# Protected Shareware:

# A Solution to the Software Distribution Problem

by
Ralph C. Merkle
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304
merkle@parc.xerox.com
www.merkle.com

## Abstract

Shareware is abstractly a good concept but works poorly in practice because shareware author's can't enforce their terms. We present a combination of inexpensive hardware, cryptographic protocols, and internal software integrity checks which, when combined, provide a convenient and low cost enforcement mechanism for shareware distribution. We call this new approach "protected shareware." Protected shareware supports casual copying of binary software by friends, distribution of binary software over networks, or any other distribution method that happens to be convenient. Software need not be "individualized" or "personalized," a single version of the binary can be distributed to all users. At the same time, it enforces the software author's right to be fairly compensated by the users of the software. User's that fail to pay for their software can't continue to use it.

## Introduction

The distribution of software fundamentally involves two parties: the user and the author. The user wants software available when and where it's convenient, without unnecessary delays. The author of software wants fair compensation from the user.

In an ideal world the user would simply get a copy of the software from whatever source was convenient, whether that source was from the local store, from a network, or from a friend. Typically, after a short trial period, the user would pay the author the asking price or would stop using the software. For some kinds of software, the user might be asked to pay a "transaction cost" whenever they accessed a database, or some other charging algorithm might be used to determine a fair price for the service being provided.

If everyone were honest we might approach this ideal world using shareware: software that is made available to any user in the world who wants it, with the admonition to pay the purchase price if the user likes it and wants to keep using it. Unfortunately, while this approach is very convenient for the user, the temptation to use the software without paying for it is often too strong to resist. Shareware is not (today) the path to riches that it “should” be in an ideal world.

To combat this, much commercial software is sold much like we sell physical objects: in stores. The user comes to the store, browses through the available boxes and purchases the one that has the best advertising, that is most touted in the computer journals, or simply has the most brightly colored box. After taking it home, the user tries out the software and finds out whether they actually like it. Returning software because the user “doesn’t like it” is often not possible, meaning users are sometimes stuck paying for software that doesn’t actually do what they want.

If a user visits a friend and watches that friend use some software that the user wants, the user is systematically dissuaded from doing the obvious: copying the software onto a floppy and taking it home. While onerous copy protection systems are now relatively rare, software often refuses to run unless the software is “individualized” with the users name and address. This mechanism is a not-so-subtle reminder to the user that it’s illegal to pass a copy of the software along to a friend, and if he does he might get caught. Before each execution, many programs ask “What’s on page 37 of the user manual?” This effectively makes the software useless without the manual -- often a rather bulky document, easily lost and not available to the user in a digital format.

While it’s technically quite easy to make software available over a network (as is often the case for shareware), much commercial software is either unavailable over networks, or is available only in a “crippled” or “demo version” form.

A method that has been tried and has largely failed (except for very expensive software) is the “dongle” (supposedly named after Don Gall, though this etymology is likely apocryphal). A dongle is a relatively low-cost device that plugs into some port on the computer. The software checks for the presence of the dongle, and refuses to run if it’s absent. While not grossly inconvenient, this approach means that users who want to use several different types of software must have some method of communicating with any one of several dongles: a daisy-chain of dongle’s is required. The use of dongles effectively links the program to a physical object, and so forces us to treat the purchase and sale of the program much like we treat the sale of a physical object. If a friend of the original user wants a copy of the software, they have to purchase an additional dongle. This forces the new user to physically purchase a physical device before being allowed to use a non-physical entity. By forcing us to treat software in the traditional framework used for physical objects, this approach artificially limits the convenience and availability of software.

In short the most convenient methods of software distribution --making a copy from a friend or over the network-- are blocked either by technical means, social means, or some combination of both; primarily because of fears that the user won’t pay for the product. Products are systematically designed to be either unusable or less convenient in the absence of some physical token (such as a dongle, a “protected” floppy, the user manual, or whatever).

This “physicalization” of the software product creates another major drawback: added cost. Software is information, and when retail stores pretend that it is a physical object that comes in a box, they add a whole range of unnecessary costs to the product. Shipping 200 copies of a software product is an intriguing exercise in foolishness, for what is truly being transmitted is bits. A truck filled with boxes filled with floppies all holding identical bit patterns is not the most efficient method of moving information. And yet, if the retail store doesn't have enough copies, they will "stock out" and lose sales. The boxes also occupy shelf space, creating another artificial restriction. Because of limited shelfspace only the most popular software packages can be displayed on the shelf, thus limiting the available selection of software. Finally, users typically come to the store to examine, purchase, and physically walk off with the software. How much time and money is wasted by physically travelling to a store to purchase information that could have been more easily provided by a friend or over a modem or network?

