GH41316   *CI-0494899

CISTI   ICIST

Document Delivery Service   Service de fourniture de Documents
in partnership with the **Canadian Agriculture Library**   en collaboration avec **la Bibliothèque canadienne de l'agri**

THIS IS NOT AN INVOICE / CECI N'EST PAS UNE FACTURE

MARION WELLS
LIBRARY DOCUMENT DELIVERY SVC
PRINCETON UNIVERSITY
ONE WASHINGTON RD
PRINCETON, NJ 08544
UNITED STATES

ORDER NUMBER:	**CI-04948994-3**
Account Number:	FGH41316
Delivery Mode:	ARI
Delivery Address:	128.112.207.55
Submitted:	2004/09/17 11:23:57
Received:	2004/09/17 11:23:57
Printed:	2004/09/17 13:14:30

rect   Conference   WWW Catalogue   UNITED ST
.00
      50.00

nt Number:	TN155196
le:	**FINANCIAL CRYPTOGRAPHY : SECOND INTERNATIONAL CONFERENCE, FC '98,**
	ANGUILLA, BRITISH WEST INDIES, FEBRUARY 23-25, 1998 : PROCEEDINGS
hor:	FC '98 (1998 : ANGUILLA)
Ref. No.:	IRN18454719
/Issue:	1465
e:	1998
jes:	73-89
cle Title:	USING A HIGH-PERFORMANCE, PROGRAMMABLE SECURE COPROCESSOR
cle Author:	SMITH, S. W.
port Number:	ISBN3540649514 IRN18454719 LNCS 1465
es Title:	LECTURE NOTES IN COMPUTER SCIENCE, 1465
olisher:	SPRINGER,
rmation Source:	INNOPAC

timated cost for this 17 page document: $11.4 document supply fee +
2 copyright = $43.4

Using a High-Performance, Programmable Secure Coprocessor

Sean W. Smith, Elaine R. Palmer, and Steve Weingart

Secure Systems and Smart Cards
IBM T.J. Watson Research Center
P.O Box 704, Yorktown Heights NY 10598   USA
{sws, erpalmer, c1shw}@us.ibm.com

Abstract. Unsecure computational environments threaten many financial cryptography implementations, and other sensitive computation. High-performance secure coprocessors can address these threats. However, using this technology for practical security solutions requires overcoming numerous technical and business obstacles. These obstacles motivate building a high-performance secure coprocessor that balances security with easy third-party programmability—but these obstacles also provide many design challenges. This paper discusses some of issues we faced when attempting to build such a device.

1   Introduction

Using secure coprocessors to build practical e-commerce applications requires practical secure coprocessors.

Previous work [6,13,18,20,21] explores the feasibility of building general-purpose, computationally powerful secure coprocessors that carry out computation without observation or interference by a well-funded adversary with direct physical access. Subsequent work [11,14,16,22,23] demonstrates that such hardware could be the foundation for secure applications in electronic commerce—including nearly every aspect of financial cryptography.

However, bringing any one application vision to reality requires addressing the problem of building and distributing the hardware. Simultaneously enabling the practical realization of a broad family of applications by building general-purpose, customizable hardware introduces the additional trust and security problems of multiple, mutually suspicious authorities, and the inevitability of flaws in complex software.

This paper discusses some of our research on these issues, as part of our efforts to produce such a tool as a mass-produced, commercial product. [10, 15] Sect. 2 discusses the fundamental threat that unsecure and untrusted computing platforms present to financial cryptography implementations, where the stakes may be considerable but where trust and privacy are also significant issues. Section 3 discusses how high-performance secure coprocessors would comprise a near-universal tool for solving these problems. Section 4 discusses the

practical barriers that must be overcome for such a tool to be widely used. Section 5 quickly reviews the design decisions we made in building our architecture. Section 6 discusses the flexibility this architecture brings to development and deployment of security solutions.

2   The Threat of Unsecure/Untrusted Environments

2.1   Security

It is probably indisputable that security is critical for many financial and e-commerce systems. However, an often overlooked aspect of these systems are the machines and data storage environments themselves. No one would argue that it is inadvisable to leave a stack of $100 bills lying on a counter while one walks away to get a deposit form; the time-honored method of storing money in a mattress is certainly less secure than keeping that money in a vault. In the same way, an unsecured computer is not a much better place to leave money than is a counter or a mattress. Data or algorithms can be modified—and control over the assets may be usurped or lost.

Because of this situation, many computational and storage operations must take place in an environment that is secure, in the sense that the parties who need to can trust that the operations satisfy some set of correctness properties, despite potential malice.

