# Secure Coprocessors in Electronic Commerce Applications

Michael Kropfberger

December 18, 2000

## Contents

## Abstract

This paper describes a new way to harden todays security issues: secure coprocessors. We will see general hardware and software needs, two prototype implementations and some profiting applications.

# 1 Introduction

Talking about real life, we all agree on the fact of insecurity of storing money under a pillow, and when we go shopping, we wouldn't hand over a pile of money bills to the counter, and ask the shopping assistant to take what ever he thinks the goods are worth and let him return the rest.
Also in E-Commerce, we have to guarantee on some security basics, which include correctness and trustworthiness of transactions.
This article will present a hardware secure coprocessor, which helps enforcing security starting with turning on the computer. We will see the hardware and software internals, prototype implementations, and E-commerce solutions in different applications.

# 2 Threat Of Unsecured/Untrusted Environments

First of all, we have to outline the main threats we have to eliminate.

## 2.1 Physically Exposed Machines

Physically exposed machines are open to attacks, either by vandals or serious hackers. Those systems might be rebooted with a modified kernel, files might be altered and/or read, and additional back doors for later reentry might be installed. It's also possible to tap the circuitry to gain knowledge of cryptographic keys, passwords or transactions.

## 2.2 Software Flaws

Not only talking about malicious hackers, but also "simple" bugs might cause serious danger to secure and important data. On multitasking systems, buggy applications might write into another programs' address space and might hereby delete some private keys. Also the operating system itself has to be prohibited from writing to arbitrary memory, since the OS has no reason to interfere with cryptographic key storage at all.

## 2.3 Dishonest Employees, Contractors, and Customers

All employees must have access to information and systems in order to carry out their jobs, but often their access cannot be finely controlled. System administrators need the rights to add users, make backups and so on, but this normally includes the tempting possibility to copy sensitive documents stored in clear text. Contractors are even more tempted to sell valuable information to the competitors. With new business models, customers gain access to online product databases and prices, which also might be of interest to competitors.

## 2.4 Backup and Disaster Recovery

System backups have to be made often, and deposited widely spread and even off-site, because of security reasons. But what secures those backups from theft, modification or misuse?

# 3 How Secure Coprocessors Solve Above Mentioned Problems

In figure 1 we outline a generic secure coprocessor. It is covered in a protective shell, which detects physical intrusion and, if that happens, clears the secure memory area. The secure memory area might store secret private keys or personal information. To securely operate with this data, a specially designed operating system with crypto support runs on a built-in CPU.

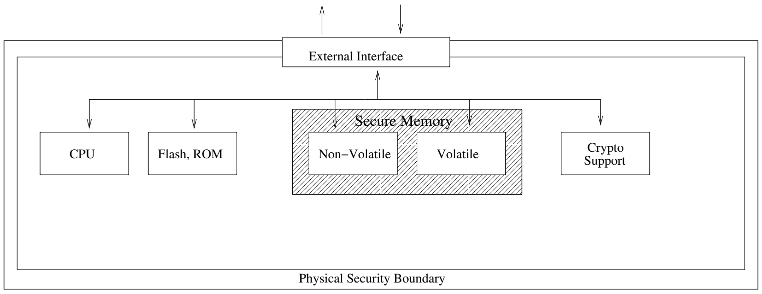


Figure 1: A generic secure coprocessor

## 3.1 Physically Exposed Machines

Trying to physically attack[1] or upload malicious programs onto a secure coprocessor will lead to tamper-response and hereby to the deletion of its secrets.

[1]eg. open and try to tap the circuitry

Point-of-sale terminals are very exposed and endangered by tamper. Find more details in section 9.2.2.

## 3.2 Software Flaws

Separated address spaces for operating system and applications prohibit (read and write) access to other applications' memory by malicious or buggy code. The threat of compromised cryptography can be addressed by hardware support for basic operations like a random number generator or on-chip DES cryptography.

## 3.3 Dishonest Employees, Contractors, and Customers

Because of the extra CPU power and programmability, secure coprocessors can add fine-grained, application-specific access rights and information hiding for both employees and customers.