There are other problems, but the fundamental point is clear: trying to buy and sell software as though it were a physical entity is the wrong model in almost every respect.

Work by Mori et. al.[1] on "superdistribution" shares many of the goals of the present work, but is distinct. He proposes to encrypt the software and as a consequence his proposal is quite different from the current proposal, which does not require encryption of the distributed software. This difference is likely to be crucial to the convenience and utility of actual use. While [1] did not describe the key distrubution method, if it is possible to freely distribute encrypted software, and if this encrypted software can be used by any user, then the key for decryption of the software must also be available to any user. Mori proposes the use of stringent physical security methods to reduce the risk that key material might be divulged to the user, but system security cannot be high if a single common key is put into all hardware systems.

Work by Cox[2] on superdistribution emphasizes a broader framework for collecting revenue for the use of any type of digital property, regardless of its nature (including books, courseware, applications, etc.) This framework does not specify whether any particular object is or is not encrypted (or what specific methods should be employed to protect it), but leaves such issues to the object's owner. The methods discussed in this paper (in contrast) focus on specific protocols for dealing with binary software by taking advantage of its particular properties.

Previous work by the author[3, 4] proposed protocols in which unsigned software would not be executed, but left the responsibility for checking the validity of the signature (and performing any other validity checks) with either the operating system or the underlieing hardware. The program was viewed as a passive object. In contrast, the present proposal requires the active participation of the program both to check signatures and to verify the integrity of the program itself.

## The Idea

How, then, can we treat software as what it is, information, and yet insure that the user will pay a fair price? As mentioned earlier, if everyone were honest shareware would be a good approximation to the ideal. Could we design a system with the distribution advantages of shareware and yet somehow insure that users would have to pay for the software or else not use it?

The short answer is "Yes!"

In protected shareware, the software calls a set of accounting routines to specify the "asking price." A simple request to the accounting routines might specify that the user could use the software free for the first 30 days, but after that would either have to pay $25.00 or stop using it. A user, by copying the software, would automatically copy the calls to the accounting routines that specify the payment terms the user must meet to use the software.

Equally clearly, if we assume the user's computer is sometimes or often cut off from a network, or that it is either uneconomical or inconvenient to constantly communicate with some central site, then the information provided by these accounting routines must be stored somewhere. We can't charge a user for using software if we don't know (or forgot) what was used. This implies that somewhere connected to the computer must be a non-volatile memory that remembers the software usage.

Unless we take additional precautions, it would be relatively easy for a user to avoid paying his bill. The user would simply erase the billing information. Thus, the memory holding the billing information must be inaccessible to the user.

And finally, the billing information needs to be periodically passed along to a billing agency. Passing along billing information need only be done infrequently: once a month using any convenient form of communications (such as a low speed modem) would be quite sufficient.

This combination of requirements suggests the solution: we sell a cheap one chip "billing computer" with some onboard non-volatile memory, some ROM and some RAM. The billing computer plugs into the "real" computer (an IBM PC or Macintosh, for example) using some low to moderate speed port (a parallel or serial port). The ROM holds the accounting routines, the non-volatile memory holds the usage data, and any program that wants to run on the main computer would first pass along the accounting information to the billing computer and then wait for the billing computer to approve execution. If the user has been paying his bills, the billing computer approves execution. If the user hasn't paid his bills for the last few months, the billing computer tells the software that execution is not approved and the software doesn't run.

Periodically, the billing computer must get in touch with some billing agency. Any means of communications will do. As an extreme example, the billing computer could write the billing information to a floppy, which the user could then mail to the billing agency. The billing agency would then mail a floppy to the user, which could be read by the billing computer. Once having received confirmation that the bills had been paid, the billing computer would then permit further software to run.

More attractively, the billing computer might include a low-cost modem that would plug into a standard telephone outlet. For convenience, this modem would pass along the telephone signal to other equipment when the modem was not in use (which would be most of the time). Thus, the "telephone interface" would not interfere with normal use of the telephone line.

