SOFTWARE PROTECTION:

MYTH OR REALITY?*

James R. Gosler
Sandia National Laboratory
Division 7233
Albuquerque, N.M. 87185

Abstract:

Staggering amounts of commercial software are marketed to fulfill needs from the PC explosion. Unfortunately, such software is trivial to duplicate! From the vendors' viewpoint a way to protect profit is needed. Typically, they have resorted to various schemes that attempt to inhibit the duplication process.

Although protection of future profit is important, so is protection against current loss. Commercial and business related software must be adequately protected lest data be stolen or manipulated. However, more important than any of these classes is protection of government computer resources, especially classified and operational software and data. Loss of control in this realm could be detrimental to national security.

This paper addresses current technologies employed in protection schemes: signatures (magnetic and physical) on floppy disks,

*This work performed at Sandia National Laboratories, supported by the Department of Energy under contract No. DE-AC04-76DP00789.

H.C. Williams (Ed.): Advances in Cryptology - CRYPTO '85, LNCS 218, pp. 140-157, 1986.

Software Analysis Denial (SAD), Hardware Security Devices (HSD), and Technology Denial Concepts (TDC) are presented, with an emphasis on SAD. Advantages and disadvantages of these schemes will be clarified.

1.0 INTRODUCTION

Software piracy, unauthorized penetration and system modification[12,13] are areas of threat to government and business computer systems, even the economic survival[6] of many software vendors is in peril. Vendors are typically using three main strategies to combat the piracy dilemma. The strategies[16], usually used in combination, include marketing, legal, and technological. A typical marketing strategy is to price software at an extremely attractive figure in the hopes that each potential customer will purchase it, especially to receive the required documentation and any technical consultation. The legal ploy[7] includes sueing for copyright or licensing agreement violations. These schemes by themselves have limited effect, but are useful in combination with other strategies. Technological schemes are extremely varied in detail, though they can be typically grouped into a few categories. The effectiveness of these technical schemes vary substantially and this is a major topic of the paper. The technological arena provides the only substantial methodology to combat software threats in government and business application fields.

Concerns other than just preventing duplication of software are very important. For instance, software vendors may wish to protect against disclosure of proprietary algorithms, banking executives must prevent their system programmers from being able to examine or modify bank accounts, and government entities must design defense systems software with an intrinsic ability to prevent tampering of critical components or information. There are many examples where the hiding of critical information in software or detection of modified software is desirable, possibly mandatory. These conditions are found in all

types of computer related applications, but have yet to receive the attention they deserve. One of the greatest flaws current banking and government computer security systems have in common is an implicit assumption that the adversary does not have access to a system. There are many recent examples that show this is not a good assumption! Obviously, this assumption, concerning potential adversary access, is not the case with copy protection schemes where the system is essentially thrown into a den of "wolves".

The overall intent of this paper is to discuss current capabilities of both the defense and the threat, provide a comparison between them, and suggest a set of goals for the ultimate software protection system.

Examples used in this paper are fictitious, but at the same time are representative of current copy protection techniques. However, we will neither deal with the issue of how protection schemes impact the end user nor address any specific defeat methodologies to compromise current security schemes. The IBM PC, used by the author, will serve as a vehicle for examples.

2.0 FUNDAMENTAL CONCEPTS

We will provide some fundamental principles of copy protection and relate these via an analogy before discussing some of the technical concepts associated with defense and threat.

There are two broad components in a copy protection scheme. The first is a uniqueness associated with the system, which must be difficult to reproduce. Typically this is done with an unusual sector(s) on a floppy disk or by using a Hardware Security Device (HSD) that is separately attached to the system. The other component is special software that is usually embedded somewhere within application software and it is responsible for interrogating the presence of the uniqueness in the system. If present the special software may also determine if the uniqueness is pristine or altered.

A security or copy protection system of this type can be beaten by two general methods; by duplicating the unique signature associated with the system or by modifying the software (application or system) in such a manner that application software will operate without the unique signature being present. The adversary need use only one of the above methods and generally the easiest is chosen.

Thus, not only must the defense design a difficult to duplicate unique signature but, must also make it hard for an adversary to analyze and/or modify the software. Interestingly, many software packages, which employ some form of copy protection, do nothing to make analysis and/or modification of software difficult for the adversary -- these schemes are easily defeated. Importantly, it is this component of a good security scheme that has many applications outside the field of copy protection.

