[5/10]

Computer Language
Vol. 5 No. 9 Sep. 1988

John Rex

Software keys: hardware-based software protection

It would be fair to say that most PC users detest copy-protected software. That the software is protected does not irritate the customer nearly as much as the *way* it is protected. The classic problem is typified by the scheme Lotus used for years, which allowed the software to be installed only three times. If you reformatted your hard disk, rearranged your hard disk with an optimizer, accidentally deleted one of the protection files, or moved to a new machine more than the allotted three times, you suddenly found yourself unable to use Lotus 1-2-3. Fortunately, most vendors, including Lotus, have given up such protection schemes.

Still, some vendors seek to protect their software. This is not unreasonable considering the amount of time and money required to develop some highly specialized packages. What the developer needs is a protection scheme that balances the needs of users and manufacturers. Users must be able to run software freely and move it easily to different machines, while developers need assurance that only one copy of the software is running at any given time. Protection must also be cheap, reliable, and easy to install.

Several companies now market a form of copy protection that fulfills these criteria. The fundamental concept is that software checks for the presence of a special protection device—a piece of hardware called a "software key"—that the user installs on the PC. The hardware device usually has some complex inner workings that would make duplication difficult. This scheme, combined with some defensive programming, is difficult to defeat. The device takes the place of the key diskette or the oddly formatted file used in older copy-protection schemes.

Software keys all look very similar on the surface. For a typical software key, see Figure 1. To simplify installation, they plug into either a parallel or serial port, making them relatively inexpensive. From the user's standpoint, this is really all there is to the scheme—the software will work as long as the device is plugged into a PC. The user may install the software on any particular machine, so long as the key is installed before the software is used.

The following discussion will develop the terminology and concepts that describe the use of these protection devices. While each manufacturer uses slightly different nomenclature, the ideas behind them are similar.

The key is a small box, typically measuring about 1½ x 1½ x ½ inches, about the size of the interface at the end of a printer cable.

In general use, the key is totally transparent to the software and hardware. All data and control lines pass unchanged through the key. The key is capable of generating a special code response used by the protection scheme, but ordinary use of a port will not activate it. The key can be used in conjunction with almost any application.

The manufacturer supplies a set of sample drivers that demonstrate how the key is activated and queried. All of the manufacturers supply code for a variety of languages. The key's driver will first gain exclusive use of the port, then awaken the key, test it for some unique response, and report this response.

The key's response varies widely with the particular key used. The querying technique differs from key to key and details are not necessarily provided by the manufacturer. This is acceptable as long as it is known that the key will not interfere with the normal function of the port. Parallel and serial port keys have different limits in this regard and will be discussed separately.

Keys that plug into a parallel port have the best functional characteristics. If the query routine is correctly written, the key is so transparent that it can even be interrogated while a background print spooler is writing to the printer. The parallel keys require only that the parallel port and printer adhere to the IBM standard, which is basically the same as the Centronics standard. Still, it is conceivable that a key might not correctly function on some nonstandard systems that otherwise seem to be IBM compatible.

In contrast to parallel keys, serial port keys cannot be interrogated while another process is using the port. This restriction is largely due to the fact that the serial port might receive data during the query process. Thus, serial keys are best suited for situations in which a serial port can be dedicated to the key. However, it would be possible for a single package to use both a mouse and a key on the same serial port.

An additional hardware consideration is the computer itself. The process of interrogating a key often entails certain timing delays, which are typically on the order of a few milliseconds, implemented as software loops. Thus, moving the key and software to a faster CPU or one with some type of special high-speed memory could conceivably alter the performance of the delay routines so they would not function correctly. The details of each manufacturer's approach to this problem are given in the individual product reviews.

