching standards, and device identity management standards for IoT devices.
- A comprehensive cybersecurity strategy should address issues stemming from both the production and the use of electronics.
- State and federal governments should fund auditing and research efforts to better understand how coordinated public policy and private interventions can lead to significant improvements to data protection practices in a globally dispersed IoT marketplace.

6.2 Internet of Things & Cybersecurity in the U.S. Government

A review of the U.S. government policies shows that there is a need for clear guidelines regarding the internal use of IoT products and services. In particular, the federal government lacks a clear cybersecurity framework for IoT, federal agencies do not have specific guidelines on the internal use of IoT, federal contractors do not have specific security requirements regarding the sale of IoT products to federal agencies, and there are structural impediments to the coordinated implementation of IoT guidelines among federal agencies.

Crawford and Sherman conduct an analysis of the U.S. federal laws and regulations that directly address or otherwise could be applied to IoT [241]. Their findings identify significant gaps in the United States policy on IoT security and privacy. The authors cite the pace

of change and growth in the emerging IoT marketplace as the key reason why the U.S. government policy has had difficulty providing a comprehensive cybersecurity framework for the space. According to Crawford and Sherman, there are no specific cybersecurity guidelines for the internal use of IoT devices within the federal government. As a result, it is not clear what security standards federal employees must meet before using IoT devices. Similarly, there is a need for guidelines regarding the security standards contractors must meet before the sale of IoT devices and services to the federal government. The Federal Acquisition Regulation (FAR) system, which defines the set of rules governing the acquisition process of federal agencies, contains no specific cybersecurity guidelines for the use of IoT devices. The federal government should provide specific guidelines on the security standards contractors must meet before the sale of IoT devices and services to the federal government to address this gap. As the results of this analysis suggest, providing clear and stringent security standards for obtaining government contractors may also induce the positive externality of broader market innovation into IoT cybersecurity mechanisms.

The current structure for how federal agencies develop and share information security best practices results in an inefficient leveraging of existing cybersecurity resources and an asymmetrical distribution of cybersecurity preparedness across federal agencies. The Federal Information Security Modernization Act (FIMSA) requires the Chief Information Officer of each federal agency to decide how to document, implement, and audit information security programs [242]. The result is that the cybersecurity standards significantly vary across federal agencies. A more coordinated approach would encourage federal agencies to leverage each other's resources by requiring the federal government to consolidate best practices into a cross-agency cybersecurity infrastructure. Efforts to establish such a cybersecurity baseline could be led by the council of federal CIOs under the leadership of the National Cybersecurity Coordinator.

The consequences of the lack of a clear cybersecurity framework for the IoT is exemplified in the case where researchers discovered that they could track the location of U.S. military

personnel by analyzing geolocation data streamed online from wearable fitness-tracking devices [243]. This discovery led to updates in the Department of Defense (DoD) internal policy on the use of connected devices, including prohibiting the use of both non-government and government-issued geolocation-capable devices, apps, and services [244]. Although the DoD ban on geo-locatable devices was a step in the right direction, the issue remains that there is a lack of clear guidelines describing which devices are permissible or how to properly configure devices for use while on the job. The usefulness of connected devices combined with a lack of clear guidelines leaves military personnel in a bind that may result in similar security compromises going forward.

Policy Recommendations: In summary, the primary limitations to the U.S. federal policy on the internal use of IoT is the lack of clear guidelines both for government staff and government contractors use. Addressing the following items would get us closer to having a regulatory framework that addresses the security risks to federal personnel and federal agencies posed by connected device ecosystems:

- The federal government should provide specific cybersecurity guidelines on the internal use of IoT.
- The federal government should provide specific guidelines on the security standards contractors must meet before the sale of IoT devices and services to the federal government.
- The federal government should assess the potential for stimulating broader market innovation into IoT cybersecurity mechanisms by requiring stringent security standards for government contractors.
- The federal government should reduce variations in cybersecurity standards across federal agencies.
- The federal agencies should leverage each other's cybersecurity resources to implement a cross-agency baseline security infrastructure.

6.3 Fog Mediation & Technologists

We advanced the idea of fog mediation — a fog computing-based reference monitor architecture as an aid to edge device security. Hence we identify several recommendations arising from the development of fog mediation and discuss their implications for technologists developing IoT security and privacy mechanisms and for technical treatments of data protection policy.

One implication of this work is that technical solutions like fog mediation may act as an aid to the adoption of data privacy policy and edge device security in hybrid cloud computing. Consider the following two data protection policy scenarios. The first scenario is the case where there are significant barriers to the global adoption of data protection policies. The second scenario is the case where there is a movement towards the global adoption of data protection policies. Technical strategies for data protection could be instrumental in realizing increased security and data protection for individuals in both scenarios.

In the scenario where there are significant barriers to the global adoption of data protection policies, standalone technical interventions can provide a remedy for individuals and businesses who would benefit from data protection practices but are based in or operating within regions that lack the institutional regulatory support for data protection policy. The economic pull of moving towards data protection as an individual right, even if limited to a few major economies like the U.S. and the EU, results in the manufacturing and availability of compliant products globally — including in territories that lack the institutional regulatory support for data protection policy.

In the scenario where there is a global adoption of consumer-oriented data protection policies regulating the use of IoT products and services, there remains the challenge of implementing compliant solutions. For example, previously we discussed the challenge of providing user capabilities such as informed consent in a way that does not drive up costs and complexity or otherwise acts as an impediment to innovation. In this scenario, standalone protection mechanisms that address security and privacy challenges provide the means for

both delivering user-controlled data protection and meeting regulatory requirements.

The role trusted computing may play in the development of fog computing is promising and potentially crucial to the security of the IoT. We explored the premise of leveraging fog computers as platforms for mediating trust between the cloud and the edge of the network in detail. Many of the technical proposals explored in this work leverage technologies that are feasible due to the emerging industry-wide paradigm shifts in computing. There are several technical recommendations with implications for policy that need to be addressed before the promise of secure IoT applications may be realized via fog mediation.