To insure that the user does not tamper with this communication, the billing computer and the billing agency could use digital signatures to sign every message. The billing computer and the billing agency would each have their own secret signing keys, and each would know the public checking key of the other. Thus, even though communications between the billing computer and the billing agency take place over insecure lines (which could pass through several layers of untrusted network software, if that proved desirable), the integrity of the billing information would be insured.

These considerations result in a billing computer with: (1) a one chip processor with RAM (both volatile and non-volatile) and ROM, (2) an onboard clock (so the billing computer can tell that three months have passed and the bills haven't been paid) (3) two modular block plugs that let it "tap into" the telephone line, but normally pass along telephone signals from other equipment, (4) two parallel port plugs that similarly let the billing computer pass along information going from the computer to (say) the printer, but which can intercept information from the program intended for the billing computer, (5) its own unique secret signing key and (6) the public checking key of the billing agency.

An alternative protocol would be more secure, but perhaps less convenient for the user. In this protocol, the software would refuse to run unless it received a signed message from the billing agency. In this model, when the software was first run it would cause the billing computer to call up the billing agency, and the billing agency would return a digitally signed message giving the software authorization to run. This approach would result in delays simply to access the billing agency (a low cost modem dialing in to the central agency could easily result in delays of a few minutes), and if the communications channel or the central billing agency were down it would result in inability to execute the desired software. Unless the central billing agency approved *every* execution of the software, there would still have to be some method of delegating authority to the billing computer. We do not consider this approach further in this paper.

While building a low-cost modem into the billing computer is a convenient method of insuring that the billing computer can conveniently talk to the billing agency, it is not essential. If the users computer already has a modem, the billing computer could use that modem. If the users computer is already connected to a network, the billing computer could use that network. It might be convenient to have a single billing computer service many computers, particularly if those computers were all connected to a reliable local network.

The billing computer would have to be "tamper resistant," e.g., attempts to modify the billing computer would have to result in erasure of the billing computer's secret key. This would be sufficient to make the billing computer worthless, for without the secret key no protected shareware would run on the system. There is an extensive literature on making computers tamper resistant. It seems likely that, for commercial purposes, relatively modest (and low cost) security measures would be sufficient. If, on occasion, the secret signing key of some billing processor *is* compromised, then it would be possible to fool protected shareware into running without having to pay for it. Distribution of such keys would be illegal, and would subject the user to various penalties. Further, if the software requires a relatively up-to-date certificate for the corresponding public checking key, then the compromised key would no longer be useful after some pre-determined period of time.

The user would also be provided with (insecure) software to interrogate the billing computer and determine the current software charges, the status of payments, etc. This software need not be secure, for damaging it would provide no benefit to the user. The billing information cannot be altered, it can only be examined. Thus, users can be provided with convenient “billing display software” which aids them in understanding the billing information and the billing procedure.

While this provides a secure link between the billing computer and the billing agency, we must still protect the link between the software and the billing computer. Without further precautions, the user could buy an imitation “billing processor” that would permit all software to execute (or provide a similar functionality by modifying the system software). The software has to be able to verify who it’s talking to.

We do this by embedding the public checking key of the billing agency into the software. The billing processor can then provide a “certificate” to the software (signed by the billing agency) which authenticates the unique public checking key of the billing processor. The software would first confirm that this certificate was valid, and that it was reasonably recent (it might require that the certificate had been signed during the last six months, for example). The software can then confirm the validity of any messages from the billing processor. The software can then send an insecure message to the billing processor describing the payment terms that it wants, and will expect to receive a digitally signed message from the billing processor which (a) echoes the payment terms and (b) approves execution. The software would then continue to periodically interrogate the billing computer to verify that execution was still permitted, and would expect a digitally signed message each time confirming this. To prevent “replay” attacks, the software would include “random” information that the billing computer would also have to sign. (This information need not be truly random, but there must be a low probability of choosing the same information twice).

The reader should note that communications from the software to the billing computer is insecure, but that this is only a minor nuisance. In essence, it means that the software must demand that every message to the billing computer be echoed by the billing computer, and that the echoed message must be digitally signed. Thus, the presence of an authenticated channel from the billing computer to the software can be used to insure that messages from the software to the billing computer are correct.

This provides complete security *if* we assume that the software has not been modified. To insure this, the software must include internal “integrity checks.” The simplest integrity check is for the software to compute a checksum over its own code. If this checksum fails, then the software displays a message to the user “This software has been damaged, please delete and load an undamaged copy.”