Another way of looking at the analysis and modification problem is by analogy using a burglar alarm. Suppose that a valuable asset must be protected and to do this we place the asset in a vault and surround it with an alarm system having many different types of sensors. These sensors are responsible for detecting changes to the normal operating environment. Upon detection, the alarm will be triggered, guards will appear, and the burglar will be permanently detained. In this analogy the valuable asset could be special software that checks for the presence of the unique signature. The sensors could also be special software which attempts to determine if the operating environment has been modified due to the use of, for instance, dynamic analysis tools by the adversary. Finally, the alarm might be anything from displaying an **UNAUTHORIZED DUPLICATE** message to implanting a **WORM** (software which will cause harm to the system) in the software.

For the software case it is feasible that an adversary has analysis tools with special properties that will not alter the monitored environmental parameters. Thus, the software can be analyzed easily without detection. In most cases even if detection occurs nothing terrible happens. Guards do not appear nor are worms implanted. The adversary, therefore, has an unlimited try capability and through repeated experiments will eventually win.

3.0 THE DEFENSE

A vendor wishing to protect his system will want duplication of the unique signature to be difficult for an adversary. He will also want it to be difficult to analyze or modify the software, which could bypass the need to duplicate the unique signature. Even the necessity of hiding proprietary algorithms may be appropriate. Government system software designers have similar objectives, but for different reasons. In order to prevent the adversary from having a working model of the system with which he can perform analysis at his leisure, designers want duplication to be difficult. However, the most important objectives are to make it extremely arduous to modify the system in a way that bypasses critical features (checks) and to obtain sensitive information (e.g. crypto variables). For both cases the objectives of the defense are threat scenario dependent. As such, designers must consider the ways in which their systems are vulnerable to an adversary and then take steps to thwart or nullify adversarial intrusion.

It is apparent that software developers with diverse applications have similar needs from the realm of software protection although the reasons they need protection are as diverse as the applications.

How can the defense achieve his objectives? As a vehicle to unveil techniques and concepts, we will employ an IBM PC as the system and *copy prevention* as the objective.

3.1 UNIQUE SIGNATURE

To make duplication of software difficult there must be an additional component(s) included in the system specifically to provide trouble for an adversary to reproduce. This component(s) usually falls into one of two categories in the commercial world.

The first category is comprised of a unique signature on the floppy disk itself. Typically, this comes in both a physical and a

magnetic form. The physical signature involves removing a small amount of magnetic material from the floppy disk surface with a laser. The scheme is implemented by software that writes some information to this damaged area and then reads the information from the same area back into memory. If the information read is the same as that which was written, then clearly there was no laser damage on the disk at the proper location. We can conclude the software/disk combination is not the original.

A magnetic form of floppy signature involves altering the standard IBM System 34 (double density)[11,18] recording format. Besides end user data, each track contains address marks, gap bytes (sync fields), sector ID fields, Cyclic Redundancy Check (CRC) bytes, and clock bytes. All of this information must be present and correct in order for the Intel 8272A floppy disk controller (FDC)[9,14] to properly process the end user data. It is quite possible, by altering this standard format[5], to cause the FDC to return an error status message back to the microprocessor (Intel 8088)[9,14,15] as a result of a disk operation. Examples of typical errors are bad CRC, sector not found, and address mark not found. For the system to determine that the unique signature is present, the software need only perform a disk operation(s) and then determine if correct error status is returned. It is also possible to create a non-standard disk format by issuing an unusual sequence of commands to the FDC or by using special hardware which bypasses the FDC and its inherent limitations.

The second category of unique signature consists of a hardware security device (HSD)[17], which is currently being used in many of the more expensive software packages. The HSD can be connected externally to the PC via the RS 232 port, the parallel port, or even placed in series with the keyboard. It is rarely connected internally since it typically requires use of a valuable card slot.

The manner of HSD implementation within a system also varies. In its simplest form the system will send a fixed value to the HSD which then responds with a fixed value. The software will compare the response with a stored value to determine if the HSD is present. In more sophisticated versions, the system will send to the HSD and receive from it a variable value, and possibly even have part of the

software encrypted and stored in the HSD. HSD advantages (from a security point of view) are that it is more difficult to duplicate than a floppy signature and it is not as obvious to an adversary when the system is looking for the unique signature.