For the technologist, hardware security primitives minimize the complexities of protecting security-critical components. However, a recurring obstacle is a lack of access to such hardware-based security primitives. We observe that hardware-based security primitives are generally not available to app developers and end-users. In the cases where the market does offer such protections, the security design emphasizes vendor lock-in as opposed to protecting user assets [20]. We recommend the implementation of minimal trusted computing base, isolated execution support specifically for use by app developers. Similarly, we recommend the implementation of auditing capabilities like remote attestation specifically for use by end-users.

As described in this work and many related works [108, 26, 25, 24, 23, 22, 20, 21], hardware support for strong integrity and secrecy assurance is a promising component for enabling developers to build user-controlled and verifiably secure services. These components extend system-level security guarantees to developer-facing programming interfaces — making it feasible to safely execute and remotely verify security-sensitive code running alongside complex and potentially compromised software stacks [23]. When in place, these components provide transparent grounds for trust even in the cases where the incentives of service providers may be at odds with the expectations of individuals or consumer advocates.

We observe that there are significant security challenges related to manufacturing and deploying the hardware building blocks themselves. The upper-bound on the extent to which

service providers can deliver secure services using hardware-based isolated execution environments is inextricably linked with the industry controls that ensure the quality, reliability, and security of semiconductor products [240, 232]. Thus, designs for such systems should assess the risks and assumptions related to the entire device life cycle including during the manufacturing, system integration, and distribution phases.

Moreover, the extent to which the platform root of trust can be established independently of third parties improves the agency of security-sensitive service providers and, ultimately, of individuals. In particular, advances to secure development techniques that bring us closer to the ideal of hardware security primitives based on intrinsic trust such as hardware security module constructions that remove the need for secure manufacturing processes, trusted modules pre-loaded with platform secrets, or reliance on trusted third parties is of great interest and value. The use of Physically Unclonable Functions as a hardware root of trust basis for key derivation, secure key storage, and platform authentication is one such promising direction [23, 111].

We observe that code execution models that explicitly protect user data from the platform and software stack are an aid application security and facilitate communicating privacy and security to the user by minimizing the TCB. Thus, we recommend both sandboxing and isolated execution functionality [117, 111].

Composite security is a crucial aspect of delivering secure IoT products and services. Efforts aimed at enhancing security that focuses on applying security measures to individual connected devices do not adequately address the challenges of composite security. Research developments and accepted best practices are available within the cryptography community for how best to achieve composite security when constructing cryptographic protocols. However, further research into best practices for the composite security of products and services leveraging connected devices is needed. It is unlikely that device vendors will move to implement networking interfaces with composite security in mind without additional incentives to do so. This is due to the reality that composite security is a cross-cutting concern involving

coordination among multiple hardware and software vendors with little to no direct payoff for a given device vendor. Thus, composite security is a good candidate for further research by industry-wide consortium or regulatory intervention.

We observe that the notion of a domain in the IoT context is more closely aligned with physical proximity than with the logical relationship of components as in the client-server model. Further, spatiotemporally defined IoT contexts provide opportunities for novel privacy-enhanced data use models that align with our conventional non-digital notions of privacy [41, 111]. In this work, we focused on the problem of privacy-preserving deep neural networks based on a proximal domain threat model. Further research should investigate additional data protection applications of proximal domains.

Policy Recommendations:

- Technical treatments of edge device security and data protection like fog mediation act as an aid to both the individual right to data protection and the adoption of data privacy policy.
- We recommend greater availability of end-user access to hardware security primitives that provide strong integrity and secrecy assurance.
- We recommend the implementation of minimal trusted computing base, isolated execution support specifically for use by app developers.
- We recommend implementation of auditing capabilities like remote attestation specifically for use by
- Designs for secure systems should assess the risks and assumptions related to the entire device life cycle including during the manufacturing, system integration, and distribution phases.
- We recommend both sandboxing and isolated execution functionality.
- Further research should investigate spatiotemporally defined privacy threat models.

Bibliography

[1] SIA Anti-Counterfeiting Task Force, "Winning the battle against counterfeit semiconductor products." White Paper, Semiconductor Industry Association, Aug 2013. Accessed at http://www.semiconductors.org.

[2] D. Sawyer, "Counterfeit threat taking malicious turn?." Military Embedded Systems, Oct 2014. Accessed at http://mil-embedded.com.

[3] J. Bamford, "The nsa is building the country's biggest spy center (watch what you say)." Wired.com, Mar 2012. Accessed at http://www.wired.com.

[4] Associated Press, "Edward snowden timeline of events." Politico, Aug 2013. Accessed at http://www.politico.com.

[5] Y. Natis, D. Smith, E. Anderson, S. Nag, N. Cannon, and R. Buest, "Predicts 2019: Increasing reliance on cloud computing transforms it and business practices." Gartner.com, Dec 2018. Accessed at https://www.gartner.com.

[6] S. Soltan, P. Mittal, and H. V. Poor, "BlackIoT: IoT Botnet of high wattage devices can disrupt the power grid," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 15–32, 2018.

[7] D. Evans, "The Internet of Things: How the Next Evolution of the Internet Is Changing Everything." Technical Report, Cisco, Apr 2011. Accessed at https://www.cisco.com.

[8] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *Pervasive Computing, IEEE*, vol. 8, pp. 14–23, Oct 2009.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, ACM, 2012.

[10] C. Timberg, "Net of Insecurity: A Flaw in the Design," *Washington Post*, vol. 30, 2015.

[11] S. Charney and T. Computing, "Establishing End to End Trust," *The Microsoft Corporation*, 2008.

