KEYLOK®

Software Piracy
Prevention System

USER'S MANUAL

KEYLOK® II
KEYLOK® FORTRESS

Copyright

Trademarks

MAI Digital Security (KEYLOK®)

1025 W 7th Ave
Denver, CO 80204 USA
Phone: (720) 904-2252 or 1-800-4KEYLOK (1-800-453-9365)• Fax: (303) 228-0285 or (720) 294-0275
Email: info@keylok.com • Web: www.keylok.com

Last updated 12/1/2010

Contents

Windows Quick Start

Do not insert your USB dongle until after the device driver has been installed!

Follow these three steps to protect your Windows application with a KEYLOK® II device in 30 minutes:

1. **Install the software and run the demo tool** — 5 Minutes

   a) If you are using a KEYLOK® II parallel dongle, plug it in now, and

   b) Insert the CD. The software installer will auto run under Windows.

   (If it does not, run *setup.exe* from the root directory of the CD.)

   c) Go to \Program Files\KEYLOK2\Demos and run *KLTool.exe* to familiarize yourself with the features and functionality of the KEYLOK® II Security System.

2. **Compile and run the sample code for your development language** — 5 Minutes

   Examine the sample source code for your development language and environment to learn how to implement the KEYLOK® API calls.

   (The appropriate sample code can be found in the \Program Files\KEYLOK2\<OS>\<compiler> directory.)

3. **Protect your application** — 20 Minutes

   Simply copy and paste the desired features from the sample code into your application.

Networking Quick Start

Do not insert your USB dongle until after the device driver has been installed!

Perform the Windows Quick Start first, and then follow these four steps to set up your dongle server and clients so that you can access a centrally mounted dongle over a TCP/IP network:

1. **Set up your dongle server.**

   a) If you are using a KEYLOK® II parallel dongle, plug it in now.

   b) Run *INSTALL.EXE* , (located in the "Program Files\KEYLOK2\Send To End Users folder), and select the appropriate type of dongle (USB, Fortress or parallel) and the "Server" radio button. Click OK to install the server software.

   c) If you are using a KEYLOK® II USB dongle, when the install displays a message saying that it is complete, plug the dongle in and let the "New Hardware Found" wizard run.

2. **Set up your clients.**

   On each "client" computer that requires access to a dongle mounted on a remote computer, run *INSTALL.EXE* and select the "Client" radio button. Click OK to install the client software.

3. **Familiarize yourself with the network demo software.**

   Run *NetDemo.exe* or *VerifyNetworkKey.exe* (found in "Program Files\KEYLOK2\Networking) on one of your client machines to verify the network connection to the dongle server (verifynetworkkey.exe) and familiarize yourself with the network demo applications (netdemo.exe). Run *NetKeyMonitor.exe* to see which servers are running and how many active dongle sessions each one has.

4. **Set up your application code for networking.**

   Compile your application and link it with the appropriate network-enabled interface file (in the sample code for your development language and environment): nwkl2_32.dll or kfunc32MTn.lib.

   See the ***Device Access Over a Network*** section of this manual to gain a full understanding of implementing the network dongle.

Note: Any machine in your network that can be pinged by all client machines can be used as a dongle server. The dongle server does not have to run a server operating system.

Evaluation To Production Quick Start

Follow these three steps to set up your protected application to use your company-unique production dongles instead of the demo dongle:

1. **Find the demo codes in your application source code. They may be in an included file or embedded directly in the source code.**

   - C/C++: *client.h*
   - VB: *GLOBAL.BAS*

   For other languages, look for a series of eleven lines of codes starting with *ValidateCode1*.

2. **Find your company-unique codes on the company-unique disc enclosed with your first production order.**

   The company-unique files are named *client.*.* We provide the files in several different formats. If none of the client files is the right format for your language and development environment, the *YOURCODES.DAT* file has the same information in very generic format so that you can edit the "demo" dongle codes and replace them with the codes assigned uniquely to your company.

3. **Copy your company-unique codes into your application source code in place of the demo codes and recompile.**

   Simply cut and paste your company-unique codes from the file you located in Step 2. in place of the demo codes you found in Step 1. Then recompile and re-link your application, and it's ready to use with your production dongles.

Note: When switching from demo to production **parallel** dongles, you must stop and re-start the driver service in order to be able to recognize the production dongles. This can be done either by switching dongles and rebooting, or by doing the following:

1) Go to a command prompt (e.g., C:\>), type **net stop parclass**, and press Return. The system should respond that the parclass service was stopped successfully. Remove the demo dongle.
2) Attach a production dongle. Go to a command prompt, type **net start parclass**, and press Return. The system should respond that the parclass service was started successfully.

Note: When switching from demo to production USB dongles, you must also

a. 1)stop all programs that may be communicating with the dongle,
b. 2)remove the demo dongle, and
c. 3)connect the company unique dongle

S-LOK™ Quick Start

For KEYLOK® II only

To install the S-LOK™ software and protect your application, follow these steps:

1. **Install the software**

   Insert the CD. The software installer will auto run under Windows. (If the setup does not automatically, run *setup.exe* from the root directory of the CD.)

2. **Protect your application**

   a) Go to your \Program Files\KEYLOK2\S-LOK™ directory and run SHRLOK32.EXE.

   b) Click on the Input File button and select your application.

   c) Click on the Output file button and select a name for the protected version of your application. We recommend making a backup copy of your unprotected application with another name (e.g., <file name> UNPROT) if you want to use the original name for the protected version of your application.

   See the ***Protecting with S-LOK™*** section of this manual for more info.

Serial Quick Start

For KEYLOK® II only

Follow these four steps to protect your application with a serial KEYLOK® II in 30 minutes:

1. **Install the software.** 5 Minutes

   Insert the CD. The software installer will auto run under Windows. (If the setup does not automatically, run *setup.exe* from the root directory of the CD.)

2. **Run the demo.** 5 Minutes

   a) Attach your serial dongle.

   b) Run *KLSERIAL.EXE* (in the ..\KEYLOK2\Serial\Windows directory) to familiarize yourself with the features and functionality of the KEYLOK® II security system.

3. **Compile the sample code.** 5 Minutes

   a) Compile the serial demo, using the C/C++ compiler of your choice. We provide Microsoft C/C++ 6 files as well as the source code.

   b) Run the *klserial.exe* executable that you built from the serial demo code in Step 3a. It should look and behave identically to the pre-compiled version you ran in Step 2.

3. **Protect your application.** 15 Minutes

   a) Copy the *client.h* and *access.c* files from the serial demo directory to your application source directory.

   b) Copy and paste the desired features from the sample code into your application.

   See also the ***Serial Port Devices*** section of this manual.

Mac Quick Start

For KEYLOK® II only

Follow these three steps to protect your Mac OS X gcc application with a USB KEYLOK® II in 30 minutes:

1. **Install the software and the dongle.** 5 Minutes

   Go to the \SampleCode\Mac folder on the CD and drag it to the Desktop. Attach your dongle to a USB port.

2. **Compile and run the sample code in gcc.** 5 Minutes

   a) Open Terminal.

   b) Go to the Mac folder on the Desktop. Select the 32-bit/Demo or 64-bit/Demo directory

   c) Compile the demo:

   make –f Makefile

   d) Run the demo:

   ./demoA

3. **Protect your application.** 20 Minutes

   Simply copy and paste the desired features from the sample code into your application.

Linux Quick Start

For KEYLOK® II only

Follow these three steps to protect your Linux application with a USB or parallel KEYLOK® II device in 30 minutes:

1. **Install the software.** 5 Minutes

   Go to the \SampleCode\Linux directory on the CD and copy the USB and/or parallel directories to some convenient directory. They are separated between 32 and 64 bit directories

2. **Compile and run the sample code in gcc.** 5 Minutes

   a) Attach your demo dongle.

   b) Go to the appropriate directory (USB/32-bit/Demo or Parallel).

   c) Compile the demo:

   make –f makefile

   d) Run the demo:

   ./demoA for USB, ./demo for Parallel

3. **Protect your application.** 20 Minutes

   Simply copy and paste the desired features from the sample code into your application.

   See the ***Using KEYLOK® Under Linux*** section of this manual for more info.

Please Note: Follow the same steps for QNX or FreeBSD, but use the sample code and object(s) in the appropriate directory in the sample code on the CD.

See the Using KEYLOK Under Linux section for more information.

Introduction

KEYLOK® is proud to offer the most complete solution to software security in the marketplace. KEYLOK® pioneered dongle based software piracy prevention in 1980, and has continued to lead the industry throughout the history of software protection.

Our product offerings include:

KEYLOK® II: The KEYLOK® security system provides a very high degree of security with an economical hardware key. KEYLOK® devices are available for parallel, USB and serial ports on computers running DOS, WINDOWS 3.x / 9x / ME / NT / 2000 / XP / Vista / 7 / Server2003 / Server2008 / Server 2008 R2, LINUX, FreeBSD, QNX or Macintosh OS X. Serial port dongles can be used on any computer with an RS232 port and do not require a device driver. KEYLOK® devices have programmable memory, remote update, lease control algorithms, and networking capability for controlling multiple users with a single hardware lock.

FORTRESS: (Windows USB only) Backwards compatible to KEYLOK II, the dongle also offers two distinct differences. Firstly, the dongle operates in HID mode: you do NOT need to install a device driver. And secondly, Fortress allows you to migrate functions from your code to execute only on the dongle providing you with an unparalleled security solution. Fortress also provides larger user memory (5,120 bytes) and provides an added level of tamper resistance, increasing the protection against piracy through reverse engineering.

S-LOK™: S-LOK™ is a special version of KEYLOK® II that is capable of "shrink-wrapping" a protected shell around a program executable without having to make source code modifications. S-LOK™ is currently compatible with only parallel port dongles.

Throughout this manual the KEYLOK® II product will be referred to generically as KEYLOK®. The parallel port version of this electronic security device attaches to any parallel printer port on the computer (the software automatically searches each port until the device is located); the USB version may be used in any available USB port, and the serial version may be used on any standard RS232C 9-pin serial port. Software interfaces to the security system and associated demonstration programs are available for most programming languages and development environments. A demonstration program prepared in the language that you are using makes implementation within your program quite simple. Robust protection algorithms defend against determined pirates attempting to bypass security.

General Product Overview

The KEYLOK® security system protects your software applications from piracy, thereby increasing your revenues associated with software sales. The security is transparent to your end-user once the hardware device is installed on the computer's USB, parallel or serial port. Unlimited backup copies of the program can be made to protect your clients with the knowledge that for each copy of your software sold only one copy can be in use at any one time. Your clients can install the software on multiple machines (e.g. at the office and at home) without having to go through difficult and time-consuming install/uninstall operations. Your clients can easily restore or reinstall copies of your software following catastrophic events such as hard disk failures. These advantages provide your clients with the features they desire and deserve while preserving your financial interests.

The KEYLOK® security system uses a number of sophisticated techniques for verification of hardware device presence. The KEYLOK® is also provided with 112 bytes (5,120 bytes for Fortress) of field programmable read/write memory. There are NO special programming adapters required to program the dongle memory.

When first attempting to communicate with the hardware device it is necessary to successfully complete an exchange of bytes between the host and the device that differs during each device use (i.e., using an active algorithm). If this sequence is properly executed the device will return a customer unique 32-bit identification code which you can use as confirmation that one of your hardware security devices is installed on the computer. If an improper data exchange occurs then the security system returns to the host a random 32-bit code in place of the proper identification code. Upon successful completion of the authentication sequence, the host computer then sends a 48-bit customer unique password to the device to authorize memory read operations. Memory write operations require the successful transmission of yet another 48-bit customer unique password to the device. The write password must be sent AFTER transmission of the proper READ authorization sequence. If the device is sent the incorrect read/write password then subsequent memory operations are ignored and random information is returned to the program. In summary, a total of 176 bits of customer unique codes must be known in order to alter the memory within the dongle.

Up to fifty-six (56) separate 16-bit counters (values of 0-65535) can be independently maintained within the device. Counters are particularly useful for controlling demonstration copies of software, as well as pay-per-use software (e.g. testing). Some clients use the counter as a means of controlling software use up until the time they have been paid for the software, and then provide their clients a special code that 'unlocks' the device for unlimited future use.

At the time of manufacturing each device is programmed with a unique device serial number, thus providing you the capability of identifying the specific device (and thus specific end-user) for customers requiring this level of control.

The security system includes algorithms for performing very secure remote memory modifications to the device. The remote update procedure involves the exchange back and forth between the end-user and you, of a series of numeric values displayed on the system console. These values are entered into the security system to activate a desired memory change at the end-user's site. This can be used to query memory, replace memory contents, extend counters, extend lease expiration dates, add additional network licenses, etc. The specific sequence used to effect a memory change will only work one time, and only on one specific dongle. Various solutions for remote update have been provided. You should select the one most appropriate and consistent with the type of interface you expect to have with your

end users. See the Remote Update section of this manual for a more thorough explanation of the available options.

Sophisticated algorithms allow the client's system clock to be used as an economical means of controlling leased software. The most recent system clock date and time are stored internally within the KEYLOK® memory. Any attempt by the end-user to set back the date and/or time generates appropriate error codes to your application. Depending upon your needs, a real time clock option is available for the Fortress LS dongle.

The KEYLOK® price-to-feature ratio is unparalleled in the industry.

Implementation Overview

KEYLOK® protection for your application is implemented by embedding calls to our Application Programming Interface (API) into your application source code (see Fig. 1). The references to our API are satisfied by linking your code to either a Windows DLL or a library file (depending upon your programming language and development environment). Our device drivers (not required for Fortress), which can easily be installed using a single-file install routine, which we supply, provide the necessary communications interface between your code and the dongle. (Note that serial devices do not require drivers, but use a few lines of additional code to implement communications with the device).

In most cases, the bulk of the necessary modifications to your source code can be made simply by cutting and pasting from the sample code supplied with our Software Development Kit (SDK). You can choose to implement any level of security you wish, depending only on which features of KEYLOK® you want to use. The simplest and most basic security, a simple check for the presence of the KEYLOK®, can be implemented in fewer than 20 lines of code in most cases.

If you need to pre-program your devices (to set a lease expiration date or a usage counter, for example), you can do so using KLTOOL.exe that we supply. We also supply utilities for verifying that any needed drivers were installed correctly (verifykey.exe/verifynetworkkey.exe) and for remotely updating the contents of the dongle memory or the lease expiration date, either by an exchange of e-mails or over the telephone.

Networking (in which one dongle is mounted on a central dongle server and accessed remotely by clients via a TCP/IP network) is equally easy to implement. You only need to link your application with the network version of our DLL or object file and install the appropriate service. Our TCP/IP network devices are available in versions that support from 3 to an unlimited number of simultaneous users (our standard devices will support one user at a time over a network connection). We supply utilities for verifying that a remote dongle can be accessed and for diagnosing common problems with the dongle access via a network.

