A Survey of Trustworthy Computing on Mobile & Wearable Systems

Travis Peters
Dartmouth Computer Science Technical Report TR2017-823
May 25, 2017

ABSTRACT

Mobile and wearable systems have generated unprecedented interest in recent years, particularly in the domain of mobile health (mHealth) where carried or worn devices are used to collect health-related information about the observed person. Much of the information – whether physiological, behavioral, or social – collected by mHealth systems is sensitive and highly personal; it follows that mHealth systems should, at the very least, be deployed with mechanisms suitable for ensuring confidentiality of the data it collects. Additional properties – such as integrity of the data, source authentication of data, and data freshness – are also desirable to address other security, privacy, and safety issues.

Developing systems that are robust against capable adversaries (including physical attacks) is, and has been, an active area of research. While techniques for protecting systems that handle sensitive data are well-known today, many of the solutions in use today are not well suited for mobile and wearable systems, which are typically limited with respect to power, memory, computation, and other capabilities.

In this paper we look at prior research on developing trustworthy mobile and wearable systems. To survey this topic we begin by discussing solutions for securing computing systems that are not subject to the type of strict constraints associated with mobile and wearable systems. Next, we present other efforts to design and implement trustworthy mobile and wearable systems. We end with a discussion of future directions.

1 INTRODUCTION

As personal devices, mobile and wearable devices often handle data that is valuable to the user of the device; this data is potentially sensitive. As a result, many solutions have been proposed to protect systems from compromise and to ensure the protection of user data handled by the device. To provide meaningful protection, it is necessary to understand users' requirements for data protection on these mobile devices [13]. More specifically, it is necessary to look at types of data that users want to protect, understand current users' practices for protecting data, and understand how security requirements vary for different data types.

Passwords, location data, personal messages (e.g., SMS) and documents (e.g., emails), photos and videos, audio recordings, and health data may all be sensitive information. For example, password-based logins play an increasingly important role on user systems. We use passwords to authenticate ourselves to countless applications and services. For help and convenience, users turn to applications such as one-time password (OTP) generators, password managers, and two-factor authentication services. Many of these applications can be found on mobile systems today. The problem is that these applications cannot guarantee the confidentiality of passwords, authentication tokens or seeds when the mobile OS is compromised. Fortunately, there exist some trusted computing technologies today that can address some of these problems. TrustOTP [17] is a secure one-time password (OTP) solution for mobile devices that leverages ARM TrustZone security features to protect the confidentiality of the OTPs against a malicious mobile OS. Also, TrustLogin [21] is a solution for Intel platforms that uses System Management Mode to protect user login credentials from malware (e.g., keyloggers) even when the OS is compromised. These are useful but highly specific instances of trusted computing technologies put to use to protect user data. In this paper we identify past and present technologies that can be used to protect user data in light of growing concerns around security and privacy.

Isolating code execution is one of the fundamental approaches to achieving security; past work has surveyed solutions towards this end [22]. OS-based and Virtual Machine-based isolation have dependencies on operating systems and hypervisors that may have large Trusted Computing Bases (TCB); e.g., the Xen hypervisor has 532K lines of source code. Also, OS-based and VM-based isolation do not address hypervisor or firmware (e.g., BIOS) rootkits. Generally speaking, recent trends suggest that excluding large, error-prone software such as a hypervisor and the OS from the TCB, is an effective way to make system exploitations more difficult for adversaries, effectively requiring higher privilege levels to compromise the system. In this paper we focus attention on trusted computing technologies that provide isolated code execution but do so by relying on hardware assistance, as these solutions generally exclude these error-prone hypervisors and operating systems from the TCB.

Regardless of the security mechanism, we observe that any trusted-computing solution needs to meet the following non-security requirements: the system should (1) perform well – especially with respect to mobile and wearable devices – in terms of energy usage, usability, latency, and computation; and the solution should (2) work with popular operating systems that are in use by the majority of mobile device users (e.g., iOS, Android). Failure to meet either of these requirements will likely mean that the technology will not be used by most people, thus defeating the purpose of designing a technology to protect their data.

In the remainder of this paper, we present relevant background in computer architecture, trusted computing, and the general threats that are considered in trusted computing; we then provide a summary of past and current developments in trustworthy computing technologies for *unconstrained systems*—it is our belief that a proper understanding of past works, even if they are not directly applicable to constrained systems, is important for informing designs for more constrained execution environments such as smartphones, tablets,

and other IoT devices; next, we provide a summary of past and current developments in trustworthy computing technologies for the *constrained systems* in which we are primarily interested; and last, we conclude by discussing some open problems or problems for which there are only limited solutions.

2 BACKGROUND

In this section we provide background information on computer architecture and terminology used in this paper. We also review concepts from trusted computing as well as attacker and threat models relevant to the motivation for this paper.

2.1 Computer architecture background

In this paper we use nomenclature that is common in computer architecture, though a deep knowledge in this area is not required for the level at which we discuss trusted-computing solutions. While our interest is primarily in mobile-computing architectures, many concepts from desktop computing platforms carry over; mobile platforms, however, have less physical space for hardware components and a much more limited energy budget. As a result, mobile and wearable platforms often include a subset of the hardware of traditional computing platforms or they use miniaturized designs that are more suitable for mobile and wearable platforms.

We acknowledge that many computer architecture terms are used interchangeably or ambiguously which, more often than not, leads to confusion. To this end, we include Figure 1 which illustrates the major components of a computer. In the most basic sense, a computer can be viewed as a device consisting of three fundamental pieces: processors to interpret and execute instructions (Figure 1, top right), different types of storage (fast/slow) to store data and instructions (Figure 1, upper and lower left), and I/O modules for transferring data to and from the outside world (Figure 1, lower right), all connected via various buses. Furthermore, we define terms that we use throughout this paper below. Figure 1 and the provided definitions should provide a sufficient mental model of relevant architectural components and relationships between components, with the understanding that modern designs combine various components into fewer *dies* and/or *packages* (defined in Section 2.1.1) for reasons described below.