Of course, some clever user might find this simple integrity check and modify the binary to eliminate it. To prevent this, the software can have a second integrity check that verifies that the first integrity check has not been tampered with. If the user modifies this second integrity check, the software can have a *third* integrity check that checks the second integrity check.....

Ultimately, a clever user can defeat such integrity checks. It is possible, after all, to fully understand the binary version of the software and then remove all the integrity checks. However, the cost to the user of understanding the binary can likely be made high enough that very few (if any) users will bother. Thus, the protection at this level is “good enough” rather than absolute. The system as a whole will produce excellent results as long as the great majority of users don’t defeat the internal integrity checks, which seems very likely.

Distribution of modified binaries (which failed to perform the appropriate integrity checks and did not bill appropriately) would be illegal. While limited distribution of such binaries would be feasible, any widespread distribution would be readily detected and legal action taken against the distributors. The distributor would be unable to use the billing mechanisms provided for protected shareware, and so would have to establish a different distribution and charging mechanism if they hoped to make any profit. Thus, to succeed financially, such an illegal distributor would have to (a) defeat the integrity checks, (b) advertise in a covert fashion to avoid legal action and (c) establish a payment mechanism other than that normally used for protected shareware. While this is feasible in principle, the practical difficulties are likely to reduce financial losses to the legitimate author to an acceptable level.

A relatively simple method of cheating the billing company would be to (a) purchase a billing computer, (b) run up significant software bills, and then (c) destroy the billing computer. This can be controlled by several methods. The simplest would be to establish a credit limit on the magnitude of the total software bill that an individual billing computer will accept. This could be combined with a fixed charge (perhaps equal to the credit limit) for the loss of the billing computer, similar in spirit to the “you break it you buy it” sign seen in some stores. In any event, the loss of the billing computer can be reliably detected, the magnitude of the problem can be accurately assessed, and appropriate corrective measures can, if necessary, be adopted.

## System Summary

There is a billing agency with a well known public checking key and a secret signing key. The well known public key is embedded in “protected shareware,” and the protected shareware is distributed over whatever communication channels are most convenient. Each computer user who wishes to use protected shareware must have access to a “billing computer,” either by purchasing one and plugging it into some convenient port on his computer or alternatively by connecting a single billing computer into some local network to which his computer has essentially constant access. The billing computer has its own unique secret signing key and public checking key, and also has a certificate signed by the billing agency to validate its public checking key.

The protected shareware sends billing information along with a random string (a nonce to prevent replay attacks) to the billing computer. The billing computer digitally signs the billing information and returns it to the software along with a certificate. The software checks the billing computer’s certificate and verifies that the billing computer has a correct copy of the billing information, and then begins execution. Periodically, the software requires an additional signed message from the billing computer to insure that billing has not been prematurely discontinued.

The billing computer regularly summarizes the software charges (perhaps once a month) and sends them by whatever communications channel is convenient to the billing agency. This message is digitally signed by the billing computer to prevent tampering. A convenient communications channel would be a low cost modem. Insecure software provided to the user would let the user interrogate the billing computer and see the summary. When the billing computer received a digitally signed message from the billing agency confirming receipt of the months bills, it would know the bill had been paid. If the billing computer fails to receive confirmation of payment for a few months, it cuts off software service.

This is illustrated in figure 1.