The HSD is much more expensive, which constitutes its primary disadvantage, and therefore is usually not used in cheaper software.

3.2 SOFTWARE ANALYSIS DENIAL (SAD)

What is currently being done in commercial software to make adversarial analysis and/or modification of the software more difficult?

In the copy protection game, it does little good to have a unique signature, impossible to duplicate, if the adversary can easily modify software such that the signature need not be present for proper operation. Consequently, it is imperative to have a well balanced protection scheme -- the difficulty of duplicating the signature should be comparable to analyzing and modifying the software.

Since the most common tool used by an adversary is a software debugger, we will limit remarks to techniques the defense can employ to make analysis from this source more difficult. However, we will also address some techniques being used to make modification of critical or non-commercial software difficult.

Given that the defense knows what tool(s) the threat will likely use, he must determine how the normal operating environment will be altered by use of these tools. For the case of a debugger, there are available several modifications to the environment.

The first and most obvious change is that the debugger must reside in the same memory space as the application, thus, a foreign presence can be checked for. Typical debuggers depend heavily on certain interrupt vectors to single step and breakpoint the application software. Application software then could easily integrate the

use of these vectors into the application itself and thereby create difficulty in using the adversary debugger. Quite often in the analysis effort, it is convenient for the adversary to modify registers and/or memory locations to help in understanding of the software. Difficulty can be enhanced by making the proper execution of software highly sensitive to not only memory location and registers used, but to all memory locations and registers available. Finally, the application should have code which is timing sensitive because analysis of the software will alter its correct timing.

Assuming that the adversary can, through analysis, determine what he needs to modify, then the defense needs to employ techniques to make the desired modification difficult. The most common technique seems to be through use of checksums. If the defense realizes where an adversary will likely modify the software then they will perform checksums on this area of code hoping that any change to the critical code will alter the value of the checksum. Encrypting the critical software is another technique. If the adversary, through analysis, examines the decrypted form of the critical code and determines what needs modification, then he also must determine how to alter the cipher text that will yield the desired result. Public key cryptography is useful in this area. For example, if the algorithm used to encrypt the critical software was RSA and only the decrypt key was stored in the system, then the adversary would have an extremely difficult time determining how to change cipher text to achieve the desired plain text.

Numerous other techniques are currently being used in the commercial world, such as executable software movement, searching for breakpoint instructions and taking advantage of the Intel 8088 prefetch queue.

3.3 TECHNOLOGY DENIAL CONCEPTS (TDC)

Assuming acquisition of a working system, it is imperative to keep the adversary from performing dynamic analysis on it in an interactive fashion. If he is allowed to perform this interactive

dynamic analysis, he will eventually be able to locate and bypass all of the SAD features discussed in 3.2.

For this reason the adversary must be made to pay a penalty each time he is detected by SAD sensors. This penalty could be anything from destroying critical system components that would disallow further testing with that particular system to subtly altering the system in such a way as to provide disinformation, which is of no pertinent value, to the adversary.

Unfortunately, from the pure security viewpoint any commercial product containing or even suspected of containing TDC will suffer exceedingly due to consumer abhorrence and consequent economic leverage. This was typified by the irate consumer response directed against several software security vendors who boldly announced the intended use of worms in a future release of their products.

4.0 THE THREAT

Adversarial objectives of the threat are diverse. They encompass pirating commercial programs, subverting banking or government software, and stealing software-based proprietary algorithms. For example, suppose that companies A and B both produce and market an RSA encryption program. Further suppose that B's product is substantially slower than A's version primarily due to the speed of the algorithm responsible for finding large prime numbers. Company B's programming analyst could acquire a copy of A's program, reverse engineer the software, and then "borrow" the faster algorithm.

For government and perhaps business applications, the adversary's objectives are similar. Suppose the military has a computer based weapon control system. Part of its system software, responsible for access control, is password protected. To access the control system that will allow use of the weapon system without knowledge of a legitimate password, or to deny use of the weapon system to an authorized user the adversary must acquire tools to

duplicate the software and then determine what modifications would be necessary to alter the control system.

Before we discuss tools with which the threat attempts to accomplish his objectives, we need to provide a working definition of the adversary. The threat can be subdivided, in general, into insider and outsider categories with authorized access being the main difference between the two. The insider threat could be anyone from the designer of the system to an authorized end user of the fielded system. Thus, the insider threat can be broken into two categories: those intimately knowledgeable with the system, such as the design team, and those with little or no knowledge but having authorized access. However, this paper will not address the problem associated with the threat being part of the design team.