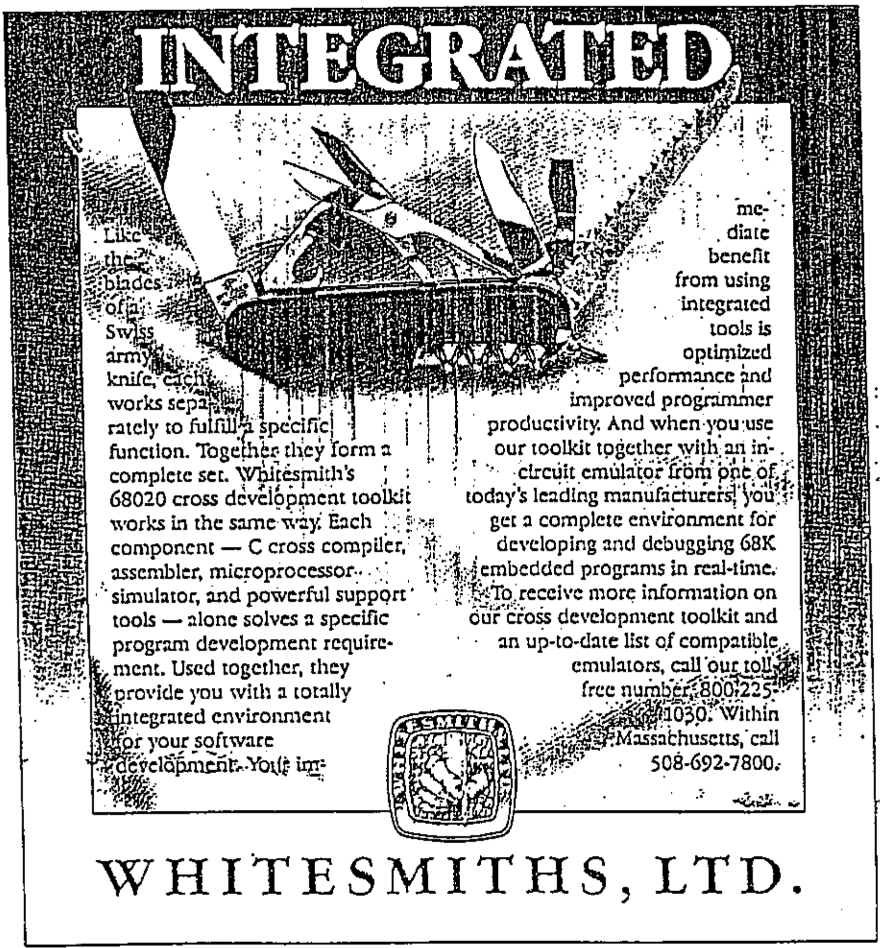
Since using multiple keys on one machine would entail plugging them into the same port, the keys have the potential to interact with each other. In general, it is possible to install different keys from the same manufacturer on one machine simply by plugging them all into a port, end on end. In my testing, I found keys from the different manufacturers to be cross-compatible. However, this cannot be guaranteed in all future situations. Finally, if keys become popular, it's easy to envision long chains of protruding keys, making it impossible to put PCs against a wall.

Two basic types of keys exist. The first might be called a "plain key." Conceptually, a plain key returns some known, predictable response. The exact response might vary with different queries, but the key always returns a specific value. This type of key is best suited for use with identically reproduced packages. To implement protection with a plain key, the software developer asks the key manufacturer to fabricate a unique key for his or her software package. The developer then incorporates appropriate test code into the package and distributes a copy of the key with each copy of the software.

The second class of key might be called a "programmable key." This key has two types of responses. The first is a unique fixed code provided by the key manufacturer to identify that series of key. Such a code will be similar to the code returned by a plain key. More important, however, is the ability of the key to store data in nonvolatile, programmable memory. The nonvolatile memory makes it possible to customize each key to a specific copy of a software package. This feature allows the key to be used in different ways.

One obvious example is a technique that allows graded access to a software package. Suppose a developer would like to make a software package available on a trial basis. The following procedure would give the customer full use of the package while ensuring that the software is either purchased or returned at the end of the evaluation period:

- The software developer has a uniquely coded programmable key fabricated for the user's package.
- Each copy of the software package has a unique serial number installed in it. This same serial number is programmed into a portion of the key's nonvolatile memory.
- When a customer requests a copy of the software, an expiration date is coded into another portion of the key's nonvolatile memory.
- The customer can now freely use



the software. It should be clear that different copies of the same software are actually unique because each serial number is different, and each copy will only run if its particular key is plugged in. Each time the software is invoked it checks the system date against the expiration date in the key.

- If the customer decides to purchase the software, the developer can either reprogram the original key or replace the key with one that contains no expiration date.
- If the customer does not buy the software, two features ensure the product can still be protected. The use of an expiration date will disable the product. However, the user could reset the system date, making the return of the physical key an important element of the protection scheme. Without a key with a matching serial number, copies of the software are useless.

