THE TRANSACTIONS OF THE IEICE, V E73(1990)July, No.7, Tokyo, JP
2334 The Transactions of the Institute of Electronics, Information and Comm. Engineers
H04L 23/06
1133
E p. 1133-1146
PAPER <Special Issue on Cryptography and Infomation Security>
G06F1/00N
G07F17/16
G07F7/00C

Superdistribution : The Concept and the Architecture

Ryoichi MORI† *and* Masaji KAWAHARA††, *Members*

SUMMARY Superdistribution is an approach to distributing software in which software is made available freely and without restriction but is protected from modifications and modes of usage not authorized by its vendor. By eliminating the need of software vendors to protect their products against piracy through copy protection and similar measures, superdistribution promotes unrestricted distribution of software. The superdistribution architecture we have developed provides three principal functions: administrative arrangements for collecting accounting information on software usage and fees for software usage; an accounting process that records and accumulates usage charges, payments, and the allocation of usage charges among different software vendors; and a defense mechanism, utilizing digitally protected modules, that protects the system against interference with its proper operation. Superdistribution software is distributed over public channels in encrypted form. In order to participate in superdistribution, a computer must be equipped with an S-box —a digitally protected module containing microprocessors, RAM, ROM, and a real-time clock. The S-box preserves secret information such as a deciphering key and manages the proprietary aspects of the superdistribution system. A Software Usage Monitor insures the integrity of the system and keeps track of accounting information. The S-box can be realized as a digitally protected module in the form of a three-dimensional integrated circuit.

1. Introduction

Superdistribution is an approach to distributing software in which software is made available freely and without restriction but is protected from modifications and modes of usage not authorized by its vendor. Superdistribution relies neither on law nor ethics to achieve these protections; instead it is achieved through a combination of electronic devices, software, and administrative arrangements whose global design we call the "Superdistribution Architecture". The concept was invented by Mori in 1983; it was first called the "Software Service System"[1],[2]. Since 1987, work on superdistribution has been carried out by a committee of the Japan Electronics Industry Development Association (JEIDA), a non-profit industrywide organization. That committee is now known as the Superdistribution Technology Research Committee.

Superdistribution of software has the following novel combination of desirable properties:

(1) Software products are freely distributed without restriction. The user of a software product pays for using that product, not for possessing it.

(2) The vendor of a software product can set the terms and conditions of its use and the schedule of fees, if any, to be charged for its use.

(3) Software products can be executed by any user having the proper equipment, provided only that the user adheres to the conditions of use set by the vendor and pays the fees charged by the vendor.

(4) The proper operation of the superdistribution system, including the enforcement of the conditions set by the vendors, is ensured by tamper-resistant electronic devices such as digitally protected modules.

From a different viewpoint, the needs of users and the needs of vendors have until now been in irreconcilable conflict because the protective measures needed by vendors have been viewed by users as an intolerable burden. The superdistribution architecture provides a resolution to that conflict that serves the interests of both parties. Wide distribution benefits vendors because it increases usage of their products at little added cost and thus brings them more income. It benefits users because it makes more software available and the lower unit costs lead to lower prices. It also creates the possibilities of additional value-added services to be provided by the software industry. Moreover, users themselves become distributors of programs that they like, since with superdistribution there is absolutely nothing wrong with giving a copy of a program to a friend or colleague.

It might seem at first that publicly distributed software such as freeware and shareware already solves the problem addressed by superdistribution. But the likelihood of the authors being paid is too small for public for public domain software to play a leading role in the software industry. Superdistribution software is much like public domain software for which physical measures are used to ensure that the software producer is fairly compensated and that the software is protected against modification. While public domain software might achieve the aims of superdistribution in an idealized world where all users paid for software voluntarily and none of them abused it, we see little hope that such an

Manuscript received February 14, 1990.
Manuscript revised April 17, 1990.

† The author is with the Institute of Information Sciences and Electronics, University of Tsukuba, Tsukuba-shi, 305 Japan.
†† The author is with Master's Degree Program in Sciences and Engineering, University of Tsukuba, Tsukuba-shi, 305 Japan.

Table 1 Levels of software protection technology.

level	key concept	remarks
4	superdistribution	SdA(Superdistribution Architecture) 1983 R.Mori(University of Tsukuba)
3	execution privileges	"Right-To-Execute":ABYSS(A Basic Yorktown Security System) 1987 S.R.White(IBM)[3]
2	customizing software with a computer ID	costomizing deciphering key (software is common) 1986 A.Herzberg PPS(Public Protection of Software)[4] 1984 D.J.Albert (Enciphering and key managemant)[5] 1982 G.B.Purdy SPS(Soltware Protection Scheme)[6]
		customizing each copy of the software
1	hardware protection	controling execution: hardware key, e.g., ADAPSO Key-ring[7]
		inhibiting duplication: copy protection, noncompatible ROM
0	no physical protection	laws & ethics

idealized world will ever come to exist.

Table 1 describes a hierarchy of levels of software protection[3]-[7]. The previous work most similar to superdistribution is the ABYSS architecture[3],[6] developed by White, Comerford, and Weingart at the IBM Thomas J. Watson Research Center in Yorktown Heights, New York. ABYSS is based on the notion of a use-once authorization mechanism called a "token" that provides the "right to execute" a software product. All or part of the software product is executed within a protected processor, and is distributed in encrypted form. Physical security for an ABYSS processor is provided by a dense cocoon of wires whose resistance is constantly monitored by the processor. A change in resistance indicates a likely attempt to penetrate the system. The chief difference between superdistribution and the ABYSS scheme is that superdistribution does not require the physical distribution of tokens or anything else to users of a software product. In other words, ABYSS requires that software be paid for in advance while superdistribution does not.

2. The Superdistribution Architecture

The superdistribution architecture we have developed provides three principal functions:

(1) Administrative arrangements for collecting accounting information on software usage and fees for software usage.

(2) An accounting process that records and accumulates usage charges, payments, and the allocation of usage charges among different software vendors.

(3) A defense mechanism, utilizing digitally protected modules, that protects the system against interference with its proper operation.

In order to participate in superdistribution, a computer must be equipped with a device known as an *S-box* (Superdistribution Box)†. An S-box is a protected module containing microprocessors, RAM, ROM, and a real-time clock. It preserves secret information such as a deciphering key and manages the proprietary aspects of a superdistribution system. An S-box can be installed on nearly any computer, although it must be specialized to the computer's CPU type. It is also possible to integrate the S-box directly into the design of a computer. We call a computer equipped with an S-box an *S-computer*.

Programs designed for use with superdistribution are known as *S-programs*. They can be distributed freely since they are maintained in an encrypted form.

In order to make it acceptable to users, software vendors, and hardware manufacturers, the superdistribution architecture has been designed to satisfy the following requirements:

(1) The presence of the S-box must not prevent the host computer from executing software not designed for the S-box. The presence of the S-box must be invisible while such software is active.