## 3.4 Backup and Disaster Recovery

Sensitive data might be encrypted before sent to backup, so employees involved with backups only see encrypted data. The needed keys could be stored on numerous secure coprocessors spread around the world, accessible only under special circumstances.

# 4 The Family of Secure Coprocessors

There are many different devices, all embodying various trade-offs between security, computational and cryptographic power, and cost. We normally are only talking about high-end devices, but still, we review the rest.

## 4.1 Chip Card (= Smart Card)

The smart card is the least powerful member, built with a 8-bit CPU, 512 bytes RAM, 8K ROM, 8K EEPROM and a hardware accelerator for cryptographic operations. Programming has to happen at the factory (burnt into the ROM) and – because of space limitations – in hand-tuned machine code. Smart cards lack support of virtual address spaces, and the interface to the outer world has a very low-bandwidth of only 9600 bit/second. Advantages are definitely the price, portability and physical robustness.

## 4.2 Personal Tokens (eg. PCMCIA)

More memory, faster CPUs and higher bandwidth make personal tokens the next step of the secure coprocessor family. Still, most personal tokens are not programmable but (if implemented) give some tamper-responsiveness.

## 4.3 Crypto Accelerators

2 MB EEPROM, 512K RAM and a small microprocessor with cryptographic hardware support make crypto accelerators a good bet for stream cyphering of large amounts of data. Secret keys are stored in secure memory, those "black boxes" are often tamper-resistant, but seldomly tamper-responding, so they are hard to pry open, but won't delete their memory. Programmability increases with special contracts with the manufacturer and sales volumes.

## 4.4 High-End Secure Coprocessor

The most sophisticated device has a powerful microprocessor (eg. Intel 486), megabytes of RAM and FLASH memory, chips to boost cryptographic performance, a real-time clock, a hardware random number generator and offers good tamper-response. It's highly programmable, but still offers separated address spaces and the immediate erasure of secure memory areas when attacked.

# 5 Needed Hardware Features

## 5.1 Tamper-Response

Most discussions about secure hardware talk about "tamper-proof", "tamper-resistant", "tamper-evident" or "tamper-responsive". But, realistically speaking, "tamper-proof" hardware is not achievable, but a secure coprocessor should *respond* to tamper by zeroizing (or somehow invalidate) some stored information.

### 5.1.1 active vs. passive tamper-response

- **Active** The device itself reacts to tamper, but this needs continuous source of power.
- **Passive** tamper-response relies on physical or chemical hardness (and sometimes on explosives)

## 5.2 Hosts

Since a secure coprocessor has to communicate with its host system, there is a considerable question of the interface. Factors are ease of installation, potential platforms and performance. PCMCIA, chip-card and PCI-bus interfaces all give different pros and cons. Also device drivers have to be installed on the host and have to be proven to be secure.

### 5.3 Cost and Durability

Using off-the-shelf parts reduces the price, but might decrease specialization. Eg. chip cards offer a small size, are highly robust and cheap (under a buck!), but lack many often needed features like programmability or good tamper-response.

### 5.4 Exportability

Powerful cryptographic devices often face compliance problems especially with the U.S. export laws.

## 6 Needed Software Features

### 6.1 Development

To write good and secure applications we have to face some challenges:

- Are first programming prototypes possible with a small number of test devices, or do we have to buy large quantities to get manufacturer support?
- Is it possible to program without revealing secure information/code parts to the manufacturer?
- Are there efficient code optimizers, good debugging and testing environments available?
- what secures core device keys, if independent programming is possible?

### 6.2 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, and further, how are software updates done? Possible problems are:

- Is the shipping channel (resellers, mail ...) secure?
- How does the device know what software to accept? (Accepting any software would allow tamper with malicious code)

### 6.3 Software Maintenance

- Are remote and wide-spread software updates possible or is a "Security officer" necessary to update every single device at its current location? And, does the deployer need to maintain a database of device-specific records of secrets?
- How does the owner of the device know, that an upgrade has occurred and fixes a possibly already exploited security hole?
- What happens to stored data during an update?
- What atomicity does the device provide for updates? Are secrets revealed or deleted when a failure occurs? What happens, if the cryptographic software that assures atomicity is updated itself?