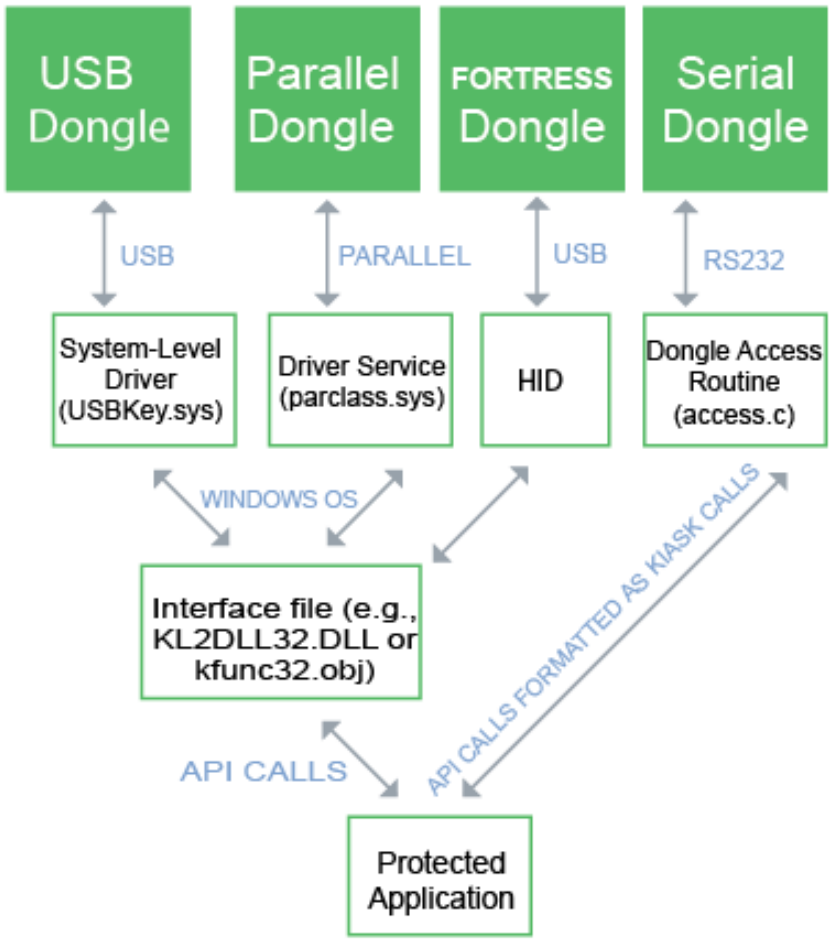

Figure 1: How Dongles Communicate with a Protected Application

Each KEYLOK® device is individually programmed by KEYLOK before shipping with a set of access codes and a serial number, which can't be changed after programming. When you first purchase production devices, we assign your company a unique set of codes that will allow you and only you to access your dongles. We maintain an extensive set of internal security procedures to protect your company-unique access codes and to ensure that they are never sent to anyone except your designated contacts without your prior written authorization. We will provide you with these codes on a disc that will be included with your first production order.

The demo devices that are included in our evaluation kits are identical in every way to our production devices, except that they are all programmed with company-unique information for the same fictitious company, and the access codes necessary to communicate with the demo devices are built into all of our sample code. To convert from using a demo dongle to using your production devices, all you need to do is cut and paste the codes we send you in place of the demo access codes and recompile your application. We also supply company-unique versions of our dongle programming and troubleshooting utilities.

Getting Started

Protecting your application can be done in 3 simple steps:

Step 1: Install the Software Development Kit

The KEYLOK® Software Development Kit (SDK) contains all of the necessary hardware and software to protect your application.

The installation program has an easy-to-use graphical interface. Insert the KEYLOK® SDK CD into the CD ROM drive. The CD will automatically run under Windows. If the CD does not automatically run, execute d*:\setup.exe* manually. ('d' is the directory of the CD ROM drive).

Step 2: Run the sample source code

Sample code is provided for over 50 different compilers and development tools. If you did not see sample code for your compiler during installation please contact technical support.

Examine the source code for the demonstration program (DEMO.xxx) written in the software development language that you are using to see how to implement the KEYLOK® API calls. Sample source code will be installed in the following directory:

Program Files\KEYLOK2\Samples\<operating system>\<language>

Application specific notes are provided in the form of a *README.TXT* file. Sample code for older programming languages and development environments can be found in:

Program Files\KEYLOK2\Samples\Archive\<language>

Step 3: Move relevant portions of the sample code to your application

Once you are familiar with the KEYLOK® API calls simply copy and paste the relevant calls into your application.

Review the ***Security Considerations*** section of this manual for ideas on how to increase the security using the KEYLOK® II system.

NOTE: Access to the KEYLOK® API calls is accomplished through a LIB file or DLL file depending on your development tool. The appropriate interface file will be found in the sample code directory for your programming language and development environment.

Security Considerations

The following suggestions are intended to help you increase the level of protection built into your application:

Enhancing Security

The ***Check for KEYLOK®*** process in the demonstration program involves a comparison of actual return values to expected values. This technique is open to debugger analysis and thus being patched around by experienced software pirates. A much more secure approach is to use the returned values in calculations performed in your program, and only much later in the program to analyze the results of the calculation to determine whether or not to proceed with program execution or to exit. The more you can bury the decision process by which you conclude the presence of a proper security device, the more effective the total security system will be.

Use of Device Memory

Memory within the security device can be used for many purposes, as follows:

- The most common use of device memory is to control licensing of multiple product and/or features within a product. You have the option of using a complete word (16 bits) of memory to control a single feature or alternately to use individual bits within a single memory address to perform the same task.
- Product revision control information can be stored within device memory. See the ***Product Version Control*** section below that provides additional suggestions regarding this capability.
- Store client name and or copyright message as text within the device for display from within your application.
- Store critical constants used in calculations within your program.
- Use memory for counter(s) to control client billing whereby charges are made for each service performed by the program, rather than a fixed purchase price for the software.

A random word can be written to the device during one part of the protected program execution, and then read back again at another point, as another means of confirming the presence of the device.

Demo Mode

Many clients place their software in 'demo' mode if the security device is not found. Clients are then encouraged to make copies of the application to distribute to their friends. This provides a great marketing strategy that pays off with additional sales.

Product Version Control

This section addresses the issue of using the KEYLOK® device to assure that your end-user purchases product upgrades.

Reserve one or more addresses within the KEYLOK® device memory for writing license information regarding which products and/or features are accessible to the user. When you create a product upgrade, change the required value in the reserved memory location that is necessary to work with the upgrade. For example, the prior product release might be version 6 and the number 6 is programmed into the device memory for version control. The software upgrade reads this memory location, and if it isn't 7 or larger (the current version) the program refuses to execute and provides the user with appropriate instructions for acquiring permission to use the upgrade.

Two techniques can be used to update the dongle memory.

- Using the **Remote Update** capabilities of the KEYLOK® system the memory containing the revision control code can be updated via an email exchange or a phone conversation with your client. Refer to the ***Remote Update*** chapter of this manual for additional information.

- Another technique would be to send out a new security device with the software upgrade. The device would be programmed with the appropriate authorization for use with the upgrade.

Extended Memory (Fortess Only)

The Fortress dongle provides 5,120 bytes of base memory, expandable up to 55,000 bytes. The increased size of memory opens up additional licensing options, including date-based licensing and counters. Additional memory allows the software developer to support multiple products with one dongle and store user information such as user settings and user preferences on the dongle, allowing it to travel with user from machine to machine. In addition, data can now be stored on the dongle rather than in the application database or file so that it also travels with the user. The KBLOCK function provides you with the ability to read and write blocks of memory at a time, greatly simplifying the management of the extended memory.

Executable Code on The Dongle (Fortess Only)

The extended memory of the Fortress dongle can also be used to store executable and data files which only reside on the dongle. At no time are these files transferred to the local computer. The smart card based device provides a secure computing environment. You select a set of key functions which when not present would render your software useless. These functions are migrated to the dongle and called through the KEXEC function so your software only operates when the dongle is present. The ability to separate the code provides you with the most secure software protection solution available.

Anti-Debugging Utility

PPMON.EXE is a utility that prevents a debugger from being attached to your executing program. The Anti-Debugger is activated by calling the KEYBD(OFF) function. Although the function name implies that the keyboard is turned off, actually the anti-debugging utility *PPMON* is launched. This adds much greater security to your protected program.

KEYLOK® API Reference

In this chapter we describe the KEYLOK® Application Programming Interface (API).

All KEYLOK® hardware can be accessed through one common interface. Whether you access the USB, parallel or serial port dongle the function calls are identical. Therefore you can access any of the KEYLOK® products using the same source code.

The KEYLOK® API has these important advantages:

- Easy-to-use function calls
- Local access through LPT, RS232 serial and USB ports
- Remote access via TCP/IP
- Hardware access via device drivers for Windows 9x/ME/NT/2000/XP/Vista/7/Server2003 / Server2008/Server2008R2
- Hardware access via standard RS232 calls for any computer or OS with RS232 support

The KEYLOK® API is Easy, Secure and Portable

Overview

Access to the KEYLOK® API is accomplished by, either linking with one of the KEYLOK® library (LIB) files or utilizing the KEYLOK® DLL (KL2DLL32.DLL). If a DLL is required for your development language the appropriate DLL(s) will be included in the same directory as your sample source code. Some development languages and operating system versions support finding the DLL in the same directory as the application, but many require that you copy the DLL to the \Windows or \Windows\system32 directory.

Most KEYLOK® API calls are made with the exposed *KFUNC* function. For ease and readability this function has been encapsulated into a function *KTASK* within the sample code.

The following calling sequence is required for most development languages. Check the KTASK function in the demonstration program for the appropriate calling sequence for your language/compiler/environment.

Command Code - KFUNC(Command Code, Arg2, Arg3, Arg4)

Argument requirements vary by function (see function description for details). The first argument is the **Command Code** and is used to select the task to be performed, as follows:

Command Code	Value	Applies To	Page
KLCHECK	1	USB, Parallel, Serial, Fortress	23
READAUTH	2	USB, Parallel, Serial, Fortress	25
GETSN	3	USB, Parallel, Serial, Fortress	26
GETVARWORD	4	USB, Parallel, Serial, Fortress	26
WRITEAUTH	5	USB, Parallel, Serial, Fortress	27
WRITEVARWORD	6	USB, Parallel, Serial, Fortress	28
DECMEMORY	7	USB, Parallel, Serial, Fortress	28
GETEXPDATE	8	USB, Parallel, Serial, Fortress	30
CKLEASEDATE	9	USB, Parallel, Serial, Fortress	31
SETEXPDATE	10	USB, Parallel, Serial, Fortress	32
SETMAXUSERS	11	USB, Parallel, Fortress	33
GETMAXUSERS	12	USB, Parallel, Fortress	34
REMOTEUPDUPT1	13	USB, Parallel, Serial, Fortress	38
REMOTEUPDUPT2	14	USB, Parallel, Serial, Fortress	38
REMOTEUPDUPT3	15	USB, Parallel, Serial, Fortress	39
REMOTEUPDCPT1	16	USB, Parallel, Serial, Fortress	39
REMOTEUPDCPT2	17	USB, Parallel, Serial, Fortress	40
REMOTEUPDCPT3	18	USB, Parallel, Serial, Fortress	40
DISABLESNCHECK	19	Parallel	41
GETNWCOUNTS	20	USB, Parallel, Fortress	42
DOREMOTEUPDATE	21	USB, Parallel,	42

		Fortress	
GETABSOLUTEMAXUSERS	**32**	**USB, Parallel, Fortress**	**43**
TERMINATE	**-1**	**USB, Parallel, Serial, Fortress**	**44**

CAUTION: Some languages require special care when generating arguments to assure that an illegal value is not assigned to the argument. An example would be attempting to assign the value 40,000 to a signed integer argument that can only have values between -32,768 and 32,767. The sample programs demonstrate how to handle these situations. An examination of function 'KTASK' in the demonstration program will provide an example of the proper calling sequence.

Extended Fortress Functions

Two additional functions are available which take advantage of the advanced features of the Fortress dongle. These are not API calls to KFUNC but are separate functions.

KBLOCK - KBLOCK(Task, Address, WordLength, pData) Fortress Only

These functions are designed to allow quick and efficient reading/writing of large amounts of data to/from the dongle without incurring significant overhead for each word of data to be exchanged

Command Code	Value	Applies To	Page
BLOCKREAD	**84**	**Fortress**	**45**
BLOCKWRITE	**85**	**Fortress**	**45**

KEXEC - KEXEC(LPSTR ExeDir, LPSTR ExeFile, LPSTR UserPin, LPSTR Buffer, USHORT BufferSize) Fortress Only

KEXEC provides you with the ability to execute your code directly on the Fortress dongle, which offers significant security advantages. The code is only executed on the dongle and data is passed between your application and the dongle using a 250 byte buffer. The code on the dongle cannot be inspected by a would-be hacker, thus providing the ultimate in security for special algorithms that make your protected application particularly valuable, and without which your program will not perform its expected functionality. Your function(s) is loaded onto the dongle as an individual program(s) in a directory structure.

Please see the KEYLOK®Code On Dongle Manual for details on the type of code you can transfer to the dongle.

Check For KEYLOK®

A successful ***Check for KEYLOK®*** process is a prerequisite to running any other security device task (e.g. reading or writing memory, etc). Many companies are content to use only the ***Check for KEYLOK®*** process for protecting their software. This task involves verifying that a device built uniquely for your company is present.

CHECK FOR KEYLOK®

All other security device functions MUST be preceded by a successful ***Check for KEYLOK®*** event. The ***Check for KEYLOK®*** involves an exchange of information between the host and the security system using a different series of bytes each time the device is interrogated (i.e. using an active algorithm). This task requires two sequential calls to *KFUNC*, as follows:

FIRST CALL:

Check For KEYLOK® (First Call)	
Prerequisite	None
Argument1	**KLCHECK = 1**
Argument2	**Validate Code 1**
Argument3	**Validate Code 2**
Argument4	**Validate Code 3**
Return Values	Each of the return arguments (ReturnValue1 and ReturnValue2) from this first call must be manipulated to create the arguments sent during the second call.

SECOND CALL:

Check For KEYLOK® (Second Call)	
Prerequisite	This call must be immediately preceded by the first call of the ***Check for KEYLOK®*** sequence
Argument1	Exclusive OR (XOR) the constant **ReadCode3** with ReturnValue2 and then XOR the resulting value with the value created by rotating the bits in ReturnValue1 left by a rotation count value established by ANDing ReturnValue2 with 7.
Argument2	The value created by rotating the bits in ReturnValue2 by a rotation count value established by ANDing ReturnValue1 with 15.
Argument3	The value created by XORing ReturnValue1 and ReturnValue2.
Argument4	Dummy argument. Whenever a dummy argument is called for, you can substitute any value (e.g., either zero or a random number).
ReturnValue1	ReturnValue1 = **ClientIDCode1**
ReturnValue2	ReturnValue2 = **ClientIDCode2**

If both parts of the customer identification code (**ClientIDCode1** and **ClientIDCode2**) are successfully retrieved from the device then you have the option of proceeding with other device operations.