These properties may describe storage of critical data. For example:

- Has the private key been exposed?
- Did the quantity of stored funds spuriously change?

The properties also include correctness of the operation itself:

- Was the keypair generated randomly—or or was it predictable by an adversary?
- Were the funds transferred transactionally—or did a communication error result in creation of spurious funds?

Even harder to articulate is the correctness of the invocation of these operations on this data:

- Did the private key generate only the signatures its owner authorized?

2.2   Threats

However, some of the security threats most difficult to thwart stem from potential adversaries with access to the hardware or software that carries out a sensitive computation, and can benefit from modifying it—an exposure which the advent of e-commerce has only amplified.

This risk can have several manifestations, including physically exposed machines and circuitry, software flaws, dishonest employees and customers, and channels for backup and disaster recovery. We visit each in turn.

Exposed Machines. Physically exposed machines are particularly vulnerable to attack, either by mischievous vandals or serious attackers. Anyone with access to a machine can attempt to reboot it, alter its files, load rogue software, or steal the whole machine to hack away at it in private.

Recent business trends exacerbate this threat. For example, the finance industry is increasing its deployment of customer self-service systems via telephone, the Internet, and at physical locations beyond the confines and protection of the conventional service counter. These remote systems allow the business to expand service locations, extend service hours, and reduce costs. The remote systems are more vulnerable to attack than those safely locked within the four walls of the home office.

Exposed circuitry is also vulnerable to attack (e.g., [2,3]). A physical attacker can quite quickly and unobtrusively clip diagnostic equipment onto a printed circuit board to observe and record cryptographic keys as they leave main memory on the way to an encryption chip. Electrical engineering lab students at the University of Delft in the Netherlands routinely practice opening, reading, and reprogramming smart card chips.

Software Flaws. Software flaws, whether inadvertent or intentional, can also offer an avenue to compromise secure computations and storage.

Consider, for example, a bug in one application of a hypothetical multi-application smart card. Current smart card processors lack the hardware memory protection required to protect the memory space of one application from that of another or the operating system. An errant frequent shopper application on today's smart cards can inadvertently destroy (or increment!) the balance of an electronic purse sharing the same card.

Flaws in operating systems are also a threat. Typically, an operating system has unlimited privileges and access to the entire memory space of the system. But what if a bug in the software causes the operating system to overwrite and thus destroy the rewritable firmware of the machine? The result can be that the system is unable to boot or reload a new, corrected operating system.

Web connectivity, and shipping functionality with Java, introduce even more issues.

Another class of threats from software flaws arises from cryptography. Cryptography is at best a panacea if the implementation itself is somehow compromised. Yet in recent years we have seen both practical demonstrations of real vulnerabilities (e.g., badly chosen random keys in Kerberos [7] and Netscape [9]) as well as theoretical demonstrations of weaknesses in key generation (e.g., [17, 24]) and other aspects of cryptographic mathematics, such as differential fault analysis (e.g., [4,5]).

Dishonest Employees, Contractors, and Customers. Dishonest employees pose a rather vexing problem. As employees, they must have access to information and systems in order to carry out their jobs, but often that access cannot be finely controlled. For example, a computer system administrator (*sysadmin*) must have unrestricted access to a machine in order to perform software updates, system

backups, add new users, etc. Unfortunately, many systems do not have fine-grained access. The sysadmin can copy files to tapes either for legitimate backup purposes or to sell to the competition. Unencrypted financial data can be quite valuable, and is an easy target for employees who succumb to blackmail or temptation. Dishonest contractors or temporary employees pose an even greater threat, since they typically neither go through the vetting nor hold the stake that regular employees do.

Dishonest customers pose an interesting challenge to a merchant. Merchants cannot usually distinguish between an honest customer and a dishonest one, but must nevertheless treat all with courtesy, not suspicion. Furthermore, merchants exert little or no control over their customers and their customers' employees, except to deny a sale or discount (or to litigate). Merchants cannot fire a dishonest customer, and the dishonest customer can return again and again to cheat. New business practices allow customers to have access to product databases, online catalogues, custom pricing algorithms, and other information potentially valuable to one's competitors. How, then, can a merchant prevent a customer's disgruntled, overworked, underpaid purchasing agent from selling the merchant's confidential price list to the merchant's competitors?

Backup and Disaster Recovery. The explicit purpose of backup and disaster recovery service is to replicate mission critical data in sufficiently many places to recover from any single point of failure. But how does one protect numerous, widely-distributed copies of critical data from theft, modification, or misuse, particularly when the backup tapes are stored offsite in facilities managed by third-party vendors?

