# Secure Coprocessing Applications and Research Issues

Sean W. Smith
Computer Research and Applications Group (CIC-3)
Los Alamos National Laboratory

**Los Alamos Unclassified Release LA-UR-96-2805**

August 1, 1996

**Abstract**

The potential of secure coprocessing to address many emerging security challenges and to enable new applications has been a long-standing interest of many members of the Computer Research and Applications Group, including this author. The purpose of this paper is to summarize this thinking, by presenting a taxonomy of some potential applications and by summarizing what we regard as some particularly interesting research questions.

# 1. Introduction

## 1.1. An Emerging Problem

Computing is transforming our society, our nation, and this Laboratory. Increasingly many services of growing importance are migrating from paper to electronic format. Private sector examples include credit card payments, catalog purchases, and journal publication; public sector examples include anonymous cash, tax and benefits transactions, and the secret ballot; Laboratory examples include access to administrative data, classified archives, and commercial research databases. This transformation will render the way we do business in twenty years unrecognizable to modern eyes.

However, the computational environments to which these services migrate are becoming distributed and universal. An unfortunate consequence of this growth and distribution is that these environments are often unreliable and unsecure, even hostile.

- In many visions of the electronic future, individuals do not use their own machines, but rather publicly accessible devices (for example, in post offices, libraries, and schools). Can users completely trust these machines?
- Viruses and other malicious software remain prevalent (for example, Word macro viruses reportedly even appear in Word documents shipped directly from Microsoft). Can users even trust their own machines?
- The above two scenarios assume that the individual user is trustworthy. What if the user is the source of malice?

---

This is a Latex/postscript version; pagination may differ from the original paper version.

## Computation in Hostile Environments

These circumstances conspire to create a security challenge. The electronic versions of these paper services and processes depend on intangible electronic and computational objects that do not naturally inherit the security and robustness properties of their electronic objects (e.g., [Sm96]). Reproducing these properties in electronic environments prone to failure is difficult enough. But we need to build in security and robustness when the environments themselves are potentially hostile, supporting malicious actions, and when none of the parties involved can necessarily be trusted!

## 1.2. The Secure Coprocessing Model

Secure coprocessing is an emerging technology that will help address the problem of securing computation in hostile environments. As the name implies, secure coprocessing involves augmenting untrusted computers with small, physically secure computational devices.

Secure coprocessing enables a new approach to security. For example, the traditional military approach to securing computation rests on several pillars:

- thorough evaluation of the software and hardware involved;
- thorough screening of all users; and
- thorough guarding of the physical installation.

However, none of these techniques apply in the emerging information world. The computing platforms and software are too numerous and emerge too quickly to be evaluated; the users are potentially the entire population;[1] and putting a fence and guards around the entire Internet is not feasible.

The secure coprocessing model addresses these challenges by establishing small, trusted bases distributed everywhere. Rather than physically guarding the entire installation, we physically guard small, individual devices. Rather than evaluating all software and hardware, we confine scrutiny to these devices. Rather than screen all users (and service providers), we establish devices that might be potentially trusted by all parties.

## 1.3. Realizing this Solution

Many steps are necessarily to achieve this vision.

### 1.3.1. Hardware

Perhaps the most fundamental step is the emergence of the secure coprocessing hardware itself. This section surveys some families of existing products and prototypes.

#### Parameters

We can organize this hardware by several parameters:

[1] Indeed, we have worked on a computer security project where the user population are convicts—precisely a category of individuals who would not pass background checks!

- **Physical Protection**

  Does the device provide active tamper resistance? For example, can it sense physical attacks and respond by zeroizing the internal memory? Does the device rely solely on an impenetrable shell or inscrutable layout? How effective are these protections?

- **Reliability**

  In a similar vein, how robust is the device against physical or electrical damage?

- **Computational Ability**

  How powerful a computational environment does the device provide? How much (if any) protected memory? How much (if any) processing power? How much (if any) hardware support for encryption? For pseudorandom number generation? Any on- board power source? A real-time clock? (If so, how accurate?)

- **Communications**

  What means does the device provide for communication? Direct connections to a host computer? To an external network? RF, infrared, or other wireless communications? Any means for direct I/O with the user?

- **Portability**

  How portable is the device—must it be permanently installed in the host, or can users carry it around in their pockets?

- **Cost**

  How much does the device cost?

### Examples

These parameters sketch out a fairly large space. Many existing products and prototypes are beginning to fill the space (although the optimal extremes are still empty). We survey a few varieties:

- **IC Chip Cards**

  Tokens the size of a credit card embedded with a small integrated circuit, IC chip cards find popular application as telephone cards in Europe and (more recently) in the U. S. They feature fairly primitive computational ability and security, and offer no direct user I/O; on the other hand, they support communication with a host both directly and via RF, and are extremely inexpensive (costing on the order of $1 per unit, which deployers often recoup by selling advertising space).

  Interestingly, the insecurity of an IC chip card application lay at the heart of possibly the biggest incident of electronic commerce fraud known to date: the theft of over $500 million from pachinko gaming in Japan. [Wa96]

- **PC (PCMCIA) Tokens**

  PCMCIA (Personal Computer Memory Card International Association) is an emerging standard for notebook and workstation peripherals. The size of a small stack of credit cards, the “Type II” PCMCIA card is the form factor for a number of commercial secure coprocessor products, including National Semiconductor’s iPower cards, Spyrus’s LYNX cards, Telequip’s CryptaPlus cards, as well as the Fortezza cards of Clipper/Skipjack infamy. PCMCIA cards are portable and provide a direct electrical interface to the host. Current PCMCIA cards do not provide direct user I/O or external connections, although no significant barrier exists here (and these features have been discussed as possibilities for future prototypes).

  The computational power in PCMCIA secure coprocessors ranges from repackaged IC chip card processors with ancillary memory to proprietary and classified encryption devices. However, sufficient real estate exists to support some interesting environments.

  The primary drawbacks to PCMCIA secure coprocessors are their price (several hundred dollars per unit) and reputed lack of physical reliability. (For example, we have heard reports of frequent connector failures—since

some applications require far more frequent insertion/removal cycles than peripherals—as well as of cards being damaged from being carried around in a wallet.)