(2) The modifications needed to install an S-box in an existing computer must be simple and inexpensive.

(3) The initial investment required to make S-programs generally available must be small.

(4) The execution speed of an S-program must not suffer a noticeable performance penalty in comparison with an ordinary program.

(5) The S-box and its supporting protocols must be unobtrusive both to users and to programmers.

(6) The S-box must be compatible with multiprogramming environments, since we anticipate that such environments will become very common in personal computers.

In our current design a program can be written without considering the S-box, but the program must be explicitly encrypted before it is distributed if it is to gain the protections of superdistribution. This encryption can be done by a programmer, using the S-box itself.

† The S-box is unrelated to the S-boxes used in the Data Encryption Standard.

The design of a superdistribution architecture can be decomposed into four tasks:

(1) Design of the superdistribution network.
(2) Logical design of the S-box and its interfaces.
(3) Design of the supporting software and file structures, including appropriate cryptographic protocols and techniques.
(4) Design of the protection for the S-box.

Technology for accomplishing each of these tasks is now nearly within to the state of the art. In the rest of this paper we describe our approach to these tasks. We have already designed and constructed two prototype S-computers with their supporting software. Prototype II, the version we are now working with, satisfies the six requirements stated above.

3. Design of the Superdistribution Network

The technical innovations that we describe here are best understood in the context of a network that provides superdistribution of S-programs and the files needed in a user's computer in order to support the use of the superdistribution services. We emphasize, however, that distributing the S-software itself is not a function of the network since a user can obtain that software by any convenient means — from a bulletin board, making a copy of a friend's program, or perhaps purchasing a collection of S-programs at a nominal price. S-programs are stored and transmitted in encrypted form, so the transmission paths need not be protected in any way.

Figure 1 illustrates a typical software distribution system that utilizes the superdistribution architecture. Each S-computer — that is, an end-user computer supporting superdistribution — is equipped with an S-box. The S-box contains a metering program, the Software Usage Monitor (SUM), that enforces the terms set by each software vendor for executing that vendor's products and keeps track of how much the user owes to each vendor. The S-box generates a payment file that contains this information. The fees charged for software usage are measured in units that we call *S-credits*. Payment files are encrypted and transmitted to the collection agency through the network as shown in the figure.

The collection agencies receive the payment files from users, process those files, and transmit payments to vendors — both the authors of the S-programs and the manufacturers of the S-boxes. Each collection agency has an S-box in its computer. The payment files are decrypted and processed under control of this S-box so as to ensure the integrity of payments to the vendors of S-programs and of the supporting hardware as well.

The clearinghouse keeps track of funds transfers engendered by superdistribution. The clearinghouse can be either a new organization or an existing one, e. g. a credit card company, that is prepared to provide the necessary services. The advantage of creating a new organization is that it provides additional degrees of freedom in the systems design. The disadvantage is that it requires a large initial investment. Using an existing organization as the clearinghouse not only saves that investment but also gives that organization the opportunity to enlarge its market.

Identification numbers (ID's) are essential to this arrangement. Each user, each software vendor, each S-box manufacturer, and each collection agency has a unique ID. In addition, users can also have ID's, either as individuals or as organizations. The ID of a user is usually in a different name space than the ID of that user's machine. User ID's provide a convenient mechanism for establishing credit, since they are associated with the individual or organization who is actually

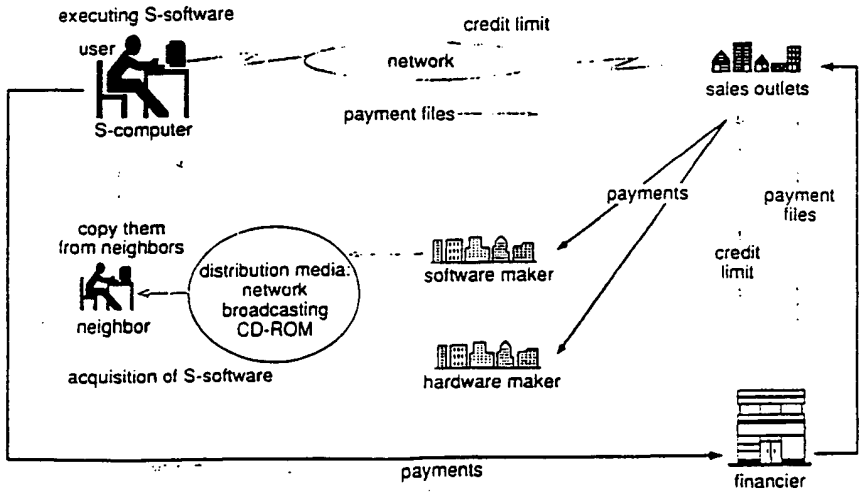


Fig. 1 Example of software distribution system utilizing the superdistribution architecture.

paying for the software. (An organization or even an individual may have more than one machine.) On the other hand, a software distribution system based purely on S-box ID's can provide a high degree of privacy for users.

Requiring prepayment for software usage avoids any risk of default, but such a requirement is unattractive to users and negates the social benefits that we hope to see provided through superdistribution. It is easy to understand why this should be so if we consider how people would react to being required to pay for all their water and electricity in advance.

The superdistribution scheme does not provide any absolute guarantee that users will pay what they owe, or even that they will return the necessary payment files to the collection agents. However, the S-box, which contains a real-time clock, will suspend its services (other than transmitting payment files) if the conditions, which are specified by the vendor or the system, are not met. Transmission can be either over a telecommunication link or with a memory card. In the event that a user transmits the payment files but does not remit the corresponding payment, a collection agent can apply appropriate sanctions.

Superdistribution does not rely on the honesty or goodwill of the manufacturer of the S-box. It is even possible to protect software designed for the S-box against attacks perpetrated by the manufacturer of the S-box. However, the methods of providing such protection are beyond the scope of this paper.

To join the system, a user need only insert an S-box, most likely in the form of a coprocessor chip, into his or her own personal computer. A collection agency can join the system in the same way. The collection agency needs a slightly different form of S-box, one that contains the software for decoding and processing the payment files. Processing at and between the collection agencies and the clearinghouse can be conveniently handled by a Credit Authorization Terminal (CAT) very similar to the ones now in use in many retail establishments.

3.1 Usage of the S-Box

Suppose that I am interested in using some S-programs. I obtain an S-box from an agent either in person or by mail. My agreement with the agent includes a credit authorization or a prepayment of, say, $100. I can now use up to $100 worth of S-programs. When I obtain the S-box, I can also obtain a memory card for it. The memory card is used to record accounting information that describes my usage of S-programs as well as the balance in my account.

To renew my usage, I must communicate the accounting information to the agent. I can do this either by establishing modem communication with the agent or by physically presenting the memory card (if I have one) to the agent. Suppose I use a modem. Then I transmit a message specifying my usage to date of each S-program that I have run. This message is generated by the S-box and enciphered so that it cannot be tampered with by anyone. The agent charges my credit card account for that usage and credits the vendors of the software that I have used. He then transmits back to me an enciphered message that resets my authorization to $100.