2.3   Hostile Environments

We reiterate that financially sensitive computations are piles of money, stacked in the platforms, networks, and disks involved, and those with access to this infrastructure have potential access to this money. In many electronic commerce applications, *everyone* with access to the infrastructure—including the *end user* himself—is potentially an adversary. (Who wouldn't like to have a bottomless electronic wallet?) The *entire environment* must be considered hostile.

3   Secure Coprocessors as a Potential Solution

3.1   The Technology

Section 2 outlined threats from adversaries tampering with computation or data storage. As a device that carries out computation despite attempts to tamper with it, a *secure coprocessor* provides a tool that helps systematically address these threats—and potentially enable new applications that are otherwise not feasible.

Figure 1 sketches a generic device. A physically secure boundary protects a CPU and memory. A subset of the memory is specified as *secure*. The physical
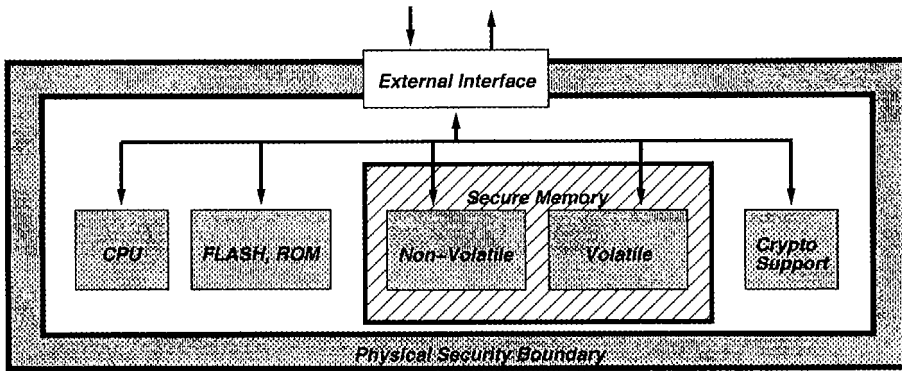
Fig. 1. A generic secure coprocessor. Tamper *zeroizes* the secure memory (or somehow renders it unavailable). As a consequence, with properly designed software, the device can provide privacy and/or integrity for data and/or operations.

security design aims to ensure that attempts to penetrate or tamper with the device will result in the contents of the secure memory being destroyed.

This secure memory enables secure applications. For example, putting a private key in this memory allows the device to act as a trusted witness, since a remote party can trust that a message correctly signed by this key must have been produced by an untampered device. Such messages might include assertions about data the device is storing, or about more complex computations it is carrying out. This foundation can extend to securing the entire on-device application; putting an entire program in secure memory might provide even more advantages. Yee's work [16,22,23] explores these issues; [14] provides a taxonomy of potential applications.

The effectiveness of a secure coprocessor as a tool for building secure applications depends on the effectiveness of its memory protections and the power of its computational engine. This previous research considered the implications of a high-end, high-performance device, which provides a general-purpose computing environment that withstands nearly all foreseeable physical and logical attacks. Such a device requires additional properties: it runs only the programs it is supposed to; it runs them unmolested; and one must be able to distinguish between a real program and a clever impersonator, even if the secure coprocessor is thousands of miles away.

Typically, secure coprocessors incorporate high-performance cryptography, but they need not be just fast crypto boxes—since computation besides cryptographic operations can go inside them.

3.2   High-End Secure Coprocessors Counteract Threats

By providing computational horsepower but resisting and responding to physical attacks, a high-end secure coprocessor is ideally suited for security in hostile environments:

- The coprocessor, in the hostile environment, can execute the part of the application that must be protected from adversaries.
- The device is sufficiently powerful for this computation to be fairly complex.
- An attacker who attempts to modify this computation will trigger tamper response.
- An attackers who disassembles a device will learn no secrets.

With proper software design, such a device provides a trusted witness, wherever trust is needed. As a result, an easily programmable, high-performance secure coprocessor is a tool can address the risks of Sect. 2.

Physically Exposed Machines and Circuitry. High-end secure coprocessors are designed to be placed in physically exposed environments. Vandals may attack at will, but will not discern secrets inside the tamper-responding coprocessor. Nor can they load rogue software into the coprocessor. The device will erase its secrets and shut itself down on any attempt to penetrate the tamper-responding enclosure, in order to learn secrets or modify execution.

Software Flaws. High-end secure coprocessors offer both hardware and software isolation of applications from each other, and of the operating system from applications. (Indeed, we found it necessary to protect regions of memory from even the operating system.) Thus, an application running inside a high-end secure coprocessor cannot inflict damage on one of its peers inside the coprocessor. Nor can the operating system damage the highly-critical microcode nor its protected storage inside the coprocessor. The threat of compromised cryptography can be addressed by hardware support for basic operations (such as a hardware source for randomness), and authentication of software configuration. Some approaches to high-end tamper protection can also shield against differential fault analysis.