The term "PC card" is supplanting the original term "PCMCIA card," due to the awkwardness of the original acronym.

- **Other Card Tokens** Other vendors have introduced secure coprocessors roughly the size of PC tokens, aimed primarily at authentication. Two in particular are SecureID and ActivCard. Both provide direct user I/O; ActivCard (currently being sued by SecureID for patent infringement) also provides optional direct host communication.
- **Smart Disks** An elegant solution to a temporary problem, smart disks provide IC chip card functionality in the form factor of a floppy disk. (PC card and IC chip card interfaces are not standard on most desktops, but floppy disk interfaces are).
- **Bus Cards** Another class of secure coprocessor devices are installed directly into the internal bus of the host device. A good example here is IBM's Citadel prototype [Pa92]; another is the new ISA-bus version of Fortezza. These devices provide the best security and computational ability of which we are aware; in the case of the IBM Citadel, the circuitry is wrapped with nichrome wire whose penetration triggers zeroization, and a 386, a megabyte RAM, and high-speed DES engine all reside within the secure envelope [Wein87, Yee94]

Although the term "smart card" is often used for nearly every item in this list, we conform to the general standard of using it only for IC chip cards. (To underscore the difference, some wags have suggested the term "smart-ass card" for the more powerful secure coprocessors.)

### 1.3.2. Applications Research

Another important aspect of deploying secure coprocessing as a solution to the emerging security challenges is using the hardware to develop and deploy applications. Since the degree of ongoing work in this area makes an exhaustive list impossible in a paper of this scope, we discuss some of the principal efforts.

In the academic sector, the pioneering work was carried out by J. D. Tygar and his students, primarily B. S. Yee, at Carnegie Mellon University, using an early version of the IBM Citadel prototype (e.g., [TyYe91, Yee94]).

IC chip cards are finding applications in numerous social and financial applications: to support telephone cards, bus fares, and credit cards. Additionally, Paul Clark and Lance Hoffman at George Washington University explored using IC chip cards to secure operating systems. [ClHo94]

The NSA has explored the use of portable Fortezza cards as part of their Multilevel Information Systems Security Initiative (MISSI), intended to use trusted systems and secure coprocessors to interconnect systems of more than one "Orange Book" security level.

National Semiconductor and Xerox have both discussed using PC card coprocessors as part of rights management for electronic publishing.

IBM's Citadel group and Telequip are jointly working with Telequip Corporation in the Financial Services Technology Consortium's eCheck project, creating checks that exist solely in electronic format and injecting them directly into the American banking clearinghouse system. [FSTC95]

## 1.4. This Paper

The potential of secure coprocessing to address many emerging security challenges and to enable new applications has been a long-standing interest of many members of the Computer Research and Applications Group, including this author. The purpose of this paper is to summarize this thinking, by presenting a taxonomy of some potential applications and by summarizing what we regard as some particularly interesting research questions.

# 2. Applications

## 2.1. Goals

Secure coprocessors provide the potential to solve a lot of interesting security problems. However, in many cases it seems that both the suppliers of the technology and the customers with security problems do not see the potential. Since beginning to pursue these ideas in 1994, our goal (to date unachieved, unfortunately) has been to establish a testbed in which to demonstrate this potential, and to build tangible proofs-of-concept.

Secure coprocessors offer a tamper-resistant computational shell. Too many smart card applications focus only on putting data (such as counters and keys) in that shell. Figure 1 sketches this style of application.

We feel this approach is too limiting: not only can secure coprocessors store and modify data behind a secure shield, they can also be an active computational partner, transforming this data and interacting with the host computer. In particular, we sketch five styles of applications that the secure-counter style omits.

- **Generalized Access**

  Many existing secure coprocessor applications center on a simple "barrier" model of access control: whether an agent is privileged to read or write data depends on whether or not they have a key or ticket. Having a computer within the secure shield allows the coprocessor to decide access based on an arbitrary function of the coprocessor's state (which can include whatever history it has maintained, along with the identity of the agent). The only limit is that this control function be a feasible computation. Figure 2 sketches this style of application.

- **Generalized Revelation**

  Similarly, many existing secure coprocessor applications center on storing data securely, and then revealing this data to privileged agents. Having a computer within the secure shield allows the coprocessor to reveal an arbitrary transformation of this data. The only limit is that the transformation be a feasible computation. Figure 3 sketches this style of application.

- **Autonomous Auditing**

  Having a computer within the secure shield allows the coprocessor to carry out secondary functionality, such as extracting and retaining state as time and transactions proceed. Figure 4 sketches this style of application.

- **Trusted Execution**

  Having a computer within the secure shield allows the coprocessor to carry out computation without manipulation by an adversary. Figure 5 sketches this style of application.