NOTE: Each time a ***Check for KEYLOK®*** is performed the *ReadAuthorization* and *WriteAuthorization* flags are reset (i.e. deactivated). This means that the authorization sequence must be re-sent prior to any memory read/write operation.

Read Operations

Upon successful completion of the ***Check for KEYLOK®*** you have the option of adding additional security to your program by making use of either the device serial number (unique to each device) and/or the programmable memory within the device. Examples of what can be done with the device memory include:

- Writing a random value to the device and reading it back later to confirm device presence
- Storing a copyright message or the actual name of your client within the device memory and reading or displaying this information from within your program,
- Storing critical constants used in program calculations within the device,
- Storing licensing information used to enable which of multiple products sold by your company can be executed, or alternately which features within an application can be executed,
- Storing a count of the number of uses available (counters)

READ AUTHORIZATION

Prior to running any read-related task, a successful ***Check for KEYLOK®*** must be completed. The first read-related task that MUST be executed is ***Read Authorization***; successful completion of this task authorizes the device to perform memory read operations. If the device does not receive the correct Read password subsequent read operations retrieve random information. It is not necessary to perform the ***Read Authorization*** call unless you intend to read the device serial number or other memory within the device after completing the ***Check for KEYLOK®***.

Read Authorization (READAUTH = 2)	
Prerequisite	Successful ***Check for KEYLOK®***
Argument1	**READAUTH = 2**
Argument2	**ReadCode1**
Argument3	**ReadCode2**
Argument4	**ReadCode3**
ReturnValue1	None
ReturnValue2	None

NOTE: After a ***Read Authorization*** there is no indication of success or failure of the operation. We intentionally avoid a success/failure code in order to complicate the task of someone writing a program to simply test all possible combinations until a success flag is returned. The only way to know that you have had success is to read some memory value (e.g. serial number, etc.) twice and confirm the same number was received both times. However, from a practical perspective, if you have run a successful ***Check for KEYLOK®*** with the device and you have sent it the proper authorization codes, then the return values from read operations will be

correct. If you fail in any of the prerequisites then the return values from read operations will be random numbers. The same discussion is applicable to the write authorization command discussed later in this manual.

READ SERIAL NUMBER

This task retrieves the unique serial number programmed into the security device. No two devices with the same company-unique information will contain the same serial numbers.

Read Serial Number (GETSN = 3)	
Prerequisite	Successful ***Read Authorization***
Argument1	**GETSN = 3**
Argument2	Dummy argument*
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	The device serial number
ReturnValue2	Undefined

*It is recommended that a random number be passed for dummy arguments.

READ MEMORY

This task allows you to retrieve information written into the programmable memory. Remember that the 112 bytes of memory are partitioned into 56 addressable memory cells (addresses 0-55).

Read Memory (GETVARWORD = 4)	
Prerequisite	Successful ***Read Authorization***
Argument1	**GETVARWORD = 4**
Argument2	Desired address (0 - 55)
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	The memory contents
ReturnValue2	Undefined

*It is recommended that a random number be passed for dummy arguments.

Write Operations

Memory within the device can be used to store information related to your program. This can either be 'static' or 'dynamic' information. An example of static information would be something you write into the device before you ship it to your client, which is subsequently read while attached to your client's computer. An example of dynamic information would be something that is written into and read from the device while it is attached to your client's computer. An example would be a counter.

In general the prerequisites to memory write operations are:

1) Successful ***Check for KEYLOK®***
2) Successful ***Read Authorization***
3) Successful ***Write Authorization***

WRITE AUTHORIZATION

Successful completion authorizes device to perform memory write operations. If the device does not receive the correct Write password the device will ignore write operations.

Write Authorization (WRITEAUTH = 5)	
Prerequisite	Successful ***Read Authorization***
Argument1	**WRITEAUTH = 5**
Argument2	**WriteCode1**
Argument3	**WriteCode2**
Argument4	**WriteCode3**
ReturnValue1	Undefined
ReturnValue2	Undefined

WRITE A VARIABLE WORD

This task is used to modify the contents of programmable read/write memory within the KEYLOK®.

Write Variable Word (WRITEVARWORD = 6)	
Prerequisite	Successful ***Write Authorization***
Argument1	**WRITEVARWORD = 6**
Argument2	Target address (0 - 55)
Argument3	Desired contents
Argument4	Dummy argument*
ReturnValue1	Undefined
ReturnValue2	Undefined

*It is recommended that a random number be passed for dummy arguments.

DECREMENT A COUNTER

This task is used to decrement the contents of a memory location. This is useful when a particular memory word is being used as a counter. The calling program receives the result of the decrement process by return of an ERROR code.

Decrement Counter (DECMEMORY = 7)	
Prerequisite	Successful ***Write Authorization***
Argument1	**DECMEMORY = 7**
Argument2	Target address (0 - 55)
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	The number of counts remaining if no error was encountered
ReturnValue2	ERROR condition codes: 0 No error 1 Memory has already been counted down to zero - no remaining counts 2 Invalid address requested 3 Write authorization not provided - must 'Write Authorize' before attempting to decrement memory

*It is recommended that a random number be passed for dummy arguments.

Date Control Tasks

The following tasks are used to control an expiration date for use in your product. They use reserved device memory, within which the last valid system date and time and an expiration date are written. Each time a successful call is made to compare the current date to the expiration date the most recent date and time information is refreshed within the device. If the date or time has been set back then you can disable your software from operating until the clock has been properly reset. You may wish to utilize a counter in conjunction with this function, and take more drastic action if the clock has been found set back more than once.

If your client is leasing your software or you have established a demonstration time period, then the remote update tasks described in the next section provide an ideal means of extending the expiration date. (You can also use the ***Remote Update*** utilities supplied by KEYLOK.) Further, if you embed the remote tasks in an application that also checks for the expiration date, then it is possible to force the client to have his system clock set properly, because unless his system date is correct (i.e. matches yours) the remote tasks will not operate.

If an end-user runs your expiration date protected program while the clock is set ahead, then the last-use date/time will be set into the future and they will no longer be able to run your software unless 1) your program expiration date is later than the date they set the computer to, and they set the computer date forward to the date/time that it was set to at the time they last ran your program, or 2) the last use date/time stored in the device is reset. The recommended solution to resolve this problem is to provide your end-user with remote update capability. When you perform a remote update using the extend expiration date task, the last use date/time will be reset to the current date/time on the end-user's computer. This is safe because remote update will only work if the end-user's computer is set to the same date as your computer. Also, the extend expiration task accepts a value of zero months for the amount of extension. Therefore, the net affect of performing this function is to simply reset the last use date/time, with no impact upon the expiration date setting.

When you set the expiration date within a security device, the last usage date/time are reset to the current system date/time on the computer on which the expiration date is programmed. This feature is useful for resetting last use date/time stored information if it becomes corrupted as a result of someone accidentally/intentionally setting his or her system date/time ahead. This often happens to our clients when testing the expiration date features of KEYLOK®.

CAUTION: When testing the lease expiration date related functions it is important that you not use an expiration date or system clock setting that is prior to the current year.

GET LEASE/DEMO EXPIRATION DATE

This task is used to read the expiration date. This date is used for comparison to the current system date as a means of establishing the remaining time period.

Get Expiration Date (GETEXPDATE = 8)	
Prerequisite	Successful ***Read Authorization***
Argument1	**GETEXPDATE = 8**
Argument2	Dummy argument*
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	The expiration date encoded in the following bit format: YYYYYYYMMMMDDDDD, where YYYYYYY + Base Year (i.e. 1990) = Year MMMM = Month of Year (1 = January) DDDDD = Day of Month (1-31) The sample code shows how to encode the date.
ReturnValue2	Undefined

TIP: Refer to the sample code for a better understanding of how to format the arguments.

****It is recommended that a random number be passed for dummy arguments.***

CHECK LEASE/DEMO EXPIRATION

This task is used to compare the expiration date stored in the KEYLOK® memory with the current date as read from the system clock. The purpose of the comparison is to establish whether or not the expiration date has been reached. This task also refreshes the last known valid date and time stored in the device as long as the current date and time are more recent. Any attempt on the part of the end-user to set back his clock will result in an error when running this task.

Check Lease Date (CKLEASEDATE = 9)	
Prerequisite	Successful ***Write Authorization***
Argument1	**CKLEASEDATE = 9**
Argument2	Dummy argument*
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	Computer's current System Date in the format YYYYYYYM MMMDDDDD (Where YYYYYYY + 1990 = Year) MMMM = Month of Year (1 = January) DDDDD = Day of Month (1-31)
ReturnValue2	Status Code - Result of comparison -2 = Lease expired (clock date greater than lease expiration date) -3 = System date has been set back -4 = No lease date (DateAddress contains '0'. This is the return code to be expected if a check is made with an as-manufactured device, to which you have not yet sent a desired expiration date) -5 = Invalid lease expiration date -6 = Last date system used is corrupt - unable to write. This error means that the device is not functioning properly. Either it has been hit by lightning, etc. and the memory has been altered, or some electrical failure occurred during memory write that prevented writing the correct value to device memory. This error code is used primarily for internal debugging purposes to identify 'bugs' in our code associated with the encryption/decryption/ update of this information. What is being stored in this memory is the last date on which a successful check-expiration-date task

	was performed. The value 'date' is only allowed to march forward. Only a trusted person can clear this error. Each time you execute the set-expiration-date task, the current system date (on the computer on which the device programming is being done) is written into this memory within the device. +n = Approximate number of days until lease expires.

TIP: Refer to the sample code for a better understanding of how to format the arguments.

It is recommended that a random number be passed for dummy arguments.

SET LEASE/DEMO EXPIRATION DATE

This task is used to initialize an expiration date. This date is used for comparison to the current system date as a means of establishing the remaining time period.

Set Expiration Date (SETEXPDATE = 10)	
Prerequisite	Successful ***Write Authorization***
Argument1	**SETEXPDATE = 10**
Argument2	Dummy argument*
Argument3	The expiration date encoded in the following bit format: YYYYYYYMMMMDDDDD (where YYYYYYY + Base Year (i.e. 1990) = Year MMMM = Month of Year (1 = January) DDDDD = Day of Month (1-31) The sample code shows how to encode the date. A value of zero for this argument will result in the expiration date check being disabled.
Argument4	Dummy argument*
ReturnValue1	Undefined
ReturnValue2	Undefined

TIP: Refer to the sample code for a better understanding of how to format the arguments.

*It is recommended that a random number be passed for dummy arguments.

Network Control Tasks

The following tasks are used to set or get the number of simultaneous authorized users of your application installed on a network. We provide a server application that communicates with the KEYLOK® device installed on the machine on which the server application is running. This technique works on any network operating system that has active support for the TCP/IP protocol. A protected program running on any node on the network can then access the device through the server application, up to the maximum simultaneous user count programmed into the dongle.

See section ***Device Access Over a Network*** of this manual for details related to using KEYLOK® security with networks.

SET MAX USER COUNT

This task initializes the count of authorized simultaneous network sessions.

Set Max User Count (SETMAXUSERS = 11)	
Prerequisite	Successful ***Write Authorization***
Argument1	**SETMAXUSERS = 11**
Argument2	The desired simultaneous session count (less than or equal to the limit set in the dongle hardware)
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	Undefined
ReturnValue2	Undefined

*It is recommended that a random number be passed for dummy arguments.

GET MAX USER COUNT

This task retrieves the maximum number of authorized simultaneous network session count as recognized by the device driver.

Get Max User Count (GETMAXUSERS = 12)	
Prerequisite	Successful ***Read Authorization***
Argument1	**GETMAXUSERS = 12**
Argument2	Dummy argument*
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	The maximum simultaneous network session count programmed into the dongle.
ReturnValue2	Undefined

*It is recommended that a random number be passed for dummy arguments.

Remote Update Tasks

Updating memory within an end-user's security device can be accomplished many ways. KEYLOK® provides Remote Update utilities that can be used either via telephone or via e-mail to update the contents of an end-user's device, but the API calls for performing remote update tasks by telephone are provided in case you wish to write your own remote update routine. One possibility would be to provide a utility that checks the security device, confirms the proper serial number, and then uses standard API calls to update dates/counters/memory etc., as required. Such a utility could be emailed, sent via diskette, or transmitted via other Internet services for execution. The remote update tasks described herein are designed to operate generically without the need for a specially tailored update utility.

The following tasks are used to provide remote device query and/or memory modifications capabilities. Two computers are required to demonstrate these tasks. One computer must be used to simulate the end-user, whereas the other is used to simulate your own facility, that being the software developer. When using a single security device (e.g. demo device) to test the sequence, the KEYLOK® device must be physically attached to the computer on which you are using the keyboard at each step of the exchange process. A complete sequence consists of six security system calls, three on each system, as well as three events involving the entering of numbers on one computer that are being displayed at the other computer facility. The three data transfer events are as follows (see Fig. 2):

1. The end-user invokes a utility for remote updates. This could be a separate program, or a function available from a pull-down menu from within the basic application. Two calls are performed to the security system '*RemoteUpdUPt1*' and '*RemoteUpdUPt2*'. Each call extracts 2 words of information about the end-users system. A checksum is computed over these 4 words, thus creating a fifth word. These 5 numbers are displayed on the end-user's system to be given to the software developer.

2. The software developer inputs the 5 numbers provided by the end-user. The application re-computes the checksum and verifies that the data were properly conveyed between the two individuals and keyed into the computer properly. If the data are correct then the developer is asked what type of remote task he wishes to perform, as follows:

    - Get the current contents of memory.
    - Add new value to existing value in memory (used to extend counters)
    - Bitwise OR new value to existing value in device memory (used to add additional licenses when individual bits are used to control access to applications or features within an application)
    - Replace existing value in device memory with a new value
    - Get the maximum network user count
    - Set the maximum network user count
    - Get the current lease expiration date
    - Extend the lease expiration date by 'n' months

    The first call to the developers security system is performed (RemoteUpdCPt1), passing the desired task, the memory address (if applicable), the data value associated with the task (if changes are to be made to the end-user's device), and the first value received from the end-user. A second security system call is made to '*RemoteUpdCPt2*' in order to pass the remaining three values received from the end-user to the security system. Provided there have been no data entry errors, and both computers are set to the same date, you will be notified as to which

serial number security device the end-user is working with, and will be provided with the first of three values required at the end-user's site. The third call is made to '*RemoteUpdCPt3*' to acquire the remaining two values. A checksum is computed over the three values to create the fourth. Each of the four numbers is displayed on the developer's system to be read to the end-user.

3. The end-user keys the four numbers conveyed to him by the developer into his computer. The application confirms that the proper checksum was entered, thus validating the data transfer process. The three data values are then passed to the security system call to '*RemoteUpdUPt3*'. Provided all of the correct information has been entered, the security system performs the requested task, and returns two arguments to the application running on the end-user's system. The raw results are encoded and a checksum is computed and displayed along with the encoded numbers to be conveyed to the software developer.