Dishonest Employees, Contractors and Customers. High-end secure coprocessors can be used to control and audit the access rights of both employees and customers to sensitive data. Furthermore, because of the programmability of the coprocessor, access controls can be quite complex, for example, limited to certain hours of the day, days of the week, etc.

Backup and Disaster Recovery. High-end secure coprocessors enable high speed encryption (backup) of sensitive data before it is sent off to remote sites for backup. High speed decryption facilitates recovery of the data. Furthermore, the keys can be stored in numerous secure coprocessors spread about the world, and accessed only on appropriate conditions. Employees at the remote backup sites see only encrypted data; the plaintext emerges only during bona fide recovery.

3.3   The Family of Secure Coprocessors

Our research interests focus on building and using high-end, programmable secure coprocessors. However, devices meeting the generic pattern of Fig. 1 encompass a wide range of capabilities; various devices in the secure coprocessor family embody various trade-offs between security, computational and cryptographic power, and cost. We review the entire family.

Chip Card. The least powerful member of the secure coprocessor family is the *smart card*[1] (also known as *chip card*). A typical smart card of today includes a tiny 8-bit processor, 512 bytes RAM, 8K bytes ROM, 8K bytes EEPROM, and a hardware accelerator for cryptographic operations. Smart cards must be programmed at the factory, since their software is permanently burned in mask ROM when the chips are manufactured—typically in quantities of at least 10,000 units. Because the memory space is so limited, the software inside the smart card is written in hand-tuned machine code.

Smart cards offer some limited detection of physical attacks. However, the response to such attacks depends on whether the device is powered on or not—for example, the response when inserted into a smart card reader may substantially differ from the response when lying unpowered on an attacker's workbench. Without power, the device is unable to erase sensitive data stored in EEPROM. Even with power, erasure within a very small time window is unlikely or impossible.

Smart cards also offer only limited protection against logical attacks. The processors on current smart cards lack the combination of "supervisor state" with hardware protection of memory regions (including RAM), the cornerstone of preventing one application from accessing and modifying the data of another application on the card. For that reason, most smart card vendors carefully inspect and test all code that must co-exist on the same card (although recent efforts at JavaCard attempt to remedy this). Cryptographic throughput is severely limited by the serial interface into and out of the card, with a typical speed of 9600 bit/second.

Smart cards offer the undeniable advantages of being inexpensive, portable, and physically robust.

Personal Tokens. The personal token, such as a PCMCIA card or "smart button," is usually the next step up in crypto performance and capability. In some cases, these devices are little more than a repackaged chip card—except the larger packaging gives the advantages of greater I/O bandwidth and real estate for more hardware, such as more memory, a battery, or stronger protections. At the other end of the spectrum, a personal token could theoretically have many of the features and capabilities of crypto accelerators and/or high-end secure processors.

---

[1] Unfortunately, the term "smart card" is sometimes used for nearly everything in this family, and other devices besides.

Personal tokens are often tamper-resistant and in some cases have reasonable tamper evident and perhaps tamper-responsive features. (However, tamper evidence is of little value without a well planned audit policy.) The typical token will be able to perform user functions such as authentication, file encryption (albeit with low performance), and storage of secrets off of the host system. Most of the PCMCIA cards available are fixed function devices and are not programmable, though some offer some programming capabilities if the work is done by, or in cooperation with the manufacturer. (Again, recent efforts JavaCard attempt to remedy this for low-end tokens.)

Crypto Accelerators. The next device in the secure coprocessor family is the cryptographic accelerator. Such devices typically include a small microprocessor (e.g. the ARM7), a reasonable amount of memory (e.g. 2 megabytes EEPROM, 512K RAM) and chips to boost cryptographic performance. The primary purpose of these devices (e.g., [1]) is to accelerate cryptographic operations while offering some degree of physical protection of cryptographic keys. These devices are often tamper-resistant, but seldom tamper-responding. (For example, their external cases resist attack because they are difficult to pry open, but they do not erase sensitive data if they are opened.) They are not widely programmable by any customer, but are sometimes programmable under special contract by the manufacturer, given sufficient sales volumes.

High-End Secure Coprocessor. The most sophisticated device in the secure coprocessor family is a high-end secure coprocessor, with a powerful microprocessor (e.g. 486), megabytes of RAM and FLASH memory, chips to boost cryptographic performance, a time-of-day clock, and a hardware random number generator, all within a tamper-responding enclosure. Such a device should provide computationally verifiable untamperedness with a nearly open programming environment, but immediately erase sensitive data when under attack.