On the other hand, suppose that I physically present the memory card either in person or by mail. I can either have it renewed on the spot or (if I choose to mail it in) can have a second card sent to me in advance so that I am never without a card.

3.2 Customer Support and Manuals in Superdistribution

Superdistribution can also be helpful in conjunction with forms of support for users of S-programs that are not yet provided electronically.

Superdistribution changes the environment for telephone support. First, superdistribution eliminates the need to discriminate illicit and authorized users, because the incidence of illicit usage can be kept reasonably low. Second, superdistribution provides a way of charging for telephone support fairly. An S-program can generate a code number, with internal checking, and charge the user for that code number. The user then calls the vendor, asks his questions, and provides the code number. The vendor validates the code number.

Superdistributed software can be supported by hypertext and other forms of electronic documentation, but such documentation is still unlikely to replace printed manuals entirely. Currently, manuals serve both to educate the user and to provide an additional safeguard for the vendor against piracy (since a user can't buy the manuals separately from the software). With superdistribution, the safeguarding function is unnecessary and paper manuals can be sold as freely as books. In particular, a user can obtain several copies of the printed manuals if he wishes, and can even purchase the manuals without purchasing the software at all. Similar remarks apply to manuals distributed on CD-ROMs, which could be used to generate customized versions of the printed manual for particular configurations of the software, the hardware, or other aspects of the working environment.

4. Design of the S-Box and Its Interfaces

The function of the S-box is to execute the Software Usage Monitor in a secure way. The S-box also contains the cryptographic keys that ensure the integrity of the system. These keys are kept secret by storing them within a digitally protected modules (see Sect. 6).

An S-box can be added to an existing computer in any of several ways:

(1) Attaching it to an I/O port.
(2) Connecting it to the bus.
(3) Placing it between the CPU and the bus.
(4) Placing it on the same chip as the CPU or inside the CPU.

Methods (1) and (2) require no modification to the computer; our Prototype I used method (1). This method has the disadvantage that it introduces significant overhead in a multiprogramming environment. Method (4), though more difficult to implement initially, offers the best performance and the lowest cost. Because methods (3) and (4) require modifying the computer, we chose to use method (2), connection to the bus, for Prototype II.

This connection is best realized by making the S-box a coprocessor and using an existing coprocessor socket. Our current implementation uses a second computer connected to the coprocessor socket of the first one; our goal is to fabricate the S-box as a single chip that can be placed in that coprocessor socket. We are using an MC68020-based computer because of its compatibility with other equipment in our laboratory, but adapting the design to other types of processors is not difficult.

All the protection needed to ensure the integrity of the payment files and of the software is concentrated in the S-box. There is no need to protect any of the communication paths shown in Fig. 2. The signals sent over the communication paths are all encrypted, so dedicated networks are not necessary to ensure the security of the system. Any public network can be used, and the added cost of a dedicated network can be avoided. In fact, since the communications links need not be kept constantly active the system is even suitable for use with portable personal computers.

The S-box provides a limited form of protection against viruses, in that programs that are specifically designed to work with the S-box cannot be modified at the user's computer and so cannot be disabled or otherwise taken over by a virus. However, the S-box does not protect against programs, distributed through superdistribution or otherwise, whose effect is on parts of the system unrelated to the S-box.

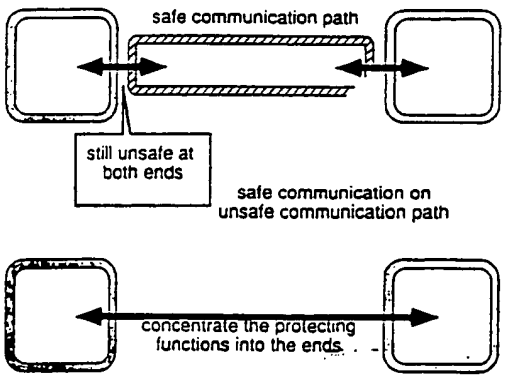


Fig. 2 No protection of communication paths results better performance/cost.

Some of the functions called for may appear to be expensive—in particular, the ability to provide the necessary physical protection and to erase the cryprographic keys recorded in the S-box in the event of an attack. However, these functions can be achieved at reasonable cost if the S-box is mass-produced as a single IC chip (or a small group of such chips).

The workings of the S-box, which we describe below, depend on the existence of cryptographic keys within the S-box. A potential problem could arise were the equipment manufacturing the S-box to be disabled and the values of the keys to be lost. This problem can be solved, although it requires the use of a different kind of protected module.

We have considered but rejected the idea of using simpler memory cards, similar to the farecards used in some transit systems, that would merely record an account balance. There are two problems with such cards. First, separate mechanisms would still be needed to record and transmit the accounting information described above. Second, the devices generally available for reading and writing such cards are insufficiently secure. A cheater could obtain such a device and modify the data on the card without using the S-box at all.

4.1 The Software Usage Monitor

The Software Usage Monitor is executed in the S-box. It performs the following functions:
(1) It monitors the execution of an S-program in order to make sure that it is only executed in the manner intended by its vendor.
(2) It encrypts and decrypts S-programs and other necessary information.
(3) It maintains an account of the charges associated with the execution of an S-program.

An S-program can issue instructions to the Software Usage Monitor. These instructions are realized as extended instructions of the CPU.

In Prototype II we have implemented the Software Usage Monitor through an emulation of the coprocessor functions. We have used an MC68020 board fitted with an MC68881 coprocessor socket, connecting a second computer (the emulator) to that socket through an interface circuit as shown in Fig. 3. The current implementation does not provide for multiprogramming because the Apple Macintosh we are using, to which an MC68020 board is plugged in, does not yet have that capability.

Further limitations on Prototype II are that it does not provide encryption and that it is not encapsulated as a digitally protected module. We consider those limitations acceptable because the purpose of our experiments at this stage is to validate the execution control and

charge control functions of the Software Usage Monitor.

4.2 Architecture of Prototype II

Figure 4 shows a conceptual diagram of the architecture of Prototype II.

- The MC68020 processor executes the S-program and communicates with the coprocessor. The coprocessor ensures the legitimacy of the execution.
- The interface section communicates with the MC68020 according to the standard MC68020 coprocessor protocol[9],[10].
- The processing section interprets and executes the coprocessor commands issued by the MC68020.
- The execution control section monitors the execution of the S-program.
- The accounting buffer contains pricing information for each currently executing S-program as well as accumulated usage information. The pricing information in the accounting buffer is a subset of the information in the tariff section of an S-program (see Sect. 5.1).
- The payment file contains a record of payments.
- The charge control section updates the S-credit balance, utilizing the information in the accounting buffer and the payment file. It also keeps a record of access violations and halts execution of the S-program if the number of violations exceeds some threshold. (Our experience suggests that access violations occasionally happen by accident, so this threshold should be nonzero). The charge control section also implements functions required to support the user utility program.
- The cryptography section performs three functions:
(1) It decrypts the information in the tariff section of the S-program and enters that information into the accounting buffer. It does this as part of the process of loading an S-program into the S-box. Either a public-key or secret-key cryptosystem methods can be used.
(2) It decrypts the encrypted portions of the S-program itself, using in this case a high-speed conventional cryptosystem.
(3) It encrypts arbitrary programs upon request. This facility is necessary so that a programmer can create new S-programs without outside assistance. In particular, no assistance is needed from the manufacturer of the S-box.