   The software developer enters the three numbers into his computer. The checksum is confirmed, the numbers decoded, and the results of the requested task are displayed.

   Prerequisites:
   a. Both computers must be set to the same date
   b. Successful write authorization or read authorization has been done, as appropriate.

   NOTE: The remote update tasks must be called in sequence with no intervening calls to other security system tasks.

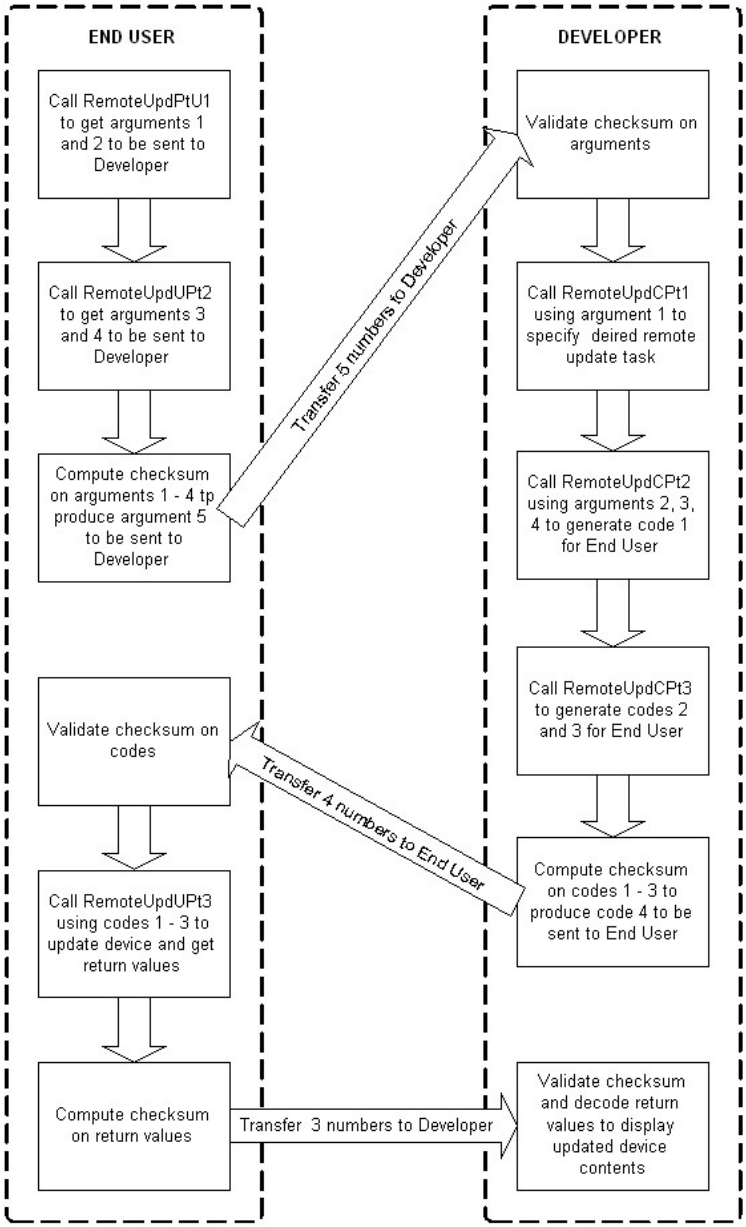

Figure 2: Remote Update Process Using API Calls

REMOTE UPDATE USER PART 1

This task is used to initialize the remote update process at an end-user's facility. It obtains two words of information relating to the configuration of the end-user's computer.

Remote Update User Part 1 (REMOTEUPDUPT1 = 13)	
Prerequisite	Successful ***Read Authorization***
Argument1	**REMOTEUPDUPT1 = 13**
Argument2	Dummy argument*
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	Argument 1
ReturnValue2	Argument 2

*It is recommended that a random number be passed for dummy arguments.

REMOTE UPDATE USER PART 2

This task is used to obtain the last two words of information relating to the configuration of the end-user's computer.

Remote Update User Part 2 (REMOTEUPDUPT2 = 14)	
Prerequisite	Successful ***Read Authorization***
Argument1	**REMOTEUPDUPT2 = 14**
Argument2	Dummy argument*
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	Argument 3
ReturnValue2	Argument 4

*It is recommended that a random number be passed for dummy arguments.

REMOTE UPDATE USER PART 3

This task is used to trigger the actual remote task. Most tasks return status information regarding the transfer activity.

Remote Update User Part 3 (REMOTEUPDUPT3 = 15)	
Prerequisite	Successful ***Write Authorization***
Argument1	**REMOTEUPDUPT3 = 15**
Argument2	Activation Code 1
Argument3	Activation Code 2
Argument4	Activation Code 3
ReturnValue1	Return Argument 1 - contents are task dependent
ReturnValue2	Return Argument 2 - contents are task dependent

The return values are encoded and contain the current value of the affected memory area within the device either queried or updated by the remote update call.

REMOTE UPDATE CLIENT PART 1

This task is used to initialize the remote update process at the developer's facility.

Remote Update Client Part 1 (REMOTEUPDCPT1=16)	
Prerequisite	Successful ***Write Authorization***
Argument1	**REMOTEUPDCPT1 = 16**
Argument2	*RemoteUpdateTask* * 8192 + Address Where 'RemoteUpdateTask' is: REMOTEADD 0 REMOTEDATEEXTEND 1 REMOTEOR 2 REMOTEREPLACE 3 REMOTEGETMEMORY 4 REMOTESETUSERCT 5 REMOTEGETUSERCT 6 REMOTEGETDATE 7 And address is the target memory address (i.e. 0 through 55).
Argument3	Value

Argument4	Argument 1 from Remote Update User Part 1
ReturnValue1	Status - '0' = success
ReturnValue2	Undefined

REMOTE UPDATE CLIENT PART 2

This task is used to pass the remaining arguments acquired from the end-user to the security system.

Remote Update Client Part 2 (REMOTEUPDCPT2=17)	
Prerequisite	Successful ***Write Authorization***
Argument1	**REMOTEUPDCPT2 = 17**
Argument2	Argument 2 from Remote Update User Part 1
Argument3	Argument 3 from Remote Update User Part 2
Argument4	Argument 4 from Remote Update User Part 2
ReturnValue1	Code 1 to be conveyed to end-user
ReturnValue2	If High bit = 1 then error encountered. Either the data was not entered correctly, or the two computers are not set to the same date. If High bit = 0, then this argument contains the serial number of the device being used at the end-user's facility to perform the remote transfer.

REMOTE UPDATE CLIENT PART 3

This task is used to acquire the remaining codes needed by the end-user to complete the remote transfer process.

Remote Update Client Part 3 (REMOTEUPDCPT3=18)	
Prerequisite	Successful ***Write Authorization***
Argument1	**REMOTEUPDCPT3 = 18**
Argument2	Dummy argument*
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	Code 2 to be conveyed to end-user
ReturnValue2	Code 3 to be conveyed to end-user

*It is recommended that a random number be passed for dummy arguments.

Disable Serial Number Checking

(Only Relevant with Parallel Port Dongles)

This task is used **strictly for internal use when programming security devices** using the *parallel* port dongle device driver.

Under 'normal' parallel port driver operation the serial number of the KEYLOK® device is read when the driver is activated. All subsequent calls to the security system check to make sure that the serial number of the attached parallel device has not changed from the original one detected. If a serial number switch is detected then security system calls simply return random numbers for return arguments. This is a security feature designed to protect you. For example, if you are performing a remote update at a client's facility that has two of your products, one inexpensive and the other expensive, each controlled by a separate device, the end-user could indicate to you a desire to acquire a lease extension on the lower price product and surreptitiously have it applied to the device associated with the more expensive product by switching the device during the remote update process. Continuous serial number checking prevents this and many similar scenarios from happening.

When this task is called, subsequent serial number checking is disabled until the driver is restarted.

DISABLE SN CHECK

Disable SN Check (DISABLESNCHECK = 19)	
Prerequisite	Successful ***Read and Write Authorization***
Argument1	**DISABLESNCHECK = 19**
Argument2	Dummy argument*
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	Undefined
ReturnValue2	Undefined

*It is recommended that a random number be passed for dummy arguments.

GET CURRENT NUMBER OF USERS

This task retrieves the current number of active sessions and the maximum authorized simultaneous network user count. A similar functionality is available in the form of a utility, *NetKeyMonitor.exe*. See section 5.1 of this manual for further details regarding this utility.

Get Network Counts (GETNWCOUNTS = 20)	
Prerequisite	Successful ***Read Authorization***
Argument1	**GETNWCOUNTS = 20**
Argument2	Dummy argument*
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	The current number of active sessions communicating with the security device.
ReturnValue2	The maximum number of authorized simultaneous network sessions programmed into the dongle.

*It is recommended that a random number be passed for dummy arguments.

DO REMOTE UPDATE

This task retrieves the AUTHORIZE.DAT file and processes the remote update against the dongle. The serial number of the dongle must match one of those within AUTHORIZE.DAT A similar functionality is available in the form of a utility, *RemoteUpdateEmailUser.exe* See the Remote Update section of this manual for further details regarding this utility.

Do Remote Update (DOREMOTEUPDATE= 21)	
Prerequisite	Successful ***Write Authorization***
Argument1	**DOREMOTEUPDATE = 21**
Argument2	Dummy argument*
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	Undefined
ReturnValue2	Undefined

*You can pass dummy arguments and RemoteUpdateEmailUser.exe will look for AUTHORIZE.DAT in the directory of the executable. However, you can set Argument2 = 1357 and then you can pass a pointer to the ANSI string containing the path and filename you want to process in Argument4. If you are building a 64-bit app, you will need to use Argument3 to pass the high address and use Argument4 for the low address. It is recommended that a random number be passed for dummy arguments if you are not setting the name and path of the AUTHORIZE.DAT file.

GET ABSOLUTE MAX USER COUNT

This task retrieves the absolute maximum number of simultaneous users set in the hardware of a multi-user network dongle. This can be used to identify a networking dongle from a non-networking dongle.

Get Absolute Max User Count (GETABSOLUTEMAXUSERS = 32)	
Prerequisite	Successful ***Write Authorization***
Argument1	**GETABSOLUTEMAXUSERS = 32**
Argument2	Dummy argument*
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	The number of allowed TCP/IP users built into the dongle.
ReturnValue2	Undefined

Examples:

KTASK(GETABSOLUTEMAXUSERS, dummy, dummy, dummy)

A 25-user TCP/IP dongle would return 25.

Demo dongles and single-user dongles will return 1.

*It is recommended that a random number be passed for dummy arguments.

Dongle Driver Termination

This task is normally only used to allow swapping of USB dongles when programming dongles using an application that you have written. The following sequence should be used to allow swapping of USB dongles for programming:

1) Call KTASK(TERMINATE, …) within your application
2) Remove 1st USB Dongle
3) Insert next USB Dongle
4) CheckForKL
5) Other programming tasks as necessary

The best practice is to call TERMINATE from your protected application.

TERMINATE

Terminate (TERMINATE = -1)	
Prerequisite	None
Argument1	**TERMINATE = -1**
Argument2	Dummy argument*
Argument3	Dummy argument*
Argument4	Dummy argument*
ReturnValue1	Undefined
ReturnValue2	Undefined

*It is recommended that a random number be passed for dummy arguments.

Extended Fortress Functions

These three additional Fortress functions are separate from the KFUNC API calls. They are separate functions and require separate declarations.

Block Read / Block Write

These functions are designed to allow quick and efficient reading/writing of large amounts of data to/from the dongle without incurring significant overhead for each word of data to be exchanged. The first argument to the KBLOCK function is the Task. It is used to identify whether you are reading or writing a block of memory. The second argument 'DongleMemoryAddress' is the starting physical address of the word within the dongle at which reading/writing is to begin. The third argument is the number of words of memory to be read/written. The fourth argument is a pointer to the array containing the data to be written, or the pointer to the array to receive the data read from the dongle.

KBLOCK (Task, Address, WordLength, pData)	
Prerequisite	Read Authorization for BLOCKREAD and Write Authorization for BLOCKWRITE
Task	**BLOCKREAD = 84 / BLOCKWRITE = 85**
Address	Starting word address of the memory within the dongle.
WordLength	The number of words of memory to be read/written. NOTE: One word can contain two characters of text, if desired.
pData	Pointer to the array that contains data to be written to the dongle, or to receive data read from the dongle.

Execute Code On Dongle KEXEC

This task provides you with the ability to execute your code directly on the Fortress dongle, which offers significant security advantages. The code is only executed on the dongle and data is passed between your application and the dongle using a 250 byte buffer. The code on the dongle cannot be inspected by a would-be hacker, thus providing the ultimate in security for special algorithms that make your protected application particularly valuable, and without which your program will not perform its expected functionality. Your function(s) is loaded onto the dongle as an individual program(s) in a directory structure.

The first argument is a pointer to a string containing the name of the folder in the directory structure that contains your code and optionally a data file(s) that will be used by your code. The name of this folder is assigned by KEYLOK personnel for your company.

The second argument is a pointer to a string containing the name of the program that contains your company unique code. One or more programs can be stored within the dongle to execute your application unique algorithms. The program file name(s) are assigned by KEYLOK personnel.

The third argument is a pointer to a string that contains an eight character (64 bit) password that must be known to execute programs stored in your company folder on the dongle. This

value can be changed from the default value assigned by KEYLOK personnel to whatever value you wish.

The fourth argument is a pointer to a buffer that is used for passing arguments to/from your company unique functions.

The fifth argument is the size of the buffer in bytes (unsigned short) containing IN/OUT arguments (maximum of 250 bytes).

KEXEC (LPSTR ExeDir, LPSTR ExeFile, LPSTR UserPIN, LPSTR Buffer, USHORT BufferSize)	
Prerequisite	Successful ***Check for KEYLOK®***
ExeDir	Directory name on the dongle
ExeFile	File name of the executable on the dongle
UserPIN	Security PIN which allows access to execute code on dongle
Buffer	Data buffer for passing data between the application and the dongle
BufferSize	Size of data buffer in bytes

Please see the ***KEYLOK®***Code On Dongle Manual for details on the type of code you can transfer to the dongle.

Get Global Unique Hardware ID

Each Fortress dongle is programmed with a 64-bit globally unique hardware id. This id can be used in addition to the Serial number to identify a dongle.

KGETGUSN (CHAR * pArray)	
Prerequisite	Successful ***Check for KEYLOK®***
Task	**GETLONGSN = 89**
pArray	Char pointer to the array that will contain the globally unique hardware id.

Anti-Debugging Utility

The **KEYBD(OFF)** function is used to activate the anti-debugging utility **PPMON.EXE**. *PPMON.EXE* is a utility that prevents a debugger from being attached to your executing program. Although the function name implies that the keyboard is turned off, actually the anti-debugging utility PPMON is launched. This adds much greater security to your protected program.

KEYBD(0) - This call launches the anti-debugging utility.