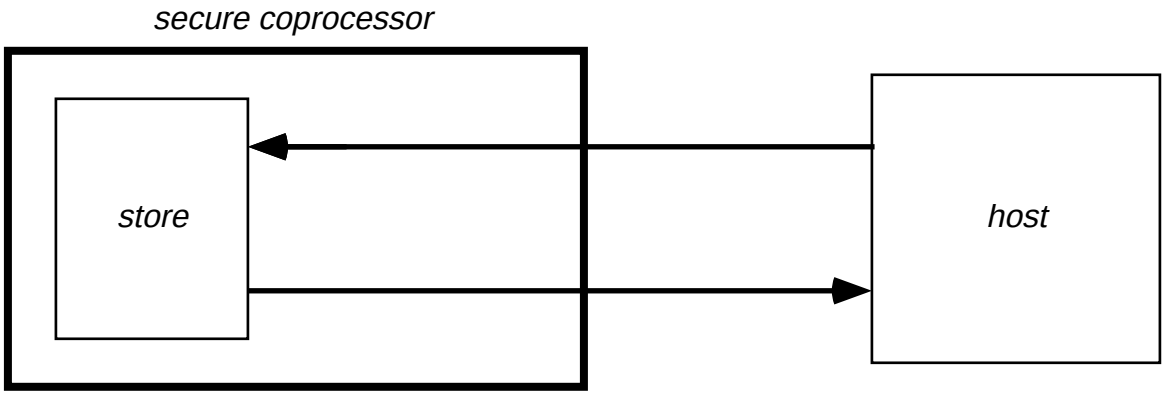


**Figure 1** In the basic style of application, the coprocessor functions as a secure store.

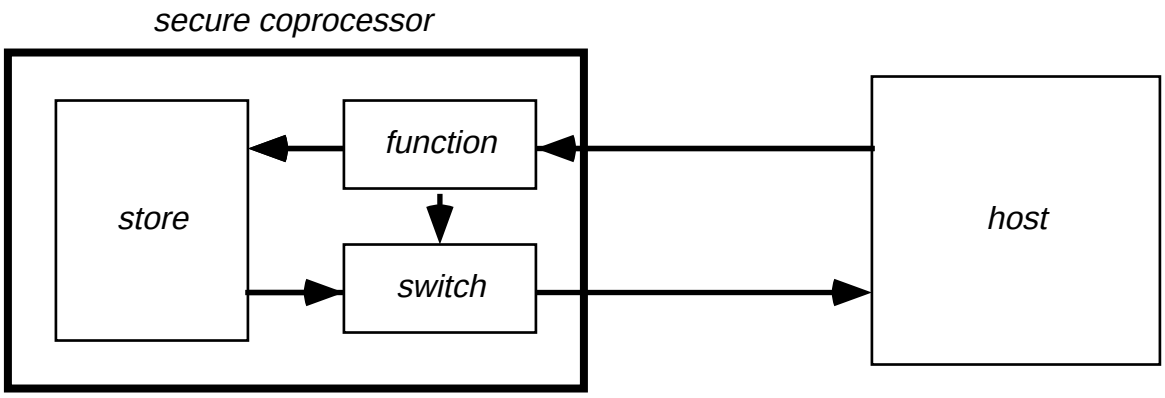


**Figure 2** The “Generalized Access” application style uses the computational ability of the coprocessor to enforce non-trivial access rules.

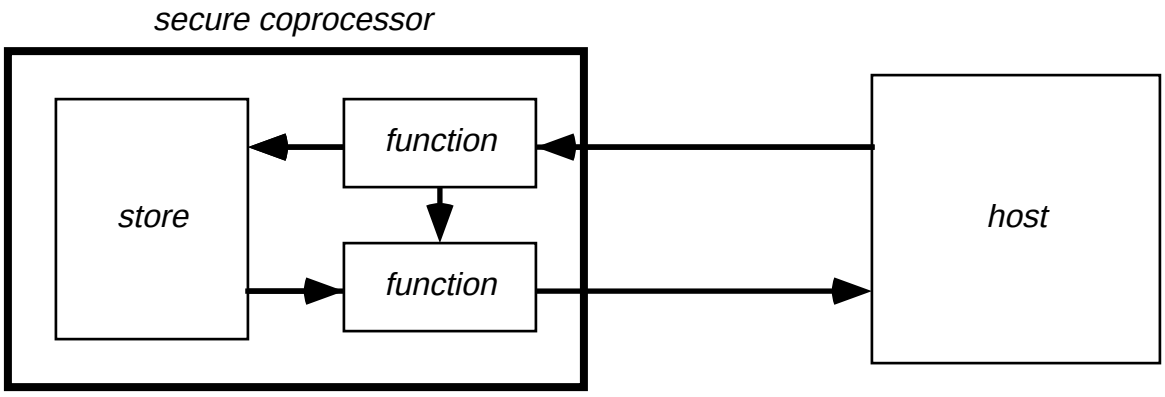


**Figure 3** The “Generalized Revelation” application style uses the computational ability of the coprocessor to transform the accessed data, depending on the context of the access.

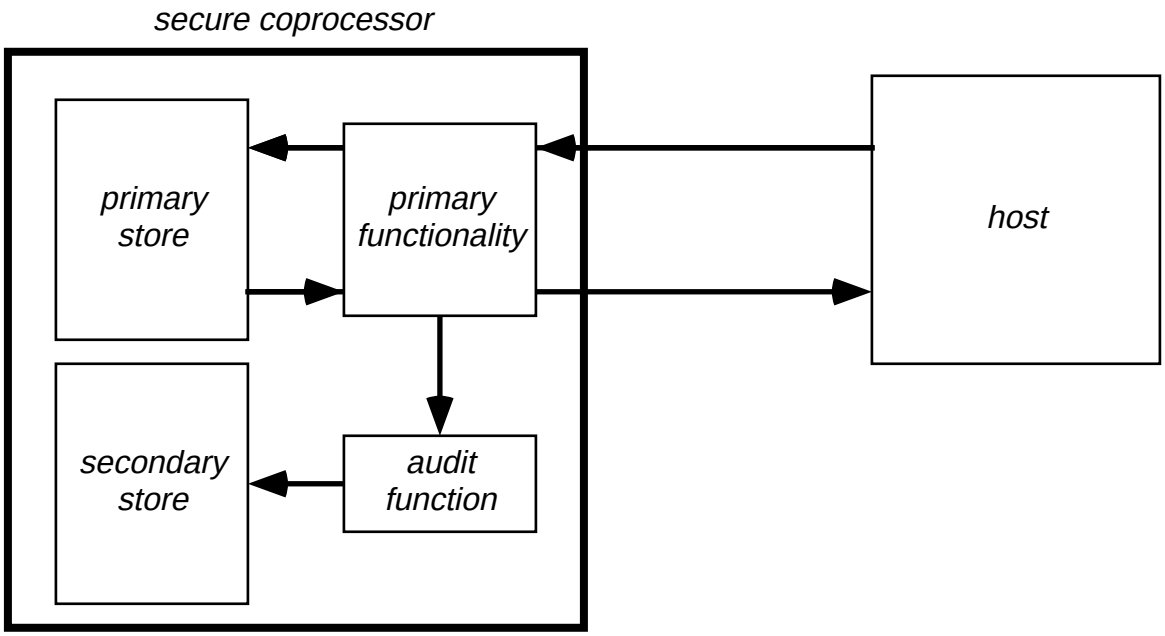


**Figure 4** The “Autonomous Auditing” application style uses the computational ability of the coprocessor to autonomously extract and archive audit data.