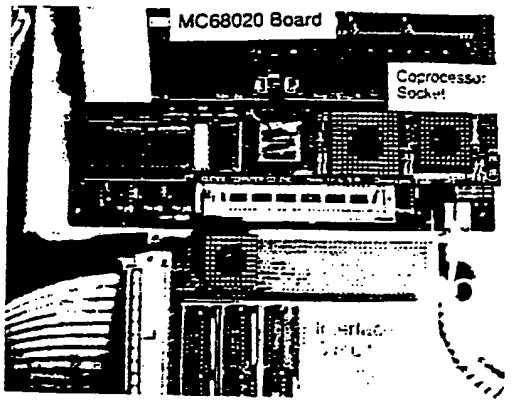


Fig. 3 Implementation of Prototype II.

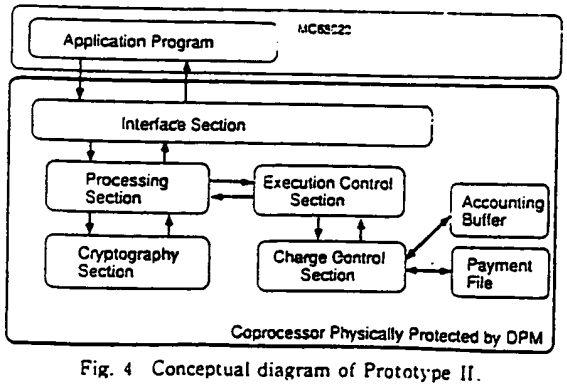


Fig. 4 Conceptual diagram of Prototype II.

5. Design of the Supporting Software and File Structures

5.1 Structure of an S-Program

The structure of an S-program as implemented in Prototype II is shown in Fig. 5. The pricing information for a S-program is contained in its tariff section. When execution of the S-program is initiated, the coprocessor verifies the information in the tariff section and copies the pricing information into the accounting buffer of the S-box. A verification routine, described below, is embedded in the body of the S-program. Execution is aborted if the verification routine detects any inconsistency in the tariff section.

The tariff section includes the following:
(1) Software ID: an identifier unique to the S-program.
(2) Access control key: A key used by the authentication routine in the S-program body. This key is checked by the coprocessor.

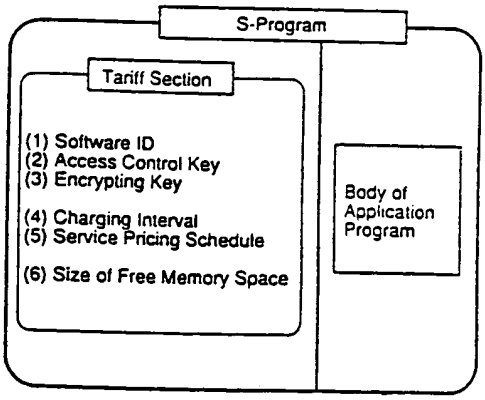


Fig. 5 Structure of S-program.

(3) Encrypting key: A key used to encrypt the machine instructions in the body of the S-program. An encrypted block of instructions is transferred to the coprocessor, where it is decrypted using this key and then executed.
(4) Charging interval: the length of execution time for which a single S-credit is charged to the user.
(5) Service pricing schedule: the number of S-credits charged to the user each time a particular software service, e. g. saving a file, has been rendered and successfully completed.
(6) The amount of memory required within the coprocessor for executing the S-program.

Note that items (4) and (5), which together constitute the pricing information, allow for two styles of charging: per unit of time (by setting (5) to zero) or per execution (by setting (4) to zero). Combinations of the two styles are also possible.

5.2 Execution of an S-Program

Execution of an S-program is monitored by the program and the coprocessor working together. The coprocessor halts execution whenever it detects a protocol violation. For proper protection, the S-software must be structured so that analysis of it by an intruder is not possible. Some methods of achieving this are:
(1) The coprocessor receives a number from the S-software and compares it with an encrypted value, returning the result of the comparison.
(2) The coprocessor receives data from the S-software and returns a jump address.
(3) The coprocessor verifies that a particular message is sent by the S-software within a prescribed period.
(4) The coprocessor receives a block of encrypted MC68020 instructions and executes them, placing the results of the execution in the registers and/or the memory of the MC68020.

These methods can be used individually or in combination.

The sequence of events that takes place as an item of S-software is executed is as follows:

- Initialization. The pricing information is sent to the coprocessor by the initialization routine when execution of the S-software commences. This can be done using general purpose coprocessor instructions[9],[10]. S-credits are then decreased periodically until execution of the S-software is terminated.
- Execution. The check routine in the software body is executed, and execution of the software is aborted if the prescribed protocol is not satisfied. Methods (1)-(3) of execution control as described above have been implemented. The implementation techniques are as follows:

—For (1), the software ID, the access control key, and a random number are sent to the coprocessor. The random number can be encrypted, although we have not yet done this. The result of comparing the random number with a known value is then returned to the processor.
—For (2), we transfer a jump address table to the coprocessor during initialization. This table can subsequently be referenced by the S-software. The coprocessor then returns the appropriate jump address.
—For (3), the access control key, the range of acceptable response times, and a random number are sent to the coprocessor. The range is represented as a lower limit and an upper limit. The coprocessor then checks that the time of the next message is within the limits. If it is not, the coprocessor aborts execution of the S-software by interrupting the main processor.

- Termination. The coprocessor is informed when the S-software has completed execution. This can be implemented using general purpose coprocessor instructions. At this time the payment file is updated to account for the usage.

5.3 Supporting Files

The *payment file* keeps track of S-credits that are owed as the result of software usage. Each record in a payment file ordinarily contains three fields: a payer, a receiver, and an amount. The collection of payment files is an essential function in a superdistribution system in order to make sure that revenues are properly distributed to the providers of services.

The *privilege file* records a user's privileges with respect to software usage. For example, if a user has purchased a program then the user has the privilege of using it thereafter without paying any additional fees. Another kind of privilege is a volume discount: after a certain amount of usage, the charging rate is decreased.

Carryover is a form of discount in which the money that a user pays for usage of a program is deducted from its purchase price if the user chooses to purchase the program at some later time. Carryover can be accommodated by associating a carryover parameter c with a particular program, where c is no greater than 1. Should the user decide to purchase the program, a credit of c times the accumulated usage charges is deducted from the purchase price. If a vendor chooses to set c to 1 for a particular program, then a user of that program need not make any decision as to whether or not to purchase the program. Carryover, if provided, is also recorded in the privilege file.