2.1.1 Foundations. If we use the CPU as an example, the term ***die*** refers to a single, continuous piece of semiconductor material (usually silicon) that contains the transistors that make up the CPU. The CPU is an example of an ***integrated circuit (IC)***, where an IC in general is nothing more than a set of electronic circuits on (integrated onto) a single ***chip***. For our purposes, the words "chip" and "die" can be used interchangeably.

The term ***package*** refers to the smallest physical parts sold to consumers. The package contains one or more dies, is made up of plastic or ceramic housing for the dies, and has gold-plated contacts on its exterior that match those on a motherboard. The package is the actual unit that is plugged into a CPU socket on a motherboard.

In the literature we review below we have noticed that the terms "die" and "chip" are used interchangeably; similarly, "chip" and "package" are used interchangeably. This is unfortunate since the terms "die" and "package" are not used interchangeably, leading to ambiguity when using the term "chip" in writing. It is usually possible to determine what an author means given more context around how they use the above mentioned terms. Thus, the context of use will determine the meaning of the term.

Note that none of these terms are exclusive to CPUs; dies, chips, and packages are meaningful terms with respect to the composition of other computer components as well.

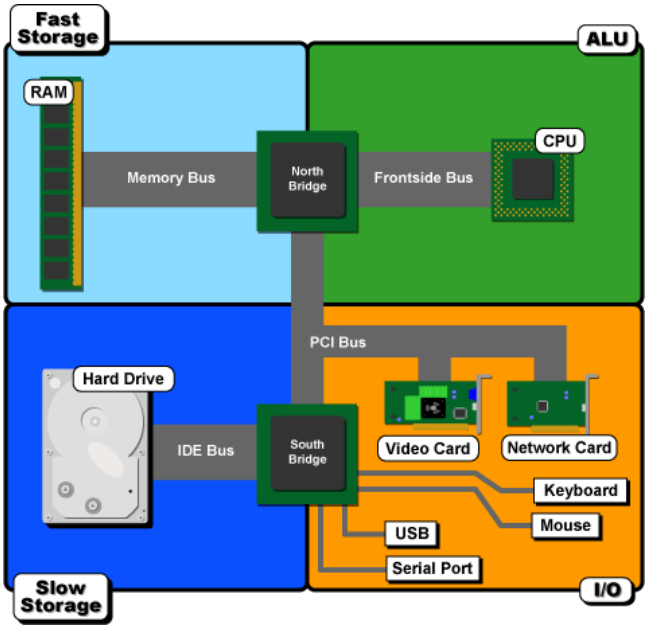

Figure 1: Traditional multi-chip computer architecture. The block labeled "CPU" depicts the CPU package which contains the processor cores. The CPU is physically connected to the other system components via buses which carry data; the actual interactions between the CPU and the rest of the system are handled by the northbridge and the southbridge. The northbridge (as rendered here) is an IC dedicated to managing the CPU's access to high-speed devices (e.g., RAM, video and network cards), whereas the southbridge is an IC that exists to manage the CPU's access to lower-speed devices (e.g., hard drives, human interface devices such as keyboards and mice) – the southbridge indirectly interacts with the CPU via the northbridge. Here, the CPU, northbridge, southbridge, RAM, and so forth, exist as separate chips that are all connected and work together to make up what we know as a modern computer. Figure from a description of a multi-chip system in response to a post on StackExchange ("What is a single-chip microcomputer?") [15].

2.1.2 Processors. Recent developments in computer architecture can especially lead to confusion concerning terms such as "CPU," "processor," "cores," "microprocessor," and "multi-core processor." We use the following definitions.

A ***central processing unit (CPU)*** is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control, and input/output (I/O) operations specified by some instruction; in the

most basic sense, a CPU *is a* processor, capable of performing a single task or running a single program at one time. Most modern CPUs are ***microprocessors***, meaning they are contained on a single ***integrated circuit (IC)*** chip.

Over the past couple of decades, CPU designs have changed in order to achieve better performance, lower power consumption, and so forth. Among those changes we've seen cache memory added to the CPU to improve execution speeds. Further, the parts of the CPU responsible for executing instructions were duplicated; components such as ALUs, fetch and decode hardware, the instruction pipeline, and cache memory were combined into what we now call ***cores***.

Thus, CPUs in general can be thought of as being made up of one or more cores. Since the terms "CPU" and "processor" can be used interchangeably, a CPU with more than one core has led to the nomenclature ***multi-core processor***. Since the advent of multi-core technology, such as dual-cores and quad-cores, the term ***processor*** has become context-sensitive, and is largely ambiguous when describing multi-core systems. A *processor* could describe either a single execution core (i.e., a single core within CPU) or a single physical multi-core chip (i.e., a CPU with multiple cores). The context of use will determine the meaning of the term.

2.1.3 Integrated Circuit (IC) designs. The term ***chipset*** broadly refers to any group of ICs that are designed to work together, and are usually marketed as a single product. This can lead to confusion, however, since the term *chipset* is most often used to refer to a specific pair of chips on the motherboard: the northbridge and the southbridge. It is increasingly common in modern systems for the ***northbridge***, which links the CPU to high-speed devices such as RAM, graphics, and network controllers, to be integrated into the main processor's die (i.e., the northbridge physically resides within the same chip as the CPU). There is also a ***southbridge***, which is generally responsible for managing access to lower-speed peripherals, [1] which may or may not be integrated into the processor package as well. In many modern chipsets, the southbridge contains integrated peripherals such as Ethernet, USB, and audio devices.

Variations in design are most often simply a reorganization of components in a multi-chip computer with several advantages. For instance, by integrating the memory controller that resides within the northbridge in a multi-chip architecture, the physical distance between the CPU and the memory controller is decreased, making memory access faster. Furthermore, if reasonable measures are made by the CPU manufacturer to protect the physical package in which the CPU resides, then any other components integrated into that package also benefit from increased security because their connections are hidden inside the package.

