Trusted Computing Platforms: Design and Applications

TRUSTED COMPUTING PLATFORMS: DESIGN AND APPLICATIONS

SEAN W. SMITH
Department of Computer Science
Dartmouth College
Hanover, New Hampshire USA

Springer

eBook ISBN: 0-387-23917-0
Print ISBN: 0-387-23916-2

Contents

List of Figures

List of Tables

Preface

We stand an exciting time in computer science. The long history of specialized research building and using security-enhanced hardware is now merging with mainstream computing platforms; what happens next is not certain but is bound to be interesting. This book tries to provide a roadmap.

A fundamental aspect of the current and emerging information infrastructure is distribution: multiple parties participate in this computation, and each may have different interests and motivations. Examining security in these distributed settings thus requires examining which platform is doing what computation—and which platforms a party must trust, to provide certain properties despite certain types of adversarial action, if that party is to have trust in overall computation. Securing distributed computation thus requires considering the trustworthiness of individual platforms, from the differing points of view of the different parties involved. We must also consider whether the various parties in fact trust this platform—and if they should, how it is that they know they should.

The foundation of computing is hardware: the actual platform—gates and wires—that stores and processes the bits. It is common practice to consider the standard computational resources—e.g., memory and CPU power—a platform can bring to a computational problem. In some settings, it is even common to think of how properties of the platform may contribute to more intangible overarching goals of a computation, such as fault tolerance. Eventually, we may start trying to change the building blocks–the fundamental hardware—in order to better suit the problem we are trying to solve.

Combining these two threads—the importance of trustworthiness in these Byzantine distributed settings, with the hardware foundations of computing platforms—gives rise to a number of questions. What are the right trustworthiness properties we need for individual platforms? What approaches can we try in the hardware and higher-level architectures to achieve these properties? Can we usefully exploit these trustworthiness properties in computing platforms for broader application security?

With the current wave of commercial and academic trusted computing architectures, these questions are timely. However, with a much longer history of secure coprocessing, secure boot, and other experimentation, these questions are not completely new. In this book, we will examine this big picture. We look at the depth of the field: what a trusted computing platform might provide, how one might build one, and what one might be done with one afterward. However, we also look at the depth of history: how these ideas have evolved and played out over the years, over a number of different real platforms—and how this evolution continues today.

I was drawn to this topic in part because I had the chance to help do some of the work that shaped this field. Along the way, I've enjoyed the privilege of working with a number of excellent researchers. Some of the work in this book was reported earlier in my papers [SW99, SPW98, Smi02, Smi01, MSWM03, Smi03, Smi04], as documented in the "Further Reading" sections. Some of my other papers expand on related topics [DPSL99, DLP+ 01, SA98, SPWA99, JSM01, IS03b, SS01, IS03a, MSW+ 04, MSMW03, IS04b, IS04a].

Acknowledgments

Besides being a technical monograph, this book also represents a personal research journey stretching over a decade.

I am not sure how to begin acknowledging all the friends and colleagues who assisted with this journey. To start with: I am grateful to Doug Tygar and Bennet Yee, for planting these seeds during my time at CMU and continuing with friendship and suggestions since; to Gary Christoph and Vance Faber at Los Alamos, for encouraging this work during my time there; and to Elaine Palmer at IBM Watson, whose drive saw the defunct Citadel project turn into a thriving research and product development effort. Steve Weingart and Vernon Austel deserve particular thanks for their collaborations with security architecture and formal modeling, respectively. Thanks are also due to the rest of the Watson team, including Dave Baukus, Ran Canetti, Suresh Chari, Joan Dyer, Bob Gezelter, Juan Gonzalez, Michel Hack, Jeff Kravitz, Mark Lindemann, Joe McArthur, Dennis Nagel, Ron Perez, Pankaj Rohatgi, Dave Safford, and David Toll; to the 4758 development teams in Vimercate, Charlotte, Poughkeepsie, and Lexington; and to Mike Matyas.