Some forms of superdistribution can be realized without having a privilege file. In these forms the charges are based on services provided, e. g. 10 credits for each program execution, 2 credits for each file save, 5 credits per printout, and 1 credit per 18 seconds of execution time.

The privilege file grows with time, since information is added to it but is rarely if ever deleted. Moreover

it must be preserved indefinitely. It is ordinarily the obligation of a software vendor to provide backup for the portion of the privilege file pertaining to that vendor's software.

A record in the privilege file is ordinarily generated at the same time as a record in the payment file and stored in the user's machine. In some environments in which users send payments separately, the vendor may generate records to be placed in the privilege file, but this arrangement is usually less convenient for users.

The portion of the privilege file pertaining to a particular vendor's software can be kept either in the user's machine or by the vendor. If privilege records are generated in one place and stored in another, then the superdistribution system must make allowances for delays in transmission and in confirmation.

If a superdistribution system provides for trial usage of software either free or at a discount, it must keep records of that usage in a *trial usage file*. Otherwise a user could use software indefinitely on a trial basis. These records need to be backed up, although backup is less critical than it is for the privilege file. The reason is that trial usage is not an obligation but rather a gratuity on the part of a vendor, and a vendor can reserve the right to cancel the privilege of trial usage at any time.

The *benefit file* keeps track of special benefits granted to the user by various vendors. The benefit file differs from the privilege file in that the information in the benefit file has no direct correspondence to the information in the payment file and is unaffected by what software is used or how much it is used. Examples of information in the benefit file are a certification entitling the user to educational discounts or a secret number entitling the user to run a certain program not available to the general public.

The *account file* is an aggregation of the particular files we have just discussed: the payment file, the privilege file, the trial usage file, and the benefit file.

A superdistribution system needs to provide secure backup for its accounting files and for the privilege file in particular. It need not make any special provision for backing up software since a user can back it up by any convenient means or get another copy without cost if the original copy is lost.

The backup requirements for the privilege file depend on whether or not privileges can decrease with time. If not, it suffices to ensure that the process of restoring the privilege file from its backup cannot generate additional privileges. This requirement can be met by associating an ID and a randomly generated validation number with each privilege file and encrypting the file before backing it up. The restoration process checks to make sure that the decrypted validation number is correct and that the backup file is indeed a proper one.

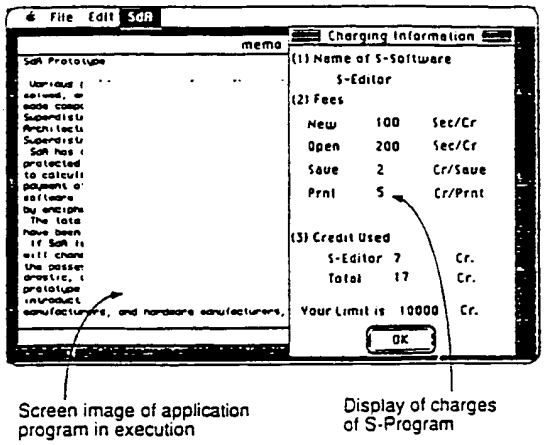


Fig. 6 Example of execution of user utility program.

5.4 Implementation of the User Interface

Since a user of an S-computer should be able to ascertain the status of his account at any time, we have built a user interface for retrieving and displaying that information. The user interface program provides two functions:

(1) Transmitting the payment file
(2) Displaying the accumulated charges for S-programs.

In Prototype I we implemented the user interface within the Software Usage Monitor; in Prototype II we have implemented it as a separate program. By making it a separate program we reduce the size and complexity of the software that needs to be permanently resident in the S-box. Prototype I allowed a great deal of flexibility in the way that usage was charged; Prototype II has adopted a simpler system based on a combination of charges for performing specific actions, e.g. opening a file, and charges per unit of time.

An example showing the execution of the user utility program is shown in Fig. 6. The user can inspect the record of software charges, the remaining credit balance, etc., through this program. Although Prototype II does not have a general multiprogramming capability, it does allow the user utility program to be executed while an application program is also executing.

6. Digitally Protected Modules

6.1 Design of the Digitally Protected Module

A protected module (PM) is a container for information that protects that information from attacks intended to read or write it in an unauthorized manner. Protected modules have also been described in the literature as "tamper resistant modules" or "protected processors". We prefer "module" to "processor" because we

are not concerned with protecting the hardware itself, but only with protecting the information being processed by the hardware.

The design presented here assumes that the S-box is implemented as a digitally protected module — that is, a module whose protective logic is based on digital technology. We plan to use three-dimensional integrated circuit technology, since we believe that a high level of physical protection can eventually be achieved by applying fabrication methods such as chip bonding to three-dimensional circuitry.

Increasingly effective levels of protection are possible:

(1) A protective method may rely on the ignorance or lack of resources of the attacker. For example, software can be stored in ROM in a form where access is difficult and does not follow the usual conventions of the computer to which it is attached.

(2) A protective method may rely on keeping a permanent record of attacks rather than on withstanding those attacks. If the proprietor of a protected module can retrieve this record, then attackers will be dissuaded by the knowledge that their activities will be discovered. Examples of devices using such methods are the counters on rented copying machines, the odometers on rented automobiles, and the electric meters placed on the main power lines of buildings.

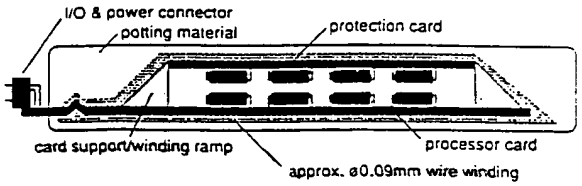
(3) A protective method may be designed to defend a protected module even when the proprietor of the module has no access to it, either physically or logically. Since no physical device can be expected to resist destruction indefinitely when the attacker has sufficient time and resources, a device using methods of this kind must be able to detect attacks and erase its contents before the attacker can retrieve them. Then an attacker is simply left with a broken module and has gained nothing.

(4) A protective method may be designed to defend a protected module even from attacks by its manufacturer. Since in some cases a protected module may contain information that the module's manufacturer is not authorized to access, this case is also of interest.

The device we discuss in this article is intended to achieve the third level of protection above, although we believe that it is even possible to achieve the fourth and highest level of protection.

Most earlier research on protected modules has either been theoretical in nature or has been subject to military classification. The μABYSS module[8] is an exception. Using this approach, an entire circuit board is enclosed in a cocoon of nichrome wire of approximately 0.09 mm diameter. Several windings, each having thousands of turns, are used. The circuit board records the protected information. The resistance of the windings is monitored in analog fashion to detect attacks (see Fig. 7).

In μABYSS, a 5 % change in resistance of the wire is taken as a threshold to determine whether an attack is taking place. This comparison is made with the original resistance value. A change of at least 1 % from the most recently measured value is also taken as an indication of an attack.