Since access control is typically a separate security issue, the outsider threat scenario considered will usually be under the assumption that the threat has already gained access to the system. Thus, for the purposes of this paper, a conservative approach is taken in that both the insider and outsider threat are considered essentially equivalent.

4.1 THREAT TOOLS

Adversarial tools that threaten commercial and perhaps other software fall into two main categories: tools used to duplicate the unique signature or its effect, and tools used to analyze software and hardware. They vary from no cost to $100K+ and are readily available.

If the unique signature is an unusual magnetic encoding on a floppy, then there are many commercial products available that will analyze the floppy and attempt to replicate the signature. However, these software tools share one deficiency: all utilize the FDC for their analysis and duplication efforts. But, there exists unique signatures that are currently being used that were not created using the FDC and all its limitations. For example, some vendors use special hardware that will generate "weak bits". These bits are

impossible to duplicate using the FDC and are thus felt to be more secure by the vendors. Unfortunately, there are also available products[3] capable of separately encoding each bit cell on a track and at a variable flux density. Such products make duplication of all magnetically encoded unique signatures on floppy disks effectively trivial.

Even if the signature is a result of physical damage to the disk, the adversary has several options. First, with appropriate equipment, he can attempt to duplicate the physical damage, which could be difficult even with expensive equipment. However, depending upon the motivation and resources of an adversary, it is certainly feasible. A simpler and cheaper approach would be to alter the system so that software which checks for damage is "fooled" into thinking that the damage is present. This might be done by front-ending certain interrupt vectors which are tied to the FDC. Such front-end software would change the status of the FDC command to the correct and expected values.

If an HSD is installed as the signature then attacks similar to the physical damage case could be employed. Usually it is a straightforward task to alter software that is communicating with the HSD using a technique that renders the HSD needless. A much more complicated technique for defeat would be to duplicate the HSD. However, this addresses the tools and techniques of analyzing and duplicating microcircuitry, which is beyond our scope.

The adversary will make use of two general classes of tools in his analysis effort: static and dynamic. Assuming he has acquired use of the system, the adversary will use these tools against the binary form of software. For example, static tools can be used to locate all branching instructions and/or all occurrences of an INT 13H (disk operation) instruction. These classes of tools can often provide a good starting point for application of dynamic analysis tools. Actually, the more structured the programming methodologies the more straightforward it is to use these tools. This situation is certainly better for the adversary.

Dynamic analysis tools are the real workhorses for the adversary. They include software debuggers[1,2,4], in-circuit

emulators (ICE)[8,10], and simulators. We have determined that the software debugger and ICE type tools are particularly useful for analyzing software systems.

These dynamic tools allow the adversary to execute the software in a controlled fashion. That is, the software can be executed one instruction at a time (single step). Then between instructions the analyst can examine/modify registers and/or memory locations. In addition to the single step mode, the analyst can also stop processing as a function of several other types of events. For example, execution could be halted and the environment examined/modified when:

1. Instructions are fetched or executed
2. Operands are fetched or modified
3. I/O ports are referenced
4. Memory/register contents reach predetermined values

Simple but powerful tools such as these give the adversary an enormous amount of information and consequently, it becomes a nearly straightforward task for the analyst to wade through the software to achieve his objective. The only difficulty that the analyst must be aware of is modification of the operating environment in a way that will trip a security sensor. Fortunately, for the adversary, even if he trips a sensor there will not be a debilitating penalty in most current systems. Thus, through an iterative process he will eventually work his way through or around all the sensors on the path to his objective. If the adversary's tools modified the environment in a detectable fashion and a significant penalty were imposed then the adversary is forced to proceed at a far slower pace. He must execute smaller blocks of code before hitting a breakpoint and he must also attempt to fix any environment modifications. Depending on the payoff, however, the adversary may well be willing to pay this extra price to analyze systems using penalties.

5.0 THREAT VS. DEFENSE

We have briefly discussed the objectives, tools and techniques of the two players. Our purpose here is to point out some strengths and weaknesses of the schemes currently being used commercially.