The second possible use for a programmable key is to allow it to protect multiple software packages. For example, vendors of multiprogram accounting packages would only need to provide their customers with one programmable key. As the customer purchased other portions of the accounting package, the same key could be updated to allow access to each additional program.

While the preceding discussion has focused on hardware, the coding techniques used to interrogate the hardware are just as important as the hardware itself. With one exception, the keys are complex proprietary circuits that would be difficult to duplicate. Thus attacks on the protection scheme will likely focus on manipulation of the software.

The simplest coding scheme provides a subroutine that interrogates the key and returns a "yes" or "no" value. Such coding design is a poor choice because, in theory, a hacker could replace the subroutine with one that always returned "yes." This scheme is easily improved in two ways.

First, the subroutine could return a value other than 1 or 0. The value would be used in a complex calculation at some point and the "yes" or "no" decision made based upon the result of this secondary calculation. This indirect approach would make it harder to locate the test routine.

Second, instead of writing the test routine as a discrete subroutine, the interrogation code could be mixed into the program's real code. In other words, lines of real code would be alternated with lines of code from the test routine. Such a technique would make it difficult to disable the test code even if it could be located—the distinction between test code and program code necessarily be clear.

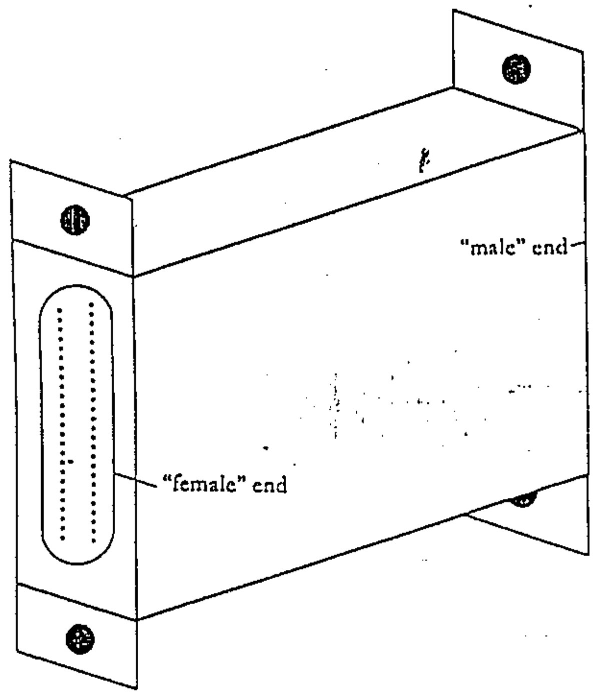
Many other schemes are possible, and the only limit is the programmer's imagination. In many ways, the programmer's implementation of the protection scheme represents the most critical element of the entire approach.

I tested the keys on a standard 4.77-MHz IBM PC-XT. This machine has two parallel ports, one connected to an IBM Proprinter, the other empty. I tested the parallel keys in both positions. The serial port is attached to a Hayes-compatible external modem. During the test period, I ran this machine with one or more of the various keys attached at all times. I never noticed any failures due to the presence of the keys on the machine.

For each type of key, I have shown the price of the key in lots of 100. Special pricing exists for larger and smaller lots, and serious developers should discuss this with each manufacturer.

FIGURE 1.

A typical software key



Software Security Inc.
The Block

Software Security Inc. markets a single device known as The Block. It is available in serial and parallel port versions, and I examined both.

Software Security takes the interesting approach of making a full disclosure of the nature of its hardware. It is patented, so in theory anyone could go to the U.S. Patent Office and examine the circuit. While this may seem counterproductive in a security scheme, it is done to give the developer the opportunity to take full control over the system. If there is ever a problem with using The Block on an off-brand machine, the developer has the tools necessary to analyze and solve the problem.

The Block is also unusual in that it comes with very few sample interrogation routines. Instead, sample code in a few languages demonstrates the techniques, but the ultimate implementation is up to the developer. By taking this approach it can ensure that no two developers use the same protection techniques.

While this is true, I felt that the sample programs were inadequate in that they only work on a 4.77-MHz PC. To use faster machines, the timing loops need to be altered, and all developers may not want to take responsibility for this. In addition, the sample code did not demonstrate the techniques for disabling interrupts during queries to The Block. Software Security says it will distribute sample code developed by customers that solves these problems for most hardware configurations.