Source: Steve H. Weingart, IBM Thomas J.Watson Research Center

Fig. 7 Physical structure of μABYSS.

6.2 Operation of the Digitally Protected Module

One of our approachs is illustrated in Fig. 8. The protective mechanism consists of a random access memory (RAM) containing a great number of closely-spaced microscopic one-bit detectors. The DPM contains one or more layers of these detectors, packed tightly so as not to leave enough room for penetration between them. Layers of detectors can be staggered so as to increase the effective density. The memory can be implemented using either IC's or printed circuits. With these technologies the center-to-center distance of adjacent detectors can be made as small as 3 μm. The detectors are individually addressable, and each one can be read or written. A possible attack is indicated when any detector is found to have lost its normal operating functions.

The testing method is simple: write a random value into the detector and read it back. Then write the complement of that value into the detector and read it back. The element is considered to be working if and only if the retrieved values agree with the written values. If the values do not agree, a possible attack is indicated. We call this event an exception.

If an exception occurs, the microprocessor examines the faulty detector several more times. Repeating the test provides protection against random transient failures such as might be caused by alpha rays. The probability of such a failure and the consequent change of state is proportional to the time between writing and reading. If a detector is read immediately after it is written, the probability becomes very small. In some environments the probability of transient errors may be low enough so that transient errors can be disregarded altogether.

If an exception is judged not to be a transient error, then the DPM can test other detectors in the physical vicinity of the erroneous one. The number of faulty elements can then be compared to a threshold. If that threshold is exceeded, we assume that the DPM has been attacked. The threshold test reduces the chance of a

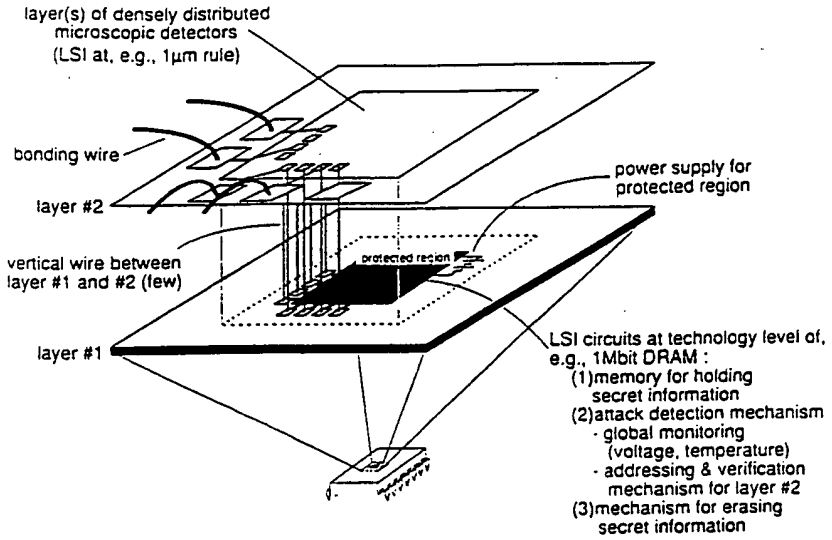


Fig. 8 Three-dimensional IC implementation of a digitally protected module.

false alarm.

Techniques that apply to RAM used in other applications can also be used for the DPM. For instance, the microprocessor can be provided with a list, resembling the list of bad tracks on a disk, that indicates elements known to be faulty. Similarly, it is possible to read and write blocks of several bits in parallel instead of operating on a single bit with each test. However, some economies are possible for the DPM that are not possible in general. For instance it is not necessary to refresh the memory because it can hold its state for at least ten milliseconds without being refreshed. That interval is much greater than the interval between writing and reading a bit. Such economies can help to reduce the cost of manufacturing the DPM.

The design of any type of protected module must address an inherent conflict. In order to make the device more likely to detect an attack it must be made more sensitive; but the more sensitive it is, the greater the likelihood of a false alarm. The more information that can be gotten about a presumed attack, the easier it is to resolve this conflict.

The digital design can provide more information than an analog design in the event of a fault because the digital design has more discrete components. In an analog design such as μABYSS there are relatively few windings. If a resistance measurement is off, there is no way to localize the fault further. In a digital design it is possible to check the detectors in the vicinity of the faulty one. The analog design can be brought closer to the digital one by using many small windings, but the control logic then becomes far more complex and it becomes much harder to manufacture the device.

We next consider various kinds of potential attacks on a digitally protected module and the protection against them:

(1) An attacker might attempt to separate the detecting layer and the protected region. Such an attack can be detected by monitoring the continuity of the interlayer wires.

(2) An attacker might aim at the detection mechanism itself rather than at the information being protected. This strategy can be foiled by placing the detection mechanism in the protected region.

(3) An attacker might attempt to manipulate the wires that connect the circuitry to the outside world. Such attacks can be prevented by a structure where the wires pass through the detecting layer, as described in the next section.

(4) It is possible to reuse a digitally protected module that has erased its cryptographic keys in response to an apparent attack, but caution is necessary in order to guard against Trojan horses or other attempts at subversion of the logical protection.

The protective capability of any protected module cannot be regarded as permanent. Once a module has been installed the hardware implementation of its defensive technology is fixed, even though the technology of attack advances continually. With time it becomes easier and easier to find ways of bypassing or otherwise neutralizing whatever defenses the module utilizes. Thus the useful life of a protected module is determined by the progress of the technology of attacks. A useful life of five years is probably not difficult to achieve. Twenty years would be desirable; ten years is probably about the longest time that we would consider practical. However, it is likely that the state of the art in attacking protected modules would be known to their proprietors, and the modules could be replaced when they are known to have become vulnerable.

6.3 Realization of Digitally Protected Modules

Three-dimensional large-scale integrated circuits[11], possibly manufactured using chip-bonding methods, provide an excellent method of realizing digitally protected modules. We have assumed such an implementation in Fig. 8.

Layer 1 consists of a protected region where the information to be protected is stored. The memory for the information can be implemented using technology at the level of 1 Mbit DRAMs. The protected region either has its own power supply or uses an external power supply. It is possible that there is more than one power supply. The protected region also contains circuitry whose function is to erase the information if an attack is detected. The erasure is accomplished by cutting off the power to Layer 1.

An attack might involve making holes in the device, etching it, or subjecting it to electro-optical reading. Layer 2 is made of devices and wirings designed to detect these kinds of attacks. It will be constructed using a technology such as 1-2 μm rule semiconductor devices.

The distance between Layer 1 and Layer 2 can be made sufficiently small, on the order of 3 μm, so that an attacker will not be able to penetrate between the layers. An attack from below Layer 1 is not feasible because any attempt to reach the components from that direction will cause the components to lose their charge. Should penetration from below be a concern, it is possible to place a layer of detectors there too, thus enclosing the protected region in a sandwich of detectors.