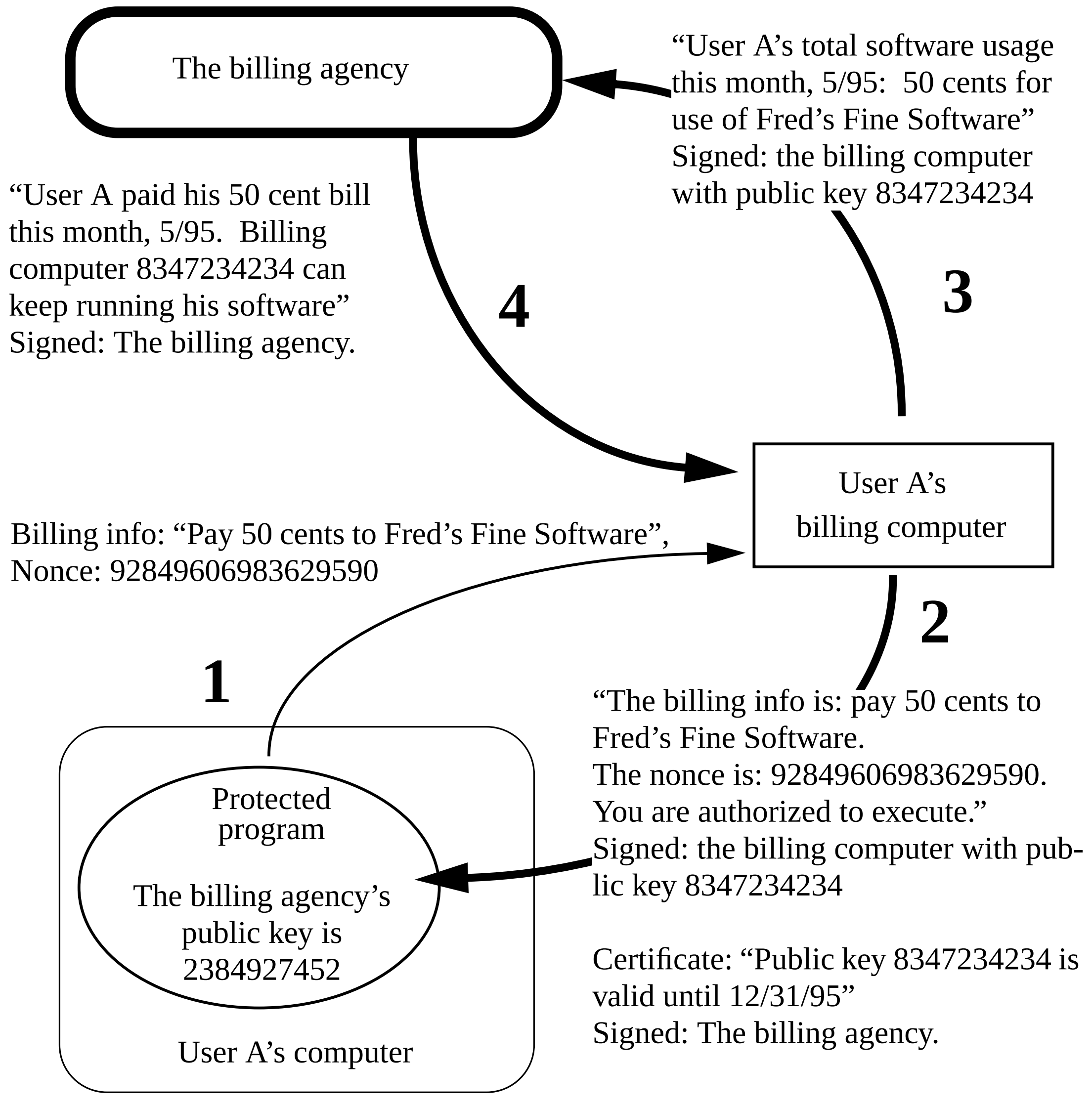
The billing agency
"User A's total software usage this month, 5/95: 50 cents for use of Fred's Fine Software" Signed: the billing computer with public key 8347234234
"User A paid his 50 cent bill this month, 5/95. Billing computer 8347234234 can keep running his software" Signed: The billing agency.
4
3
User A's billing computer
Billing info: "Pay 50 cents to Fred's Fine Software", Nonce: 92849606983629590
2
1
"The billing info is: pay 50 cents to Fred's Fine Software. The nonce is: 92849606983629590. You are authorized to execute." Signed: the billing computer with public key 8347234234
Certificate: "Public key 8347234234 is valid until 12/31/95" Signed: The billing agency.
Protected program
The billing agency's public key is 2384927452
User A's computer

Figure 1
Note that steps 1 and 2 occur immediately while steps 3 and 4 could occur much later (perhaps monthly).

Also note that message 1 is insecure (not signed).

The protected shareware uses internal integrity checks to prevent unauthorized modification, but is otherwise unprotected. While unauthorized modification is in principle feasible, it seems likely that it will be difficult enough to deter most users, which should be sufficient.