An example of such a high-end coprocessor is the recent IBM 4758, the product [10] which we explicitly designed to address the issues raised in this paper.

4   Obstacles to Practical Deployment

The feasibility of high-end secure coprocessors (e.g., [21]) and their application potential (e.g., [22,23]) had been well-established. However, realizing the vision of Sect. 3.1 and Sect. 3.2 required overcoming the significant obstacles that confront an organization wishing to develop and deploy real applications using this technology.

This section reviews these obstacles. Section 5 and Sect. 6 below discuss how we designed our coprocessor to try to eliminate them.

4.1   Hardware

We have repeated allusions to a "high-end secure coprocessor" that provides:

- *highly reliable* tamper-response
- *high-speed* cryptographic support
- a *powerful, general-purpose* computational environment

Until recently, this hardware was not available beyond prototype quantities.

Tamper-Response. Most discussions of secure hardware usually use phrases like "tamper-proof," "tamper-resistant," "tamper-evident" or "tamper-responsive." Any realistic assessment recognizes that "tamper-proof" hardware is unattainable; however, the solutions of Sect. 3.2 all assumed that the secure coprocessor somehow *responded* to tamper by *zeroizing* (or somehow rendering unrecoverable) some stored information.

But how should this happen?

- *Active* tamper-response relies on the device itself to detect tamper attempts and destroy its secrets. Active response can be computational—which requires that during tamper, the processor remain alive long enough to destroy the secrets—or depend instead on independent special-purpose circuitry that more quickly *crowbars* the memory.
- *Passive* techniques rely on physical or chemical hardness (and sometimes on explosives).

Passive protection is difficult to carry out effectively (witness the continued permeation of smart card technology) and difficult to apply to multi-chip modules. But on the other hand, using active protection requires recognizing that the device is only as secure as long as the necessary environment exists for the active protection to function. Minimally, this recognition requires grappling with some difficult issues:

- The continuous existence of this environment requires a continuous source of power.
- What exactly do we know about the device after an interval in which this environment fails to exist?
- Does the device always protect itself, or only between visits by a security officer?
- What should happen to a device that zeroizes its secrets?

Exactly how the zeroizable secrets should be stored raises additional design issues. For example, using Static RAM requires considering the *imprinting* effects of low-temperature and long-term storage.

Trusted I/O Path. Effectively using a trusted device in human-based applications often requires effective authentication of communications between a human and their trusted device. [8] A human-usable I/O path on the secure device itself makes these problems simpler—but although a nice abstraction, such a path can greatly compromise the physical security.

Hosts. Almost tautologically, a secure coprocessor requires a host system. Wide deployment of a secure coprocessor application requires considering the population of host machines:

- What physical interface should be used? How does this choice affect ease of installation, number of potential platforms, and performance of coprocessor? (For example, the PCMCIA, chip-card, and PCI-bus interfaces all give different answers to these questions.)
- What device drivers and other associated host-side software are required? How does this software get to the host? What possibilities exist for attacking the application by attacking the host-side software?

Cost and Durability. For an application to succeed, someone needs to create and distribute a population of secure coprocessors. This task requires balancing the cost of the device with its power and protections, as well as considering longer-term reliability issues. (Indeed, the often-lamented computational restrictions of chip cards are a consequence of the requirement to keep them highly robust to physical wear-and-tear.)

Depending on the tamper-protection methods used, additional environmental factors such as heat and radiation need to be considered.

Exportability. Another challenge facing any practical development and deployment of cryptographically powerful devices is compliance with the U.S. export laws.

4.2   Software

Development. A cornerstone of secure coprocessing applications is putting a substantial portion of application-specific computation into the secure device—not just using it for basic cryptographic operations. Developing and deploying secure coprocessing applications thus requires the ability to develop software for the device. This requirement leads to many challenges, even with current-generation low-end devices:

- Is development possible on a *small scale* with small numbers of devices—or must the application developer first convince a hardware manufacturer of the business case for thousands or millions of units?
- Is development possible *independent* of the hardware manufacturer—or must the application developer work closely and expose plans and code?
- Does a robust *programming environment* exist for the device, or must code be hand-tuned? What about debugging and testing?
- If independent development is possible, what *prevents* malicious or faulty software from *compromising* core device keys? Do these protections consist of verified hardware and software, or depend solely on complex software with a track record of flaws?