A ***System-on-Chip (SoC)*** is a design where all of the system components are packed into a single silicon chip. Along with a CPU, an SoC usually contains a graphics processor, memory and memory controller, USB controller, power management circuits, and wireless radios (e.g., Wi-Fi, 3G, 4G LTE). This integration is performed in a single manufacturing process; the die is then put into a package. Whereas a CPU cannot function without dozens of other chips, it's possible to build complete computers with just a single SoC. SoCs – which are common in mobile and wearable devices – are generally lower cost, lower energy, and have huge potential for improving security relative to multi-package designs.

A ***System-in-Package (SiP)*** is a further level of integration where multiple dies are integrated inside a single package. The system components (subsystems) are individual dies and they can be manufactured independently. They are assembled inside a single package by various techniques, e.g., vertical stacking or horizontal stacking. The SiP approach helps surpass the limits of the SoC designs. Benefits to SiP include user intellectual property (IP) integration, IP reuse, low design risk, reduced process complexity, low developmental cost, and shorter time-to-market. In short, SiP brings together ICs including SoCs and discrete components using lateral or vertical integration technologies.

2.1.4 Software privilege levels. Commodity CPUs implement several mechanisms to protect data and functionality from faults and malicious behavior; one mechanism of particular interest to us in this paper is *software privilege levels*. Commodity CPUs run software at different privilege levels. Each privilege level is strictly more powerful than the ones below it, so a piece of software can freely read and modify the code and data running at less privileged levels. Therefore, a software module can be compromised by any piece of software running at a higher privilege level. It follows that a software module implicitly trusts all the software running at more privileged levels, and a system's security analysis must take into account the software at all privilege levels.

In practice these sorts of hierarchical privilege levels are often called ***protection rings*** (or simply ***rings***), and they exist as mechanisms to protect data and functionality from faults and malicious behavior. A protection ring is one of two or more hierarchical levels or layers of privilege within the architecture of a computer system. This is generally hardware-enforced by some CPU architectures that provide different CPU modes at the hardware or microcode level. Rings are arranged in a hierarchy from most privileged (most trusted, usually numbered zero or with a negative number) to least privileged (least trusted, usually with the highest ring number, e.g., 3). On most operating systems, ring 0 is the level with the most privileges and interacts most directly with the physical hardware such as the CPU and memory. Programs such as web browsers running in higher numbered rings – usually ring 3 – and must request access to system resources such as the network – a resource restricted to lower-numbered rings.

2.2 Trusted Computing

Trusted Computing is by no means a new concept. "In the security engineering subspecialty of computer science, a ***trusted system*** is a system that is relied upon to a specified extent to enforce a specified security policy. As such, a trusted system is one whose failure may break a specified security policy" [20].

Some Trusted Computing designs (Figure 2) aim to enforce security policies by leveraging trusted hardware. The trusted hardware establishes a secure container, and a local or remote computation service can provision desirable computation and data into the secure container. The trusted hardware protects the data's confidentiality and integrity while the computation is being performed.

[1] Most peripherals are "lower-speed peripherals" when compared to the speeds at which, for example, memory access happens.

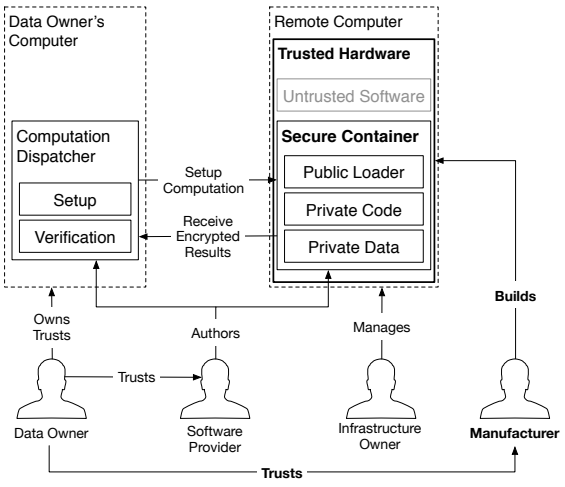

Figure 2: "Trusted Computing. The user trusts the manufacturer of a piece of hardware in the remote computer, and entrusts her data to a secure container hosted by the secure hardware." Figure and quote from *Intel SGX Explained* [2].

Trusted Computing is seeing increasing attention as a necessary technology in future computing platforms, specifically for consumer devices like PCs, laptops, tablets, smartphones, and IoT devices. In light of prominent modern computer security and privacy threats (including rootkits that compromise operating systems), researchers and system developers have explored various mechanisms to create a ***Trusted Execution Environment (TEE)*** that makes it possible to isolate a security sensitive application or service from a regular operating system. Also, a critical goal in securing systems is to reduce the attack surface by trusting only the system components and code that are *absolutely* necessary to implement the system's intended functionality; this effort is often referred to minimizing the Trusted Computing Base (TCB). Thus, we anticipate that TEE technologies will be used in the near future as they become common in commercial chips and concerns around data protection intensify, and that these technologies will provide security at an increasingly granular level (e.g., application-specific TEEs such as the application enclave approach used in Intel's SGX).

Although the concepts of trusted computing have been around for some time, the availability of this technology is relatively new and it is only recently that application developers have the tools to develop applications that leverage the power of hardware security features such as TEEs. A TEE is an environment with desirable security properties (SP) where code can be executed and data can be stored. Specifically, a TEE should provide the following properties, even in the presence of compromised system software; we adapt[2] and summarize them below according to the definitions provided by Vasudevan et al. [19].

(SP1) ***Isolated Execution*** provides secrecy and integrity of a process's code and data.

(SP2) ***Secure Storage*** provides secrecy, integrity, and freshness of a software module's data at rest.