This call need only be performed one time from within the protected application. In general this call can be placed anywhere in your application, however there are instances in which this call must be placed outside of the InitApplication() function.

Protecting with S-LOK™

KEYLOK®'s S-LOK™ product protects executable programs from piracy without having to modify the original source code. You are able to "Shrink-Wrap" a protective shell around your existing executable programs that prevents them from operating unless a proper security device uniquely built for your company is present.

S-LOK™ is a combination of two mature technologies. It combines KEYLOK® II dongle based software piracy prevention system with Blink, Inc.'s Shrinker. Shrinker is designed to compress executable programs to save space on distribution media.

The S-LOK™ security system product is designed for use with IBM personal computers (and compatibles) running under the DOS or WINDOWS 3.x/9x/NT/2000/XP/Vista operating systems. The electronic security device attaches to any parallel printer port on the computer (the software automatically searches each port until the device is located). Robust protection algorithms defend against determined pirates attempting to bypass security. The hardware security device is also protected against reverse engineering and comes with programmable memory.

The S-LOK™ security system protects your developed software applications from piracy, thereby increasing your revenues associated with software sales. The security is transparent to your end-user once the hardware device is installed on the computer's parallel port. Unlimited backup copies of the program can be made for your client's protection, with the knowledge that you have complete control over the number of copies of the application actually able to be used. Your clients can install the software on multiple machines (e.g. at the office and at home) without having to go through difficult and timely install/uninstall operations. Your clients can easily RESTORE or reinstall copies of your software following catastrophic events such as hard disk failures. These advantages provide your clients with the features they desire and deserve while preserving your financial interests.

At the time of manufacturing each device is programmed with a unique device serial number, thus providing you the capability of identifying the specific device (and thus end-user) should you desire this level of control.

We have also implemented sophisticated algorithms that allow you to use the client's system clock as an economical means of controlling leased software. The most recent system clock date and time are stored internally within the S-LOK™ device memory. Any attempt by the end-user to set back his date and/or time generates appropriate error codes to prevent your application from running. Our Remote Update utilities can be used with S-LOK™ dongles to update lease expiration dates on S-LOK™ dongles.

The S-LOK™ price to feature ratio is unparalleled in the industry.

How to Protect an Executable Program

S-LOK™ is extremely easy to use. In order to protect a program simply follow these steps (see Fig. 3):

1) Attach the S-LOK™ security device to the parallel port.
2) Run the *INSTALL.EXE* utility to install the necessary drivers and associated files. Administrator rights are required when installing on Windows NT/2000/XP systems.
3) Run *SHRLOK32.EXE* (client.h and SHRINK32.DLL must be located in the same directory as SHRLOK32.EXE).
4) Click the Input File button and select the application to be protected.
5) Click the Output File button and select the name of the protected application. We recommend making a backup copy of the unprotected application if the same file name will be used for the protected application.
6) Click the Protect button.

The Output file produced will require a dongle to be attached to run.

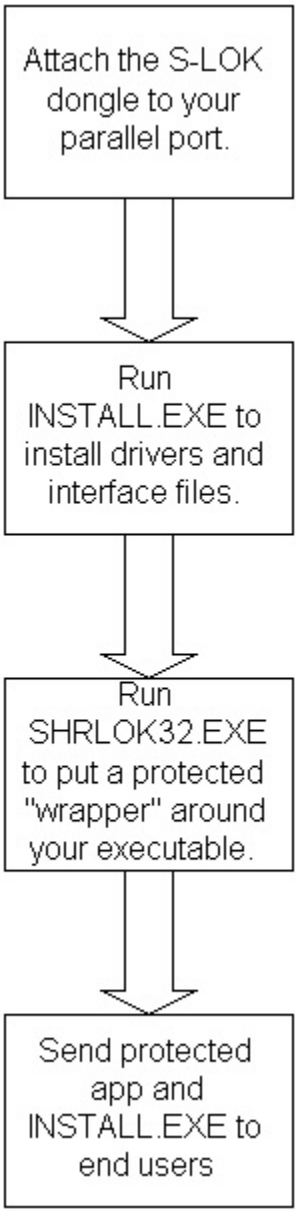

Figure 3: How to Protect an Executable with S-LOK™

Activating S-LOK™ Optional Features

Preprogramming the security device memory using the CONFIG utility described below activates the various optional features. In a couple instances (see below), this information must be supplemented by embedding an appropriate constant into the *client.h* file used during the protection process:

- Serial Number Constraint – A constant named *REQUIREDSN* can be found near the end of the *client.h* file. Constraint to permit a protected application to work only with a security device of a predefined serial number.

- Product License Constraint – A constant named *TargetLicense* can be found near the end of the *client.h* file. Constraint to permit protected application to only work with a security device containing this license value.

Configuring Device for Runtime

A utility named *CONFIG.EXE* is provided to program the security device with various options you choose to activate to control how the security system operates during run-time. The following is a summary of the optional features that can be set using this utility:

Acquiring Attached Device Serial Number

This feature is used to acquire the serial number burned into the security device at manufacturing time. Every device has its own unique serial number. This number is frequently used to track to whom a device has been sent. It is also useful if you wish to configure a protected copy of your application to only work with a predefined device serial number. The application reports the serial number using both the hexadecimal and decimal numbering systems. The range of valid serial numbers is from 0 to 65535 (decimal), or 0x0000 to 0xFFFF (hexadecimal). The serial number is provided in hexadecimal format because this is the format that must be used in the *client.h* file to constrain your protected application to work only with the specified security device.

Setting or Acquiring the Expiration Date

This feature allows you to program the security device with an expiration date. The program will stop functioning once this date has been reached. You can also use the utility to examine a device to determine what expiration date it is currently programmed for.

Setting or Acquiring Individual Product License

This feature allows you to program the security device with a predefined number that is compared to a number built into the application at protection time. There are forty-eight reserved memory locations within the device that can be used for storing product license information. The utility also allows you to examine a security device to determine which product it has been programmed to activate.

Setting or Acquiring Counters

This feature allows you to program the security device with a pre-established count of how many times the protected application is allowed to start before it becomes non-operational. The utility can also be used to examine a security device to determine how many counts remain.

Installation on End-users Computers

The ***INSTALL.EXE*** utility automatically copies the required files to the recommended locations on the target system. Please see the ***Distributing Your Application*** section of this manual for more detailed instructions.

Using S-LOK™ Devices with the KEYLOK® API

The access codes listed in the *client.h* file in the S-LOK™ directory can be used to provide device access using all of the same KEYLOK® API calls as a standard KEYLOK® device. Use the API calls described above in the ***KEYLOK® API Reference*** section of this manual.

One reason for accessing an S-LOK™ device in this way would be to provide higher security for your application. Using an S-LOK™ device, you can put a protected wrapper around your executable, and also embed API calls in your application to ensure that it can't be run without a dongle being present. Using this approach makes it more difficult to crack your code because the executable file can't easily be disassembled.

You might also use API calls to an S-LOK™ device if you distribute two different applications, one protected using S-LOK™ and the other protected using embedded API calls, and you want to be able to use the same devices for either application.

Device Access Over a Network

Overview

General Information

The KEYLOK® security system can be used to share a single device among various applications running on a network. The advantage of this technique is that multiple copies of a protected application running on different computers can be controlled through use of a single security device.

A server application is provided that communicates with the KEYLOK® device installed on the machine on which the server application is running (see Fig. 4). This technique should work on any network operating system running the TCP/IP protocol, which is currently the most widely supported network protocol. A protected program running on any node on the network can then access the device via the network, up to the established maximum simultaneous user count. (NOTE: See later section "Controlling Authorized User Count" for more details.)

Networking Components

The following components are required to implement the network security system. Each of these components is installed and configured automatically as appropriate by the KEYLOK® install utility.

Server Application: This is the program that each copy of your protected application communicates with in order to acquire remote contact with the security device. This program acts as the interface between your protected application and the device driver (except for Fortress), which actually communicates with the security device. The name of the server application is *KLSERVER.EXE*. The server runs as a Windows service. Up to three copies of this server can be run on a network. Note that the total number of sessions that can be run simultaneously on a network equals the sum of the allowed sessions programmed into each of the server dongles running on the network.

NOTE: The server application (KLSERVER.EXE) can be installed and run on any machine, it does not require a server OS.

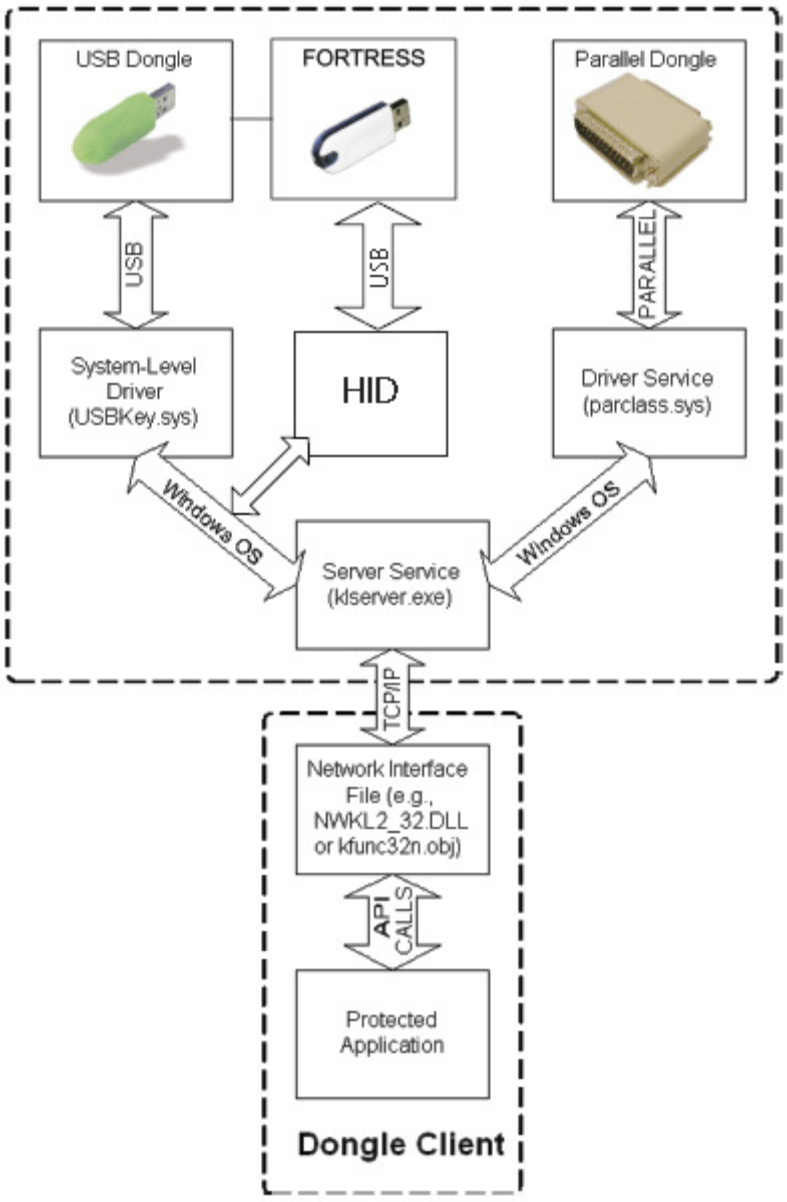

Figure 4: Dongle Access over a TCP/IP network

You can optionally set a fixed timeout (in minutes) for dongle client sessions connected to a server by including a file named *KLTCPIP.DAT* in the same directory as *KLSERVER.EXE*. This is intended to recover sessions that have inadvertently been lost in such a way that the application and/or OS did not send a signal to the server application to close the session. Once a session 'times-out' the session is then freed for access from any other client. The file should be a standard
ASCII text file containing the number of minutes that can pass with no activity from the client before a session times out. The default timeout period for a session with no dongle activity is 9 hours (540 minutes). This is intended to correspond to an individual's full work shift of 8 hours including an hour off for lunch.

Device Driver: This is the program that actually communicates directly with the security device. The server application communicates with this driver. The name of the device driver file for parallel port devices is *PARCLASS.SYS* for Windows NT/2000/XP/Vista and *PARCLASS.VXD* for Windows 95/98/ME. For USB devices other than Fortress, the device driver is *USBKey.sys*.

KEYLOK® security device: The security device required for network operation is identical to that used for standalone operation except that for more than one simultaneous user you must use special multi-session versions of our devices. The device must be physically installed on the network node that is running the server application *KLSERVER.EXE*. The device may be optionally programmed with user limits. If more than one key designed to respond to the same set of company unique codes is installed on the same network they must be physically installed on two separate platforms.

Client Application: This is the protected program that must be capable of communicating with the security device over the network in order to confirm the presence of a proper device, and to be able to read and/or write to the security device memory. The protected application can also be run on the same computer platform as the server application.

Clients can be forced to use a specific server by a *TCPIPSVR.DAT* file in the \Windows\System32 directory on Windows NT/2000/XP systems (\Windows\System on Windows 9x/ME systems). The *TCPIPSVR.DAT* file is an ordinary text file that contains the network name or IP address of the server to which you want the client to connect. Note that a blank *TCPIPSVR.DAT* file is created by default during a Client installation; if a server name or IP address is specified during the installation, it will be written into the *TCPIPSVR.DAT* file.

Network: The computers on which the protected application and server application is running must be physically connected via a network with the TCP/IP protocol supported on each platform. We strongly recommend that the server and all clients be in the same subnet; in some cases, it may be possible to access dongles across subnets.

Network Utilities

We provide three utilities to assist in implementing and monitoring networked dongles. Note that all of these utilities are company specific (i.e. the 'DEMO' device version will not work with company unique devices, and vice versa). These utilities are as follows:

NetDemo is a network-enabled version of our demonstration program. NetDemo illustrates nearly all of the capabilities of KEYLOK® devices, and may be used to check device serial numbers, check or set the contents of device memory (one 2-byte location at a time), check or set lease expiration dates, or, on multi-user network devices, to check or set the maximum number of simultaneous users (up to the limit of the device hardware). **NOTE:** This utility should ***not*** be sent to end-users.

NetKeyMonitor is a utility that can be run on the dongle server or client platform. The utility shows all active dongle servers detected on the network and reports the maximum number of allowed sessions as programmed into each corresponding security device, as well as the current number of active sessions on each dongle server (for your company-unique devices). Its only function is to check for the presence of network dongles, so you can safely send it to your end-users as a diagnostic tool.

VerifyNetworkKey is a utility that allows you to make sure that a client on a network can communicate with a dongle mounted on a remote server. Its only function is to check for the presence of a network dongle, so you can safely send it to your end-users as a diagnostic tool.

Network Protected Program Preparation

The primary difference between a protected program designed to look for the device locally and the network version is the interface file with which the application is linked, as follows:

- If a 32-bit application is linked with *KFUNC32_xx.OBJ*, checks for the device will be made only on the local machine. If the application is linked with the network version of the object file, *KFUNC32N_xx.OBJ*, then checks will be made both locally and on the network. Successful linking of 32-bit applications with *KFUNC32N_xx.OBJ* also requires the WIN32 SDK supplied libraries of *NETAPI32.LIB and WSOCK32.LIB*.