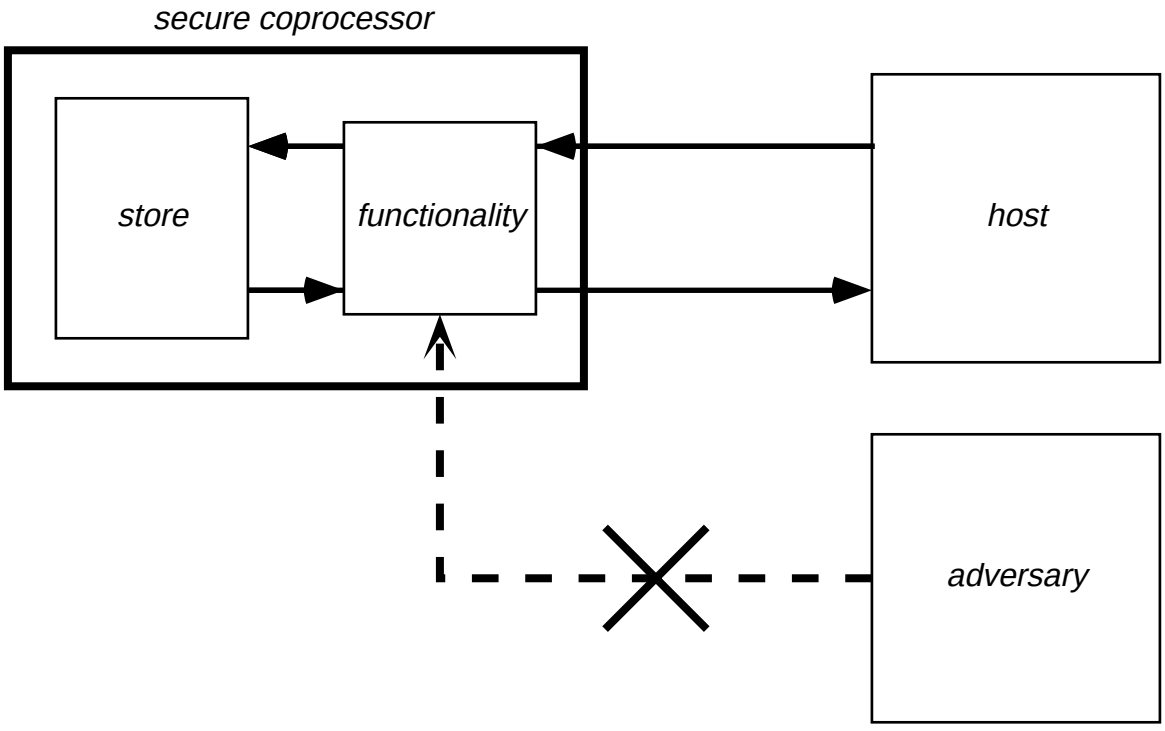


**Figure 5** The “Trusted Execution” application model uses the computational ability of the coprocessor to carry execution without manipulation by an adversary.

- **Hidden Execution**

  Having a computer within the secure shield allows the coprocessor to carry out computation without observation by an adversary. Figure 6 sketches this style of application.

## 2.2. Sample Applications

We now sketch several potential secure coprocessor applications, that have emerged based on our discussions and research here (some fairly early, e.g., [Sm94b]). We note that the purpose here is to sketch a diversity of potential applications, not to present completed implementations nor to present a thorough survey of other activity in this field (for example, ongoing work in rights management by IBM, in rights management and PCMCIA cards by Xerox, and in information metering and authentication by National).

We group the sample applications into three general areas: securing data access, securing authentication and securing computation.

### 2.2.1. Data

#### Application: Dictionary

A challenge facing electronic publication is how to reproduce electronically the controls and rules present in paper publications, often implicitly [Sm96]. A particular concern is copy protection: the owner of a copy of the publication is permitted "legitimate use" of a single copy. (However, the electronic age has complicated even the formalization

