
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

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

©2005 Springer Science + Business Media, Inc.

Print ©2005 Springer Science + Business Media, Inc.
Boston

All rights reserved

No part of this eBook may be reproduced or transmitted in any form or by any means, electronic, mechanical, recording, or otherwise, without written consent from the Publisher

Created in the United States of America

Visit Springer's eBookstore at:
and the Springer Global Website Online at:

<http://ebooks.springerlink.com>
<http://www.springeronline.com>

Contents

List of Figures	xiii
List of Tables	xv
Preface	xvii
Acknowledgments	xix
1. INTRODUCTION	1
1.1 Trust and Computing	2
1.2 Instantiations	2
1.3 Design and Applications	5
1.4 Progression	7
2. MOTIVATING SCENARIOS	9
2.1 Properties	9
2.2 Basic Usage	10
2.3 Examples of Basic Usage	12
2.4 Position and Interests	14
2.5 Examples of Positioning	15
2.6 The Ideological Debate	18
2.7 Further Reading	18
3. ATTACKS	19
3.1 Physical Attack	21
3.1.1 No Armor	22
3.1.2 Single Chip Devices	23
3.1.3 Multi-chip Devices	23
3.2 Software Attacks	24
3.2.1 Buffer Overflow	25

3.2.2	Unexpected Input	25
3.2.3	Interpretation Mismatches	26
3.2.4	Time-of-check vs Time-of-use	27
3.2.5	Atomicity	28
3.2.6	Design Flaws	29
3.3	Side-channel Analysis	30
3.3.1	Timing Attacks	30
3.3.2	Power Attacks	33
3.3.3	Other Avenues	34
3.4	Undocumented Functionality	35
3.4.1	Example: Microcontroller Memory	36
3.4.2	Example: FLASH Memory	37
3.4.3	Example: CPU Privileges	38
3.5	Erasing Data	38
3.6	System Context	39
3.7	Defensive Strategy	41
3.7.1	Tamper Evidence	41
3.7.2	Tamper Resistance	41
3.7.3	Tamper Detection	41
3.7.4	Tamper Response	42
3.7.5	Operating Envelope	42
3.8	Further Reading	42
4.	FOUNDATIONS	43
4.1	Applications and Integration	43
4.1.1	Kent	44
4.1.2	Abyss	44
4.1.3	Citadel	45
4.1.4	Dyad	46
4.2	Architectures	48
4.2.1	Physical Security	48
4.2.2	Hardware and Software	49
4.3	Bootting	50
4.4	The Defense Community	52
4.5	Further Reading	52

5. DESIGN CHALLENGES	55
5.1 Context	55
5.1.1 Personal	55
5.1.2 Commercial	56
5.2 Obstacles	57
5.2.1 Hardware	57
5.2.2 Software	59
5.3 Requirements	63
5.3.1 Commercial Requirements	63
5.3.2 Security Requirements	64
5.3.3 Authenticated Execution	66
5.4 Technology Decisions	67
5.5 Further Reading	71
6. PLATFORM ARCHITECTURE	73
6.1 Overview	73
6.1.1 Security Architecture	74
6.2 Erasing Secrets	75
6.2.1 Penetration Resistance and Detection	76
6.2.2 Tamper Response	76
6.2.3 Other Physical Attacks	77
6.3 The Source of Secrets	78
6.3.1 Factory Initialization	78
6.3.2 Field Operations	79
6.3.3 Trusting the Manufacturer	81
6.4 Software Threats	81
6.4.1 Software Threat Model	82
6.4.2 Hardware Access Locks	82
6.4.3 Privacy and Integrity of Secrets	85
6.5 Code Integrity	85
6.5.1 Loading and Cryptography	86
6.5.2 Protection against Malice	86
6.5.3 Protection against Reburn Failure	87
6.5.4 Protection against Storage Errors	88
6.5.5 Secure Bootstrapping	89
6.6 Code Loading	90
6.6.1 Authorities	91
6.6.2 Authenticating the Authorities	92

6.6.3	Ownership	92
6.6.4	Ordinary Loading	93
6.6.5	Emergency Loading	96
6.7	Putting it All Together	97
6.8	What's Next	99
6.9	Further Reading	99
7.	OUTBOUND AUTHENTICATION	101
7.1	Problem	101
7.1.1	The Basic Problem	102
7.1.2	Authentication Approach	102
7.1.3	User and Developer Scenarios	103
7.1.4	On-Platform Entities	104
7.1.5	Secret Retention	104
7.1.6	Authentication Scenarios	105
7.1.7	Internal Certification	107
7.2	Theory	108
7.2.1	What the Entity Says	109
7.2.2	What the Relying Party Concludes	109
7.2.3	Dependency	110
7.2.4	Soundness	111
7.2.5	Completeness	112
7.2.6	Achieving Both Soundness and Completeness	112
7.2.7	Design Implications	113
7.3	Design and Implementation	114
7.3.1	Layer Separation	115
7.3.2	The Code-Loading Code	115
7.3.3	The OA Manager	116
7.3.4	Naming	119
7.3.5	Summary	119
7.3.6	Implementation	120
7.4	Further Reading	121
8.	VALIDATION	123
8.1	The Validation Process	124
8.1.1	Evolution	124
8.1.2	FIPS 140-1	125
8.1.3	The Process	126
8.2	Validation Strategy	126