- Similarly, if the application normally uses a DLL for device communications, then the non-network DLL must be replaced with the network version of the DLL. For 32-bit applications, replace *KL2DLL32.DLL* (for local device checks) with *NWKL2_32.DLL* (for network checks).

The network interface files (NWKL2_32.DLL or KFUNC32N.OBJ) first search the local machine for a dongle. Then, if a local device is not found, they search the network. The specific search order is:

1) Search for local USB device
2) Search for local parallel device
3) Search for TCP/IP network key for remote dongle over the network

Note that the non-network search is as follows:

1) Search for local USB device
2) Search for local parallel device

Controlling Authorized User Count

There are two methods that can be used to control the maximum number of simultaneous users of your application on the network, as follows:

- Device Limited (Absolute Maximum User Count): Multi-user KEYLOK® devices are pre-programmed with an absolute maximum number of simultaneous users that can be supported. This limit is physically set in the device and cannot be changed. Single-user devices are accessible over a network, but will only support one user at a time.

- Programming Limited (Maximum Authorized Users): The KEYLOK® device can be programmed with the maximum number of sessions you wish to authorize with the device (API SETMAXUSERS). This number can be set to any value from 1 up to the device-limited maximum number pre-programmed into the dongle. NOTE: Fortress

dongles have a maximum allowed session limit of 127 and do not utilize the conversion table referenced above. For unlimited network dongles, the number of session counts is limited to 65,535 and are set as follows:

101 150
102 200
103 250
104 300
105 350
106 400
107 450
108 500
109 750
110 1000
111 2000
112 3000
113 4000
114 5000
115 7500
116 10000
117 15000
118 20000
119 25000
120 30000
121 35000
122 40000
123 45000
124 50000
125 55000
126 60000
127 65535

The maximum number of sessions of a network-enabled application that can be run simultaneously is equal to the total number of sessions programmed into all devices accessible on the network.

When using a 32-bit application linked with the network version (i.e. KFUNC32N.OBJ) of the security system each active session that communicates with the device is counted against the user limit. When an application linked with the standalone version (i.e. KFUNC32.OBJ) of the security system is used, then sessions of that application run on any machine including the one running the server application will not be counted against the session limit. There is no limit on the number of simultaneous sessions that can be run on a single machine when linking with *KFUNC32.OBJ*.

Installing the Server Application

The server application *KLSERVER.EXE* must be installed and started on the platform on which the KEYLOK® security device is attached. This is accomplished by running the ***INSTALL*** program with the Server checkbox checked or from the command line with the /NP option (for parallel devices) or the /NB option (for USB devices) or /NF option (for Fortress devices). Please review the detailed instructions for running *INSTALL.EXE* in the section of this manual entitled ***Distributing Your Application***.

Installing the Client Drivers

The client software must be installed in order to allow the protected application to access a server-mounted dongle via the network. This is accomplished by running the ***INSTALL*** program with the Client checkbox checked or from the command line with the /C option. Please review the detailed instructions for running *INSTALL.EXE* in the section of this manual entitled ***Distributing Your Application***.

Serial Port Devices

The serial port dongle uses the same API functions and call structures for accessing dongle memory and authenticating the presence of a unique dongle as documented in the ***KEYLOK® API Reference*** section of this manual. However, the serial port dongle does not require the use of any external device driver to communicate with the dongle, so that the API functions are passed directly to the dongle via RS232 rather than via a DLL or object file (see Fig. 5). Serial port dongle communication is based on the use of standard RS232 protocols. Networking of serial port dongles is not supported.

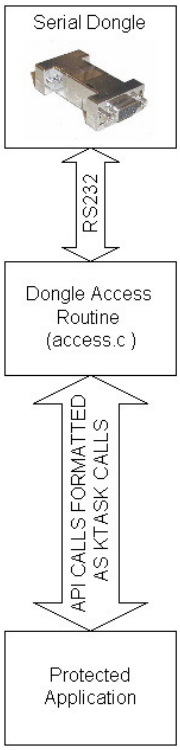

Figure 5: Serial Dongle Access

Device Select Sequence

It is recommended that the provided sample serial port source code be used to gain an understanding of the dongle communication.

Dongles are available in either of two configurations. One configuration allows dongles to be chained together. This configuration requires that the serial number of the target dongle be included in the selection sequence in order to assure that only the target dongle responds. The 'default' configuration does not use the serial number in the selection sequence and is limited to communication with only that dongle which is closest to the physical PC serial port connection. The discussion that follows does not include the serial number as part of the select sequence.

The dongle is designed to be transparent to communication not directed to the dongle. As such, it is necessary that there be a recognized command sequence to notify the dongle that a command is about to be sent. This sequence consists of four characters: FS (0x1c), SUB (0x1a). FS (0x1c) and SUB (0x1a). This command sequence tells the dongle that the data which follow are addressed to the dongle. The purpose of SUB characters is to notify devices downstream from the dongle that the prior character is to be ignored.

Once the dongle has been selected, it then switches the transmit and receive data lines such that data transmitted to the dongle is blocked from downstream devices, and data from downstream devices is blocked until the dongle communications (i.e. the current command/task) have been completed.

Command Sequence

The second portion of the select sequence is used to identify the desired API task. The character (8-bit byte) contains the code for the desired API task to be performed, as documented in the ***KEYLOK® API Reference*** section of this manual.

The remainder of the command sequence is made up of an eight-character sequence. The first two characters are random numbers used by the host to encode the arguments sent to the dongle so that the dongle can use this information to decode the arguments. The other 6 characters contain the encoded 3 (16-bit word) arguments associated with the desired task. Each of the three arguments is transmitted in low-byte, high-byte sequence. Some tasks require no arguments, whereas others require fewer than 3 arguments. Unused arguments should be transmitted as random numbers in order to introduce as much randomness into the communication sequence as possible as a means of further increasing security. The ***KEYLOK® API Reference*** delineates the use of task arguments.

Note that the *access.c* routine provided with your company-unique codes performs all of the necessary encoding and decoding, allowing you to access the device using simple *KTASK* calls rather than having to manually encode and decode all the information passed to and from the dongle.

The character sequence is encoded utilizing a company-unique sequence. The encryption key required for the dongle to decode the command and its associated arguments is embedded in supplemental characters added to the command sequence. This encoding is accomplished using the company-unique codes and *access.c* module supplied with the serial device software.

Dongle Response

The dongle responds to each standard API command by returning two 16-bit words to the calling application. Interpretation of these return arguments is as defined in the ***API Reference***. The supplemental functions described below are used to acquire and decode these two arguments.

The returned arguments are encoded using an encryption scheme based upon the same encryption key information embedded within the outgoing arguments to the dongle so that no additional information is required from the dongle for the calling application to decode the return arguments. The specifics of the decoding scheme are contained in the serial port sample code at the end of the *KTASK* subroutine.

Dongle Programmable Memory

The memory within the dongle is addressable as 2-byte words. The lowest valid memory address is 'zero' and the highest addressable memory address is 55, thus providing 112 bytes of programmable *EEPROM* memory. An attempt to read an *EEPROM* memory address outside of the defined range will return random numbers. A delay of a minimum of 75 milliseconds must occur between device reset and any attempt to write to the device memory. Each individual memory location is capable of being written to a minimum of 1,000,000 times with a typical count of 10,000,000 allowable writes per address.

Supplemental KEYLOK® II Functions:

Block Memory Read:

A special function with task code of decimal 21 is used to activate a block memory read from the dongle.

Argument 2 = number of words to be read
Argument 3 = starting address to begin reading from
Argument 4 = undefined

The words are returned in low byte/high byte sequence. There are no 'normal' *ReturnValue1* or *ReturnValue2* arguments returned by this function. The addition of this function significantly increases the speed at which data can be read from the dongle. A special decoding scheme is required to restore the data to usable form.

Block Memory Write:

An additional special function with task code of decimal 22 is used to activate a block memory write to the dongle.

Argument 2 = number of words to be written
Argument 3 = starting address to begin written
Argument 4 = undefined

The words are sent in high byte/low byte sequence. There are no 'normal' *ReturnValue1* or *ReturnValue2* arguments returned by this function.

Communications Parameters

KEYLOK® Serial devices are designed to communicate at 19,200 BPS using 8 data bits, 1 stop bit, and no parity (19200,8,N,1).

Event Timing

The following measurements represent typical times expected to accomplish various API calls using a data exchange baud rate of 19,200 BPS:

Check for company unique Dongle:	75 milliseconds
Serial Number retrieval:	30 milliseconds
Memory read:	30 milliseconds
Memory write:	40 milliseconds
Block read of all memory:	100 milliseconds
Block write of all memory:	2.25 seconds

Transmission Error Level

The calculated baud rate error is +0.16%. This is the difference between the target of 19,200 and the actual transmitting baud rate of 19,231 based upon the nominal oscillator frequency of the CPU and the internal baud rate generator. The total error is based upon the sum of this error plus any variation in the CPU oscillator speed from the nominal level used to calculate the expected baud rate. The resonator used to create the CPU oscillator speed is rated accurate to within 0.5%. Therefore, the total deviation from the expected baud rate is + 0.66% / - 0.34%.

Transmission Signal Level

Data is transmitted from the dongle at a level of a minimum of +/-5 Volts in order to exceed the levels required to achieve compliance with RS232 standards.

Remote Update

KEYLOK® provides two sets of **Remote Update** utilities that will allow you to securely read, write, or modify any of the programmable features of KEYLOK® devices remotely. The utilities both use exchanges of codes between the developer and the end-user; one set of utilities is optimized for exchanges in real time (e.g., over the telephone), while the other is set up for e-mail exchanges.

Telephone Remote Update

The **Telephone Remote Update** utility consists of two routines, one for the software developer and one for the end-user. When a remote update is required, the developer sends the end-user a copy of the end-user **Telephone Remote Update** module (RemoteUpdateUser.exe) and tells the end-user to telephone the developer to conduct the update session. Although not required, this utility is provided to reduce the effort on the part of the developer. However, developers may choose to build the remote update sequence into their distributed applications.

A typical telephone remote update session is as follows (see Fig. 6):

1) The end-user runs *RemoteUpdateUser.exe* on a machine that has the dongle to be updated attached (and the dongle drivers installed) and clicks "Update Security Device". The program will respond with a set of five numeric codes. The end-user telephones the developer, reads the codes to the developer, and clicks OK. The program then displays a screen for input of four codes to be provided by the developer.

2) The developer runs the developer module (RemoteUpdateDev.exe) and clicks "Remote Update Software Developer". The developer then enters the five codes given by the end-user and clicks OK.

3) The developer's program responds with a screen that allows the developer to select the remote update action to be done. This action may be to alter the dongles memory contents, or simply to query the dongles memory contents. The developer selects the desired operation and clicks OK. If additional information is needed for the update (e.g., if a lease expiration date is extended or a memory location is accessed), an additional window will appear for input of the necessary data. When all data input is complete, the developer clicks OK and the program responds with a set of four numeric codes. The developer then reads the four codes to the end-user. The developer's module will display a screen for entry of three codes to be provided by the end-user.

   NOTE: Only one action at a time can be done during a single exchange of codes. If more than one action is necessary, the entire process must be repeated until all desired remote update actions have been completed.

4) The end-user enters the four codes provided by the developer and clicks OK. The program will respond with three numeric codes.

5) The end-user reads the three numeric codes to the developer, who enters them and clicks OK. The program will respond with a message indicating whether or not the remote update was accomplished successfully and display the result of the update (i.e., new expiration date, new counter value, new memory contents, etc.).

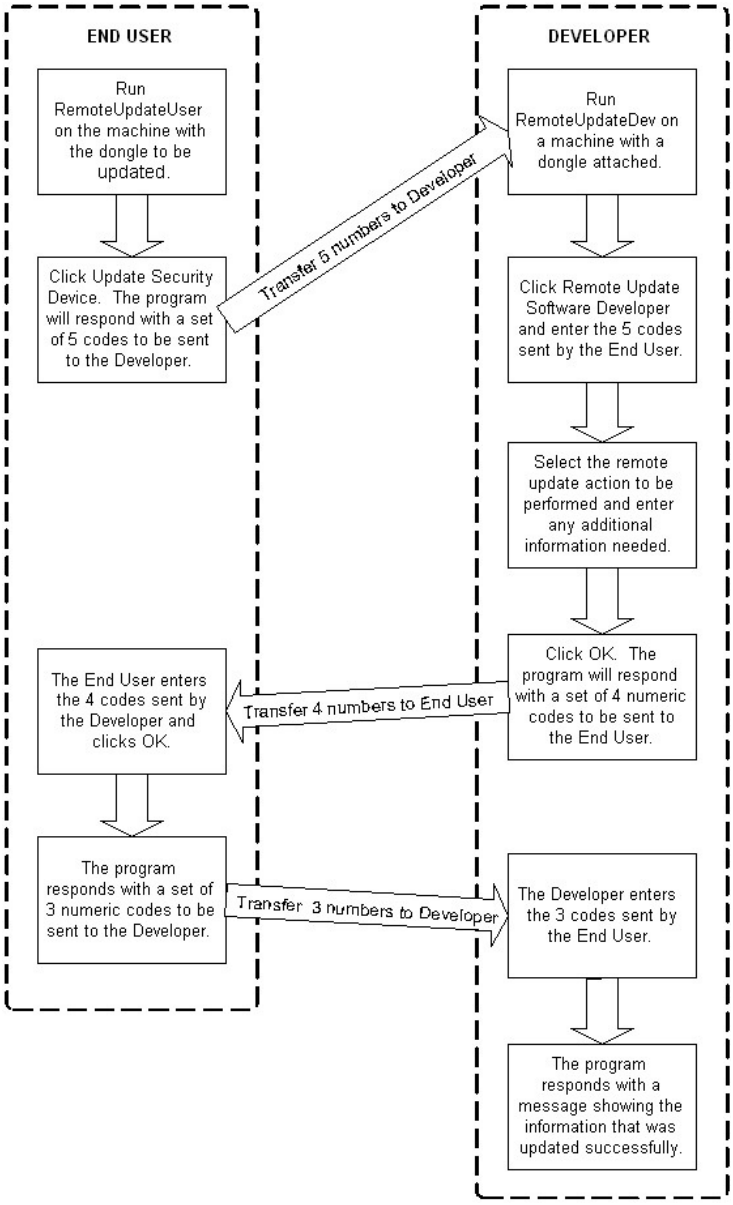

Figure 6: Telephone Remote Update Process

E-Mail Remote Update

The **E-Mail Remote Update** utility consists of two modules:

1) the software developer module (RemoteUpdateEmailDev.exe)
2) the end-user module (RemoteUpdateEmailUser.exe)

When a remote update is required, the developer runs the developer module (RemoteUpdateEmailDev.exe) to generate an update file, AUTHORIZE.DAT, and sends the end-user a copy of the end-user E-Mail Remote Update module (RemoteUpdateEmailUser.exe) and a copy of the AUTHORIZE.DAT file. The end-user runs the module and processes the received file against the device to update it.