(SP3) ***Local & Remote Attestation*** allows local and remote parties to verify that a particular message originated from a particular process – an attestation based on factors such as a measurement of the application code itself.

(SP4) ***Secure Provisioning*** is a mechanism to send data to a specific process from a trusted party, running on a specific device, while protecting the data's secrecy and integrity. The trusted party that provisions the data can be a remote party (e.g., a content provider that uses secure provisioning to construct a DRM[3] scheme) or a local party (e.g., an application that uses secure provisioning to migrate secrets to an updated version of the application).

(SP5) ***Trusted Inter-process Communication (IPC)*** protects authenticity, secrecy, and availability of communication between trusted processes.

(SP6) ***Trusted Path*** protects authenticity, secrecy, and availability of communication between a trusted process and a peripheral (e.g., keyboard, touch screen, or health device).

TEE technology promises to make it difficult to access code and data that is being executed and stored in a system, even in light of capable adversaries who have control over all untrusted software components, including the operating system and hypervisor.

2.3 Threats & attacker model

We provide an overview of common adversaries considered in trustworthy computing work; these adversaries have varying capabilities and motivations. In this discussion we attempt to capture the *general* assumptions that are made in the trusted computing systems and designs, as well as identify *general* capabilities and motivations of adversaries.

We denote static (machine) code and associated data and metadata as an ***application***. The ***Trusted Computing Base (TCB)*** of an application is the set of components (hardware and software) that must be secured to assure desired security properties over the application. Applications that are designed and believed to implement a particular function for the user (e.g., health data repository), and that use a set of trustworthy hardware and software components to protect its computation and data, are referred to as ***trusted applications***.

The works that we cover in the following sections of this paper attempt to implement systems that provide the desired security properties (SP1) – (SP6), though none of the systems provide all of the desired properties today. In trusted computing, it is assumed that any deployed software and hardware in the TCB, and any cryptographic mechanisms used, are secure and implemented correctly.

With respect to platforms and the sensitive data they store and process, it is generally the goal of an adversary to compromise applications running on a platform that are not owned by the adversary; here, to ***compromise*** applications means to obtain unauthorized access to their code or data, or to affect the underlying trusted

[2]While the list is largely a summary of the original list, we add *local attestation* (SP3) and *secure IPC* (SP5) to the list presented by Vasudevan et al. [19].

[3]Digital Rights Management (DRM) schemes are access-control technologies that can be used to restrict access to copyrighted works such as software and multimedia content.

components of the platform in such a way that violates the desired functional and security properties. To this end, it is generally assumed that the adversary has full control over the (untrusted) OS and other software running on the platform. In addition, it is not unreasonable to assume that the adversary also controls all communication with the platform and can eavesdrop, manipulate and intercept any network links or I/O channels.

The physical security of platforms often arises as a topic of interest in trustworthy computing. In trusted computing it is generally assumed that the high levels of integration achievable with modern IC fabrication processes render chip-level invasive attacks such as tampering, on-chip bus probing, extracting keys from on-chip memory, or fault injection out of scope for economically motivated attackers and that mitigations are in place against side-channel leakage through power, electromagnetic emissions or timing behavior.

When we discuss trusted computing solutions and their weaknesses or limitations it is useful to have a specific adversary and her respective capabilities in mind. For this we adapt a list of progressively capable adversaries (AD) from Mirzamohammadi et al. [12]:

- ***(AD1)*** The first attacker can only use the application API in the operating system, e.g., the Android API. This attacker can run native code but without root privileges.
- ***(AD2)*** The second attacker runs native code with root privileges in user space, but cannot run code with kernel privileges or secretly modify the system image (for future boots).
- ***(AD3)*** The third attacker leverages some vulnerabilities of the kernel to compromise it and hence can run code with kernel privileges.
- ***(AD4)*** The fourth attacker is a more advanced version of the third attacker that, after compromising the kernel, leverages some vulnerabilities of the hypervisor to compromise it and hence can run code with hypervisor privileges.
- ***(AD5)*** The fifth attacker is a root user in a system without any sort of verified boot feature, which would allow him to rewrite the kernel and hypervisor images (to be used after a reboot).
- ***(AD6)*** The sixth attacker has physical access to the device and can manipulate the hardware. This attacker is assumed to have the necessary knowledge and capabilities to carry out chip-level invasive attacks such as tampering, on-chip bus probing, extracting keys from on-chip memory, or fault injection.

Trusted computing solutions are almost always resilient to (AD1) and (AD2). Few solutions can protect against (AD6). Hence, the trusted-computing solutions of greatest interest are those that can protect against as many of (AD3) – (AD5) as possible.

3 TRUSTWORTHY COMPUTING ON UNCONSTRAINED SYSTEMS

To best understand the state of trustworthy computing on constrained mobile and wearable systems, we first touch on solutions that are relevant to computing systems that are less constrained – such as those designed for PCs and servers. Of particular interest are hardware-based solutions that offer security properties relevant to our goals and the promise of being implemented as efficiently as possible; specifically, we look at Hardware-assisted Isolated Execution Environments (HIEEs). HIEEs provide isolated execution, sometimes referred to as a TEE, for code execution even on a compromised system. Note that while the terms HIEEs and TEEs are sometimes used interchangeably, they are not the same in all cases; a TEE can be enforced in software, and not all HIEEs are designed for security.

In the remainder of this section we briefly describe relevant projects and build atop an organization of these types of works presented in an SoK paper on HIEEs [22], and an informative white paper that reviews Intel's SGX technology in great detail along with other related work [2].

3.1 Legacy solutions

We begin by reviewing some of the earliest work in HIEEs.

3.1.1 System Management Mode (SMM). System Management Mode (SMM) is a mode of execution similar to Real and Protected modes available on x86 platforms. It provides a hardware-assisted isolated execution environment for implementing platform-specific system control functions such as power management. SMM is triggered by asserting the System Management Interrupt (SMI) pin on the CPU. It is initialized by the Basic Input/Output System (BIOS). The BIOS loads the SMI handler into SMRAM (dedicated RAM in main memory for SMM) at boot time. The SMI handler has unrestricted access to the physical address space and can run privileged instructions; for this reason, SMM is often referred to as ring -2 (pronounced "ring negative-two").