To avoid all the above mentioned problems, one might generally forbid updates, but if software flaws occur, way higher hardware and resource costs will be necessary to fix the problems.

### 6.4 Multi-Party Issues

On a high-end secure coprocessor, there runs the operating system, and multiple applications, and it's very common, that they all are maintained by different vendors. We already agreed on separate memory areas, which prevent malicious applications accessing another apps' secrets. But this has also to be true for the operating system itself, so there have to be possibilities to secure and hide some memory even for the operating system.

## 7 Internals of the IBM Research Project of a Secure Coprocessor

The following secure coprocessor [3] was build in the IBM Research Labs and went into a real product. It is a PCI card, which goes into a host PC. It offers all features for high-end secure coprocessors like a Intel 486, megabytes of memory, hardware crypto support, fast PCI interface and is highly tamper-responsive (see section 7.3.1). Figure 2 shows the hardware architecture of the IBM device.

### 7.1 Application Support Architecture

To achieve highest security and to maintain flexibility, there are multiple layers, which are (except layer 0, which is stored in ROM) all exchangeable/updatable (see figure 3). The kernel in layer 2 communicates with special-purpose managers, which themselves communicate with the underlying hardware parts (see

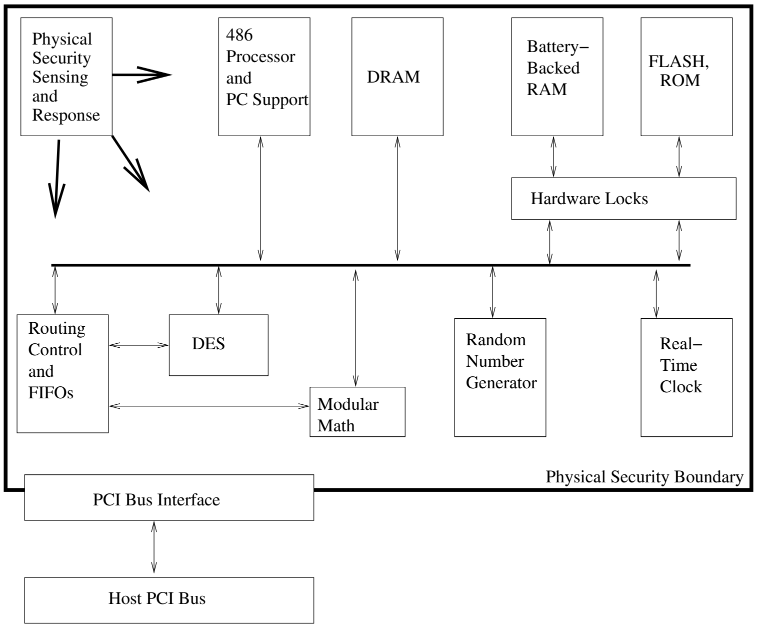


Figure 2: detailed hardware architecture

figure 4). All managers are different modules running in their own address space and were developed separately.

### 7.1.1 Secure Bootstrap

Layer 0, the secure bootstrap, is permanent, so tamper- (but also update-) proof, and initializes all hardware devices properly, before the layer 1 bootstrap part is loaded.

### 7.1.2 Kernel

The IBM people chose CP/Q, a mature OS for industrial embedded systems, because it fulfilled all of the following needs:

- provide separation between address spaces for different computational entities
- provide support for multiple threads of execution, even within the same address space
- provide debugging tools for both user and supervisor code
- work with standard tools (compiler, linker, etc)