There are various options for performing this task:

1) the end user can generate a REQUEST.DAT file by running the RemoteUpdateEmailUserRequest.exe utility and send this file to developer as an 'aid' in creating the AUTHORIZE.DAT file. In this case each requested action is displayed to the developer for authorization, and the developer is also allowed to add additional update tasks.
2) the developer utility (RemoteUpdateEmailDev.exe) can be run in GUI mode as described below,
3) the developer utility can be data driven by a file named 'DeveloperRemoteUpdate.dat', in which case all of the desired update actions are written into a text file along with the target serial number(s) of the dongle(s) to be updated. The developer utility simply converts the input .dat file into an encrypted AUTHORIZE.DAT file. The utility can be run in quiet mode (i.e. no displayed messages [/q command line parameter activates quiet mode]) to further automate this task with a minimal of effort on the part of the developer.

A typical e-mail remote update session is as follows:

1) The developer runs the developer module *RemoteUpdateEmailDev.exe*. When the program opens, the developer clicks ***Check for KEYLOK®***, clicks OK, clicks Memory Read>Read Authorization, Memory Write>Write Authorization, and Email Remote Update>Software Developer. The developer is prompted to input the target dongle serial number(s). Up to 60 dongles can be updated from a single AUTHORIZE.DAT file.

2) The developer module will then display a list of possible remote update tasks. The developer clicks the radio button next to the selected task, supplies any additional information needed (e.g., memory address and value, new expiration date, etc.), and repeats this until all desired update operations have been selected. The developer than clicks "No More Requests", and the program responds with a message giving the serial numbers and update numbers for this update session. The update information is written to an encrypted *AUTHORIZE.DAT* file.

3) The developer sends the end-user a copy of the end-user **E-Mail Remote Update** module *RemoteUpdateEmailUser.exe* and a copy of the *AUTHORIZE.DAT* file.

4) The end-user puts the *AUTHORIZE.DAT* file and the end-user module into the same directory and runs the end-user module. When the program displays its window, the end-user clicks on the "Program Security Key" button. The program responds with a message saying that the security key has been updated and also displays the serial number of the dongle that has been updated.

Rather than using RemoteUpdateEmailDev.exe, you can build your own end-user utility and use the API function DOREMOTEUPDATE (Value = 21) which will update the dongle with the AUTHORIZE.DAT file located in the same directory as your utility. You will have to check for the dongle and get read and write authorization before processing this function.

CUSTDATA.DAT and AUTHORIZE.DAT

The *CUSTDATA.DAT* file is used to keep track of how many update sessions have been conducted for a given dongle. For this reason, **it is critical that the CUSTDATA.DAT file be backed up frequently and regularly**. A new *CUSTDATA.DAT* file is created the first time that the developer module is run, and updated by each run thereafter.

A new *AUTHORIZE.DAT* file is generated for each update session and contains the actual encrypted update instructions used by the end-user module to update the dongle.

From a high level, the way that *CUSTDATA.DAT* and *AUTHORIZE.DAT* are used is as follows:

Each update to a customer will increase the "Update Number" within the *CUSTDATA.DAT* file. The contents of the file are dongle serial numbers and update numbers, for example:

Serial #	Update #
1234	7
1235	0
1236	3
1237	1

When an *AUTHORIZE.DAT* file is used to update a dongle, it first checks to see if the dongle serial number matches, and then compares the *AUTHORIZE.DAT* update number with the number of the last update stored on the dongle (in encrypted form in a location not accessible to the user). This prevents the same update or older updates from being run on the same dongle twice. For example:

Before the first update:

	Serial Number	Update Number
AUTHORIZE.DAT	1234	1
Dongle	1234	0

This will update the dongle and store the last update number 1 on the dongle.

After the first update, using the same *AUTHORIZE.DAT*:

	Serial Number	Update Number
AUTHORIZE.DAT	1234	1
Dongle	1234	1

This will not update the dongle because the last update number stored on the dongle is the same as the number in *AUTHORIZE.DAT*.

To address situations in which the *CUSTDATA.DAT* file is lost or corrupted, it is possible to re-generate/re-synchronize the files. For example, if the *CUSTDATA.DAT* file in the above example was lost, it would be necessary to create an *AUTHORIZE.DAT* with an update number

of 2 or more. This can be done by resetting the version counter (click Email Remote Update>Erase Last File Version) and repeatedly generating new *AUTHORIZE.DAT* files until the required sequence number is reached. Alternatively you can use the utility named CustDataFileMaintenance.exe to create and update CUSTDATA.DAT

NOTE: File regeneration only applies if the *CUSTDATA.DAT* is lost. This file should be backed up often. Regeneration sets all update numbers to zero. It is then necessary to edit the file using the CustDataFileMaintenance.exe utility for each dongle that has been sent remote update AUTHORIZE.dat files.

DEVELOPERREMOTEUPDATE.DAT

The creation of AUTHORIZE.DAT can be automated by providing RemoteUpdateEmailDev.exe with a text file containing the tasks to be processed on the remote dongles.

DeveloperRemoteUpdate.dat must be in the following format:

```
Count: 2
SerialNumbers: 1234, 5667
Task: 0
Address: 1
Value: 5
Task: 1
Address: 15
Value: 3
Task: 2
Address: 9
Value: 1
Task: 9
Address: 4000
Value: 12/28/2018
Task: 10
Address: 45
Value: 5, 99, 33, 56, 44556, 23231, 1, 10, 88, 90
Value: 65, 47
```

Count = the number of serials numbers to be processed. The maximum number in one DeveloperRemoteUpdate.dat is 60.
SerialNumbers = the serial numbers to be updated, separated by a comma and a space
Task = Task ID (see table below). If using Task 10, it must be the last task
Address = memory address to be updated (see table below). For Task 10, this is the first address in the block to be updated. For Fortress, address must be less than 4096. If you need to update higher memory addresses, use Block update (Task 10) with a starting address of 4096 or lower.
Value = value to be used for the task. Maximum of 10 values per line for Task 10

DeveloperRemoteUpdate.dat must be less than 170,000 characters. If using Task 10, the number of values must be less than 28,000.

Task	Task ID	Address	Value
Add to memory contents	0	Actual address	Positive integer
Extend expiration date "n" months	1	Any value (ignored)	Number of months
Bitwise "OR" of memory contents	2	Actual address	Any value
Replace memory contents	3	Actual address	Any value
Set maximum number of sessions	5	Any value (ignored)	1-127 see page 57
Set expiration date	9	Any value (ignored)	MM/DD/YYYY or 0 to remove expiration date
Block Memory Update	10	Actual address (Fortress Only)	Any value

The maximum number of tasks which can be included in DeveloperRemoteUpdate.dat is 60.

Distributing Your Application

Using the KEYLOK® Install Utility

The easiest and recommended method to install the required KEYLOK® driver and interface files onto your end-users' computers is to use the KEYLOK® Install Utility, *Install.exe*. This utility may be run from its own Graphical User Interface (GUI) or invoked, using command-line switches to select installation parameters, from your existing installation program such as InstallShield or WISE.

The installer determines which operating system is running, and then copies the appropriate individual driver and/or KEYLOK® files to the proper directories.

NOTE: Modification of the NT/2000/XP/Vista system registry requires that a user with Administrator permissions runs the utility.

USB Installations (except for Fortress)

It is very important that the USB dongle NOT be attached to the computer until after the INSTALL utility has been run. To install the USB drivers, please do the following:

1) Run *Install.exe*. Select the "USB" checkbox.

2) When the install displays a message saying that it is finished, attach the dongle. When you do so, the Windows New Hardware Found wizard should start and ask to install the drivers for a USB Security Protection device. Accept all the defaults and let the wizard finish.

If the dongle was inadvertently attached before running the driver install, it will be necessary to uninstall and reinstall the drivers. To do so, please do the following:

1) Remove all dongles from the system.

2) Run *Install.exe*. When the opening screen comes up, make sure that the "Uninstall" checkbox is selected. Click OK.

3) When the uninstall displays a message that it is finished, reboot the system.

4) Make sure that your USB dongle is NOT attached.

5) Re-run *Install.exe*. Select the "USB" checkbox.

6) When the install displays a message saying that it is finished, attach the dongle. When you do so, the Windows New Hardware Found wizard should start and ask to install the drivers for a USB Security Protection device. Accept all the defaults and let the wizard finish.

Parallel Port Installations

To install the parallel port drivers, please do the following:

1) Attach your parallel port dongle.

2) Run *Install.exe*. Select "Parallel". Click OK. The installer will display a message box when the installation is finished.

If the dongle was not attached when the install utility was executed the driver can be started manually using the following command at a DOS prompt (in Windows NT, 2000, or XP):

net start parclass

This command can be added to your application to ensure the driver is started before checking for the dongle.

Install Utility Command Line Arguments

Install.exe Utility	
Delimiters	Valid command line delimiters are **'/'** or '****' or **'-'** A space is required before each delimiter and options cannot be combined (i.e. '/QN' is illegal, but '/Q /N' is legal).
/A	Install both parallel port and USB driver files **Note:** This option will result in a parclass error 0x37 if a parallel port dongle is not attached during the installation process.
/B	Install USB driver files only
/C	Install network Client files and USB drivers Allows local USB dongle or remote USB or parallel port dongle
/F	Install Fortress files only
/NP	Install parallel port TCP/IP server networking drivers to allow remote access to a parallel port dongle via a TCP/IP network
/NB	Install USB TCP/IP server networking drivers to allow remote access to a USB dongle via a TCP/IP network
/NF	Install Fortress TCP/IP server networking files to allow remote access to a Fortress dongle via a TCP/IP network.
/P	Install parallel port driver files only
/Q	Quiet mode install – displays only fatal errors
/S:<IP address>	Valid only for a network client install. Forces the client to attempt to connect to the dongle server at <IP address>. You may also specify the network name of the server instead of its IP address.
/U	Uninstall Removes all previously installed files

Custom INF Files

If you put a custom *USBKey.INF* file (e.g., an INF file with your company name in it) in the same directory as *INSTALL.EXE*, the custom file will be installed and used instead of the default file built into the install utility.

Manual Installation

In some cases, it may necessary or desirable to install the driver files manually rather than using our installer. The following tables indicate what needs to be done to install the drivers and interface files manually on a Windows NT/2000/XP/Vista/7 32-bit system. For other configurations, please contact Technical Support.

All Installations

The following files must always be installed, regardless of device type or network use.

File Name	Description	Location
KL2DLL32.DLL	32-bit KEYLOK® II DLL	\Windows\System32
PPMON.EXE*	Anti-debugger	\Windows\System32

USB

The following files must be installed where indicated in order for the Windows New Hardware Found wizard to operate properly to install and configure the USB driver.

File Name	Description	Location
USBKey.sys	USB device driver	\Windows\System32\Drivers
USBKey.INF	INF file	\Windows\INF

Parallel

The following file must be installed where indicated to provide parallel port support. The driver runs as a Windows service.

File Name	Description	Location
Parclass.sys	parallel port device driver	\Windows\System32\Drivers

The registry entry for the parallel port dongle is created by a Win32 call to *CreateService()*. This sets the **parclass.sys** service to start up automatically when the machine is started. The C call that is made is as follows:

CreateService(SchSCManager,	// SCManager database
KEY_NAME,	// name of service
KEY_NAME,	// name to display
SERVICE_ALL_ACCESS,	// desired access

* Only required if the developer has chosen to implement calls to the anti-debugging utility **(strongly recommended).**

SERVICE_KERNEL_DRIVER,	// service type
SERVICE_AUTO_START,	// start type
SERVICE_ERROR_IGNORE,	// error control type
driverName,	// service's binary
group,	// load ordering group
NULL,	// no tag identifier
depends,	// dependencies
NULL,	// LocalSystem account
NULL	// no password[1]
);	

Network Server

File Name	Description	Location
KLSERVER.EXE	Network server service	\Windows\System32
NWKL2_32.DLL	32-bit KEYLOK® II network DLL	\Windows\System32

The registry entry for the network server service is created by a Win32 call to *CreateService()*. This sets the *klserver.exe* service to start up automatically when the machine is started. The C call that is made is the same as is shown above for parallel port dongles, with the exception that the KEY_NAME and driverName values should be: xxxx and yyyyy.

The network server installation will also install the parallel or USB drivers (depending on which was selected) as shown above.

Network Client

The following file must be installed where indicated in order for the protected application (if using DLL instead of linkable object) to be able to communicate with a dongle server.

File Name	Description	Location
NWKL2_32.DLL	32-bit KEYLOK® II network DLL	\Windows\System32

[1] The actual values of KEY_NAME and driverName should be provided.

KEYLOK® Utility Programs

KEYLOK® supplies a number of utilities that you can use to program dongles, diagnose common driver or network problems, and explore the capabilities of KEYLOK® devices. All of these utilities except *INSTALL.EXE* are company specific (i.e. the 'DEMO' device version will not work with company unique devices, and vice versa). The utilities are:

WinDemo

WinDemo.exe is a pre-compiled version of our standard sample demonstration program. The source code for *WinDemo.exe* may be found in the ..\Samples\Windows\VisualStudio32 directory in our sample code. **WinDemo** illustrates nearly all of the capabilities of KEYLOK® devices, and may be used to check device serial numbers, check or set the contents of device memory (one 2-byte location at a time), check or set lease expiration dates, or, on multi-user network devices, to check or set the maximum number of simultaneous users (up to the limit of the device hardware).

VerifyKey

VerifyKey.exe is a utility that allows you to make sure that the device drivers and interface files have been installed correctly so that your application will be able to communicate with a dongle. Its only function is to check for the presence of the dongle, so you can safely send it to your end-users as a diagnostic tool to make sure that the dongle drivers are working correctly.

KLTool

KLTool.exe is a very versatile general purpose programming and demonstration tool. In addition to all of the capabilities of **WinDemo**, it also allows reading and writing text strings to and from device memory, resetting of lease expiration dates (to no date set, so that the dongle never expires), programming of multiple keys using stored profiles, and simulation of a telephone **Remote Update** session.

NetDemo

NetDemo.exe is the network-enabled equivalent of **WinDemo**. It has the same functionality as **WinDemo**, except that it is designed to work with a dongle mounted on a remote dongle server that is accessed via a network.

VerifyNetworkKey

VerifyNetworkKey.exe, the network-enabled equivalent of **VerifyKey**, is a utility that allows you to make sure that a client on a network can communicate with a dongle mounted on a remote server. Its only function is to check for the presence of a network dongle, so you can safely send it to your end-users as a diagnostic tool.

NetKeyMonitor

NetKeyMonitor is a utility that can be run on the dongle server or client platform. The utility shows all active servers detected on the network and reports the maximum number of allowed sessions as programmed into the security device, as well as the current number of active sessions (for your company-unique devices).