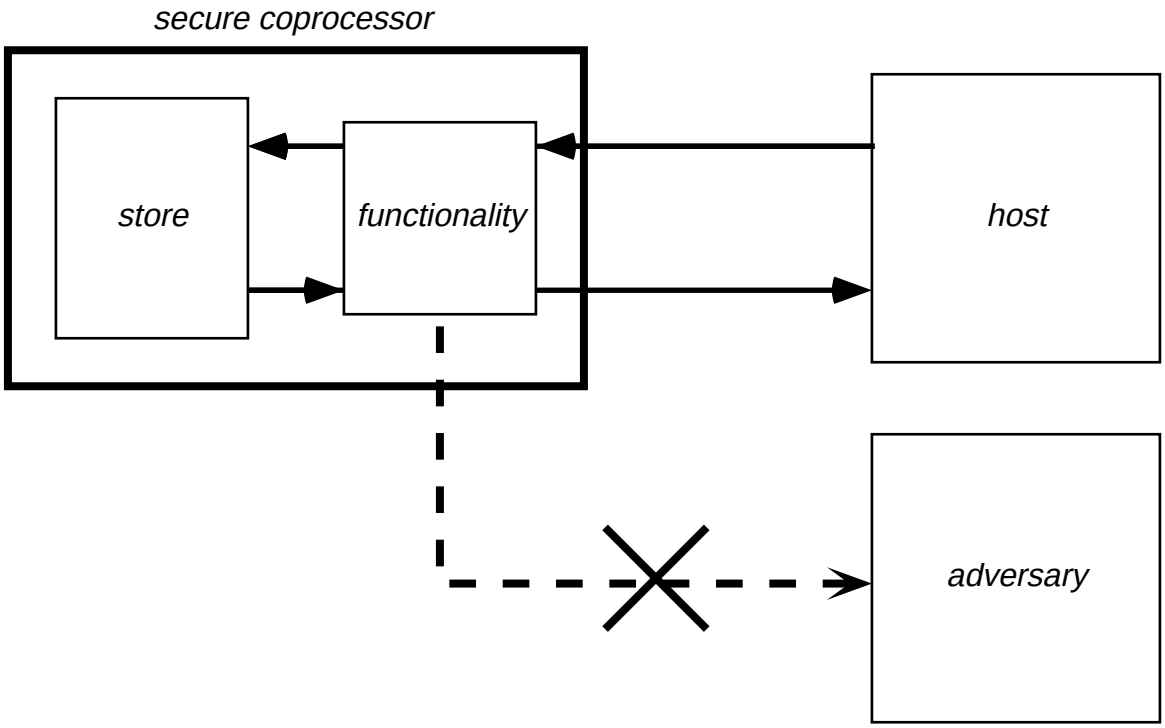


**Figure 6** The "Hidden Execution" application model uses the computational ability of the coprocessor to carry execution whose details are hidden from an adversary.

of "legitimate use." For example, eliminating "no duplication" eliminates making an emergency back-up of one's software, and—depending on some interpretations—also eliminates reading copyrighted material over the Web—since reading requires downloading and possibly caching, both of which might be considered duplication.)

Enforcing legitimate use for electronic versions of linear media, like novels, is difficult in the presence of potentially hostile users: ordinary use of the object requires disclosing the entire object to presumably a hostile environment (the user's machine). However, secure coprocessing might enable enforcing legitimate use for electronic versions of non-linear media, such as a dictionary, where ordinary use only discloses a small portion of the overall structure.

Legitimate use for object such as a dictionary might be closely characterized by some straightforward rules. For example, a sequence of fairly infrequent, somewhat random queries might likely be legitimate; a rapid sequence of alphabetized queries might reflect systematic copying—unless, of course, the user is running "spellcheck" on a word processor that batches and alphabetizes queries.

The "Generalized Access" application style can support a secure coprocessor implementation of an electronic dictionary: the dictionary resides in the internal store, and the access function (probably with state and real time) enforces the rules. This application extends the secure physical shell to a virtual shell that ensures that usage of the data conforms to patterns of legitimate use.

Of course, this sketch does not suffice to fully reproduce all properties of physical dictionaries. For example, physical books have the property that they can be lent or given away. At first glance, our secure coprocessor implementation could support this by having two coprocessors perform transactional exchange of the dictionary item. However, this feature may provide too much power: the slowness of physical actions prevents the "giving away" property of physical books from supporting virtual timesharing (e.g., whenever Alice needs the dictionary, she near-instantly borrows it from whoever has it, looks up her word, and then returns it). Thus, the giving-away functionality inside the coprocessors would also need to be governed by access rules ensuring that lending and giving conform to legitimate use.

### Application: Secure File Systems

A variation on the electronic dictionary (coupled with the technique of cryptopaging) is a secure file system. A coprocessor can function as a gateway that encrypts files to be stored, decrypts them if appropriate authentication is provided; otherwise, the keys to the data remain locked inside the tamper-resistant environment. This application might prove useful in scenarios where the entire computing environment (including storage media) suddenly falls into the hands of an adversary.

### Application: Health Card

A dictionary has the property that when the rules permit the owner to make a query, the owner may see the complete data record. Not all datasets have this property. A simple example within the Laboratory are publication lists and projects on CV's: some publications have titles that change, depending on the status of the reader; some project details, client names, and even projects themselves are too sensitive to be disclosed without sanitization, except in certain contexts.

The notion of a health record card provides a more sophisticated example of an application where not just access to a record but the view of that record needs to follow non-trivial rules. For example, how much and what fields of a patient's health history should be sanitized (or even randomized) depends on the context and identity of the querier: the family doctor, a dentist, a specialist in foot care, an employer, a nurse checking prescription dose, or a medical student making a research query.

The "Generalized Revelation" style can support a secure coprocessor implementation of a health data card; the access function enforces the rules of access and, from context, directs the output transform function to act accordingly. This application extends the secure physical shell to a virtual shell that ensures that usage and views of the data conforms to patterns of legitimate use.

The health card application also suggests more advanced topics. Issues of back-up and destruction are discussed in Section 3. The unconscious-at-the-side-of-the-road scenario suggests the need for "emergency room" access to the complete and unsanitized record; back-up auditing and automatic alerts (examples of remaining application styles) would be necessary to detect and announce any such access, to prevent fraud (e.g., by unscrupulous prospective employers or insurers). Considering health care and sensitive data also suggests the need for some creative applications of cryptography, supported by coprocessors. For example, an HIV+ individual from a small town may wish to go to a pharmacy in a large city to fill his prescription of AZT; however, revealing his name and the doctor's name may leak too much information. How do we support such "authenticated anonymity?" How do we so while preventing a dishonest patient from filling his prescription repeatedly?

Generalized access with context-based views departs significantly from the lattice models of security traditional in military installations. As Section 3 discusses, fully considering the implications of this now-feasible departure suggests some interesting research problems.

#### Application: Pay-as-you-use Software

An extreme example of non-linear media is software. Ordinary execution of many programs only discloses a small path through the graph of possible executions. This fact, coupled with the expense of software, suggests that secure coprocessors may make feasible a novel application: software that users pay for as they use. This application could take several forms: as rental of software, as superdistribution, or simply as users paying a fractional cost (converging to the software purchase price) depending on how often and how much of the program they use.

The "Autonomous Audit" style can support a basic secure coprocessor implementation of pay-as-you-use software; the audit function tracks and records program execution. This application extends the secure physical shell to a virtual shell that ensures the software vendor accurately gauges the amount of use of their software, despite potentially hostile user environments.

Protecting the software from copying (and hence illegitimate use) suggests many more advanced topics. How much can a hostile user learn about the software from observing his or her execution traces? Can adding "Generalized Revelation"-style randomization and masking counter such attacks? What impact do protection measures have on performance? Indeed, one might argue that secure coprocessors are crucial to realization of the long-term vision of superdistribution (e.g., [Co95]).