3.1.2 Dynamic Root of Trust for Measurements (DRTM). DTRM is an alternate to Static Root of Trust Measurements, which allows the root of trust measurement to be initialized at any point. DTRM was introduced to the TPM v1.2 specification in 2005.

Two well-known implementations of DTRM are (a) Intel's Trusted eXecution Technology (TXT), which implements a trusted way to load and execute system software (e.g., OS or VMM); and (b) AMD's Secure Virtual Machine (SecVM), which implements new CPU instructions to enter/exit a secure environment for code execution. Intel's TXT and AMD's SecVM are similar and are both hardware-assisted isolated execution environments used for running security-sensitive tasks. The drawback to these solutions, however, is that they introduce significant performance overhead due to the expensive CPU instructions (e.g., `SENTER`, `SKINIT`) that control the transition between secure and non-secure environments.

3.2 Recent solutions

Next, we review popular developments from the last 10-15 years that are still in use today.

3.2.1 Intel Management Engine (ME) & AMD Platform Security Processor (PSP). Intel ME and AMD PSP (and AMD System Management Unit (SMU)) are similar solutions. This design consists of embedding a micro-computer (i.e., co-processor) into the main processor. This coprocessor is commonly integrated into the northbridge, which is commonly integrated into the main processor package. This design creates a completely isolated environment for code execution and data storage (i.e., a TEE). The ME, PSP, SMU solutions are interesting since the embedded computer really is a

computer that contains its own dedicated processor, internal Static Random-Access Memory (SRAM), Read-Only Memory (ROM), a cryptography engine, Direct Memory Access (DMA) engine, Host-Embedded Communication Interface (HECI) engine, a timer, and other I/O devices. With these resources the Intel ME, for example, can execute instructions on its own processor; it has code and data caches to reduce the number of accesses to the internal SRAM; it uses its own internal SRAM is to store the firmware code and runtime data; it is capable of using its DMA and HECI engine to access the main memory of the computer; and it can run Intel security applications (e.g., Enhanced Privacy Identification [5], Identity Protection Technology [6]).

Boot code stored in internal ROM is used as the root of trust for these embedded devices. The boot code loads and runs code from external flash memory (usually accessed over SPI); flash devices are typically "locked" by Original Equipment Manufacturers (OEMs) to prevent (malicious) modifications, but researchers have shown that it is possible to modify this code. Thus, while this approach is common, it is not without flaws.

3.2.2 ARM TrustZone (TZ). The ARM TrustZone (TZ) technology [8] technically falls into the domain of "recent solutions;" TrustZone technology, however, is primarily aimed at ARM's mobile processors. Please refer to Section 4 below for more information on TrustZone.

3.3 Latest solutions

To conclude our summary of Trusted Computing on unconstrained systems we review promising efforts from the last few years.

3.3.1 Intel Software Guard Extensions (SGX). Announced in 2013, SGX [11] is a set of CPU instructions and memory access mechanisms added to Intel Architecture (IA) processors. These extensions allow an application to instantiate a protected container referred to as an ***enclave***. An enclave could be used as a TEE, which provides confidentiality and integrity even without trusting the BIOS, firmware, hypervisors, or operating systems.

This solution is considered to be the "next generation of TXT" and has aroused a lot of attention recently. Not everyone, however, is convinced that SGX is the future of trusted computing. Costan et al. [2] examine SGX in great detail and identify many potential concerns with the technology; in response, they propose Sanctum [3].

3.3.2 Sanctum. Sanctum [3] offers the same promise as SGX, namely strong provable isolation of software modules running concurrently and sharing resources, but protects against an important class of additional software attacks that infer private information from a program's memory-access patterns. The authors of Sanctum claim to "follow a principled approach to eliminating entire attack surfaces through isolation, rather than plugging attack-specific privacy leaks" and that "strong software isolation is achievable with a surprisingly small set of minimally invasive hardware changes, and a very reasonable overhead."

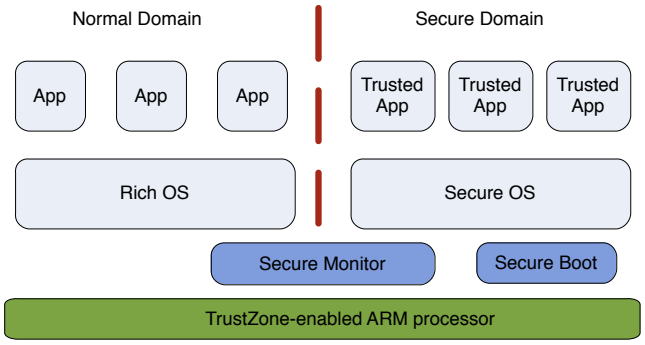

Figure 3: TrustZone architecture. Figure from *TrustICE* [18].

4 TRUSTWORTHY COMPUTING ON CONSTRAINED SYSTEMS

The work reviewed in Section 3 is helpful for trying to understand various approaches that have been considered when trying to realize trustworthy computing on platforms with few limitations. Keeping these approaches in mind, we now turn our attention to the state of trustworthy computing on constrained mobile and wearable systems.

4.1 ARM TrustZone

We begin by briefly discussing the ARM TrustZone (TZ) security technology [8]. In the text that follows we introduce some technical background on TrustZone, then review trustworthy-computing solutions built atop TrustZone. Figure 3 presents a high-level overview of the ARM TrustZone architecture, and Figure 4 illustrates an ARM-based smartphone SoC design based on TrustZone.