Installation. How does the deployer ensure that the potentially untrusted user, in a potentially hostile environment, ends up with an authentic, untampered device that is programmed with the right software?

Installing the software at the factory forces the application developer to have a substantial presence in the factory, and the factory to customize their processes to individual application developers. This approach may complicate small-scale development.

However, installing the software at any later point raises a number of additional issues.

- What about the security of the shipping channel? What if the device is modified between the time it leaves the factory and the time the software is installed?
- How does the device know what software to accept? (Accepting just anything opens the possibility of tamper via false software load.)
- Does a device carry a key whose exposure compromises that device, or other devices?
- Installing the software at the deployer's site forces the deployer to ship the hardware to the end-users.
- Installing the software at the end-user's site requires the need for security officers, or for the device itself to exert fairly robust control, authentication, and confirmation of software loads.
- With general-purpose programmable hardware intended for multiple deployers, installation after the factory needs to ensure that hardware loaded with one deployer's software cannot claim to be executing software from a different deployer.

The software installation process may also have unpleasant interactions with the desired security model. For example:

- If the application developer requires that their software itself be secret, but the hardware only provides a limited amount of tamper-protected storage, then the installation process must include some way of installing the software decryption key in that storage.

Most post-factory installation scenarios require that the devices leave the factory with some type of security/bootstrap code, which raises additional issues.

Software Maintenance. After installation, how does the deployer then proceed to securely carry out the maintenance and upgrades that such complex software inevitably requires?

- How does the device authenticate such requests? Must the deployer use an on-site "security officer," or can they use remote control? If the latter, how much interaction is required? Does the deployer have to undergo a lengthy handshake with each deployed device? Does the deployer need to maintain a database of device-specific records or secrets?

- How can participants in an application know for certain that an upgrade has occurred? (The purpose of the upgrade might be to eliminate a software vulnerability which an adversary has already used to explore the contents of privileged memory.)
- What should happen to stored data when software is upgraded? Not supporting "hot updates" is cleaner, but can greatly complicate the difficulty of performing updates.
- What atomicity does the device provide for software updates? Can failures (or malice) leave the device in a dangerous or inoperable state? What if the software that cryptographically verifies updates is itself being updated?

To avoid grappling with these issues, some deployers may choose simply to not allow updates. However, the decision certainly needs to be balanced against hardware expense and software complexity (hence likelihood of upgrade).

Multi-Party Issues. The foregoing discussion largely focused on a model where basic device hardware had one software component that needed to be installed and updated. In reality, this situation may be more complicated. Multiple software layers may lie beneath the application software:

- The presence of a device operating system (in order to make software development easier) raises the questions of when, where, and how the OS is installed and updated.
- The OS may come from an independent software developer, like the application does.
- The more tasks assigned to the basic bootstrap/configuration, the more likely this foundational software might also require update.

Multiple layers each controlled by a different authority makes the software installation and update problem even more interesting. For example, the ability to perform "hot-updates" potentially gives an OS vendor a backdoor into the application secrets.

The simple answer of "not allowing any OS updates" avoids these risks, but introduces the problem of what to do when a flaw is discovered in security-critical system software—especially if this software is too complex to have been formally verified.

Some scenarios may additionally require multiple sibling software components, at the same layer (although this flexibility must be balanced against the risks of potentially malicious sibling applications, the hardware expense and the sensitivity of the application).

Comparison to PCs. It might be enlightening to compare this situation with software development for ordinary, exposed machines, such as personal computers. For PCs, software developers do not need to build and distribute the computers themselves. Software developers never need meet or verify the identity of the user. Software developers do not need to worry about how or where the user obtained the machine; whether it is a genuine or modified machine, or

whether the software or its execution is being somehow modified. Furthermore, developers of the application software usually do not also have to develop and maintain the operating system or the ROM BIOS.

5   Building Technology to Overcome These Obstacles

The list of obstacles to deploying secure coprocessor applications naturally leads to design issues for those hoping to minimize these obstacles by building a *high-end, programmable* secure coprocessor. Although Sect. 4 presented a quick enumeration, we stress that the design choices often interact in subtle ways. For just one example, the business decisions to support remote update of potentially buggy supervisor-level software requires the ability to remotely authenticate that this repair took place, which in turn may require changing the hardware to provide a region of secure memory that is private even from a defective supervisor-level operating system.

Indeed, the issues in Sect. 4 arose from our group's long-term research into secure coprocessing technology, and more recent efforts to produce a commercially available, general-purpose device. Our design choices for this product were driven not just by academic analysis, but also by more practical factors such as expected business case for such devices, and "lessons learned" from previous IBM experience with similar technology. Some of these factors include:

- Software is less stable than hardware—especially if the time delay between manufacture and end-user installation is considerable.
- The complexity of manufacturing and maintenance support appears to increase exponentially with each shippable variation of a commercial product.
- No one wants to trust anyone else more than necessary.
- Expensive hardware must be repairable.