[12] N. R. Council *et al.*, *Trust in Cyberspace*. National Academies Press, 1999.

[13] R. Oppliger, *Internet and Intranet Security*. Norwood, MA, USA: Artech House, Inc., 1998.

[14] D. D. Clark and D. R. Wilson, "A comparison of commercial and military computer security policies," in *Security and Privacy, 1987 IEEE Symposium on*, pp. 184–184, IEEE, 1987.

[15] D. C. Latham, "Department of defense trusted computer system evaluation criteria," *Department of Defense*, 1986.

[16] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, *et al.*, "Meltdown: Reading kernel memory from user space," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 973–990, 2018.

[17] J. Horn *et al.*, "Reading privileged memory with a side-channel," *Project Zero*, vol. 39, 2018.

[18] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, *et al.*, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1–19, IEEE, 2019.

[19] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "Dawg: A defense against cache timing attacks in speculative execution processors." Cryptology ePrint Archive, Report 2018/418, 2018. https://eprint.iacr.org/2018/418.

[20] A. Vasudevan, J. M. McCune, and J. Newsome, *Trustworthy execution on mobile devices*. Springer, 2014.

[21] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 3, p. 8, 2015.

[22] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution.," *HASP@ ISCA*, vol. 10, 2013.

[23] E. Owusu, J. Guajardo, J. McCune, J. Newsome, A. Perrig, and A. Vasudevan, "OASIS: On Achieving a Sanctuary for Integrity and Secrecy on Untrusted Platforms," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 13–24, ACM, 2013.

[24] A. Vasudevan, J. McCune, J. Newsome, A. Perrig, and L. van Doorn, "CARMA: A Hardware Tamper-Resistant Isolated Execution Environment on Commodity x86 Platforms," in *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2012.

[25] S. Chhabra, B. Rogers, Y. Solihin, and M. Prvulovic, "SecureME: A Hardware-Software Approach to Full System Security," in *ACM International conference on Supercomputing (ICS)*, 2011.

[26] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, "TrustVisor: Efficient TCB reduction and attestation," in *IEEE Symposium on Security and Privacy (S&P)*, 2010.

[27] Department of Homeland Security, "Us-cert cyber security bulletin." Available at www.us-cert.gov.

[28] S. Zhang, D. Caragea, and X. Ou, "An empirical study on using the national vulnerability database to predict software vulnerabilities," in *International Conference on Database and Expert Systems Applications*, pp. 217–231, Springer, 2011.

[29] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing.," *HotCloud*, vol. 9, no. 9, p. 3, 2009.

[30] V. Haldar, D. Chandra, and M. Franz, "Semantic remote attestation: a virtual machine directed approach to trusted computing," in *USENIX Virtual Machine Research and Technology Symposium*, vol. 2004, 2004.

[31] M. Fort, F. Freiling, L. D. Penso, Z. Benenson, and D. Kesdogan, "Trustedpals: Secure multiparty computation implemented with smart cards," in *European Symposium on Research in Computer Security*, pp. 34–48, Springer, 2006.

[32] S. Bugiel, S. Nurnberger, A. Sadeghi, and T. Schneider, "Twin clouds: An architecture for secure cloud computing," in *Workshop on Cryptography and Security in Clouds (WCSC 2011)*, vol. 1217889, 2011.

[33] K. A. Küçük, A. Paverd, A. Martin, N. Asokan, A. Simpson, and R. Ankele, "Exploring the use of intel sgx for secure many-party applications," in *Proceedings of the 1st Workshop on System Software for Trusted Execution*, p. 5, ACM, 2016.

[34] F. J. Krautheim, D. S. Phatak, and A. T. Sherman, "Introducing the trusted virtual environment module: a new mechanism for rooting trust in cloud computing," in *International Conference on Trust and Trustworthy Computing*, pp. 211–227, Springer, 2010.

[35] E. P. Komarla and V. J. Zimmer, "Secure booting and provisioning," Apr. 17 2007. US Patent 7,207,039.

[36] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: A virtual machine-based platform for trusted computing," in *ACM SIGOPS Operating Systems Review*, vol. 37, pp. 193–206, ACM, 2003.

[37] M. Bhargava and K. Mai, "An efficient reliable puf-based cryptographic key generator in 65nm cmos," in *Proceedings of the conference on Design, Automation & Test in Europe*, p. 70, European Design and Automation Association, 2014.

[38] K.-W. Park, J. Han, J. Chung, and K. H. Park, "Themis: A mutually verifiable billing system for the cloud computing environment," *IEEE Transactions on Services Computing*, vol. 6, no. 3, pp. 300–313, 2013.

[39] X.-F. Wang, Y. Mu, and R.-M. Chen, "An efficient privacy-preserving aggregation and billing protocol for smart grid," *Security and Communication Networks*, vol. 9, no. 17, pp. 4536–4547, 2016.

[40] V. Kulkarni, B. Chapuis, and B. Garbinato, "Privacy-preserving location-based services by using intel sgx," in *Proceedings of the First International Workshop on Human-centered Sensing, Networking, and Systems*, pp. 13–18, ACM, 2017.

[41] J. Chi, E. Owusu, X. Yin, T. Yu, W. Chan, P. Tague, and Y. Tian, "Privacy Partitioning: Protecting User Data During the Deep Learning Inference Phase." Under Review. Preprint online at arxiv.org, 2018.

[42] N. Kuntze and A. U. Schmidt, "Trusted computing in mobile action," *arXiv preprint cs/0606045*, 2006.

[43] N. Asokan, J.-E. Ekberg, K. Kostiainen, A. Rajan, C. Rozas, A.-R. Sadeghi, S. Schulz, and C. Wachsmann, "Mobile trusted computing," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1189–1206, 2014.

[44] C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Capkun, "Secure enrollment and practical migration for mobile trusted execution environments," in *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*, pp. 93–98, ACM, 2013.