This still leaves the user vulnerable to software provided by unscrupulous vendors who misrepresent the charges. Several methods of protecting the user are available. First, the software provided over the net could be digitally signed by the author. The user could then use software only from reputable authors who did not engage in such practices. This also provides some protection against viruses. Second, the billing computer could (through additional software on the main computer) inform the user of the price before the software was executed, and block execution if the user thought the price was too high. Third, because the user would have a copy of the software, it would be easy to verify that the software was misrepresenting the price. The software might say “Pay only $5.00 to run this software!” in a message displayed to the user, but would tell the accounting routines “Charge the user $5,000 dollars.” The user could complain to the billing agency, which could readily verify the duplicity of the software and take appropriate action (press charges against the author, for example, or systematically remove the offending software from publicly accessible archives). Fourth, the billing agency could let some other agency (the courts, for example) deal with disputed bills.

## Other hardware implementations

Once protected shareware catches on, it would be a relatively simple matter for computer manufacturers to include the billing computer in the design of the main computer. The simplest method of doing this is to provide a small “secure kernel” which has protected memory that is inaccessible to the user (except through the approved accounting routines). While secure operating software is in general difficult to write, a small secure kernel with limited functionality (it just takes care of the billing information) would be relatively easy to write even today. This approach means the hardware component of the protected shareware system could be provided to the user at a very low cost.

## Software only implementations

While less secure, it would be possible to embed the operations of the billing computer into the operating system. This approach has the advantage that it could be implemented easily by software changes in the operating system rather than by the addition of new hardware. Particularly if the operating system were relatively complex, and if new versions or updates to the operating system checked the old version to remove any unauthorized modifications, it would be possible to rapidly make the capabilities described here widely available, with a level of security that would be adequate for many applications. As the same interface could then be used for more secure hardware versions, this would provide a migration path from today’s insecure environment (where the kind of billing computer described here is not available) to a future environment where secure hardware implementations were widely available.

Programs desiring higher security could insist that the billing computer be implemented in secure hardware -- for this purpose, the billing computer’s certificate would identify it as a software or hardware implementation. (More generally, the certificate could identify the level of security provided by the implementation of the billing computer). Programs could then decide (based on such factors as the likely theft rate and the market share of computers providing that level of security) whether or not to run.

## Negotiating the Price

While we have considered only the simplest case, protected shareware allows the software to charge whatever price it feels like. For example, the software might initially present the user with the message: “$25.00 to use this software.” After the user declined, the software might come back and say “Only $15.00 to use this software with the “SAVE” feature disabled.” When the user again declines, the software might say “How about a penny a minute?” Not only can the software negotiate a price, it can vary the price depending on circumstances. A scientific package that crunched numbers could charge more on a faster processor, while a database package could charge more if there were more items in the database.

Ultimately, the ability of the author and the user to agree on any “deal” that they both thought was reasonable would be a major boon to the software market. Today, the technical issues surrounding software distribution and use sharply constrain the terms that the author and user can agree upon. With protected shareware, the terms can be as flexible as their needs, desires, and imaginations.

## Summary

Protected shareware provides the distribution advantages of shareware but allows the software’s author to enforce the purchase price and, more generally, the terms of the purchase. The user is free to accept or reject those terms, but cannot use the software in violation of those terms. Distribution overhead is lower, payments to the author are more direct, convenience and availability to the user are higher, software authors have easier access to larger markets, and the sale and distribution of low volume software becomes easier and more profitable.

Protected shareware avoids many problems that other software distribution methods have had. The programs can all be identical: there is no need to “customize” each program for each user. There is no physical “artifact” that must follow the software to insure that it works. While the user must purchase and install the billing computer, once installed it is left in place and is used for *all* protected shareware. A single billing computer could be used for many computers if those computers were connected by some reliable network. Security is difficult to defeat, even for those who fully understand how the system works. The user is not forced to wait for “approval,” but can download or copy any software from anywhere and run it immediately *if* they are willing to pay for it. Existing software is undisturbed: programs that don’t use the billing features will continue to run as before.

In short, the only inconvenience imposed on the user is the need to pay his bills.

## REFERENCES

1. *Superdistribution: The Concept and the Architecture*, by Ryoichi Mori and Masaji Kawahara, The Transactions of the IEICE, Vol. E 73, No. 7, July 1990, page 1133-1146.

2. *Superdistribution*, by Brad Cox, 1996, Addison-Wesley

3. *Secrecy, authentication, and public key systems*, by Ralph C. Merkle, UMI Research Press, 1982.

4. Unpublished proposal to prevent unapproved “cartridge” based software (e.g., third party game cartridges) from running on a manufacturer’s base unit; by Ralph C. Merkle, 1981.