We decided on building a board-level coprocessor assembled, as much as possible, from existing commercial technology. For protection, we chose active tamper response (electrical, not computational). In an attempt to broaden the family of compatible hosts, we use a PCI interface and are developing host-side software for several popular operating systems. Our device supports separate bootstrap, operating system, and application layers, each potentially controllable by different authorities with minimal involvement by IBM. To simplify the production process and comply with export laws, all devices are shipped the same: with only the bootstrap layer. Software installation (and update) can occur at any point thereafter, including at the end-user site, via broadcast-style commands from remote authorities.

The device is shipped already initialized with device-generated secrets protected by the tamper-response circuitry, in order to defend itself. Each untampered device computationally establishes its untamperedness beginning with its creation at the factory, and continuing throughout shipment, software installation and updates—even if some of this software turns out to be defective or malicious.

Our device is not a portable user token (although we see no substantial engineering barriers to moving this to PCMCIA format). Because of our choice of active tamper response, factors such as low temperatures, x-rays, and bungled battery changes may all trigger zeroization—since otherwise, these are avenues for undetectable tamper.

A separate report [15] discusses the technical details of our architecture.

6   Usage Scenarios

From our analysis in Sect. 4, we concluded that enabling widespread development and deployment secure coprocessing applications required a tool that was easily programmable. Adding sufficient computational power and physical security resulted in a device sufficiently expensive that the ability to update software became necessary. The fact that our device is more a fixed extension of the host[2] than a highly portable user token (due to the large form-factor of a PCI-card) makes supporting field installation of software a necessity.

Essentially, we converged on the generic PC model discussed in Sect. 4, attempting to maximize independence between the development/distribution of the hardware, and the development/distribution of the software. As with PCs, end users can obtain their hardware from anywhere, and install the software on their own. But unlike PCs, bona fide software installed into an untampered device can authenticate itself as such—thus providing the necessary cryptographic hooks for the solutions of Sect. 3.2.

Designing for this "worst-case" approach—software from multiple parties gets installed and updated in the hostile field, without security officers—permits a wide range of development, deployment, and usage scenarios:

Application Development.

- **Off-the-shelf Applications**   A party wishing to deploy an application may find that suitable software (for example, application-layer code that transforms the device into a crypto accelerator providing whatever crypto API and algorithms are currently fashionable) is already available.
- **Off-the-shelf Operating Systems**   A party wishing to deploy a more customized application may choose an OS that is already available, register with that vendor, and build on that programming environment.
- **Debug and Development**   Debug and development environments become just another variation of the operating system—the developer can use an off-the-shelf device, with a different OS load.
- **On the Metal**   A party wishing complete on-the-metal control of the device can register with the manufacture, and take control of the OS layer in new off-the-shelf devices.

---

[2] In fact, some IBM hosts may be shipped with the device already installed.

- **Incremental FIPS**   Secure field upgradability makes a device not a "FIPS 140-1 Module" per se [12], but rather a partially certified "meta-module." If the hardware and bootstrap software have FIPS validation, a developer need only submit their additional code to obtain a fully validated module for their application.

Application Deployment.

- **Remote Broadcast**   A deployer can *avoid the hassle of distributing hardware themselves* by just registering as a code vendor and publishing a download command on the Web. The end-users can purchase the hardware from any standard manufacturer channel.
- **Remote Handshake**   If the deployer would like more control, he can use targeting and authentication features of the device bootstrap to interactively install software into a particular device, remotely over an open network.
- **Local Security Officer**   The deployer can always eliminate the open network, and send an authorized security officer with their own trusted device to perform installation.
- **Direct Shipment**   A deployer can also follow the traditional model of obtaining the devices, install the software, and ship them to their users.

Use.

- **Commerce among people who have never met**   The ability for devices to authenticate themselves and their software configurations permits users of such secure devices to securely interact with each other remotely, across an open network—even if these users have met neither each other nor the application deployer.

7   Conclusions

With the growth of the Internet, electronic commerce is becoming central to the way business is conducted. However, this great advance for business carries with it tremendous risks. Specifically, the electronic representations of value can often be modified or stolen or otherwise fraudulently manipulated more easily than physical assets—and in many cases this malfeasance is undetectable.