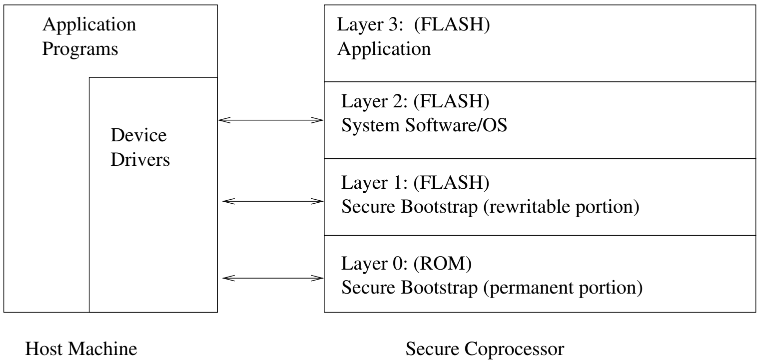


Figure 3: layered software architecture

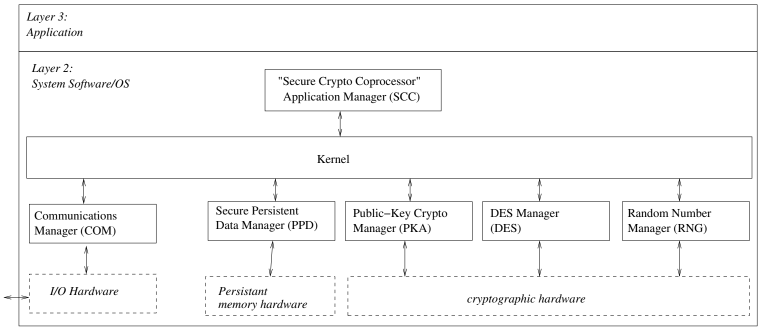


Figure 4: application support architecture within Layer 2

## 7.2 Communications

The COM Manager and SCC Manager set up some application IDs and channels, maintaining an agent table. For fast data movement there are hardware FIFOs available, accessed by the hosts' DMA hardware to transfer data in the background.

## 7.3 Secure Persistent Storage

Applications need storage that persists over hardware reboots, crashes and between the invocations of that application. Three needed characteristics are known:

- **integrity** the stored data will not change due to error or malice
- **secrecy** the stored data has not been revealed to an unauthorized party, including someone who physically attacks the device

- **atomicity** the data changes as an atomic unit, no inconsistent, intermediate states are visible

#### 7.3.1 FLASH and Battery-Backed RAM

FLASH provides large amounts of non-volatile, non-zeroizable memory. It is accessible in 4 to 64 KB blocks and is too slow to react and zeroize memory on tamper. If the FLASH is disconnected from power, there is no way to zeroize its memory.
BBRAM provides small amounts of non-volatile, zeroizable memory. To erase the BBRAM, only the power supply has to be disconnected. So BBRAM holds a session key, which cyphers large amounts of secret data on FLASH.

### 7.4 Cryptography Support

Relying on cryptographic hardware, PKA and DES Manager allow public key cryptography and DES streaming cypher/decipher using the hardware FIFOs and host DMA for data transfer.

### 7.5 Random Number Generation

For the hardware RNG, a thermal noise source generates a serial stream of random bits. Because of performance and even reproducibility for testing (using the same starting seed), there is also a pseudo RNG ([7] and [8]) implemented.

## 8 Internals of the M$ Research Project "Dyad"

In [1], this prototype is discussed in more detail. Online documentation is available under [2]. In this prototype, they also use a secure boot ROM to assure basic tamper-proofness. Using securely stored signatures, Dyad checks and verifies the integrity of the host operating system and later started applications. It supports data encryption to storage media and encrypted communication channels for key exchange, authentication, private key encryption etc.

### 8.1 Secure Coprocessor Architecture

To verify, that the loaded operating system and applications are in the correct version, the secure coprocessor needs to have securely stored checksums to compare. If keyless checksums (e.g. MD5) are stored, secure memory must be write-protected to unauthorized programs. Keyed checksums (e.g. Karp and Rabin's technique of fingerprinting ([6])), authentication, key exchange, private key encryption all also require read-protection to unauthorized programs. But, how much memory is necessary to store all secret data ever needed?