[45] R. Sandhu and X. Zhang, "Peer-to-peer access control architecture using trusted computing technology," in *Proceedings of the tenth ACM symposium on Access control models and technologies*, pp. 147–158, ACM, 2005.

[46] M. C. Mont, S. Pearson, and P. Bramhall, "Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services," in *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*, pp. 377–382, IEEE, 2003.

[47] D. Song, E. Shi, I. Fischer, and U. Shankar, "Cloud data protection for the masses," *Computer*, vol. 45, no. 1, pp. 39–45, 2012.

[48] S. Gueron, "A memory encryption engine suitable for general purpose processors.," *IACR Cryptology ePrint Archive*, vol. 2016, p. 204, 2016.

[49] B. Kauer, "Oslo: Improving the security of trusted computing.," in *USENIX Security Symposium*, pp. 229–237, 2007.

[50] D. Thilakanathan, S. Chen, S. Nepal, R. A. Calvo, D. Liu, and J. Zic, "Secure multiparty data sharing in the cloud using hardware-based tpm devices," in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pp. 224–231, IEEE, 2014.

[51] S. Gajek, H. Löhr, A.-R. Sadeghi, and M. Winandy, "Truwallet: trustworthy and migratable wallet-based web authentication," in *Proceedings of the 2009 ACM workshop on Scalable trusted computing*, pp. 19–28, ACM, 2009.

[52] S. Weiser and M. Werner, "Sgxio: generic trusted i/o path for intel sgx," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pp. 261–268, ACM, 2017.

[53] C. Landwehr, D. Boneh, J. Mitchell, S. Bellovin, S. Landau, and M. Lesk, "Privacy and Cybersecurity: The Next 100 Years," *Proceedings of the IEEE*, vol. 100, pp. 1659–1673, May 2012.

[54] R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols." Cryptology ePrint Archive, Report 2000/067, 2000. http://eprint.iacr.org/.

[55] K. Thompson, "Reflections on trusting trust," *Communications of the ACM*, vol. 27, no. 8, pp. 761–763, 1984.

[56] Brian Krebs, "Coordinated ATM Heist Nets Thieves $13M," 2011. Available at http://krebsonsecurity.com.

[57] Lucian Constantin, "One year after DigiNotar breach, Fox-IT details extent of compromise," 2012. Available at www.wired.com.

[58] "The CDW 2011 Cloud Computing Tracking Poll," 2011. Available at www.cdw.com.

[59] D. Song, E. Shi, I. Fischer, and U. Shankar, "Cloud data protection for the masses," *IEEE Computer*, 2012.

[60] Jason Kincaid, "Google Confirms That It Fired Engineer For Breaking Internal Privacy Policies," 2010. Available at http://techcrunch.com.

[61] Symantec, "Symantec-Sponsored Ponemon Report Finds Negligent Employees Top Cause of Data Breaches in the U.S. While Malicious Attacks Most Costly," 2012. Available at www.symantec.com.

[62] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon Physical Random Functions," in *ACM conference on Computer and Communications Security (CCS)*, 2002.

[63] R. S. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical One-way Functions," *Science*, 2002. Available at web.media.mit.edu.

[64] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Controlled Physical Random Functions," in *Proceedings of Annual Computer Security Applications Conference (ACSAC)*, 2002.

[65] P. Tuyls, G.-J. Schrijen, B. Skoric, J. van Geloven, N. Verhaegh, and R. Wolters, "Read-Proof Hardware from Protective Coatings," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2006.

[66] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA Intrinsic PUFs and Their Use for IP Protection," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2007.

[67] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers," *IEEE Trans. Computers*, 2009.

[68] K. Kursawe, A.-R. Sadeghi, D. Schellekens, B. Skoric, and P. Tuyls, "Reconfigurable Physical Unclonable Functions – Enabling Technology for Tamper-Resistant Storage," in *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*, 2009.

[69] Y. Wang, W. kei Yu, S. Wu, G. Malysa, G. E. Suh, and E. C. Kan, "Flash Memory for Ubiquitous Hardware Security Functions: True Random Number Generation and Device Fingerprints," in *IEEE Symposium on Security and Privacy (S&P)*, 2012.

[70] A. Juels and M. Wattenberg, "A Fuzzy Commitment Scheme," in *ACM conference on Computer and Communications Security (CCS)*, 1999.

[71] J.-P. Linnartz and P. Tuyls, "New Shielding Functions to Enhance Privacy and Prevent Misuse of Biometric Templates," in *International conference on Audio and Video Based Biometric Person Authentication (AVBPA)*, 2003.

[72] Y. Dodis, M. Reyzin, and A. Smith, "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data," in *Advances in Cryptology (EUROCRYPT)*, 2004.

[73] G. Taylor and G. Cox, "Behind Intel's New Random-Number Generator," *IEEE Spectrum*, 2011. Available at http://spectrum.ieee.org.

[74] Y. Lu, L.-T. Lo, G. Watson, and R. Minnich, "CAR: Using Cache as RAM in LinuxBIOS," 2012. Available at http://rere.qmqm.pl/ mirq.

[75] X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, and A. Smith, "Secure Remote Authentication Using Biometric Data," in *Advances in Cryptology (EUROCRYPT)*, 2005.

[76] Y. Dodis, J. Katz, L. Reyzin, and A. Smith, "Robust Fuzzy Extractors and Authenticated Key Agreement from Close Secrets," in *Advances in Cryptology (CRYPTO)*, 2006.

[77] X. Boyen, "Reusable cryptographic fuzzy extractors," in *ACM Conference on Computer and Communications Security (CCS)*, 2004.

[78] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient Helper Data Key Extractor on FPGAs," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2008.

[79] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-Overhead Implementation of a Soft Decision Helper Data Algorithm for SRAM PUFs," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2009.