Many advantages in this game reside with the threat. As always, the adversary plays his cards last and thereby gets to attack a static security design. Beyond this there is another major obstacle for any cryptographic solution to the security dilemma. Even though the defense uses cryptographic schemes to scramble the executable software he must include not only the decryption algorithm but also all of the necessary cryptographic keys as part of the system.

The weakest characteristic of these schemes is the fact that the adversary never has to pay a penalty and in effect has an unlimited number of tries in order to achieve his objectives. To make matters worse some schemes typically broadcast to the outside world that a security violation has occurred. They will usually provide for the adversary a detailed road map to the sensor location. This weak characteristic alone makes defeat of these security schemes significantly easier!

Currently, many systems use a security front-end to their application software. This is done for several reasons, one of which is that it does not require modification to the application software, which makes the addition of protection easier for the vendor. Unfortunately, these front-ends are typically very easy to completely remove leaving the adversary with the unprotected application. Also, due to the proliferation of security schemes numerous software vendors purchase and use the same security package. Consequently when one package is defeated the rest will fall in short order and with minimal effort.

As previously stated, it appears that all of the more advanced security schemes rely on the use of clever programming tricks to detect an adversarial presence. This tends to make the reverse engineering process more difficult. It is not clear, however, as was pointed out in Simmons[19], how effective these defensive tricks can be designed to preclude or significantly delay the adversary from ultimately achieving his objectives.

Fortunately, there are several techniques that could be employed to make software analysis/modification more difficult. Most importantly, the adversary must be made to pay for his mistakes. A suitable penalty in the commercial world would simply be to make the

application software nonfunctional. The best way to alter the software to a nonfunctional state is to cause the software to fail intermittently with subtle problems. For example, suppose the XYZ corporation produces and markets a CAD/CAM program. Upon detection of an adversarial presence the penalty to be invoked might be to alter the software so that the drawings sent to a plotter will randomly miss pen strokes. In the case of spreadsheet software, the numerical calculation associated with the spreadsheet columns could be subjected to random errors.

With proper implementation of this type of penalty the adversary will not be tipped off that he has been caught. Later when he or his customer is using the application software there is a good chance that he will not associate the sporadic (flaky) operation to the pirated copy. This sort of tactic prevents another and perhaps more intensive attack on the target software.

Many other techniques could be used to improve the security of software using current methodologies. However, we feel they, at best, provide very limited protection from a sophisticated opponent. The security of the system should not depend heavily on how cleverly the designer implemented his tricks. An enormous need exists for software security systems that provide a high degree of predictable protection. What we really need are methodologies whose security is comparable to that of a good cryptographic system.

6.0 RESEARCH GOALS

Research applications in this area will impact software based systems in four distinct domains: 1) security level 2) cost 3) reliability 4) performance. Obviously, the optimum objective of SAD/TDC is to provide maximum security at minimal cost, with no impact on system reliability or degradation of system performance. This is an impossible task. However, depending on the application, the above optimum objective could be relaxed and realistic requirements could still be achieved.

Ideally, the level of security provided by SAD/TDC is equivalent to security associated with modern cryptographic based systems. That is, the compromise of a system should be dependent on the adversary dealing with the computational complexity issue. As mentioned, however, all SAD/TDC currently being employed involve the use of clever tricks and attempts to conceal information from the adversary. After refinement, perhaps even these techniques may be adequate for limited situations. For example, suppose our secure system, which we have control over, is one in which the unique information (a special algorithm perhaps) can be made obsolete within one week after detecting loss. If so, a security system which provides at least two weeks of delay to the adversary may be adequate. In the limit then our secure system could have its uniqueness changed inside the cycle time of an adversary.

Costs of these sorts of systems can be broken into three categories: 1) development 2)production and 3) administration. Development, that is the cost to design and integrate the security subsystems into the applications, is a one-time item and thus, this cost increase will usually have the most latitude. However, additional production costs will be incurred for each system produced and as such, may receive much closer scrutiny from management. On the other hand, in an extremely high security application, as is the case with control for nuclear weapons, costs become of secondary importance, so an increase of perhaps a few thousand dollars for the security system becomes acceptable. Administrative costs are associated with maintaining system security requirements, such as the need for key management. Costs such as these are recurring and potentially substantial. The security designer should always be attentive to this area.

Many of the application systems that need added security have requirements for extremely high reliability. For this reason the security designer must be very careful with use of certain techniques such as timing tricks. Many times a security system undergoes an independent review process, which is designed to determine if subversive features (trapdoors, trojan horses, etc.) are present. Unfortunately, it may be more difficult to detect designer induced subversive constructs due to current techniques being used by the designers to improve system security.