The responses generated by the keys are actually quite simple. In effect, each key is an eight-bit counter that is set to send a signal after a certain number of cycles. After being reset, The Block is queried by incrementing its counter 256 times. The Block will generate a response. The software watches for a response on, for example, the 191st increment.

One problem with this scheme is that only 255 different versions of The Block exist. Furthermore, if two different versions of The Block are plugged into the same port, they will both cycle when the interrogation routine runs. Unwary software might be fooled by the presence of extra responses.

Both versions of The Block run on IBM compatibles and the PS/2 and are compatible with MS-DOS, OS/2, and XENIX. Versions compatible with the NEC 9801 can also be ordered. Given the availability of hardware information, porting The Block to almost any combination of hardware and software is also possible.

Software Security is currently beta-testing versions of The Block that, while retaining the same basic counter concept, provide multiple counters that can be logically tied together in various ways. This should vastly increase the number of possible responses and eliminate some of the problems just described.

ProTech
The ProTech Key and the ProTech Memory Key

ProTech, formerly known as Secom, markets both a plain key (ProTech Key) and a programmable key (ProTech Memory Key). Both are available in either serial or parallel versions. I examined the parallel versions.

ProTech provides very little information about how its keys work. This is a deliberate effort on the company's part to increase system security. Each version of the key comes with a few pages of directions and a diskette containing the code used to interrogate the keys. Code compatible with a wide variety of languages is available—I asked for the C language version. This dearth of information makes a first encounter with the package a bit confusing, but once all the sample code is listed and studied, using the package is straightforward.

The sample interrogation routines demonstrate the most aggressive software protection techniques of any of the three packages. These techniques could, of course, be applied to the use of any key, but it is convenient to have them integrated into the demonstration code. The interrogation of ProTech's keys consists of about 10 lines of set-up code and a call to an assembly language routine. For increased security, ProTech recommends that the lines of set-up code be intermingled with lines of regular code.

The set-up code does two important things. First, it inserts random bytes into the interrupt vectors used by debuggers, effectively disabling them. Second, it decrypts the encrypted assembly language query routine. This routine never exists as executable code except just before it is called. After the routine interrogates the key, it encrypts itself again. Thus it is difficult even for the developer to learn very much about the internal function of the ProTech devices.

The interrogation routine supplied by ProTech takes responsibility for dealing with all hardware timing issues. ProTech supplies upgrades as needed to deal with new or unusual hardware situations.

The ProTech Key—ProTech's plain key—responds by making four numbers available to the caller. Though the exact values of the four numbers will vary from call to call, the result of a certain calculation involving the four numbers is guaranteed to generate a number that is a constant for a given key.

The ProTech Memory Key offers 62 bytes of programmable memory. The Memory Key is unique in that in can be reprogrammed using just the power available at the parallel or serial port. This feature can be very powerful. For example, it could make software leasing a simple proposition. A package could be leased for a certain number of usages or through a certain date and the key could be programmed with the package's serial number and its expiration date. Extending a user's lease would be as simple as mailing an update program or sending it by modem. The update program would only update one specific key because it would check for the presence of the unique serial number in the key's memory.

The ProTech inquiry routines search all available parallel or serial ports for the presence of a key. The manufacturer states that the software and hardware are compatible with MS-DOS, OS/2, and XENIX. In addition to working on the full range of IBM PCs and compatibles, the ProTech keys work on the IBM PS/2. Serial port versions of the keys are available for the NEC 9801.

All ProTech keys are compatible with each other; that is, the interrogation code and the keys are designed so that two different ProTech keys will never interact.

One potential drawback of the ProTech scheme is that the very code that makes life difficult for someone trying to penetrate the key's protection scheme also makes it difficult for software developers to debug their own code. The destruction of the interrupt vectors needed by a debugger is not an optional part of the interrogation routine. Thus the development code and final production code would have to be compiled slightly differently, which might introduce sub-

PRODUCTS MENTIONED

The Block—$28 per unit in lots of 100
Software Security Inc.
870 High Ridge Rd.
Stamford, Conn. 06905
(203) 329-8870

ProTech Key—$27.95/unit in lots of 100
ProTech Memory Key—$39.95/unit in lots of 100
ProTech Marketing Inc.
1804 W Southern Pkwy., Bldg A-112
Durham, N.C. 27707
(800) 843-0413

SentinelPro, Software Sentinel-S,
and Software Sentinel-C—$33/unit in lots of 100
Rainbow Technologies
18011-A Mitchell S.
Irvine, Calif. 92714
(714) 261-0228

Smalltalk/V
Object-Oriented Programming System (OOPS)
digitalk inc.
World's #1 Best Seller. More people solve more problems with Smalltalk/V than any other OOPS.

tle bugs that would be difficult to analyze.