Since I left IBM, this journey has been helped by fruitful discussions with many colleagues, including Denise Anthony, Charles Antonelli, Dmitri Asonov, Dan Boneh, Ryan Cathecart, Dave Challener, Srini Devadas, John Erickson, Ed Feustel, Chris Hawblitzel, Peter Honeyman, Cynthia Irvine, Nao Itoi, Ruby Lee, Neal McBurnett, Dave Nicol, Adrian Perrig, Dawn Song, and Leendert van Doorn. In academia, research requires buying equipment and plane tickets and paying students; these tasks were supported in part by the Mellon Foundation, the NSF (CCR-0209144), AT&T/Internet2 and the Office for Domestic Preparedness, Department of Homeland Security (2000-DT-CX-K001).

Here at Dartmouth, the journey continued with the research efforts of students including Alex Barsamian, Mike Engle, Meredith Frost, Alex Iliev, Shan Jiang, Evan Knop, Rich MacDonald, John Marchesini, Kazuhiro Minami, Mindy Periera, Eric Smith, Josh Stabiner, Omen Wild, and Ling Yan. My colleagues in the Dartmouth PKI Lab and the Department of Computer Science also provided invaluable helpful discussion, and coffee too.

Dartmouth students Meredith Frost, Alex Iliev, John Marchesini, and Scout Sinclair provided even more assistance by reading and commenting on early versions of this manuscript.

Finally, I am grateful for the support and continual patience of my family.

Sean Smith
Hanover, New Hampshire
October 2004

Chapter 1

INTRODUCTION

Many scenarios in modern computing give rise to a common problem: why should Alice trust computation that's occurring at Bob's machine? (The computer security field likes to talk about "Alice" and "Bob" and protection against an "adversary" with certain abilities.) What if Bob, or someone who has access to his machine, is the adversary?

In recent years, industrial efforts—such as the *Trusted Computing Platform Association (TCPA)* (now reformed as the *Trusted Computing Group, TCG*), Microsoft's *Palladium* (now the *Next Generation Computing Base, NGSCB*), and Intel's *LaGrande*—have advanced the notion of a "trusted computing platform." Through a conspiracy of hardware and software magic, these platforms attempt to solve this remote trust problem, for various types of adversaries. Current discussions focus mostly on snapshots of the evolving TCPA/TCG specification, speculation about future designs, and idealogical opinions about potential social implications. However, these current efforts are just points on a larger continuum, which ranges from earlier work on *secure coprocessor* design and applications, through TCPA/TCG, to recent academic developments. Without wading through stacks of theses and research literature, the general computer science reader cannot see this big picture.

The goal of this book is to fill this gap. We will survey the long history of amplifying small amounts of hardware security into broader system security. We will start with early prototypes and proposed applications. We will examine the theory, design, implementation of the IBM 4758 secure coprocessor platform, and discuss real case study applications that exploit the unique capabilities of this platform. We will discuss how these foundations grow into the newer industrial designs such as TCPA/TCG, as well as alternate architectures this newer hardware can enable. We will then close with an examination of more recent cutting-edge experimental work.

1.1 Trust and Computing

We should probably first begin with some definitions. This book uses the term *trusted computing platform (TCP)* in its title and throughout the text, because that is the term the community has come to use for this family of devices.

This terminology is a bit unfortunate. "*Trusted* computing platform" implies that some party trusts the platform in question. This assertion says nothing about who that party is, whether the platform is worthy of that party's trust, and on what basis that party chooses to trust it. (Indeed, some wags describe "trusted computing" as computing which circumstances force one to trust, like it or not.)

In contrast, the devices we consider involve trust on several levels. The devices are, to some extent, *worthy of trust:* physical protections and other techniques protect them against at least some types of malicious actions by an adversary with direct physical access. A relying party, usually remote, has the ability to *choose to trust* that the computation on the device is authentic, and has not been subverted. Furthermore, typically, the relying party does not make this decision blindly; the device architecture provides some means to *communicate* its trustworthiness. (I like to use the term "trustable" for these latter two concepts.)

1.2 Instantiations

Many types of devices either fit this definition of "trusted computing platform," or have sufficient overlap that we must consider their contribution to the family's lineage.