Install

INSTALL.EXE is the utility that installs the KEYLOK® drivers and interface files and makes all necessary registry entries as appropriate for the type of installation selected. The utility will automatically detect the version of Windows on which it is running and install the appropriate versions of the INF files, drivers, and interface files. **INSTALL.EXE** can be run either from its own GUI or from a command line. If run from a command line, it is possible to suppress all messages except fatal error messages.

CustDataFileMaintenance

CUSTDATAFILEMAINTENANCE.EXE is the utility that the developer uses to synchronize an authorize.dat file with custdata.dat. It is used to correct custdata.dat if it has gone out of sync with information stored in a target dongle. Lack of synchronization could be caused by 1) lost custdata.dat file, 2) experimentation with the remote update utilities by developers, 3) failure to apply previous remote update files by end-users, etc. Authorize.dat files that have an embedded remote update sequence number more than '3' greater than the last update to the dongle will not be processed, nor will they be processed if the authorize.dat file contains a sequence number less than or equal to that stored in the target dongle.

RemoteUpdateEmailDev

REMOTEUPDATEEMAILDEV.EXE is the utility that the developer uses to create authorize.dat files which are used to update the devices remotely.

RemoteUpdateEmailUser

REMOTEUPDATEEMAILUSER.EXE is the utility that processes the authorize.dat on the client machine to update the dongle remotely.

RemoteUpdateEmailUserRequest

REMOTEUPDATEEMAILUSERREQUEST.EXE is the utility the end user runs to create a request.dat which is emailed to the developer and processed by RemoteUpdateEmailDev. Use of this utility is purely optional, and any actions converted to authorize.dat file contents require confirmation by the developer.

Using KEYLOK® Under Open Source OS's[2]

We do not currently have a native LINUX install utility. To install on a Linux machine, go to the \SampleCode\Linux directory on the CD and copy the USB and/or parallel directories to some convenient directory on your Linux machine.

To implement KEYLOK® protection in your Linux code, simply copy the appropriate sections from the correct version (USB or parallel) of our Linux sample code into your application and link with our Linux static library, *libkfunc32.a*. We can also supply a shared object (.SO) libkfunc32.so which should be copied to /usr/lib directory.

Note that our Linux object also supports Java under Linux, using standard Java Native Interface (JNI) calls to *KFUNC* to access the dongle.

We support parallel port devices under Linux using the standard Linux system parallel port drivers. USB support is provided using the libusb library, and assumes use of the usbdevfs file system.

The KEYLOK® module *libkfunc32.a* is configured as a user-level driver,
i.e. it accesses I/O ports directly rather than through a kernel driver.
This has been done for simplicity, reliability and speed. The
user->kernel interface was deemed an all too easy hacking vulnerability.
As a result, root access is required. This has been resolved with the Keylok_install script which removes the root access requirement. Udev rules are loaded at start-up so the Linux machine needs to be restarted after the script has been run.

QNX

If you are using the QNX operating system, our standard Linux sample code should run, with the following addition:

unsigned long userDelay = x;

where x is some number. This will add a required delay to allow communication with the dongle.

Only parallel devices are currently supported with QNX.

[2] E.g., OS A, LINUX, FreeBSD, QNX, etc.

Appendices

A. File Descriptions

Certain files are necessary in order for a protected application to access KEYLOK® devices. These files are automatically installed in the correct directories as appropriate for the type of installation by the KEYLOK® installation utility. The files are described in the following table.

LIB files and DLLs	
KFUNC32MT.LIB KFUNC32MD.LIB	Library file to be linked into 32-bit applications to provide the communications interface to local (non-network) USB or parallel port devices. Linked with application
KFUNC32MTN.LIB KFUNC32MDN.LIB	Library file to be linked into 32-bit applications to provide the communications interface to a remote dongle via a network. Also provides support for locally mounted dongles. Successful linking of 32-bit applications with KFUNC32MTN.LIB or KFUNC32MDN.LIB also requires the WIN32 SDK supplied library of NETAPI32.LIB. Linked with application

KFUNC64MT.LIB KFUNC64MD.LIB	Library file to be linked into 64-bit applications to provide the communications interface to local (non-network) USB or parallel port devices. Linked with application
KFUNC64MTN.LIB KFUNC64MDN.LIB	Library file to be linked into 64-bit applications to provide the communications interface to a remote dongle via a network. Also provides support for locally mounted dongles. Successful linking of 64-bit applications with KFUNC64MDN.LIB or KFUNC64MTN.LIB also requires the WIN64 SDK supplied library of NETAPI64.LIB. Linked with application
KL2.OBJ	Object file to be linked into 16-bit applications to provide the communications interface to local (non-network) devices. Some versions of KFUNC32.OBJ require linking with KL2.OBJ for resolution of external calls. If this is necessary, both object files will be in the sample code directory for the specific development language

	and environment. We are in the process of phasing out the requirement for KL2.OBJ for 32-bit applications. KL2.OBJ provides backwards compatibility to the very old WIN32S environment, which is rarely (if ever) used today. Linked with application
KL2.DLL	16-bit DLL interface for locally mounted devices. \Windows\System32
KL2N.DLL	16-bit DLL interface for local or remote (network) devices. \Windows\System32
KL2DLL32.DLL	32-bit DLL interface for locally mounted devices. \Windows\System32
NWKL2_32.DLL	32-bit DLL interface for local or remote (network) devices. \Windows\System32
KL2DLL64.DLL	64-bit DLL interface for locally mounted devices. \Windows\System32
NWKL2_64.DLL	64-bit DLL interface for local or remote (network) devices. \Windows\System32

Networking	
KLSERVER.EXE	KEYLOK® TCP/IP server networking service that interfaces between your protected program and the device driver that actually communicates with the KEYLOK® device. 32-bit - \Windows\System32 64-bit - \Windows\SysWow64
KLTCPIP.DAT	An optional file used to set the maximum number of minutes a client session can be idle before being disconnected. \Windows\System32
TCPIPSVR.DAT	An optional file used to force a network client to attempt to attach to a specific dongle server. Contains the IP address of the desired server. \Windows\System32

Device Driver Files	
PARCLASS.VXD	KEYLOK® Windows 98 parallel port driver. Used to communicate with the KEYLOK®. \Windows\system
USBKey.sys	32-bit KEYLOK® USB device driver. \Windows\System32\Drivers
USBKey.INF	INF file that provides Windows with driver installation information for USB keys. \Windows\INF
USBkey64.sys	64-bit KEYLOK® USB device driver for Windows XP64 on AMD or compatible processors. \Windows\System32\Drivers
USBKey64.INF	INF file that provides Windows with driver installation information for USB keys. \Windows\INF
PPMON.DLL	VDD providing 16-bit interface to driver. This DLL allows non-NT applications (e.g. 16-bit DOS/Windows, 32-bit DOS extender, etc.) to communicate with the NT device driver. Normal 32-bit applications (i.e. not using a DOS extender) do not require the use of this file. \Windows\System32

Other Files	
PPMON.EXE	Anti-debugger routine for 16- or 32-bit systems. If a kernel-level debugger is detected, PPMON will terminate the application within 10 seconds of detection. \Windows\System32
PPMON64.EXE	Anti-debugger routine for 64-bit systems. If a kernel-level debugger is detected, PPMON will terminate the application within 10 seconds of detection. \Windows\System32

B. Protecting 64-bit Applications

KEYLOK® has device drivers that support USB devices on 64-bit Windows XP64 / Server 2003 / Windows 7 / Server 2008 / Server 2008 R2 running on AMD or AMD-compatible processors. No 64-bit support is available for parallel port devices.

In order to implement KEYLOK® protection in a 64-bit application, it is only necessary to link your application with the 64-bit version of the interface files (library files or DLLs as appropriate for your programming language and development environment). No changes are necessary to your source code or to the KEYLOK® API calls used.

The KEYLOK® driver installer, **INSTALL.EXE**, will automatically detect a 64-bit operating system and install the 64-bit interface files instead of the 32-bit default versions.

C. Protecting 16-bit Applications

KEYBD Function

In 16-bit environments, the functionality of **KEYBD** is slightly different from its 32-bit behavior. The **KEYBD** function is used by 16-bit applications to disable the keyboard prior to calls to **KFUNC**, to obtain keyboard status, and to re-enable the keyboard after calls to **KFUNC**.

Appropriate arguments are as follows:

0 - Turns off the keyboard for 16-bit applications, or launches the anti-debugging utility for 32-bit applications

1- Checks the current status of the keyboard
It is recommended that a call to **KEYBD** with an argument of '1' be conducted prior to checking return arguments for validity. The system will return the status of the keyboard so that you can determine if the keyboard has been re-enabled as part of a piracy attempt. If the keyboard is found enabled then you should immediately exit from your program. If the return argument ANDed (a bitwise AND operation) with '2' equals '2' the keyboard is properly disabled. The demo programs show how this is done.

2 - turns on the keyboard for 16-bit or 32-bit DOS extended applications. Several consecutive calls can be made to **KFUNC** by 16-bit applications prior to making this call to **KEYBD** to re-enable the keyboard.

Please note that 16-bit WINDOWS programs require inclusion of appropriate lines from the 'demo.def' into your module definition file.

Omission of the keyboard disabling calls, or insertion of debugging break points between security system calls can result in your system freezing during checks for the device.

Networking 16-bit Applications

NOTICE: If you have a 16-bit application that requires the ability to run multiple copies on the same Windows NT/2000/XP/Vista platform, then the application MUST be run in its own memory space if it uses *KL2N.DLL*. There are two ways to inform NT to run the program in a separate memory space, as follows: (1) When starting the application use **START—RUN—Filespec**, check the box beneath the path to the file, or (2) create a Shortcut to the file, and change the properties of the Shortcut to specify run in a separate memory space.

If a 16-bit application is linked with *KL2.OBJ*, checks for the device will be made only on the local machine. If the application is linked with the network version of the object file (i.e. KL2N.OBJ) then a check will first be made locally for the device. If it is not found, an attempt is made to find the device on the network. Applications linked for network access will only find a device on a local platform if a KEYLOK® driver (PARCLASS.VXD or .SYS) is running on the platform.

For 16-bit applications, replace *KL2DLL.DLL* (for local device checks) with *KL2N.DLL* (for network checks).

Terminate

This task is applicable when:

1) Running 16-bit applications in a Windows 95/98/ME/ NT/2000/XP/Vista environment, and/or

2) Using network security routines to access a single device from multiple computers. It notifies the security system to free up resources allocated to the application. There are three situations where this is required.

Windows 95/98/ME fails to properly release resources allocated to 16-bit applications at the time they terminate. Therefore the 'handle' to the device driver is not released. As you start and terminate your program, each closure will leave a handle to *PARCLASS.VXD* open. If the open handle count exceeds the number provided for by the operating system, subsequent copies of your program will fail to find the security device. The 'TERMINATE' call forces closure of the handle to the *PARCLASS.VXD* device driver to prevent this problem.

D. Protecting DOS Applications

32-bit DOS extended (not true 32-bit) applications

32-bit DOS extended (not true 32-bit) applications that run under Windows 9x/ME/NT/2000/XP/Vista must be linked in such a way that the KEYLOK® data resides within the first 64K of the data space and the KEYLOK® code resides within the first 64K of the code space. This can be confirmed by examining the link map. Applications that violate this constraint will receive 32-bit return values from the **KFUNC** call during the **CheckForKL** task of 0xfefefefe if the data is located above 64K, or 0xfefdfefe if the code is located above 64K.

E. Special Language Support

There are two approaches used to support languages that do not have provisions for linkable object modules.

For **Parallel Port Dongles with 16-bit applications running under MS DOS**, protection can be implemented using a **TSR** (Terminate and Stay Resident) security module that installs itself as a DOS interrupt handler. Communication with the security module is done through the processor registers (see sample programs for details).

For **Parallel Port and USB Dongles**, KEYLOK® can supply a security routine in .EXE form. Shelling out of the protected application to run the executable security program and then returning back into the protected program provides security device communications. Communication between the protected program and the security module is accomplished through an intermediate data file, which can be read and written by both programs.

F. Selecting a Specific Parallel Port

The following are the rules for creating a port search criterion for those rare instances in which it may be required.

Load the first argument with the task identifier (i.e. KLCHECK = 1). For all 16-bit applications and 32-bit applications using parallel port dongles and linked with the *KL2.OBJ* file for WIN32S support a port-searching director must be added to the task identifier.

Multiply the **Port Select Option** by 8192 and add it to the constant **KLCHECK** to create the argument value. This moves the port option bits to the high order bit positions in the argument.

The Port Select Option is defined as follows:

0 Search LPT1, LPT2 & LPT3 for device
1 Search only LPT1 for device
2 Search only LPT2 for device
3 Search only LPT3 for device
4 Search addresses 378H, 3BCH and 278H for device.
5 Search only address 378H for device
6 Search only address 3BCH for device
7 Search only address 278H for device

Options '0' and '4' work best for most situations. LPT port addresses are those established by the ROM BIOS during the boot of the computer. The addresses of the ports found are stored in a data table at segment 40 offset 8. If you have a situation where the security device search is encountering a printer port that does not have the device, then the interrogation routine can result in a reset signal being sent to the printer. Using the **Port_Select_Option** that corresponds to the actual port on which the device is installed can eliminate this. We recommend that the port option be stored in a data file that can be easily altered by the end-

user (e.g. using a simple text editor). Please remember that the port option is ignored by KEYLOK® device drivers.

There is a known problem associated with the use of **Watcom C** to generate AutoCAD add-ins. For the AutoCAD environment it is necessary to use a port searching option that uses port addresses, ***not*** LPT ports, or else a GPF is encountered during the attempt to read the port address from the port table.

G. Technical Specifications

Environment	
Storage Temperature	-10° F to 175°F (-23° C to 80° C)
Operating Temperature	32° F to 157° F (+0° C to +70° C)
Dimensions / Connectors	
Parallel	½" x 1 ¾" x 2 3/16", Body 1 ¼" (15 x 47 x 55mm, Body 33mm) DB25 on both ends. 1.2oz
USB	3/8" x 5/8" x 1 7/8" (9 x 17 x 49mm) Standard USB port
Serial Port	5/8" x 1 ¼" x 2 3/8", Body 2 1/16" (16 x 33 x 63mm, Body 53mm) DB9 on both ends
Memory	
Data retention	At least 10 years
Programmable EEPROM	• 112 Bytes • 1 Million Write cycles per location • Unlimited Read cycles
Security	
Encryption	Random Proprietary Encryption Algorithms
Authentication Password	2^{96} possibilities
Read Password	2^{128} possibilities
Write Password	2^{176} possibilities

H. Troubleshooting

For up-to-date troubleshooting guides and frequently asked questions please visit our support website:

http://www.keylok.com/support/default.aspx

If after reviewing our support website your questions have still not been answered you may send an email to

support@keylok.com

Or call to speak to one of our technical support representatives at

+1 (720) 904-2252
Mon - Fri 8am – 5pm MST (GMT –7)

Index