The wires between the two layers serve two purposes: to provide signal paths between the circuitry in the layers and to provide signal paths from the protected layer to the outside world. These wires are connected to small pads on the inner surface of each layer. They cover the entire perimeter of the DPM. Any disruption of their signals will have the same effect as a disturbance to the detectors, so they provide an additional level of protection to the device.

The larger pads on the upper surface of the detecting layer are bonding pads like those of a conventional IC. Their purpose is to provide connections for external wiring. Although external connections are logically necessary only for the protected circuits in Layer 1, the wires are routed via Layer 2 in order to prevent an attacker from gaining access to the protected region via the external connections.

The DPM can be encapsulated using an appropriate resin, preferably one that is not transparent. By mixing a substance such as alumina with the resin we can increase the resistance of the device to attacks using mechanical or chemical methods, or even attacks using lasers or plasmas. Simpler measures may, however, suffice for some environments.

The RAM in Layer 1 depends on a continuous supply of power to maintain its state. If the protective logic of the DPM concludes that an attack has taken place, it grounds the power supply and erases the secret information in the protected region. It is pointless for an attacker to disrupt the power supply because the effect of doing that will be to erase the protected information.

The DPM is not a demanding application of three-dimensional IC technology. The number of lines of inter-layer wiring is small, particularly in comparison with typical applications of three-dimensional IC's. For example, an image processor would usually require several interlayer connections for each pixel. An image processor capable of handling large images would thus need thousands of inter-layer lines. Far fewer lines are needed for the DPM: power supplies, inputs, outputs, and addresses.

Furthermore, the DPM is a general-purpose device that will be produced in quantity. Mass production of DPM's will lead to economies of scale, just as it has with memories, microprocessors, and personal computers.

6.4 How to Detect Attacks

A likely form of attack is to attempt to make a hole in the device. Because of the spacing of the detectors, a hole whose diameter is as small as ten microns will still destroy at least one detector, and a hole not larger than twenty microns will destroy at least four. The design of the DPM makes it possible to determine not only that four detectors are faulty, but that the four are all in the same region.

The DPM has several different memories, and these memories have distinct purposes. The memory in Layer 1, the protected region, stores the information that is to be protected. The memory in Layer 2 is devoted to detecting attacks.

The order in which the detectors are tested is arbitrary, but the total test duration must be kept short. The detectors can be tested individually or in groups. Group testing helps to keep the testing interval short because the detectors in a group can be tested in parallel.

If we have a million elements and test one element every microsecond, then it will take just one second to test all the elements. If we test in parallel, the testing time can be made much less than that. The time required to erase the protected information is negligible by comparison.

There are at least three levels of protection that can be provided for the information in the protected region:
(1) The lowest level of protection is provided by using the same physical structure in every DPM. The protected information is represented by the presence or absence of electrical charges at particular physical locations in the structure.
(2) The second level of protection is provided by

storing the protected information in a way that is different for each individual DPM. One way of doing this is as follows: Each module contains a random number, which is generated independently for each module. The information is encoded in such a way that it can be decoded by using the random number. Full cryptographic protection is not necessary for this encoding; even such a simple method as exclusive or'ing the stored information with the random number is sufficient.

(3) The third level of protection is provided in a similar way as the second, but the random number and the decoding function change with time and are not always stored in the same location.

The choice among these levels should be made according to the defensive strength that is required and the cost that is acceptable.

In some applications a less effective method of protection may suffice. The protected information is kept on a mask ROM, and the chip containing the ROM is encapsulated in resin. For applications in which the attackers are not expected to be sophisticated, or in which the protecting device can be inspected by the owner of the protected information, this method may be appropriate since it is simpler and cheaper. Further discussion of it, however, is outside of the scope of this article.

6.5 The Protected Region

Layer 1 of the DPM is the protected region. Layer 2 of the DPM covers the protected region.

The purpose of the detectors is to insure that nobody can physically invade the protected region without destroying one of the detectors. However, if it is known that physical invasion will not result in the theft of the protected information, the invasion can be tolerated.

It is highly desirable to use RAM rather than EPROM for the memory in the protected region, at least in the initial stage of commercial production of the DPM. The advantage of RAM over EPROM is that RAM can be erased almost instantaneously by cutting off its power. EPROM retains its information even when it is not powered.

Near-instantaneous erasure is particularly important because the protected information typically includes an encryption key, and that key will be common among many DPM's. If the key in one DPM is stolen then the information in any device using the same type of DPM becomes vulnerable. Because the rewards of a successful attack are so great, we can expect the DPM to be subject to thorough, well-planned, and prolonged attacks. The explicit action required to erase EPROM makes it far more susceptible to attack.

At very low temperatures, a RAM may hold an electrical charge for some length of time even without power. Therefore the protected region can contain a monitoring circuit that will erase the secret information should the temperature drop below a specified threshold.

The DPM requires a continual supply of electrical power, whether it is constructed using RAM or EPROM. Once the DPM has been loaded with its protected information, power is needed for the memory that contains the secret information, the addressing logic, and the testing circuitry.

6.6 The Power Supply

Maintaining the power supply is not necessary for protection, since if the power is disrupted the device simply loses its information. It is necessary, however, to insure that in the event of a power loss the protected information is erased before the protective mechanism itself fails. This can be achieved by having the protective mechanism monitor the power supply and grounding it entirely if its voltage is not within acceptable limits.

Nevertheless it is very important to protect the device against accidental loss of power, since such an event can lead to the loss of cryptographic keys, and will be a great inconvenience to the user. Thus it is very desirable that the protected region have its own power supply. Using current CMOS and battery technology, we can build power supplies that have a lifetime of ten years. As we noted above, the useful lifetime of the DPM is unlikely to be longer than that because of the progress we can anticipate in the technology of attacks.

However, it is also possible to use an external power source provided certain precautions are taken. An external backup battery must be connected at all times once the device has been loaded with its information. It is possible to safeguard against momentary power loss by attaching a condenser to the power line. This condenser can be integrated into the device or external to it.

It will still be necessary occasionally to replace the backup battery. A device such as an electromagnetic plunger can be used to prevent the backup battery from being removed accidentally when the external power line is not connected.

7. Current Status and Future Plans

Prototype I (Fig. 9)[2] has been operating in our laboratory for three years and is available for demonstration. The S-box of Prototype I is connected to a NEC 9801 personal computer through an RS232C interface, but any other personal computer could be used. Prototype I was constructed when we were using the "Software Service System" name for these ideas.

Prototype II (Fig. 10) is now working. We expect to fabricate the S-box chip during 1990. We further plan to demonstrate the chip and distribute a limited number of copies of it to groups that will test it. We then hope

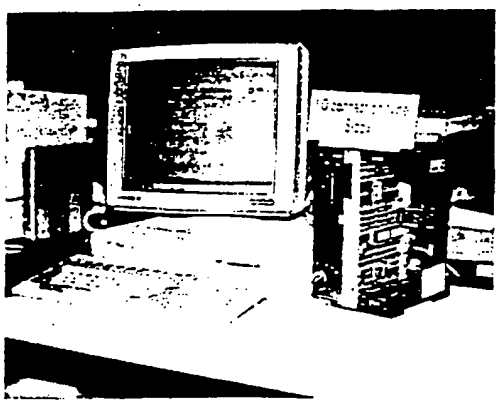


Fig. 9 Prototype I for superdistribution architecture.