### 2.2.2. Authentication

#### Application: Authentication

A popular application of secure coprocessing is authentication. How can users log in over insecure networks without transmitting passwords in the clear? One approach is to use time-based smart cards that generate passwords good for only a short period of time; another is to use passwords that change with each use; another is to use public-key cryptography for a challenge-response protocol. However, these approaches have some limitations: an eavesdropper can still gain useful information; an adversary can attack a user by causing the coprocessor and the host to become unsynchronized (perhaps by attempting a login under the user's name); a user that loses their card needs to obtain a new one that knows how far the other card got; an attacker can use chosen text attacks, and (if the coprocessor only supports one key pair) the coprocessor cannot use keys for signatures.

The "Generalized Revelation" application style can support a secure coprocessor implementation of authentication, by carrying out true zero-knowledge authentication of users (e.g., [Bl86, FFS88]). Eavesdroppers provably gain no knowledge, and suppressed or delayed messages will cause no synchronization problems. This application extends the secure physical shell to a virtual shell that ensures that authentication can be carried out despite insecure networks.

However, user authentication only solves part of the problem. Penetration specialists will happily point out that hijacking a session is only marginally harder that sniffing a password, as is hijacking the user's host (and feeding arbitrary packets to the coprocessor). Effective security would also require the coprocessor to encrypt every packet, as well as perform some type of real-time anomaly detection on these packets. These problems could prove difficult.

### Application: Authentication for Virtual Secure Networks[2]

Sensitive computing environments—like the Laboratory's—give rise to some difficult security and logistical problems. Users, machines, networks, and traffic all possess different security levels. Security requires enforcement of appropriate level- based access rules; however, the size of the Laboratory, the number of workstations, and the portability of users creates logistical problems. For example, Scientist $X$ wants to use the nearest machine to check a record, rather than having to go find the nearest machine of security level $Y$ that is currently functioning and is connected to a network of level $Z$.

The "Trusted Execution" application style can support a secure coprocessor solution for this problem, by having the coprocessor act as a combined authentication token and controlled interface. Scientist $X$ could insert his coprocessor into the nearest machine (and, to meet the letter of regulation, connect a network cable to the coprocessor); the coprocessor could authenticate to the remote secure server, perhaps perform an integrity check of the workstation (a la [Yee94]), and, by performing the appropriate CI barrier functions, transform the workstation into the properly secure computing environment. This application extends the secure physical shell to a virtual shell that ensures officially "secure" computation can occur in open environments.

### Application: Migrating Environments[3]

A consequence of distribution—both in computing, as well as in society—is the breakdown of a firm correlation between users and machines. Nonetheless, users would like to preserve that fiction, and have a correlation between users and computing environments (even if the actual hardware changes). Similar to virtual secure computation, it would be convenient for a scientist at the Laboratory to grab the nearest workstation—or a businessmen to grab a workstation at the airport club—and to have it quickly and reliably transform to their familiar environment.

However, this application raises security questions. How does a user safely carry their environment with them? How do they trust the machine they are using? How do they know that no traces of their environment remain on the machine when the session is finished?

The "Trusted Execution" application style can support a secure coprocessor implementation of migrating environments. This implementation would be similar to virtual secure computation, but with a few additional issues:

- evaluating the integrity of the host machine may be particularly difficult for scenarios, such as the "airport club computer," where the host may be particularly untrustworthy;
- maintaining a continuous network presence despite discontinuous, distributed sessions raises many design and security topics similar to mobile computing;
- preventing loss or damage of the user's portable coprocessor from significantly damaging the user's stored environment requires developing algorithms for effective and secure back-up using distributed and possibly hostile hosts.

### Application: Biometrics[4]

A topic of some ongoing research (e.g., the still-unfinished [FaSm95]) is the security of biometric authentication systems. Our concern derives from the fact that physical features (such as fingerprints or hand geometry) cannot be changed, and (currently) each installation of a particular biometric reader is testing for the same key data. (We have called this the *common template* problem [HSMP95].) This situation creates some security hazards:

- If Alice builds a prosthesis for Bob's fingerprint/hand/etc., she can use it to impersonate Bob.

[2] Based on discussions with G. Christoph.

[3] Based on discussions with E. Knill.

[4] Based on discussions with V. Faber.

- If Alice compromises the template database at any installation that uses a particular biometric, then she may have sufficient information to impersonate that installation's users anywhere else they are registered and that uses the same biometric. For example, suppose Bob's workplace and school both use fingerprints; if Alice breaks into the school's database, she might learn enough to impersonate Bob at his workplace. (The prevalence of biometric demonstrations at conferences and trade shows is another area of concern—just who are you giving your thumb to?)
- Once she has Bob's template, Alice does not need to test her prosthesis in a live situation—she can steal a biometric reader, or even buy one off the shelf.

Solution strategies rest on hiding details of algorithms: what view of biometric data a particular site examines (so compromising the school gives no information about the workplace) and what subview a site examines on a particular authentication attempt (so Alice can never be sure if her prosthesis will work on a given attempt). These data sets would need be structured non-trivially, so that knowledge of some reasonable number of template views (subviews) would not suffice to construct a prosthesis that would successfully match another view (subview)—trivial solutions, like splitting the biometric into four quadrants, will not work.

Hidden algorithms naturally suggest secure coprocessors (the "Hidden Execution" application style). A number of implementations suggest themselves:

- A coprocessor inside the device accepts the raw data from the user's biometric, and uses its secret algorithm to compress that data to a tractable signature.
- A coprocessor inside the device explicitly requests various fields and parameters from the user's biometric, and then compresses that data to a tractable signature. (This approach prevents the raw data from ever existing in one place—since otherwise an adversary could just build a prosthesis matching the raw data, which would then match all compressed signatures.)
- A coprocessor the user carries has its own biometric reader. The coprocessor then measures the user's biometric, computes the compressed signature, performs some mutual authentication with the biometric device, and reports the signature to the device. (This application should be feasibile to prototype—fingerprint readers that fit in PCMCIA format have been announced, and a colleague of ours who wrote the FBI standard fingerprint compression algorithm estimates it could be efficiently performed in a Citadel-like environment.)