TrustZone is a hardware feature that creates an isolated execution environment, similar to other hardware isolation technologies. Namely, TrustZone provides security extensions for hardware components including the CPU, memory, and peripherals. The processor on a TrustZone-enabled ARM platform has two security modes: the "secure world" mode (i.e., a TEE) and "normal world" mode. Each processor mode has its own memory access region and privilege. Code running in normal world cannot access memory in secure world, but secure world code can access normal world memory. A *secure bit* (also known as the NS bit) in the Secure Configuration Register (SCR) is used to identify the secure/normal worlds; it can only be modified in the secure world. An interface known as the "Monitor Mode" (which technically resides in the secure world domain) is the gate keeper between normal and secure worlds, managing transitions between the worlds. The normal world invokes a Secure Monitor Call (`SMC`) to enter the monitor mode, which can modify the NS bit to switch into the secure world.

In Figure 4, the red IP blocks are TrustZone-aware. The red connections ignore the TrustZone secure bit in the bus address.

We now review trusted-computing solutions built atop TrustZone.

4.1.1 Sentry. Projects such as the Sentry [1] system are great examples of interesting work that grows out of developers having access to trustworthy computing technologies. Sentry is a system

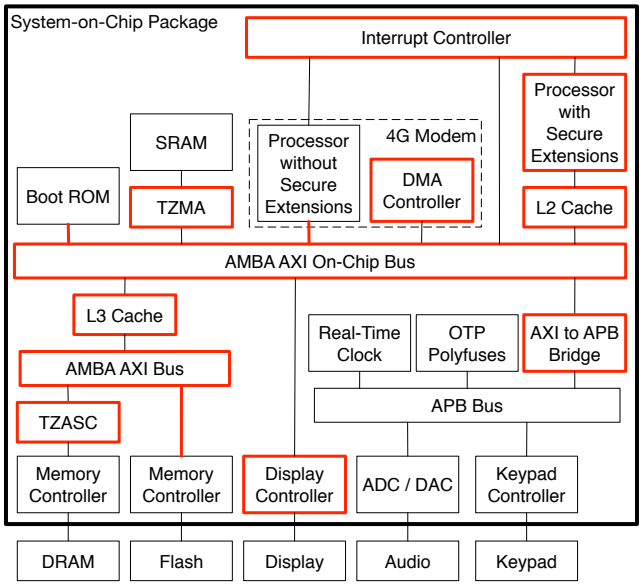

Figure 4: Smartphone SoC design based on TrustZone. The red IP blocks are TrustZone-aware. Figure from *Intel SGX Explained* [2].

that protects against DRAM attacks by leveraging on-SoC storage mechanisms originally intended for realtime predictable performance. Sentry can bootstrap additional secure storage by safely encrypting regions of memory much larger than the capacity of the ARM SoC. Sentry allows applications and OS components to store their code and data on the System-on-Chip (SoC) rather than in DRAM, thus enabling protections for applications and OS subsystems from memory attacks.

4.1.2 TrustICE. TrustICE [18] is a TrustZone-based isolation framework that creates isolated computing environments (ICEs) in the normal world. The authors of TrustICE anticipate that as more secure code is designated to run in the secure world, the attack surface of the secure domain will increase along with the size of secure code, and it will be an arduous process to negotiate with OEMs to get new secure code installed.

TrustICE securely isolates the secure code in an ICE from an untrusted Rich OS in the normal domain. The TCB of TrustICE remains small and unchanged regardless of the amount of secure code being protected. Their prototype shows that the switching time between an ICE and the Rich OS is less than 12 ms. Also, TrustICE proposed the use of LEDs to protect against spoofing attacks that “pretend" to switch between normal/secure world; i.e., LEDs are actuated in a meaningful way to indicate to the user that the system has really switched from one world to the other.

4.2 Flicker

Flicker [10] is an infrastructure for executing security-sensitive code in complete isolation while trusting as few as 250 lines of additional code. Flicker can also provide meaningful, fine-grained attestation of the code executed (as well as its inputs and outputs) to a remote party. Flicker guarantees these properties even if the BIOS, OS and DMA-enabled devices are all malicious. Flicker leverages new commodity processors from AMD and Intel and does not require a new OS or VMM. The authors of Flicker demonstrate a full implementation on an AMD platform.

4.3 TrustVisor

TrustVisor [9] is a special-purpose hypervisor that provides code integrity as well as data integrity and secrecy for selected portions of an application. TrustVisor achieves a high level of security, first because it can protect sensitive code at a fine granularity, and second because it has a small code base (only around 6K lines of code), which makes verification feasible. TrustVisor can also attest the existence of isolated execution to an external entity. In their work, the authors of TrustVisor observe less than 7% overhead on the legacy OS and its applications when protecting security-sensitive code blocks.

4.4 Self-Protecting Modules (SPM)

Strackx et al. propose self-protecting modules (SPM) [16], a design for a generic and lightweight hardware mechanism that can support an efficient implementation of isolation for several subsystems that share the same processor and memory space.

4.5 SMART

The Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust (SMART) [4] is an efficient (lightweight and low cost) and secure approach for establishing a dynamic root of trust in a remote embedded device. SMART is primarily intended for low-end micro-controller units (MCU) that lack specialized memory management or protection features. The authors of SMART demonstrate that a simple measurement routine in ROM with exclusive access to a protected secret key can provide remote attestation and trusted execution. The implementation of SMART requires minimal changes to existing MCUs (while providing concrete security guarantees) and imposes few limitations on adversarial capabilities when arguing for the quality of security provided by SMART. Their work shows that SMART implementations require only a few changes to memory bus access logic.

4.6 Sancus

Sancus [14] proposes additional CPU instructions that can be used to set up trusted software modules at runtime. For this purpose, they implement multiple memory-protection regions, each containing a code and data section. An extended processor instruction set enables dynamic measurement and loading of code into protected regions to query the protection status of modules and request tokens for authenticated communication between processes.