We now survey the principal classes.

Secure Coprocessors. Probably the purest example of a trusted computing platform is a *secure coprocessor.*

In computing systems, a generic *coprocessor* is a separate, subordinate unit that offloads certain types of tasks from the main processing unit. In PC-class systems, one often encounters *floating-point coprocessors* to speed mathematical computation. In contrast to these, a *secure coprocessor* is a separate processing unit that offloads security-sensitive computations from the main processing unit in a computing system. In hindsight, the use of the word "secure" in this term is a bit of a misnomer. Introductory lectures in computer security often rail against using the word "secure" in the absence of parameters such as "achieving what goal" and "against whom."

From the earliest days, secure coprocessors were envisioned as a tool to achieve certain properties of computation and storage, despite the actions of *local* adversaries—such as the operator of the computer system, and the computation running on the main processing unit. (Dave Safford and I used the term *root secure* for this property [SS01].) The key issue in secure coprocessors is

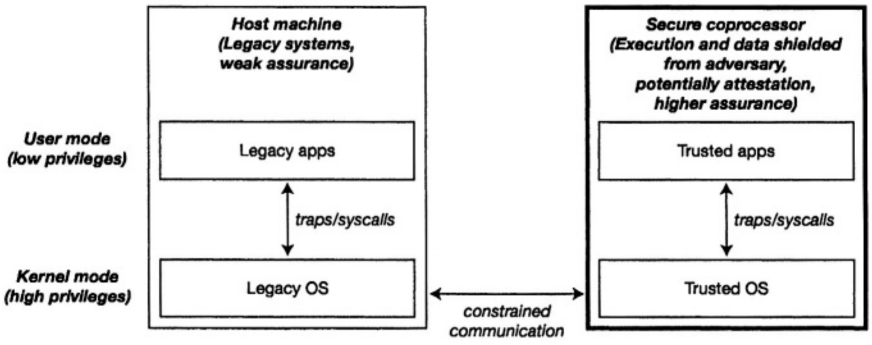

Figure 1.1. In the secure coprocessor model, a separate coprocessor provides increased protections against the adversary. Sensitive applications can be housed inside this protected coprocessor; other helper code executing inside the coprocessor may enhance overall system and application security through careful participation with execution on the main host.

not *security* per se, but is rather the establishment of a trust environment *distinct* from the main platform. Properly designed applications running on this computing system can then use this distinct environment to achieve security properties that cannot otherwise be easily obtained. Figure 1.1 sketches this approach.

Cryptographic Accelerators. Deployers of intensively cryptographic computation (such as e-commerce servers and banking systems) sometimes feel that general-purpose machines are unsuitable for cryptography. The modular mathematics central to many modern cryptosystems (such as RSA, DSA, and Diffie-Hellman) becomes significantly slower once the modulus size exceeds the machine's native word size; datapaths necessary for fast symmetric cryptography may not exist; special-purpose functionality, like a hardware source of random bits, may not be easily available; and the deployer may already have a better use for the machine's resources.

Reasons such as these gave rise to *cryptographic accelerators:* special-purpose hardware to off-load cryptographic operations from the main computing engines. Cryptographic accelerators range from single-chip coprocessors to more complex stand-alone modules. They began to house sensitive keys, to incorporate features such as physical security (to protect these keys) and programmability, (to permit the addition of site-specific computation). Consequently, cryptographic accelerators can begin to to look like trusted computing platforms.

Personal Tokens. The notion of a *personal token*—special hardware a user carries to enable authentication, cryptographic operations, or other services—

also overlaps with the notion of a trusted computing platform. Personal tokens require memory and typically host computation. Depending on the application, they also require some degree of physical security. For one example, physical security might help prevent a thief (or malicious user) from being able to learn enough from a token to create a useful forgery. Physical security might also help to prevent a malicious user from being able to amplify his or her privileges by modifying token state. Form factors can include smart cards, USB keyfobs, "Dallas buttons" (dime-sized packages from Dallas Semiconductor), and PCMCIA/PC cards.