This application extends the secure physical shell to a virtual shell that reduces the potential for adversaries, even with complete access to the biometric device, to build prostheses sufficiently accurate to spoof the device.

### 2.2.3. Computation

#### Application: Agents

Some distributed computation work, including our Los Alamos Information Systems Technologies project, consider intelligent agents: objects with specific missions migrating through distributed, networked environments A quick consideration of this area yields several interesting questions regarding mutual suspicion:

- Why should Alice believe that agent X has authority to examine sensitive data?
- Why should Alice trust agent X to execute on her machine?
- Why should Alice trust agent X to read sensitive data and then migrate to another address space?
- Can the creator of agent X trust Alice to let it run unmolested on her machine?

The "Trusted Execution" and "Hidden Execution" application styles can support a secure coprocessor implementation of migrating agents. Alice can trust agent X to run in a sheltered place inside her coprocessor—which would

shield X from Alice, as well as provide a mechanism for Alice's coprocessor to monitor the actions of the X. When ready, agent X could migrate in encrypted form to Bob's coprocessor.

Advanced topics here include extending the application to support multiple varieties of coprocessors; the migrating object could evaluate the power of its new environment, and expand accordingly. [5]

### Application: Applets

The advent of Java and the World Wide Web creates a set of more practical, immediate migrating-agent issues: the security issues raised by migrating applets. In particular:

- How does Alice know that the applet she just downloaded won't do anything dangerous?
- How can Bob deploy applets that need, in order to fulfill their purpose, to access data or functionality at Alice's machine, in a way that violates the "clean room" standards of Java?

The "Trusted Execution" and "Hidden Execution" application styles can support a secure coprocessor implementation of migrating applets. Alice might trust applet X to run inside a sheltered space in her coprocessor, which can monitor its actions, preventing anything illegitimate but perhaps permitting actions beyond the "clean room" constraints.

### Application: Piracy Auditor[6]

Worldwide, a significant fraction of the software in use is pirated. DOE Laboratories address this concern by actually sending human auditors to examine the software and files on users' computers. However, this approach is both inefficient—humans need to visit every office—and intrusive. This intrusiveness might hinder application of this approach in less controlled corporate and academic environments, and may also trigger a confrontational attitude that might impair effectiveness.

The "Trusted Execution" application style can support a secure coprocessor implementation of such an auditor, more efficient and less intrusive than the human version. Coprocessors on each user's machine might systematically examine the file system to evaluate the answers to a set of published questions (e.g., "is all software properly licensed?"). Management can trust the coprocessors to report the correct answers; users can trust the coprocessors to report the correct answers, and nothing else. Malicious users might still try to spoof coprocessors, but we suspect coprocessor auditors should be at least as effective as their human counterparts (especially since coprocessors can audit all the time). Coprocessor auditors would offer the additional advantage of preserving compartmentalization.

### Application: Capabilities Management

The problem of granting, delegating, and then revoking capabilities can be tricky, especially if machines cannot be trusted. Our earlier work [Sm94a] observes how this problem reduces to optimistic rollback in a partial order time model, which can be addressed using secure coprocessor techniques [Sm94a, SmTy94].

### Application: Distributed Security Everywhere[7]

A common security weakness in computing systems is that security protections exist in only a few isolated places: a firewall, a login challenge. Conversely, a mantra we need to often repeat to customers is "belt and suspenders"—effective security requires more than just one fence.

---

[5] Based on discussions with M.F. Jones.

[6] Based on discussions with V. Faber.

[7] Based on discussions with V. Faber.

In a long-term vision, distributing secure coprocessors everywhere—in every step of a communications and computation path—can take this approach to its extreme. (barring a trivial way of defeating the tamper-resistance).

# 3. Research Questions

These questions fall into many areas. Here are some ones we are currently thinking about:

## 3.1. Applications Issues

What if a user loses their coprocessor? How do we securely back-up their data? What about "accidental" loss or destruction?

Why should a user believe that a coprocessor is performing as advertised? ([Yee94] suggests cut-and-choose, with destructive testing; [YoYu96] warns against public key generation in coprocessors. Some of our suggestions in 3.3 might apply here.)

How do we express rights and behavior properties for malleable electronic objects emerging in the information infrastructure?

How do we model rights and rights management with secure coprocessing?

- For example, rather than saying agent $X$ can see object $Y$ if the $X, Y$ entry in a matrix is 1, we can say that agent $X$ can see transformation $T(Y)$ if function $F(X, Y, \text{history})$ is 1. How does this functional approach relate to classical approaches with regard both to impossibility results as well as relative efficiency/complexity?

## 3.2. Achieving Secure Coprocessing

How do we secure the operating environment?

- Many low-end devices, such as chip-cards, derive their clocks from the host and thus can be extremely susceptible to timing attacks [e.g., Ko95].
- High-end devices, such as the IBM Citadel, use the technique of cryptopaging to augment memory space—but the frequency, location, context, and content of page swaps can be observed by the adversary. How do we construct the coprocessor's cryptopaging strategy to minimize the information an adversary can learn?

What can we do about the lack of trusted communication channels in and out of the card [GSTY96]?

- For example, while on travel, a Lab employee might use a PCMCIA card to establish a secure connection from her notebook to the Lab in order to read email. But what about the end of the tunnel? How does the token know what input is legitimate—and what input came from a Trojan Horse planted on the user's notebook? Effective porting of anomaly detection tools into secure coprocessing environments would be a significant achievement.

What can we do about malicious suppression of communication in and out of a coprocessor? (Random, encrypted back-references to earlier communications seems like a promising technique.)

### 3.3. Limits of Secure Coprocessing

Is secure coprocessing necessary?

- Some researchers object to solving security problems with hardware. Yet, intuition suggests that many problems (e.g., transactional exchange of encrypted data objects) can only be solved with secure hardware. Can we formally prove that this intuition is correct?

What minimal amount of power is necessary?

- Chip card researchers [Gu96] argue that computer science has a long history of prophets predicting wonders when just a bit more computational power is available. Is the number of potential applications of secure coprocessing relative to computational power linear, or a step function? (Answering this question either way is a win.)