Clearly, cryptography and other computational tools can protect electronic representations of value. But tamper-protection of the hardware itself is necessary for these tools to be effective—the computation that occurs must be the computation that was intended to occur, despite physical and logical attack. But to be useful and acceptable, tamper-protected hardware must fit into the emerging business models. Flexibility and configurability are as necessary as security and integrity for acceptance.

We offer our research as a step toward meeting these new requirements, and our resulting technology as a tool to allow data and computations to be protected in a way sufficiently flexible and configurable to meet the new needs of business.

Acknowledgments

References

1. D. G. Abraham, G. M. Dolan, G. P. Double, J. V. Stevens. "Transaction Security Systems." *IBM Systems Journal.* 30:206-229. 1991.
2. R. Anderson, M. Kuhn. "Tamper Resistance—A Cautionary Note." *The Second USENIX Workshop on Electronic Commerce.* November 1996.
3. R. Anderson, M. Kuhn. *Low Cost Attacks on Tamper Resistant Devices.* Preprint. 1997.
4. E. Biham, A. Shamir. *Differential Fault Analysis: A New Cryptanalytic Attack on Secret Key Cryptosystems.* Preprint, 1997.
5. D. Boneh, R. A. DeMillo, R. J. Lipton. *On the Importance of Checking Computations.* Preprint, 1996.
6. D. Chaum. "Design Concepts for Tamper Responding Systems." *CRYPTO 83.*
7. B. Dole, S. Lodin, E. H. Spafford. "Misplaced Trust: Kerberos 4 Session Keys." *ISOC Conference on Network Security.* 1997.
8. H. Gobioff, S. W. Smith, J. D. Tygar and B. S. Yee. "Smart Cards in Hostile Environments." *The Second USENIX Workshop on Electronic Commerce.* November 1996.
9. I. Goldberg, D. Wagner. "Randomness and the Netscape Browser." *Dr. Dobb's Journal.* January 1995.
10. *IBM PCI Cryptographic Coprocessor.* Product Brochure G325-1118. August 1997.
11. M. F. Jones and B. Schneier. "Securing the World Wide Web: Smart Tokens and their Implementation." *Fourth International World Wide Web Conference.* December 1995.
12. National Institute of Standards and Technology. *Security Requirements for Cryptographic Modules.* Federal Information Processing Standards Publication 140-1, 1994.
13. E. R. Palmer. *An Introduction to Citadel—A Secure Crypto Coprocessor for Workstations.* Computer Science Research Report RC 18373, IBM T. J. Watson Research Center. September 1992.
14. S. W. Smith. *Secure Coprocessing Applications and Research Issues.* Los Alamos Unclassified Release LA-UR-96-2805, Los Alamos National Laboratory. August 1996.
15. S. W. Smith, S. H. Weingart. *Building a High-Performance, Programmable Secure Coprocessor.* Resarch Report RC21102, IBM T.J. Watson Research Center. February 1998.

16. J. D. Tygar and B. S. Yee. "Dyad: A System for Using Physically Secure Coprocessors." *Proceedings of the Joint Harvard-MIT Workshop on Technological Strategies for the Protection of Intellectual Property in the Network Multimedia Environment.* April 1993. (A preliminary version is available as Computer Science Technical Report CMU-CS- 91-140R, Carnegie Mellon University.)

17. S. Vaudenay. "Hidden Collisions on DSS." *CRYPTO 1996.* LNCS 1109.

18. S. H. Weingart. "Physical Security for the μABYSS System." *IEEE Computer Society Conference on Security and Privacy.* 1987.

19. S. H. Weingart, S. R. White, W. C. Arnold, and G. P. Double. "An Evaluation System for the Physical Security of Computing Systems." *Sixth Annual Computer Security Applications Conference.* 1990.

20. S. R. White, L. D. Comerford. "ABYSS: A Trusted Architecture for Software Protection." *IEEE Computer Society Conference on Security and Privacy.* 1987.

21. S. R. White, S. H. Weingart, W. C. Arnold and E. R. Palmer. *Introduction to the Citadel Architecture: Security in Physically Exposed Environments.* Technical Report, Distributed Systems Security Group. IBM T. J. Watson Research Center. March 1991.

22. B. S. Yee. *Using Secure Coprocessors.* Ph.D. thesis. Computer Science Technical Report CMU-CS-94-149, Carnegie Mellon University. May 1994.

23. B. S. Yee, J. D. Tygar. "Secure Coprocessors in Electronic Commerce Applications." *The First USENIX Workshop on Electronic Commerce.* July 1995.

24. A. Young and M. Yung. "The Dark Side of Black-Box Cryptography— or—should we trust Capstone?" *CRYPTO 1996.* LNCS 1109.