### 8.1.1 Crypto-paging and Sealing

Instead of installing megabytes of memory to the secure coprocessor, one might trade memory to speed. Memory pages are encrypted and then stored on the hosts' physical memory (or to an external disk). To ensure integrity, the paged memory might be crypto-sealed by storing checksums of the pages. When needed, the paged-out memory is brought back into the secure coprocessors' memory, the checksums are compared and then the whole block is decrypted.
This only needs a little space for an encryption key and a data cache, but needs much computational power just for paging.

## 8.2 Secure Coprocessor Software

A secure coprocessor only has to care about resource management, communications, key management and encryption services. It provides separate address space, but no support for terminals, network, disk drives, and other device drivers.

## 8.3 Key Management

Key management is a core feature of the secure coprocessor software. Authentication, key management, fingerprints, and encryption protect the integrity of the secure coprocessor and the secrecy of private data. So, first thing to ensure basic security, the bootstrap loader has to verify the loaded coprocessor kernel before transferring control to it. For kernel updates, the coprocessor has to know keys of following releases and versions. This might be achieved by a cryptographically secure pseudo-random number generator ([7] and [8]) with its internal state entirely in secure non-volatile memory. Anyways, all software updates have to appear transactional, so it must have the properties of *permanence*, *serializability* and *failure atomicity*.

# 9 High-Level Applications

Now we have talked about all the necessary hardware and software features to make secure coprocessors possible, have seen two prototypes, but we didn't present any applications really taking advantage out of secure coprocessors.
The main secure coprocessor feature used by all of the following software application is the fact, that it is possible to store secure data (private keys, session keys, symetric keys) in a real secure place.
Note, that also a chip-card is a simple secure coprocessor, and everyone can imagine a chip-card per each user with his authentication key securely stored on it.

## 9.1 Copy Protection For Software

Commercial software products normally are sold on a per-CPU, per-site or per-use basis. But to enforce this, ther are only some possibilities: Dongles (little hardware devices plugged into the parallel port or USB), CPU-IDs, validity checks via the (insecure) internet or faith in the honesty of all persons accessing the computer. Dongles are exactly for one single software product and they have to be shipped as well, so they are totally infeasible for rental or per-use charging. CPU-IDs have to be compiled into the software. Communication via the internet normally is insecure, sending personal data and statistics is against any privacy ideas and often not possible because of company-wide firewall/security restrictions.
With secure coprocessors, (which could be seen as built-in multi-function dongles), it is easy to store a software key in non-volatile secure memory and encrypt the software binary. This also allows backup of the binary, so even if stolen, the encryption only has to resist plain-text attacks.
Better adapted software may also take advantage of the secure coprocessor and lets some parts of the software run in secure memory. If on-chip memory is not enough, possible solutions would be crypto-paging (section 8.1.1) or better splitting of security-critical and security-uncritical code segments.
This all assumes an untampered device operating system and host OS, which is enforced by device kernel and host kernel fingerprints in the secure coprocessor at boot-up from the layer 0 ROM bootstrap.

## 9.2 Electronic Currency

To efficiently implement e-commerce applications with above mentioned pay-per-use or e-shopping, some kind of electronic currency is necessary. The use of secure coprocessors is necessary, since tampering and hereby generating/deleting money is unbearable to vendors and customers.

### 9.2.1 Electronic Money Models

Basic properties of all electronic money models are well known from databases (the ACID properties):

- **Failure atomicity** If a transaction is interrupted, all results are undone.
- **Permanence** After a successful transaction all changes will remain until changed again by another successful transaction.
- **Serializability** Concurrent transactions will lead to the same final state, ignoring the order of execution or even by running them serially.

**Cash Analogy** Electronic cash can be effectively anonymous, since no centralized authority is necessary. E-cash has serial numbers, so it can't be copied or destroyed, except by national treasuries. The vendors' and buyers' secure coprocessors start an atomic transaction via a secure channel. ACID conform Roll-back mechanisms are enforced.