[80] L. Chen, "Recommendation for Key Derivation Using Pseudorandom Functions (Revised)." NIST Special Publication 800-108, 2009.

[81] H. Krawczyk, "Cryptographic Extraction and Key Derivation: The HKDF Scheme," in *Advances in Cryptology*, CRYPTO, 2010.

[82] IEEE, "IEEE Standard Specifications for Public-Key Cryptography — IEEE Std 1363^{TM}-2000," 2000. Available at www.ieee.org.

[83] A. K. Lenstra and E. R. Verheul, "Selecting Cryptographic Key Sizes," *Journal of Cryptology*, 2001. 10.1007/s00145-001-0009-4.

[84] H. Cramér, "On the order of magnitude of the difference between prime numbers," *Acta Arithmetica*, 1937.

[85] H. Maier, "Primes in short intervals," *Michigan Math J.*, 1985.

[86] T. R. Nicely, "Tables of prime gaps." Available at http://www.trnicely.net/index.html#TPG. Accessed July 2012.

[87] V. Shoup, "OAEP Reconsidered," in *Advances in Cryptology (CRYPTO)*, 2001. Available at www.shoup.net.

[88] D. Dolev, C. Dwork, and M. Naor, "Non-Malleable Cryptography," in *SIAM Journal on Computing*, 2000.

[89] V. Shoup, "A Proposal for an ISO Standard for Public Key Encryption." Version 2.1, 2001. Available at www.shoup.net.

[90] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An Execution Infrastructure for TCB Minimization," in *ACM European Conference in Computer Systems (EuroSys)*, 2008.

[91] B. Parno, J. R. Lorch, J. R. Douceur, J. W. Mickens, and J. M. McCune, "Memoir: Practical state continuity for protected modules," in *IEEE Symposium on Security and Privacy (S&P)*, 2011.

[92] J. Li, M. Krohn, D. Mazières, and D. Shasha, "Secure Untrusted Data Depository (SUNDR)," in *USENIX Symposium on Operating Systems Design & Implementation (OSDI)*, 2004.

[93] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, 2002.

[94] Virtutech, "Simics x86-440BX Target Guide," 2010.

[95] J. Schmitz, J. Loew, J. Elwell, D. Ponomarev, and N. B. Abu-Ghazaleh, "TPM-SIM: A Framework for Performance Evaluation of Trusted Platform Modules," in *ACM Design Automation Conference (DAC)*, 2011.

[96] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. D. Gligor, and A. Perrig, "TrustVisor: Efficient TCB Reduction and Attestation," in *IEEE Symposium on Security and Privacy (S&P)*, 2010.

[97] "ARM Security Technology - Building a Secure System using TrustZone Technology," 2009. Available at http://infocenter.arm.com/.

[98] "Intel Trusted Execution Technology (Intel TXT) - Software Development Guide," 2013. Document Number: 315168-009 Available at www.intel.com.

[99] S. P. J. Ittai Anati, Shay Gueron, "Innovative Technology for CPU Attestation and Sealing," in *Workshop on Hardware Architecture for Security and Privacy*, 2013.

[100] K. El Defrawy, A. Francillon, D. Perito, and G. Tsudik, "SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust," in *Network and Distributed System Security Symposium (NDSS)*, 2012.

[101] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural Support for Copy and Tamper Resistant Software," *ACM SIGPLAN Notices*, 2000.

[102] R. Lee, P. Kwan, J. McGregor, J. Dwoskin, and Z. Wang, "Architecture for Protecting Critical Secrets in Microprocessors," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2005.

[103] J. S. Dwoskin and R. B. Lee, "Hardware-rooted trust for secure key management and transient trust," in *ACM conference on Computer and communications security (CCS)*, 2007.

[104] P. Williams and R. Boivie, "CPU Support for Secure Executables," in *Trust and Trustworthy Computing*, 2011.

[105] B. Gassend, M. van Dijk, D. Clarke, and S. Devadas, "Controlled physical random functions," in *Security with Noisy Data*, pp. 235–253, Springer, 2007.

[106] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, *FPGA intrinsic PUFs and their use for IP protection*. Springer, 2007.

[107] G. E. Suh, C. W. O'Donnell, and S. Devadas, "Aegis: A single-chip secure processor," *Information Security Technical Report*, vol. 10, no. 2, pp. 63–73, 2005.

[108] G. E. Suh, C. W. O'Donnell, and S. Devadas, "AEGIS: A Single-Chip Secure Processor," *Information Security Technical Report*, 2005.

[109] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and A. Seshadri, "How Low Can You Go? Recommendations for Hardware-Supported Minimal TCB Code Execution," in *ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008.

[110] S. W. Smith and S. Weingart, "Building a High-Performance, Programmable Secure Coprocessor," *Computer Networks*, 1999.

[111] E. Owusu, Y. Tian, J. Guajardo, and P. Tague, "MEDIA: Mediating Edge Device-based Integrated Access." Under Review. Preprint online at http://mews.sv.cmu.edu, 2018.

[112] Verizon, "IP Latency Statistics." Verizon Enterprise, 2018. Accessed at http://www.verizonenterprise.com/.

[113] K. Phemius and M. B. Thales, "Openflow: Why latency does matter," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pp. 680–683, IEEE, 2013.

[114] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. A. Lee, and J. Kubiatowicz, "The Cloud is Not Enough: Saving IoT from the Cloud.," in *HotStorage*, 2015.

[115] R. Miller, "The Internet of Things May Create A New Breed of Data Centers." DataCenterFrontier.com, 2016. Accessed at http://datacenterfrontier.com.

[116] F. Meunier, A. Wood, K. Weiss, K. Huberty, S. Flannery, J. Moore, C. Hettenbach, and B. Lu, "The 'Internet of Things' Is Now: Connecting the Real Economy." Technical Report, Morgan Stanley, Apr 2014.