However, because personal tokens typically are mass-produced, carried by users, and serve as a small part of a larger system, their design tradeoffs typically differ from higher-end trusted computing platforms. Mass production may require lower cost. Transport by users may require that the device withstand more extreme environmental stresses. Use by users may require displays and keypads, and may require explicit consideration of usability and HCISEC considerations. Use within a larger system may permit moving physical security to another part of the system; for example, most current credit cards have no protections on their sensitive data—the numbers and expiration date—but the credit card system is still somehow solvent.

Dongles. Another variation of a trusted computing platform is the *dongle*—a term typically denoting a small device, attached to a general purpose machine, that a software vendor provides to ensure the user abides by licensing agreements. Typically, the idea here is to prevent copying the software. The main software runs on the general purpose machine (which presumably is at the mercy of the malicious user); this software then interacts with the dongle in such a way that (the vendor hopes) the software cannot run correctly without the dongle's response, but the user cannot reverse-engineer the dongle's action, even after observing the interaction.

Dongles typically require some degree of physical security, since easy duplication would enable easy piracy.

Trusted Platform Modules. Current industry efforts center on a *trusted platform module (TPM):* an independent chip, mounted on the motherboard, that participates and (hopefully) increases the security of computation within the machine. TPMs create new engineering challenges. They have the advantage of potentially securing the entire general purpose machine, thus overcoming the CPU and memory limits of smaller, special-purpose devices; they also let the trusted computing platform more easily accommodate legacy architectures and software. On the other hand, providing effective security for an entire system by physically protecting the TPM and leaving the CPU and memory exposed is a delicate matter; furthermore, the goal of adding a TPM to every commodity machine may require lower cost, and lower physical security.

Hardened CPUs. Some recent academic efforts seek instead to add physical security and some additional functionality to the CPU itself. Like the industrial TPM approach, this approach can potentially transform an entire general purpose machine into a trusted computing platform. By merging the armored engine with the main processing site, this approach may yield an easier design problem than the TPM approach; however, by requiring modifications to the CPU, this approach may also make it harder to accommodate legacy architectures.

Security Appliances. Above, we talked about types of devices one can add to a general-purpose machine to augment security-related processing. Other types of such specialized *security appliances* exist. For example, some commercial firms market hardened *network interface cards (NICs)* that provide transparent encryption and communication policy between otherwise unmodified machines in an enterprise. For another example, PC-based postal meters can also require hardened postal appliances at the server end—since a malicious meter vendor might otherwise have motive and ability to sell postage to his or her customers without reimbursing the postal service. Essentially, we might consider such appliances as a type of trusted computing platform pre-bundled with a particular application.

Crossing Boundaries. However, as with many partitions of things in the real world, the dividing line between these classes is not always clear. The IBM4758 secure coprocessor platform drew on research into anti-piracy dongles, but IBM marketed it as a box that, with a free software application, the customer could turn into a cryptographic accelerator. (Nevertheless, many hardened postal appliances are just 4758s with special application software.) Some senior security researchers assert that secure coprocessing experiments on earlier generation IBM cryptographic accelerators predate the anti-piracy work. Some engineers have observed that current TPM chips are essentially smart card chips, repackaged. Other engineers assert that anything can be transformed into a PCMCIA token with enough investment; secure NICs already are.

1.3 Design and Applications

Many questions play into how to build and use a trusted computing platform.

Threat Model. Who are the adversaries? What access do they have to the computation? How much resources and time are they willing to expend? Are there easier ways to achieve their goal than compromising a platform? Will compromise of a few platforms enable systematic compromise of many more? Might the adversary be at the manufacturer site, or the software developer site, or along the shipping channel?

Deployment Model. A trusted computing platform needs to make its way from its manufacturer to the user site; the application software also needs to make its way from its developer to the trusted computing platform. The paths and players involved in deployment create design issues. Is the device a generic platform, or a specific appliance? Does the software developer also ship hardware? If software installation happens at the user site, how does a remote party determine that the executing software is trustworthy? Is the device intended to support multiple applications, perhaps mutually hostile?