**Bank Rendezvous Analogy** Like in real life, all transactions run via a centralized "bank" server. This simplifies access controls and managing account data, but is widely open to several problems: very limited scalability, central point of failure/attack (eg. denial of service) and loss of anonymity.

**Credit Card/Check Analogy** Vendor and buyer authenticate and hereby guarantee payment. This needs not only secure authentication keys but also some check-in-advance, finding out, if the money is available at all. This means additional complexity concerning transaction concurrency and communication for bank account statements. All satisfying solutions using secure coprocessors lead back to the cash or bank rendezvous analogy.

### 9.2.2 Point-of-Sale Terminals

Until now we were always describing networked computers with built-in secure coprocessors. We have to trust our host computers, eg. we are transferring secure data on "insecure" data paths like money statements on their way from the coprocessor to the display (eg. you see a $1 battery, but the coprocessor always sees and buys a $10,000 gold watch), passwords typed on "insecure" keyboards (open to tapping) and so on.

When we want to allow secure transactions at point-of-sale terminals, we have no reason to trust the terminals' integrity at all. What we need is a portable device with a secure display, which we can plug into a point-of-sale terminal. Today's smartcards and PCMCIA cards do not incorporate secure displays. Interestingly, to also permit secure input of passwords *and* correct display of eg. purchase information, a secure display suffices.

To allow secure password entry, we generate a cryptographically random string like "QZKNCFLX", and the user has to change each key with the arrow keys up and down to the real password (eg. "SHOELACE", so on "Q" hit the arrow-up twice to a "S", then arrow-left to the next character). This idea is borrowed from [9].

### 9.2.3 Previous Work

Two alternatives without taking advantage of secure coprocessors are DigiCash ([10], [11]) and NetBill ([12]). Both try to implement their protocol without use of extra hardware, relying on current internet security, which leads to either insecure crypto-key usage and/or centralization.

## 9.3 Electronic Contracts

Normally, e-commerce only deals with two-party contracting, but using secure coprocessors and an expressive electronic contract language also allows multi-party contracts for electronic marketplaces or superdistribution. Here every buyer is allowed to resell on the manufacturers' behalf, for at least a minimally higher price, but can't tamper with the product as a reseller.

Secure coprocessors enable the creation of electronic markets not previously possible. For example, for air travel, a user might set up a request for a destination, time and maximum price and sets this up for an auction. Travel agents bid for this request, and buy the right (and obligation) to fulfill such contracts. To continue this thought, the flight tickets might also be cryptographically signed tokens, which are transferred from the flight agency via the travel agent to the customer.

In general, all protected objects might be represented with tokens, stored in secure coprocessors, underlying some minimal transactional guarantees (ACID) and authentication issues. To describe the behavior of those tokens, a expressive description language is needed, offering primitives for authentication, token transfer to other communication partners, time restrictions and deadline constraints, and so on.

## 9.4 Secure Postage

We associate cryptography with securing the contents of a message, maximally also encrypt the sender and/or recipient, to withstand traffic analysis. But sending a physical message involves costs for stamps, so there has to be some security for franking, too.

The US Postal Service has nearly 40,000 autonomous post office facilities, handling over 165 billion pieces of mail annually [13]. Every post office has a postage metering machine. With every letter stamped by the machine, it deducts the appropriate amount off an internal credit counter. Postal meters are subject to at least four types of attack:

1. a tampered credit counter allows to steal postage
2. forged or copied stamps allow replay
3. unauthorized usage of a valid postage meter
4. a postage meter may be stolen[2]

[2] 82,000 franking machines in the U.S. are currently reported as lost or stolen

### 9.4.1 Cryptographic Stamps

Today's technology easily allows digitally readable (barcodes or PDF417 encoding [14]) stamps, which can be printed eg. with normal laser printers. A normal PC with user authorization solves point 3.

If we encrypt identifying information like return and destination address, postage amount, a software serial and sequence number and a timestamp into the digital stamp, the stamp is unforgeable and copying only allows multiple letters going to the same destination (and only, if we don't check for duplicate serial and sequence numbers, see section 9.4.2). Using the timestamp also limits the time frame for sending to the same destination.