What maximal amount of power is useful?

- If trusted central sites exist, then at some point it becomes more efficient to establish a secure channel from the coprocessor to a central site and to ship functionality there.

What can be done to address the failure of security assumptions?

- We are not convinced by the tamper-resistance claims of chip-card manufacturers. How can we address the challenge of the adversary duplicating modified cards? For example, consider a card that contains a circuit calculating some function $F$. What can we do to make a circuit self-verifying (to prevent adversary modification) without sacrificing efficiency or security? (One extreme is replacing the circuit with a truth table, with each $< x, F(x) >$ pair signed by a trusted authority; this leads to an exponential blowup in circuit size. Another is to replace the circuit with a signed copy of a description or program of the circuit, and uploading that program to the card reader; this sacrifices a type of security, by giving away the entire function with each use. We speculate[8] that a zero-knowledge approach to "proofs of correctness" might work.)
- Many players are considering deploying secure coprocessors to the population at large in order to generate, store, and apply private keys for use in legally binding signatures. Since it might be the case that private keys can be learned by an adversary (e.g., through a timing attack or other covert channel), citizens might legitimately deny selected signatures. What can be done inside the coprocessor to help law enforcement distinguish between genuine denial and fraudulent denial? (We speculate[9] that a Haber/Stornetta-style timestamping scheme [BHS93, HaSt91] can be extended to link together uses of a key, so that a malicious user could not selectively repudiate signatures.)
- The "give and forget" techniques of [Sm94a] might also extend to limit the mischief possible for an adversary who opens a tamper-resistant box. (Basically, a compromised coprocessor cannot be relied upon to forget data, but an uncompromised one can.) Using the standard cryptographic technique of iterated one-way functions on a secret seed might also yield similar results.

## Acknowledgments

The author is extremely grateful to many colleagues for their helpful advice, comments, and discussion. Particular thanks go to Doug Tygar (of CMU) and B. S. Yee (now at UCSD); Howard Gobioff (at CMU); Vance Faber, Gary Christoph, Paul Pedersen, and Manny Knill (at Los Alamos); Elaine Palmer (at IBM); and Michael Jones (of Telequip).

[8] Based on discussions with B.S. Yee.

[9] Based on discussions with V. Faber.

## References


[BHS93] D. Bayer, S. Haber and W. S. Stornetta. "Improving the Efficiency and Reliability of Digital Time-Stamping." *Sequences II: Methods in Communication, Security and Computer Science.* Springer-Verlag, 1993.

[Bl86] M. Blum. "How to Prove a Theorem so that No One Else Can Claim It." *Proceedings of the International Congress of Mathematicians.* 1986.

[ClHo94] P. C. Clark and L. J. Hoffmann. "BITS: A Smartcard Protected Operating System." *Communications of the ACM.* 37: 66-70. November 1994.

[Co95] B. Cox. *Superdistribution: Objects as Property on the Electronic Frontier.* Addision Wesley, 1995.

[FaSm95] V. Faber and S. W. Smith. *A Framework for Biometric Security.* Paper in progress (still). Los Alamos National Laboratory. 1995?

[FFS88] U. Feige, A. Fiat and A. Shamir. "Zero Knowledge Proofs of Identity." *Journal of Cryptology.* 1: 77-94. 1988.

[FSTC95] Financial Services Technology Consortium. *Electronic Check Proposal: Public Document.* 1995.

[GSTY96] H. Gobioff, S. W. Smith, J. D. Tygar, B. S. Yee. *Smart Cards in Hostile Environments.* Draft, July 1996. (A preliminary version appeared as Los Alamos Unclassified Release LA-UR-96-1297, Los Alamos National Laboratory, March 1996.)

[Gu96] S. Guthery, Schlumberger. Personal communication, April 1996.

[HaSt91] S. Haber and W. S. Stornetta. "How to Time-Stamp a Digital Document." *Journal of Cryptology.* 3: 99-111. 1991.

[HSMP95] J. Hochberg, S. W. Smith, et al. *Kiosk Security Handbook.* Los Alamos Unclassified Release LA-UR-95-1657, Los Alamos National Laboratory. 158 pp. May 1995.

[Gu96] S. Guthery, Schlumberger. Personal communication, April 1996.

[Ko95] P. Kocher. *Cryptanalysis of Diffie-Hellman, RSA, DSS, and Other Systems Using Timing Attacks.* Extended abstract. December 7, 1995.

[Pa92] 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.

[Sm94a] S. W. Smith. *Secure Distributed Time for Secure Distributed Protocols.* Ph.D. thesis. Computer Science Technical Report CMU- CS-94-177, Carnegie Mellon University. September 1994.

[Sm94b] S. W. Smith. *A Secure Coprocessor Initiative.* Los Alamos National Laboratory, internal use only, December 1994.

[SmTy94] S. W. Smith and J. D. Tygar. "Security and Privacy for Partial Order Time." *Proceedings of the ISCA International Conference on Parallel and Distributed Computing Systems.* 1994.

[Sm96] S. W. Smith. "Expressing and Enforcing Robust Behavior for Electronic Objects." Los Alamos Unclassified Release LA-UR-96- 1705. *The Federal Networking Council/MIT Internet Privacy and Security Workshop,* May 1996.

[TyYe91] 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.)

[Wa96] P. Wayner. "TILT! Counterfeit Pachinko Cards Send $588M Down the Chute." *The Risks Forum Digest.* Volume 18: Issue 15. May 24, 1996.

[Wein87] S. H. Weingart. "Physical Security for the $\mu$ABYSS System." *Proceedings of the IEEE Computer Society Conference on Security and Privacy.* 1987.

[Yee94] B. S. Yee. *Using Secure Coprocessors.* Ph.D. thesis. Computer Science Technical Report CMU-CS-94-149, Carnegie Mellon University. May 1994.

[YoYu96] A. Young and M. Yung. "The Dark Side of Black-Box Cryptography— or—should we trust Capstone?" *CRYPTO 1996.*