4.7 TrustLite

TrustLite [7] is a security architecture for flexible, hardware-enforced isolation of software modules. The main contribution behind TrustLite is an execution-aware memory protection unit with a secure exception handler that protects the state of “tasks." The developers

of TrustLite acknowledged that tiny devices cannot afford sophisticated hardware security mechanisms, and therefore new hardware protection mechanisms were (and are) needed to provide the required resilience and dependency at low cost. This work is particularly necessary as tiny devices are increasingly embedded in complex control infrastructures, medical support systems, and health and wellness consumer products; our dependency on these tiny devices, as well as our willingness to allow them to collect copious amounts of personal data, has made these devices a popular target of attack. TrustLite includes mechanisms for secure exception handling and communication between protected modules, enabling seamless interoperability with untrusted operating systems and tasks. TrustLite scales from providing a simple protected firmware runtime to advanced functionality such as attestation and trusted execution of user space tasks. The authors demonstrate their design using an FPGA prototype playing the role of a low-cost embedded system.

5 DISCUSSION

From the work that we survey in this paper we observe that solutions tend to follow one of two approaches: (1) isolation or (2) re-engineering shared resources; we describe these categories – and the challenges that are present in each approach – below. To conclude our discussion, we also address open challenges.

5.1 Isolation

Generally speaking, isolation-based security achieves security through (complete) physical separation from the application processor. This sort of isolation is realized through the inclusion of a trusted co-processor that executes security-sensitive tasks using its own dedicated resources such as SRAM and caches. The co-processor may have access to a shared bus, allowing it to interact with software running on the application processor. To ensure protection from untrusted software, custom bus logic is implemented to prevent the application processor from accessing privileged resources.

This approach seems to only be used in the context of the unconstrained trusted-computing solutions we describe in Section 3; this may be due to the expensive[4] cryptographic operations that security-oriented co-processors perform; or it may be possible due to the larger area available inside of processor packages to include an additional secure co-processor.

5.2 Re-engineering shared resources

Security through re-engineering shared resources seems to be a more frequently pursued approach to providing trustworthy computing on mobile and wearable devices, according to the work we survey. This approach relies on hardware modifications that can logically partition a system's resources, allowing both trusted and untrusted software to share processors, memory, and so forth. This approach is desirable because it builds security features into hardware that is common in mobile and wearable systems. Other benefits include keeping manufacturing costs low, little reliance on energy-consuming cryptographic operations to realize and enforce security, and so forth.

Drawbacks to this approach include the complexities of managing access of software in different *security domains*. For instance, this approach may require at least the following considerations:

System Bus modifications. Memory accesses can be performed from different security domains. Thus, the system bus address lines may need to be extended – as has been done to the AMBA AXI system bus for ARM TrustZone-based SoCs – to contain contextual information about the security domain in which memory accesses occur. For instance, in TrustZone-based systems, "the address in each bus access executed by a core reflects the world in which the core is currently executing" [2].

Secure boot scheme. The trustworthiness of the system is ultimately rooted in how the system boots. Initially, a system should boot into the highest-privilege security domain so that it can load and execute the boot code that measures, loads, and executes subsequent software (e.g., non-secure domain bootloaders, operating systems). This secure boot scheme can be realized by having a first-stage bootloader verify a signature in the second-stage bootloader against a public key whose cryptographic hash is burned into on-chip One-Time Programmable (OTP) fuses; the first-stage bootloader and the hash burned into the OTP fuses constitutes the system's root of trust.

Context switching between security domains. There should exist some trusted component to handle how the system context-switches between security domains. For instance, a solution should implement a "monitor" that controls context switching between security domains; the monitor is all-powerful in these designs, having access to resources across all security domains. The monitor can also be used to implement secure IPC between the security domains (e.g., between the normal world and secure world in TrustZone).

Instruction set modifications. Ideally untrusted software should have a way to invoke the secure monitor so it can initiate secure IPC or use some service provided by software running in a higher-security domain (e.g., software and platform attestation). For this purpose, new CPU instructions can be added that cause code running in less-secure domains to jump to (invoke) the monitor; this prevents direct access to higher-security domains.

Introducing or modifying CPU instructions for the aforementioned reasons requires additional considerations. For instance, it is important that each (logical) execution core maintains information about the security context in which it is executing, as well as maintain separate address translation units (e.g., for the secure and normal worlds). Furthermore, this can require, for example, extending physical addresses in page-table entries to include security context information; in this case, one must protect the integrity of page tables for different security domains. Regarding memory, these designs must also consider implications for shared caches, SRAM, DRAM, flash, etc., and their respective memory controllers.

Threat models. Mobile and wearable devices present unique challenges with respect to threats. They are often carried or worn, and are easily lost or stolen. They are often cheap, which means they are likely to have little or no countermeasures for physical attacks.

Generally, threat models in trusted-computing trust the processor package. Unconstrained systems can include tamper-resistant

[4]Expensive in terms of processor cycles and energy consumption.

hardware and software, but these solutions are often too expensive or bulky for mobile and wearable devices.

5.3 Open problems

Trusted computing for mobile and wearable systems is clearly on the mind of many hardware vendors and security researchers. Even with all of the existing work that has been done to date, however, there remain open problems or at least areas where improvement is greatly needed; we briefly discuss some of these below.

Trusted Switching Path. The HIEE-based systems we discuss above generally assume attackers have ring 0, and maybe even ring -1, privileges. Given that level of privilege, attackers can intercept the switching from the normal environment to TEE and provide a fake switching process to deceive users. Or attackers can perform a Denial of Service (DoS) attack against a system by simply disabling the switching (since they have ring 0 privilege or below). These attacks are challenging to overcome because trusted-computing solutions are primarily concerned with ensuring the desired security goals hold over sensitive user data even if the system has been compromised.

Trusted I/O. TEE technology, especially in the form of HIEE-based TEE technology, provides hope that applications can perform sensitive computations within a secure container that protects sensitive code and data. Unfortunately, ensuring that sensitive code and data can be delivered to and from a secure container has proven to be a more challenging problem. As in the rest of the trusted-computing space, existing solutions are highly specific to a particular architecture, hypervisor, and/or operating system.