### 9.4.2 Detecting Replays

To detect replays, the post office has to store all incoming stamp information in a database, all until expiration. Storing a kilobyte of data per stamp multiplied by millions, makes replay detection seem totally infeasible, but we can exploit the natural distribution of mail delivery and sorting.

In the U.S., mail is first sorted by zip code and sent to regional offices, where it is resorted by carrier route to the local post offices. Since digitally encrypted stamps can't be changed, a replayed copy has to arrive at the same destination as its parent.

Simplifying some assumptions for the calculation, with 80 billion pieces of national mail, distributed over 600 regional offices, and a very generous expiration time of 6 months (7 days are normally the maximum), every regional office sees about 130,000,000 stamps. This would only require less than 130 gigabytes, which is well within the size of a single disk array system. The stamp database can be viewed as a sparse boolean matrix indexed in one dimension by software instance serial number, and in the second dimension by a hash on the stamp sequence number for that software instance.

To make replay detection even faster, we could also check for duplicates at the regional office. It's pretty likely that mail enters the system at the same primary sorting site. To get a grip on mail sent somewhere else, on-line or batch comparison via network connections have to be done. The network issues are based on how fast we have to detect replays.

## 10 Conclusion

After pointing out some of the todays threats, we showed a solution to overcome them: secure coprocessors. We discussed all family members like Chip cards, PCMCIA up to high-end devices with fast CPUs and loads of memory. We also defined the hardware and software needs for high-end secure coprocessors and show two prototype implementations.

Finally we were looking on applications that really gain full advantage from secure coprocessors.
Everybody knows that todays security is unbelievably bad, just because of non-existent user acceptance for additional complications like manually signing emails or storing data on a server into secure containers. All these needed security issues do exist, but have to be hidden to the user, and that's what secure coprocessors should automate.
For us, this means to wait and hope, that every PC is equipped with a standardized secure coprocessor soon, used by adapted mainstream software products like email clients, file systems etc.

# References


[1] Bennet Yee, J.D. Tygar, "Secure Coprocessors in Electronic Commerce Applications"

[2] "Dyad" Homepage: http://www.cs.cmu.deu/afs/cs/project/dyad/www/

[3] Joan Dyer, Ron Perez, Sean Smith, Mark Lindemann, "Application Support Architecture for a High-Performance, Programmable Secure Coprocessor"

[4] Elaine R. Palmer, Sean W. Smith, Steve Weingart, "Using a High-Performance, Programmable Secure Coprocessor"

[5] Sean W. Smith, "Secure Coprocessing Applications and Research Issues"

[6] Richard M. Karp and Michael O. Rabin, "Efficient randomized pattern-matching algorithms" Technical Report TR-31-81, Aiken Laboratory, Harvard University, December 1981

[7] Blum, Blum, and Shub, "Comparison of two pseudo-random number generators", Advances in cryptology: CRYPTO-82, pages 61-79, 1983

[8] Namuel Blum and Silvio Micali, "How to generate cryptographically strong sequences of pseudo-random bits", SIAM Journal on Computing, 13(4):850-864, November 1984

[9] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson, "Authentication and delegation with smart-cards", Technical Report 67, DEC Systems Research Center, October 1990

[10] Stefan Brand, "An efficient off-line electronic cash system based on the representation problem", Technical Report CS-R9323, Centrum voor Wiskunde en Informatica, 1993

[11] David Chaum; "Security without identification: Transaction systems to make big brother obsolete", Communications of the ACM, 28(10):1030-1044, October 1985

[12] Marvin Sirbu and Doug Tygar, "Netbill: An internet commerce system optimized for networked delivered services", IEEE Compcon '95 Coference, pages 20-25, March 1995

[13] U.S. Postal Service, "Annual report of the postmaster general", fiscal year 1991

[14] Stuart Itkin and Josephine Martell, "A PDF417 primer: A guide to understanding second generation bar codes and portable data files", Technical Report TR-31-81, Aiken Laboratory, Harvard University, December 1981