Rainbow Technologies SentinelPro, Software Sentinel-P, and Software Sentinel-S

Rainbow Technologies markets a plain key as SentinelPro (for parallel ports) and Software Sentinel-S (for serial ports). Its programmable key, Software Sentinel-C, is only available in a parallel port version. I examined only SentinelPro.

Rainbow Technologies takes the middle ground when it comes to the information the company supplies about the functioning of its keys. While the circuits within the keys are kept secret, the nature of the interface between the software and the key is described in the documentation. The key comes with disks that contain sample drivers for many different languages, and the documentation indicates that Rainbow will assist in porting the interrogation routines to other languages if needed. The documentation is relatively straightforward, and I had little trouble installing the interrogation routine into a sample program.

The supplied interrogation routines were implemented as a simple subroutine call. As discussed in the introduction, such a technique is less than optimal, and in practice the code should be modified to further increase its reliability. The documentation does not, however, spend very much time describing more advanced forms of software protection. Company officials indicated that they would rather supply the more advanced information directly to the developer.

The interrogation routines are provided in source code form, so it would be a straightforward proposition to modify them to accommodate faster hardware with different timing considerations. In addition, Rainbow regularly updates the routines to deal with these problems.

Rainbow's plain keys are plain keys with a twist. The interrogation routine is passed an arbitrary string such as "Computer Language." The key algorithmically converts this string into a number. Other strings would produce different numbers. Thus anyone trying to duplicate the key would be faced with the task of discovering the algorithm used by the key to generate the correct number for an infinitely large number of possible input strings.

Rainbow's programmable key, Software Sentinel-C, has 126 bytes of nonvolatile memory available. Unfortunately, reprogramming this memory requires a special programming adapter; keys would have to be returned to the developer to be reset for use in a leasing arrangement. Rainbow Technologies indicates that it is presently testing a key that can be reprogrammed with just the power available at the parallel port.

The query routine Rainbow supplies does not automatically search all parallel/serial ports, but it is easy enough to program in this feature. The manufacturer states that all its software and hardware is compatible with MS-DOS, OS/2, and XENIX and runs on all IBM compatibles and the PS/2. In addition, SentinelPro is compatible with the NEC 9801.

Rainbow's keys suffer from one drawback. Logically, the keys are grouped into 26 different families, and keys that belong to the same family cannot be plugged into the same port at the same time. While this is not a problem for any given software developer, the possibility that two separate packages might interact on a user's machine should be kept in mind.

For those on machines other than IBM PCs and compatibles, Rainbow markets Software Sentinel-W. This key is basically the same as Software Sentinel-S but does not come with any software. Instead, all the timing diagrams needed to understand the interface are supplied so that a driver suitable for use on any given target machine can be written.

Decisions, decisions

No clear winner emerges here. Rather, a wide variety of choices allow the software developer to select a protection scheme appropriate for the application. Perhaps most important is that software developers contemplating using of these devices carefully consider both their benefits and drawbacks. While the benefits are obvious, the following limitations should be kept firmly in mind:

- How much the key costs.
- What the possible interactions are between the key and the port or peripheral.
- The key may not run on newer, faster, or nonstandard machines. The cost of updating software to allow users to work on these machines should be considered.
- The code to interrogate the key could introduce bugs in the real application code.
- The key might fail due to interaction with other keys.

Because of these problems, it would appear that these keys are best used in certain specific market areas. Of particular note would be specialized, high-price packages where a single lost sale due to unauthorized copying is vital—such as CAD/CAM packages—and lower-priced packages ($750 and up) used in settings where the PC tends to be a single-purpose device (such as accounting and integrated office management packages).

Developers contemplating the protection of more commonplace utility programs (such as word processors, spreadsheets, compilers, or editors) should keep in mind the possibility that interactions between keys or between keys and other hardware might limit users' acceptance of their products. ■

John Rex is a computer consultant specializing in C. In addition to being the technical editor of The C Gazette, *he is the author of* C:LINES/C:TREE, *a C source code analysis tool.*