[117] Y. Li, J. M. McCune, J. Newsome, A. Perrig, B. Baker, and W. Drewry, "Minibox: A two-way sandbox for x86 native code.," in *USENIX Annual Technical Conference*, pp. 409–420, 2014.

[118] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An Execution Infrastructure for TCB Minimization," in *ACM European Conference on Computer Systems (EuroSys)*, 2008.

[119] F. Petitcolas, "La cryptographie militaire," 1883.

[120] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *Journal of Cryptographic Engineering*, pp. 1–27, 2016.

[121] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "Brand and ip protection with physical unclonable functions," in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pp. 3186–3189, IEEE, 2008.

[122] P. Tuyls and L. Batina, "Rfid-tags for anti-counterfeiting," in *Cryptographers' Track at the RSA Conference*, pp. 115–131, Springer, 2006.

[123] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[124] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas, "Extracting secret keys from integrated circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 10, pp. 1200–1205, 2005.

[125] A.-R. Sadeghi, I. Visconti, and C. Wachsmann, "Enhancing rfid security and privacy by physically unclonable functions," in *Towards Hardware-Intrinsic Security*, pp. 281–305, Springer, 2010.

[126] M. Bhargava, C. Cakir, and K. Mai, "Comparison of bi-stable and delay-based physical unclonable functions from measurements in 65nm bulk cmos," in *Custom Integrated Circuits Conference (CICC), 2012 IEEE*, pp. 1–4, IEEE, 2012.

[127] Thales Communications & Security, "Foundations for Forgery-Resistant Security Hardware ," 2012.

[128] C. Helfmeier, C. Boit, D. Nedospasov, and J.-P. Seifert, "Cloning physically unclonable functions," in *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, pp. 1–6, IEEE, 2013.

[129] Y. Oren, A.-R. Sadeghi, and C. Wachsmann, "On the effectiveness of the remanence decay side-channel to clone memory-based pufs," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 107–125, Springer, 2013.

[130] S. Zeitouni, Y. Oren, C. Wachsmann, P. Koeberl, and A.-R. Sadeghi, "Remanence Decay Side-Channel: The PUF Case," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1106–1116, 2016.

[131] D. E. Holcomb, W. P. Burleson, and K. Fu, "Initial SRAM State as a Fingerprint and Source of True Random Numbers for RFID Tags." Conference on RFID Security 07, 2007.

[132] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[133] B. Iannucci, P. Tague, O. J. Mengshoel, and J. Lohn, "Crossmobile: A cross-layer architecture for next-generation wireless systems," 2014.

[134] Y. Ngoko, "Heating as a cloud-service, a position paper (industrial presentation)," in *European Conference on Parallel Processing*, pp. 389–401, Springer, 2016.

[135] W. Hu, Y. Gao, K. Ha, J. Wang, B. Amos, Z. Chen, P. Pillai, and M. Satyanarayanan, "Quantifying the impact of edge computing on mobile applications," in *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*, p. 5, ACM, 2016.

[136] J. Nielsen, "Website response times," *Nielsen Norman Group*, vol. 21, no. 06, 2010. Accessed at https://www.nngroup.com/.

[137] G. Čandrlić, "How Website Speed Affects Conversion Rates." Global Dots, 2012. Accessed at http://www.globaldots.com/.

[138] K. Eaton, "How One Second Could Cost Amazon $1.6 Billion In Sales," *Fast Company*, vol. 14, 2012. Accessed at https://www.fastcompany.com.

[139] Apple, "Final Cut Pro 7 User Manual - Appendix D: Frame Rate and Timecode." Final Cut Pro Help, 2010. Accessed at https://documentation.apple.com.

[140] E. Chester, "Monitor Refresh Rates: Why higher isn't always better." Trusted Reviews, 2017. Accessed at http://www.trustedreviews.com/.

[141] A. Wiltshire, "The latency problem: why modern gaming PCs are slower than an Apple II." PC Gamer, 2018. Accessed at https://www.pcgamer.com.

[142] vrs.org.uk, "Head-mounted Displays (HMDs)." Virtual Reality Society, 2017. Accessed at https://www.vrs.org.uk/.

[143] appleinsider.com, "Apple VR/AR." Apple Insider, 2017. Accessed at https://appleinsider.com.

[144] digitaltrends.com, "Oculus Rift vs HTC Vive." Digital Trends, 2017. Accessed at https://www.digitaltrends.com.

[145] N. Hughes, "iOS 10.3 beta hints at more fluid Apple Pencil with next-gen iPad Pro models." Apple Insider, 2017. Accessed at https://appleinsider.com.

[146] N. Antonopoulos and L. Gillam, *Cloud computing*. Springer, 2010.

[147] N. Davies, N. Taft, M. Satyanarayanan, S. Clinch, and B. Amos, "Privacy Mediators: Helping IoT Cross the Chasm," in *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, pp. 39–44, ACM, 2016.

[148] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. Stillwell, *et al.*, "Scone: Secure linux containers with intel sgx.," in *OSDI*, vol. 16, pp. 689–703, 2016.

[149] V. Costan and S. Devadas, "Intel sgx explained.," *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[150] E. Nallusamy, "A framework for using processor cache as ram (car)," 2005.

[151] Intel, "Software guard extensions programming reference," 2014. Accessed at https://software.intel.com.

[152] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt, "Rethinking the library os from the top down," in *ACM SIGPLAN Notices*, vol. 46, pp. 291–304, ACM, 2011.

[153] D. Kuvaiskii, O. Oleksenko, S. Arnautov, B. Trach, P. Bhatotia, P. Felber, and C. Fetzer, "Sgxbounds: Memory safety for shielded execution," in *Proceedings of the Twelfth European Conference on Computer Systems*, pp. 205–221, ACM, 2017.