8.3	Formalizing Security Properties	129
8.3.1	Building Blocks	130
8.3.2	Easy Invariants	131
8.3.3	Controlling Code	131
8.3.4	Keeping Secrets	132
8.4	Formal Verification	134
8.5	Other Validation Tasks	136
8.6	Reflection	138
8.7	Further Reading	139
9.	APPLICATION CASE STUDIES	141
9.1	Basic Building Blocks	141
9.2	Hardened Web Servers	142
9.2.1	The Problem	142
9.2.2	Using a TCP	144
9.2.3	Implementation Experience	149
9.3	Rights Management for Big Brother's Computer	152
9.3.1	The Problem	152
9.3.2	Using a TCP	153
9.3.3	Implementation Experience	154
9.4	Private Information	155
9.4.1	The Problem	155
9.4.2	Using a TCP: Initial View	157
9.4.3	Implementation Experience	158
9.4.4	Using Oblivious Circuits	160
9.4.5	Reducing TCP Memory Requirements	163
9.4.6	Adding the Ability to Update	165
9.5	Other Projects	167
9.5.1	Postal Meters	167
9.5.2	Kerberos KDC	167
9.5.3	Mobile Agents	167
9.5.4	Auctions	167
9.5.5	Marianas	168
9.5.6	Trusted S/MIME Gateways	169
9.5.7	Grid Tools	169
9.6	Lessons Learned	170
9.7	Further Reading	171

10. TCPA/TCG	173
10.1 Basic Structure	175
10.2 Outbound Authentication	178
10.3 Physical Attacks	179
10.4 Applications	180
10.5 Experimentation	180
10.6 TPM 1.2 Changes	181
10.7 Further Reading	181
11. EXPERIMENTING WITH TCPA/TCG	183
11.1 Desired Properties	184
11.2 The Lifetime Mismatch	184
11.3 Architecture	185
11.4 Implementation Experience	189
11.5 Application: Hardened Apache	190
11.6 Application: OpenCA	191
11.7 Application: Compartmented Attestation	193
11.8 Further Reading	194
12. NEW HORIZONS	195
12.1 Privilege Architectures	195
12.2 Hardware Research	197
12.2.1 XOM	197
12.2.2 MIT AEGIS	198
12.2.3 Cerium	199
12.2.4 Virtual Secure Coprocessing	199
12.2.5 Virtual Machine Monitors	199
12.2.6 Others	200
12.3 Software Research	201
12.3.1 Software-based Attestation	202
12.3.2 Hiding in Plain Sight	202
12.4 Current Industrial Platforms	203
12.4.1 Crypto Coprocessors and Tokens	203
12.4.2 Execution Protection	203
12.4.3 Capability-based Machines	204
12.5 Looming Industry Platforms	204
12.5.1 LaGrande	204

<i>Contents</i>	xi
12.5.2 TrustZone	206
12.5.3 NGSCB	206
12.6 Secure Coprocessing Revisited	208
12.7 Further Reading	209
Glossary	211
References	221
About the Author	235
Index	237

List of Figures

1.1	Secure coprocessing application structure	3
5.1	The basic hardware architecture.	68
5.2	The basic software architecture.	69
6.1	The authority tree.	91
6.2	Contents of a layer.	93
6.3	Statespace for a layer.	93
6.4	Ordinary code-load command.	94
6.5	Countersignatures.	95
6.6	Authorization of code-load commands.	95
6.7	An emergency code-load command.	97
7.1	<i>Epochs</i> and <i>configurations</i> .	105
7.2	Replacing untrusted software with trusted software creates problems.	106
7.3	Replacing trusted software with untrusted software creates problems.	107
7.4	Sketch of the proof of our outbound authentication theorem.	113
7.5	When the code-loading layer updates itself.	116
7.6	Having the certifier outlive a code change creates problems.	117
7.7	Having the certifier outlive the certified can cause problems.	118
7.8	We regenerate certifier key pairs with each code change.	118
8.1	The formal verification process, as we envisioned it before we started.	128
8.2	The “safe control” invariant.	132
8.3	The “safe zeroization” invariant.	133
8.4	The formal verification process, as it actually happened.	135

8.5	Validation documentation tools.	136
9.1	Revising the SSL handshake to use a trusted co-server.	150
9.2	A switch	160
9.3	Oblivious shuffles with a Benes network	162
11.1	Flow of protection and trust in our TCPA/TCG-based platform.	188
12.1	The standard CPU privilege structure.	196
12.2	The revised CPU privilege structure.	197

List of Tables

6.1	Hardware ratchets protect secrets.	85
6.2	Hardware ratchets protect code.	87
9.1	Performance of an SSL server with a trusted co-server.	151
9.2	Slowdown caused by adding a trusted co-server.	151

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 Cathcart, Dave Challener, Srinu 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 ideological 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

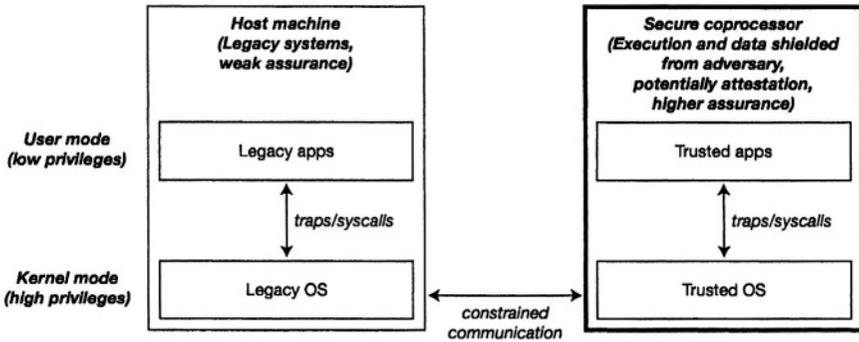


Figure 1.1. In the secure coprocessor model, a separate coprocessor provides increased protections against the adversary. Sensitive applications can be housed inside this protected coprocessor; other helper code executing inside the coprocessor may enhance overall system and application security through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