Based on this current state of affairs, if the software designer were a covert agent, then he could compromise the integrity of the system while appearing to increase its security!

Finally, the issue of system performance can be of overriding importance depending on the application. The key idea is to minimize or eliminate such adverse effects. If an application is the guidance subsystem software for a defensive missile then performance degradation could not be tolerated. For instance, the systems reduced capability to update the missiles parameters may result in an unacceptable reduction of kill ratio.

Now, we would like all parameters of a security subsystem skewed in our favor in an optimal fashion, but realistically this does not seem feasible. Compromises will need to be made with the security design as a function of application system requirements so that an optimum balance is achieved.

7.0 SUMMARY

Considerable effort and resources are expended to prevent "hackers" or outsiders from attaining illegal access to computer systems. The same is not true, unfortunately, concerning the insider adversary having access to a computer system. Partial or complete access can lead to unauthorized duplication or modification of the systems software.

Current defense methodologies are not adequate to prevent or even significantly delay an insider adversary from achieving unethical to illegal objectives. Many software applications in both business and government sectors are in dire need of effective techniques to thwart an insider or outsider (who has acquired access) attack. Although the level of security offered through current methodologies can be enhanced to some degree, the results will still be unsatisfactory because the problem stems from these marginal methodologies.

We must provide new developments that, for example, explore the world of cryptology and exploit the limits of numerical complexity to the extent the security of a system is provable, or at least predictable. With this sort of focus perhaps, the myths of software protection and security can be transformed to reality.

8.0 REFERENCES

1. S. Armbrust and T. Forgeron, "Entymological Explorations", PC Tech Journal, vol. 3, no. 1, (Jan. 1985), pp. 88-109.

2. S. Armbrust and T. Forgeron, "Untangling Problems", PC Tech Journal, vol. 3, no. 4, (Apr. 1985), pp. 81-95.

3. COPYIIPC Option Board Manual, Central Point Software, (1985).

4. D. Daftwyler, "Professional Debugging", PC Tech Journal, vol. 3, no. 3, (Mar. 1985), pp. 60-73.

5. Disk Mechanic Technical Manual for the IBM Personal Computer, MLI MICROSYSTEMS, (May 1985).

6. D. Gabel, "Copy Protection", PC Week, vol. 2, no. 34, (Aug. 1985), pp. 35-37.

7. G. Geruaise Davis III, Esq., Software Protection: Practical and Legal Steps to Protect and Market Computer Programs, Van Nostrand Reinhold, New York, (1985).

8. HP 64000 Logic Development System Model 64620A Logic State/Software Analyzer Reference Manual, P/N 64620-90903, Colorado Springs, HEWLETT-PACKARD, (1982).

9. IBM Personal Computer Technical Reference Manual, IBM, (Apr. 1983).

10. I^2ICE Integrated Instrumentation and In-Circuit Emulation System Reference Manual, P/N 163252-003, INTEL, (1984).

11. ISBC 204 Flexible Diskette Controller Hardware Reference Manual, P/N 9800568-02, INTEL, (1979).

12. B. Landreth and H. Rheingold, Out of the Inner Circle: A Hacker's Guide to Computer Security, Microsoft Press, Bellevue, Washington, (1985).

13. S. Levy, Hackers Heroes of the Computer Revolution, Anchor Press/Doubleday, Garden City, New York, (1984).

14. Microsystem Components Handbook: Microprocessors and Peripherals, vol. 1&2, P/N 230843-002, INTEL, (1985).

15. S. Morse, The 8086/8088 Primer: An Introduction to their Architecture, System Design, and Programming, Hayden, Rochelle Park, New Jersey, (1982).

16. D. Parker, Fighting Computer Crime, Charles Scribner's Sons, New York, (1983).

17. W. Rosch, "Internal Security", PC Week, vol. 2, no. 18, (May 1985), pp. 89-108.

18. Shugart OEM Manual: SA 810/860 Single/Double-Sides Half-Height Diskette Storage Drives, (1982).

19. G. Simmons, "How to (Selectively) Broadcast a Secret", Proceedings of the Symposium on Security and Privacy, Oakland, California, (Apr. 22-24, 1985), pp. 108-113.