[154] S. Checkoway and H. Shacham, *Iago attacks: Why the system call API is a bad untrusted RPC interface*, vol. 41. ACM, 2013.

[155] Intel, "Intel core™ i7-6600u processor," 2015. Accessed at https://ark.intel.com.

[156] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "Vc3: Trustworthy data analytics in the cloud using sgx," in *Security and Privacy (SP), 2015 IEEE Symposium on*, pp. 38–54, IEEE, 2015.

[157] U. F. Mayer, "Linux/unix nbench," 2003. Accessed at https://www.math.utah.edu/ mayer.

[158] Y. Fu, E. Bauman, R. Quinonez, and Z. Lin, "Sgx-lapd: thwarting controlled side channel attacks via enclave verifiable page faults," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 357–380, Springer, 2017.

[159] S. MALLICK, "Facemark : Facial landmark detection using opencv." learnopencv.com, 2018. Accessed at https://www.learnopencv.com.

[160] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *CVPR*, 2014.

[161] G. Tzimiropoulos and M. Pantic, "Optimization problems for fast aam fitting in-the-wild," in *Proceedings of the IEEE international conference on computer vision*, pp. 593–600, 2013.

[162] S. Ren, X. Cao, Y. Wei, and J. Sun, "Face alignment at 3000 fps via regressing local binary features," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1685–1692, 2014.

[163] "Embedded tls library for applications, devices, iot, and the cloud." Accessed at https://www.wolfssl.com/.

[164] G. B. H. E. Learned-Miller, "Labeled faces in the wild: Updates and new reporting procedures," Tech. Rep. UM-CS-2014-003, University of Massachusetts, Amherst, May 2014.

[165] J. S. Chung, A. W. Senior, O. Vinyals, and A. Zisserman, "Lip reading sentences in the wild.," in *CVPR*, pp. 3444–3453, 2017.

[166] S. Suwajanakorn, S. M. Seitz, and I. Kemelmacher-Shlizerman, "Synthesizing Obama: Learning Lip sync from Audio," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 95, 2017.

[167] Z. Wojna, A. Gorban, D.-S. Lee, K. Murphy, Q. Yu, Y. Li, and J. Ibarz, "Attention-based extraction of structured information from street view imagery," *arXiv preprint arXiv:1704.03549*, 2017.

[168] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.

[169] Y. Zhang, Y. Sun, P. Phillips, G. Liu, X. Zhou, and S. Wang, "A multilayer perceptron based smart pathological brain detection system by fractional Fourier entropy," *Journal of medical systems*, vol. 40, no. 7, p. 173, 2016.

[170] M. Drange and R. Albergotti, "At Ring's R&D team, security gaps and rookie engineers," 2018. Accessed at www.theinformation.com.

[171] J. Vlahos, "Smart talking: Are our devices threatening our privacy?," 2019. Accessed at www.theguardian.com.

[172] M. Day, G. Turner, and N. Drozdiak, "Amazon workers are listening to what you tell Alexa," 2019. Accessed at www.bloomberg.com.

[173] S. Schechner and M. Secada, "You give apps sensitive personal information. then they tell Facebook.," 2019. Accessed at www.wsj.com.

[174] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 619–631, ACM, 2017.

[175] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*, pp. 201–210, 2016.

[176] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[177] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pp. 1310–1321, ACM, 2015.

[178] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pp. 1054–1067, ACM, 2014.

[179] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1322–1333, ACM, 2015.

[180] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the GAN: information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 603–618, ACM, 2017.

[181] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*, pp. 265–284, Springer, 2006.

[182] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[183] Y. LeCun, "The MNIST database of handwritten digits," *http://yann.lecun.com/exdb/mnist/*, 1998.

[184] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[185] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," *arXiv preprint*, 2018.

[186] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Security and Privacy (SP), 2017 IEEE Symposium on*, pp. 3–18, IEEE, 2017.

[187] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[188] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 625–660, 2010.

[189] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," in *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008.

[190] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.

[191] S. Moschoglou, A. Papaioannou, C. Sagonas, J. Deng, I. Kotsia, and S. Zafeiriou, "AgeDB: the first manually collected, in-the-wild age database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop*, vol. 2, p. 5, 2017.

[192] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[193] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[194] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[195] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," 2010.

[196] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Advances in neural information processing systems*, pp. 3320–3328, 2014.

[197] L. Fan, "Image pixelization with differential privacy," in *IFIP Annual Conference on Data and Applications Security and Privacy*, pp. 148–162, Springer, 2018.

[198] A. C. Gilbert, Y. Zhang, K. Lee, Y. Zhang, and H. Lee, "Towards understanding the invertibility of convolutional neural networks," *arXiv preprint arXiv:1705.08664*, 2017.

[199] A. Hore and D. Ziou, "Image quality metrics: PSNR vs. SSIM," in *Pattern recognition (icpr), 2010 20th international conference on*, pp. 2366–2369, IEEE, 2010.

[200] A. Dosovitskiy and T. Brox, "Generating images with perceptual similarity metrics based on deep networks," in *Advances in Neural Information Processing Systems*, pp. 658–666, 2016.

[201] H. R. Sheikh, M. F. Sabir, and A. C. Bovik, "A statistical evaluation of recent full reference image quality assessment algorithms," *IEEE Transactions on image processing*, vol. 15, no. 11, pp. 3440–3451, 2006.

[202] N. Ponomarenko, V. Lukin, A. Zelensky, K. Egiazarian, M. Carli, and F. Battisti, "TID2008 - a database for evaluation of full-reference visual quality assessment metrics," *Advances of Modern Radioelectronics*, vol. 10, no. 4, pp. 30–45, 2009.