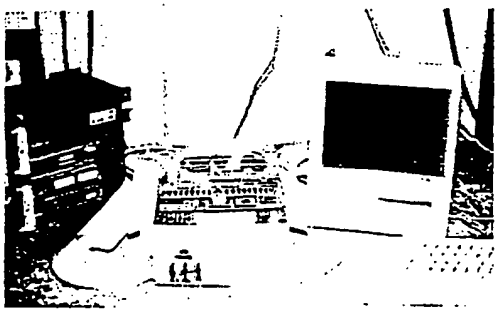


Fig. 10 Prototype II for superdistribution architecture.

to make available sufficient technical information so that others can design and produce S-boxes with the same function and purpose as ours. These S-boxes would be coprocessors for the 68020, 68030, 80286, and 80386 chips. The coprocessor implementation is clearly superior to the free-standing implementation because it has less overhead, would make multitasking possible, and is nearly invisible to users.

Prototype II does not provide encryption, nor does it physically protect the coprocessor against attacks. However, we anticipate no difficulty in using existing ASIC technology as embodied in encrypting units using established cryptographic technology to satisfy all the requirements of Section 1 of this paper.

The accounting algorithms of Prototype II are simple—simpler, even, than those of Prototype I. We are now investigating methods by which various charging methods, e. g. outright purchase and free trials, can be gotten by having a collection agent process the payment files.

The extension of these methods to multiprogramming is not difficult, given appropriate hardware. The MC68020 coprocessor interface has save-restore instructions for the coprocessor status, and these can be adapted to halt or resume processes within the coprocessor. It is also possible to disable interrupts in the main processor while the coprocessor is executing instructions, provided that the coprocessor can return control within a suitably short period of time. The time can be kept short because it usually requires only one general purpose coprocessor instruction to carry out the authentication required by the checking routine.

We have implemented Prototype II using the MC68020, but the interface section of Fig. 4 is the only part of the system that depends on the coprocessor interface of the MC68020. Thus our architecture can be adapted to other MPU's just by changing the interface.

Our work on Prototype II has verified that superdistribution can be achieved without a large initial investment, and reinforces our belief that a production system will be feasible in the near future. Our future plans include the following:

(1) Investigating the details of integrating the entire system.

(2) Systematic investigation of the software for the emulating computer.

(3) Fabrication of the coprocessor chip that provides encryption and decryption functions using ASIC technology.

(4) Distributing the coprocessor chips to other experimenters in order to verify that the system functions as expected and to learn from the experiences of others.

Although we have directed our work towards superdistribution of computer software, our architecture can be applied to other forms of digital information as well. One example is music in the forms that it is recorded on digital audio tapes.

8. Conclusions

We have developed an implementation for one step in the realization of a superdistribution architecture. This implementation has been achieved by inserting an emulated coprocessor containing the Software Usage Monitor into the coprocessor socket of an existing computer. Most personal computers already have such a socket in order to support a floating point coprocessor. It is possible to construct a single chip that combines the S-box with a floating point coprocessor, thus avoiding conflicts over the use of the socket. For those computers lacking a coprocessor socket, a special-purpose board inserted into an expansion slot can be used instead. In this case software is needed that simulates the coprocessor interface, but the overhead is not excessive.

Although we have described our system in terms of application programs, it can also be applied to system programs as well. Our methods, moreover, can be adapted to a virtual computer system that supports multiple operating systems, an arrangement that is likely to be supported in personal computers of the future.

Containers for tangible items reflect the value and importance of their contents. No similar containers have existed for digital information despite the progress of microelectronics and communications networks. The lack of such containers has invited problems such as

computer viruses and software piracy.

Digitally protected modules promise to provide such containers and thus to become one of the most fundamental applications of integrated circuits in the information society. We have viewed them as a fundamental support for our work on superdistribution, which aims to provide unrestricted distribution of commercial software.

Acknowledgement

We wish to thank Dr. Paul Abrahams, Consulting Computer Scientist, Deerfield Massachusetts and Professor Takashi Tokuyama, University of Tsukuba for their advice on this paper.

References

(1) R. Mori and S. Tashiro: The concept of 'Software Service System (SSS)'", Trans. IEICE, J70-D, 1, pp. 70-81 (Jan. 1987).
(2) S. Tashiro and R. Mori: "The implementation of a small scale prototype for Software Service System (SSS)", Trans. IEICE, J70-D, 2, pp. 335-345 (Feb. 1987).
(3) S. R. White: "ABYSS: A trusted architecture for software protection", Proc. 1987 IEEE Symp. on Security and Privacy, Oakland, CA, pp. 38-51 (April 1987).
(4) A. Herzberg and G. Karmi: "Public protection of software", in Advances in Cryptology; Proc. Crypto 85, ed. H. C. Williams, p. 158 (1986).
(5) D. J. Albert and S. P. Morse: "Combating software piracy by encryption and key management", IEEE Computer, pp. 68-73 (April 1984).
(6) G. B. Purdy, G. J. Simmons and J. A. Studier: "A software protection scheme", Proc. 1982 Symp. on Security and Privacy, Oakland, CA, pp. 99-103 (April 1982).
(7) "Proposal for software authorization system standards", Ver. 0.30, ADAPSO, Association of Data Processing Service Organization, (1985).
(8) S. H. Weingart: "Physical security for the μABYSS system", Proc. 1987 IEEE Symp. on Security and Privacy, Oakland, CA, pp. 52-58 (April 1987).
(9) MC68020 32-Bit Microprocessor User's Manual, Motorola Inc. (1986).
(10) MC68881 Floating-Point Coprocessor User's Manual, Motorola Inc. (1988).
(11) Y. Akasaka and T. Nishimura: "State of the art of three dimensional ICs", J. IEICE, 72, pp. 1417-1421 (1989).

Ryoichi Mori was born in 1929. He received the B. E. and Ph. D degrees from The University of Tokyo in 1953 and 1962, respectively. In 1959 he joined ETL, MITI then moved to the Institute of Information Sciences and Electronics, University of Tsukuba, in 1978, where he is a professor. He has been a Chairman of the Microcomputer Technology Committee of the Japan Electronic Industry Development Association since 1974. He is a Director of Euromicro. He received the Silver Core from IFIP in 1986.

Masaji Kawahara was born in Kagawa, on April 9, 1962. He received the B. S. degree from University of Tsukuba, Tsukuba, Japan, in 1985. From 1985 to 1988, he joined Fujitsu Program Laboratory Ltd., Japan. He is now the first year graduate student of the master cource of University of Tsukuba.