More issues arise once the platform is actually configured and deployed. Should the platform support code maintenance? Can the platform be re-used for another application? Can an installation of an application be migrated, with state, to another trusted computing platform? Can physical protections be turned on and off—and if so, what does this mean for the threat model? Can we assume that deployed platforms will be audited?

Architecture. How do we balance all these issues, while building a platform that actually does computation?

Basic performance resources comprise one set of issues. How much power does the CPU have? Does the platform have cryptographic engines or network connections? More power makes life easier for the application developer; however, more power means more heat, potentially complicating the physical security design. User interfaces raise similar tradeoffs.

Memory raises additional questions. Besides the raw sizes, we also need to consider the division between types, such as between volatile and non-volatile, and between what's inside the physical protection barrier, and what lies outside (perhaps accessible to an adversary). Can a careful physical attack preserve the contents of non-volatile memory? What can an adversary achieve by observing or manipulating external memory?

Security design choices also interact with architecture choices. For example, if an on-the-motherboard secure chip is intended to cooperate with the rest of the machine to form a trusted platform, then the architecture needs to reflect the mechanics of that cooperation. If a general-purpose trusted platform is intended to persist as "secure" despite malicious applications, then we may require additional hardware protection beyond the traditional MMU. If we intend the platform to destroy all sensitive state upon tamper, then we need to be sure that all components with sensitive state can actually be zeroized quickly.

Applications. All these issues then play into the design and deployment of actual applications.

Is the trusted platform secure enough for the environment in which it must function? Is it economically feasible and sufficiently robust? Can we fit the application inside the platform, or must we partition it? Can we assume that a platform will not be compromised, or should we design the application with the idea that an individual compromise is unlikely but possible? How does the application perform? Is the codebase large enough to make updates and bug fixes likely—and if so, how does this mesh with the platform's code architecture? Will the application require the use of heterogeneous trusted computing platforms—and if so, how can it tell the difference? Finally, why should anyone believe the application—or the trusted computing platform underneath it—actually works as advertised?

1.4 Progression

In what follows, we will begin by laying out the big picture. Modern computing raises scenarios where parties need to trust properties of remote computation (Chapter 2); however, securing computation against an adversary with close contact is challenging (Chapter 3). Early experiments laid the groundwork (Chapter 4) for the principal commercial trusted computing efforts:

- High-end secure coprocessors—such as the IBM 4758—built on this foundation to address these trust problems (Chapter 5 through Chapter 9).
- The newer TCPA/TCG hardware extends this work, but enables a different approach (Chapter 10 through Chapter 11).

Looming industrial efforts—such as the not-yet-deployed NGSCB/Palladium and LaGrande architectures—as well as ongoing academic research explore different hardware and software directions (Chapter 12).

Chapter 2

MOTIVATING SCENARIOS

In this chapter, we try to set the stage for our exploration of trusted computing platforms. In Section 2.1, we consider the adversary, what abilities and access he or she has, and what defensive properties a trusted computing platform might provide. In Section 2.2, we examine some basic usage scenarios in which these properties of a TCP can help secure distributed computations. Section 2.3 presents some example real-world applications that instantiate these scenarios. Section 2.4 describes some basic ways a TCP can be positioned within a distributed application, and whose interests it can protect; Section 2.5 provides some real-world examples. Finally, although this book is not about ideology, the idealogical debate about the potential of industrial trusted computing efforts is part of the picture; Section 2.6 surveys these issues.

2.1 Properties

In its classic conception, a trusted computing platform such as a secure coprocessor is an armored box that does two things:

- It protects some designated data storage area against an adversary with certain types of direct physical access.
- It endows code executing on the platform with the ability to prove that it is running within an appropriate untampered environment.

What types of attacks the platform defends against, and exactly how code does this attestation, are issues for the platform architect.

In an informal mental model of a distributed computing application, we map computation and data to platforms distributed throughout physical space. Users (including potential adversaries) are also distributed throughout this space. Co-location of a user and a platform gives that user certain types of access to