We believe the trusted computing community is in need of a trusted I/O solution that can be broadly relevant, regardless of the TEE solution that is used by some platform. In keeping with the TEEs we review above, any solution claiming to provide trustworthy I/O (i.e., a trusted path from peripherals to trusted software) should ensure — at a minimum — the confidentiality and integrity of I/O data even when running on a system in which the OS or hypervisor has been compromised.

Trust rooted in manufacturers. One trust relationship that is difficult to avoid is that of the user of a device trusting the hardware vendor(s) of a device. Human user trust is inevitably rooted in the hardware vendors; many hardware-based solutions are “black boxes" and there is no way to verify the trustworthiness of their implementations or manufacturing processes. For example, the Intel ME is largely a mysterious black box that only the hardware vendor really understands. The closed nature of some of these devices raise questions about how to reliably evaluate the trustworthiness of these (often mysterious) hardware security technologies.

6 CONCLUSION

It is clear from the research and industry efforts reviewed in this paper that realizing more trustworthy computing on mobile and wearable systems is necessary but challenging. In this paper we review the most relevant systems and architectures that have been proposed as solutions for realizing more trustworthy computing on mobile and wearable devices. We also discuss open challenges and potential areas of future work.

By surveying legacy and current state-of-the-art trusted computing systems, we hope our summaries and observations provide helpful insights into the design of future systems.

REFERENCES

[1] Patrick Colp, Jiawen Zhang, James Gleeson, Sahil Suneja, Eyal de Lara, Himanshu Raj, Stefan Saroiu, and Alec Wolman. Protecting data on smartphones and tablets from memory attacks. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 177–189. ACM, March 2015. DOI 10.1145/2694344.2694380.

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

[3] Victor Costan, Ilia Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *Proceedings of the USENIX Security Symposium*, pages 857–874, August 2016. Online at https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan.

[4] Karim Eldefrawy, Aurélien Francillon, Daniele Perito, and Gene Tsudik. SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust. In *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*, pages 1–15, February 2012. Online at http://www.eurecom.fr/publication/3536.

[5] Intel. Enhanced Privacy ID. https://software.intel.com/en-us/node/702985, March 2017.

[6] Intel. Identity Protection Technology. http://www.intel.com/content/www/us/en/architecture-and-technology/identity-protection/identity-protection-technology-general.html, March 2017.

[7] Patrick Koeberl, Steffen Schulz, Ahmad R. Sadeghi, and Vijay Varadharajan. TrustLite: A Security Architecture for Tiny Embedded Devices. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, pages 1–14. ACM, April 2014. DOI 10.1145/2592798.2592824.

[8] ARM Limited. ARM Security Technology - Building a Secure System using TrustZone Technology, 2009.

[9] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. TrustVisor: Efficient TCB reduction and attestation. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 143–158. IEEE, May 2010. DOI 10.1109/sp.2010.17.

[10] Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. Flicker: an execution infrastructure for TCB minimization. *SIGOPS Operating Systems Review*, 42(4):315–328, April 2008. DOI 10.1145/1352592.1352625.

[11] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*. ACM, June 2013. DOI 10.1145/2487726.2488368.

[12] Saeed Mirzamohammadi and Ardalan A. Sani. Viola: Trustworthy sensor notifications for enhanced privacy on mobile systems. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 263–276. ACM, June 2016. DOI 10.1145/2906388.2906391.

[13] Ildar Muslukhov, Yazan Boshmaf, Cynthia Kuo, Jonathan Lester, and Konstantin Beznosov. Understanding users' requirements for data protection in smartphones. In *IEEE International Conference on Data Engineering Workshops*, pages 228–235. IEEE, April 2012. DOI 10.1109/icdew.2012.83.

[14] Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony V. Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, and Frank Piessens. Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In *Proceedings of the USENIX Security Symposium*, pages 479–498, August 2013. Online at https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/noorman.

[15] StackExchange. Multi-chip Architecture. https://i.stack.imgur.com/3HIfQ.png, February 2017.

[16] Raoul Strackx, Frank Piessens, and Bart Preneel. Efficient isolation of trusted subsystems in embedded systems. In *Proceedings of the International Conference on Security and Privacy in Communication Systems (SecureComm)*, pages 344–361. Springer, 2010. DOI 10.1007/978-3-642-16161-2_20.

[17] He Sun, Kun Sun, Yuewu Wang, and Jiwu Jing. TrustOTP: Transforming smartphones into secure one-time password tokens. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 976–988. ACM, October 2015. DOI 10.1145/2810103.2813692.

[18] He Sun, Kun Sun, Yuewu Wang, Jiwu Jing, and Haining Wang. TrustICE: Hardware-assisted isolated computing environments on mobile devices. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 367–378. IEEE, June 2015. DOI 10.1109/dsn.2015.11.

[19] Amit Vasudevan, Emmanuel Owusu, Zongwei Zhou, James Newsome, and Jonathan M. McCune. Trustworthy execution on mobile devices: What security properties can my mobile platform give me? In *Trust and Trustworthy Computing*, volume 7344 of *LNCS*, pages 159–178. Springer, 2012. DOI 10.1007/978-3-642-30921-2_10.

[20] Wikipedia. Trusted system. https://en.wikipedia.org/w/index.php?title=Trusted_system, March 2017.

[21] Fengwei Zhang, Kevin Leach, Haining Wang, and Angelos Stavrou. TrustLogin: Securing password-login on commodity operating systems. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIA CCS)*, pages 333–344. ACM, 2015. DOI 10.1145/2714576.2714614.

[22] Fengwei Zhang and Hongwei Zhang. SoK: A study of using hardware-assisted isolated execution environments for security. In *Proceedings of the International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*. ACM, June 2016. DOI 10.1145/2948618.2948621.