[203] E. C. Larson and D. M. Chandler, "Most apparent distortion: full-reference image quality assessment and the role of strategy," *Journal of Electronic Imaging*, vol. 19, no. 1, p. 011006, 2010.

[204] N. Ponomarenko, L. Jin, O. Ieremeiev, V. Lukin, K. Egiazarian, J. Astola, B. Vozel, K. Chehdi, M. Carli, F. Battisti, *et al.*, "Image database TID2013: Peculiarities, results and perspectives," *Signal Processing: Image Communication*, vol. 30, pp. 57–77, 2015.

[205] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[206] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[207] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," *CoRR*, vol. abs/1702.07464, 2017.

[208] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 587–601, ACM, 2017.

[209] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig, "Resolving individuals contributing trace amounts of dna to highly complex mixtures using high-density snp genotyping microarrays," *PLoS genetics*, vol. 4, no. 8, p. e1000167, 2008.

[210] M. Backes, P. Berrang, M. Humbert, and P. Manoharan, "Membership privacy in microRNA-based studies," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 319–330, ACM, 2016.

[211] C. Dwork, A. Smith, T. Steinke, J. Ullman, and S. Vadhan, "Robust traceability from trace amounts," in *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pp. 650–669, IEEE, 2015.

[212] A. Pyrgelis, C. Troncoso, and E. De Cristofaro, "Knock knock, who's there? membership inference on aggregate location data," *arXiv preprint arXiv:1708.06145*, 2017.

[213] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs.," in *USENIX Security Symposium*, pp. 601–618, 2016.

[214] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," *arXiv preprint arXiv:1802.05351*, 2018.

[215] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 308–318, ACM, 2016.

[216] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," *arXiv preprint arXiv:1610.05755*, 2016.

[217] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data.," in *NDSS*, 2015.

[218] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *Security and Privacy (SP), 2013 IEEE Symposium on*, pp. 334–348, IEEE, 2013.

[219] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh, "Privacy-preserving matrix factorization," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 801–812, ACM, 2013.

[220] C. J. Bennett, *Regulating privacy: Data protection and public policy in Europe and the United States*. Cornell University Press, 1992.

[221] European Parliament, "Eu regulation 2016/679: General data protection regulation," 2016. Accessed at eur-lex.europa.eu.

[222] B. Schneier, "The Internet of Things is wildly insecure — and often unpatchable," 2014. Accessed at www.wired.com.

[223] D. O'Shea, "48% of consumers say hurdles deter loyalty program sign-up," 2018. Accessed at www.retaildive.com.

[224] R. Roman, P. Najera, and J. Lopez, "Securing the internet of things," *Computer*, no. 9, pp. 51–58, 2011.

[225] US DOJ Office of Privacy and Civil Liberties, "Overview of the privacy act of 1974," 2015. Accessed at www.justice.gov.

[226] United States Code, "Children's online privacy protection act of 1998," 1998. Accessed at www.law.cornell.edu.

[227] National Conference of State Legislatures, "Security breach notification laws," 2019. Accessed at http://www.ncsl.org.

[228] National Conference of State Legislatures, "State laws related to internet privacy," 2019. Accessed at http://www.ncsl.org.

[229] IEEE, "California consumer privacy act of 2018," 2018. Accessed at leginfo.legislature.ca.go.

[230] J. Lazzarotti, J. Gavejian, and M. Atrakchi, "State law developments in consumer privacy," 2019. Accessed at www.workplaceprivacyreport.com.

[231] T. Connor and K. Dent, "Offside! labour rights and sportswear production in asia," *Oxfam Policy and Practice: Private Sector*, vol. 3, no. 1, pp. 1–112, 2006.

[232] R. M. Locke, B. A. Rissing, and T. Pal, "Complements or substitutes? private codes, state regulation and the enforcement of labour standards in global supply chains," *British Journal of Industrial Relations*, vol. 51, no. 3, pp. 519–552, 2013.

[233] Industrial Internet Consortium, "Industrial internet of things volume g4: Security framework," 2016. Accessed at www.iiconsortium.org.

[234] IEEE, "Ieee internet of things related standards," 2019. Accessed at standards.ieee.org.

[235] OpenFog Consortium, "IEEE Approved Draft Standard for Adoption of OpenFog Reference Architecture for Fog Computing," *IEEE Std. P1934*, 2018.

[236] 116th US Congress, "US Congress S.734: IoT Cybersecurity Improvement Act of 2019," 2014. Accessed at www.congress.gov.

[237] P. Overeem, "Reset: Corporate social responsibility in the global electronics supply chain," *Amsterdam: GoodElectronics and the Dutch CSR Platform (MVO Platform). Retrieved June*, vol. 8, p. 2014, 2009.

[238] R. M. Locke, F. Qin, and A. Brause, "Does monitoring improve labor standards? lessons from nike," *ILR Review*, vol. 61, no. 1, pp. 3–31, 2007.

[239] R. Locke, M. Amengual, and A. Mangla, "Virtue out of necessity? compliance, commitment, and the improvement of labor conditions in global supply chains," *Politics & Society*, vol. 37, no. 3, pp. 319–351, 2009.

[240] R. Locke, G. Distelhorst, T. Pal, and H. Samel, "Production goes global, standards stay local: Private labor regulation in the global electronics industry," *MIT Political Science Department Research Paper*, 2012.

[241] D. Crawford and J. Sherman, "Gaps in united states federal government IoT security and privacy policies," *Journal of Cyber Policy*, vol. 3, no. 2, pp. 187–200, 2018.

[242] United States Federal Government, "US Public Law 113–283: The Federal Information Security Modernization Act of 2014," 2014. Accessed at www.congress.gov.

[243] J. Hsu, "The Strava heat map and the end of secrets," 2018. Accessed at www.wired.com.

[244] US Department of Defense, "Use of geolocation-capable devices, applications, and services," 2018. Accessed at admin.govexec.com.