

KEY-LOK II[®]

Software Piracy Prevention System

USER'S MANUAL

Copyright

©Copyright 1982-2003 - All Rights Reserved. This documentation and the accompanying software are copyrighted materials. Making unauthorized copies is prohibited by law.

Trademarks

MAI Digital Security owns a number of registered and unregistered Trademarks and Service Marks (the "Marks"). These Marks are extremely valuable to MAI Digital Security and shall not be used by you, or any other person, without MAI Digital Security's express written permission. The Marks include, but are not necessarily limited to the following: KEY-LOK™; S-LOK™; COMPU-LOCK™; and the MAI Digital Security Logo.

S-LOK™ is a joint trademark of Microcomputer Applications, Inc. and A.S.M. Inc.

You shall not use any of the Trademarks or Service Marks of MAI Digital Security without the express written permission of such Trademark or Service Mark owner.

MAI Digital Security

1025 W 7th Ave
Denver, CO 80204 USA
Phone: (720) 904-2252 • Fax: (303) 228-0285
Email: Info@keylok.com • Web: www.keylok.com

Contents

Contents	3
Introduction.....	5
Getting Started.....	8
Installing the KEY-LOK Developer's Kit	9
Security Considerations.....	10
Enhancing Security	10
Use of Device Memory	10
Demo Mode	11
Product Version Control	11
Anti-Debugging Utility.....	12
KEY-LOK API Reference	13
Overview	14
Check For KEY-LOK	16
Read Operations	18
Write Operations.....	22
Date Control Tasks	26
Network Control Tasks	32
Remote Update Tasks	36
Application Termination	46
KEYBD	48
USB and Parallel Port.....	49
Protecting with S-LOK Wrapper	50
Device Access Over a Network.....	57
Overview.....	57
Networking Components	58

Network Protected Program Preparation	60
Controlling Authorized User Count	61
Installing Server Application	63
Device Search Sequence.....	63
Serial Port	64
Remote Update.....	69
Distributing Your Application	70
Using the KEY-LOK Install Utility.....	70
USB Installations	70
Parallel Port Installations	70
Install Utility Command Line Arguments.....	71
Appendix	72
File Descriptions	73
Protecting 16-bit Applications	75
Networking 16-bit Applications	76
Terminate.....	77
Protecting DOS Applications	78
32-bit DOS extended (not true 32-bit) applications	78
Special Language Support	79
Selecting Particular Parallel Port	80
Troubleshooting.....	82

Introduction

We at Microcomputer Applications, Inc. (MAI) are proud to offer the most complete solution to software security in the marketplace. MAI pioneered dongle based software piracy prevention in 1981, and has continued to lead the industry throughout the history of software protection.

Our product offerings include:

KEY-LOK II™

The KEY-LOK security system provides a very high degree of security with economical hardware key. Available for parallel, USB and serial ports on computers running DOS/WINDOWS 3.x/WIN95/98/ ME/NT/2000/XP/ LINUX. Serial port dongles can be used on any computer with an RS232 port and do not require a device driver. Provided with programmable memory, remote update, lease control algorithms, and network compatible routines (parallel port) for controlling multiple users with a single hardware lock.

S-LOK

A special version of KEY-LOK II that is capable of shrink-wrapping a protective shell around a program executable without having to make source code modifications. The S-LOK is currently compatible with only parallel port dongles.

Throughout this manual the KEY-LOK II product will be referred to generically as KEY-LOK. The electronic security device attaches to any parallel printer port (NOTE: USB port dongles are also available) on the computer (the software automatically searches each port until the device is located). Software interfaces to the security system and associated demonstration programs are available for most computer languages. 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. The hardware security device is also protected against reverse engineering.

General Product Overview

The KEY-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 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 given point in 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 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.

The KEY-LOK security system uses a number of sophisticated techniques for verification of hardware device presence. The KEY-LOK is also provided with 112 bytes 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. 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 the end-user's computer that is used to generate the initially displayed numeric sequence.

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 KEY-LOK memory. Any attempt by the end user to set back the date and/or time generates appropriate error codes to your application.

KEY-LOK's price to feature ratio is unparalleled in the industry.

Getting Started

Protecting your application can be achieved in 3 simple steps

Step 1 – Install Software Developer’s Kit

The KEY-LOK Software Developer’s Kit contains all of the necessary hardware and software to protect your application.

Step 2 – Run 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.

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

Once you are familiar with the KEY-LOK API calls simply copy and paste the relevant calls to your application.

Installing the KEY-LOK Developer's Kit

Step 1 Install the SDK software

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

Note: To prevent the CD from automatically running, hold SHIFT while inserting the CD

Step 2 Examine Demonstration Source Code

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 KEY-LOK API calls. Sample source code will be installed in the following directory:

Program Files\KEYLOK2\Compiler

Application specific notes are provided in the form of a README.TXT file.

Step 3 Extract Features to Use in Your Program

Strip the code for device features you wish to use from the demonstration program and include it within your program.

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

NOTE: Access to the KEY-LOK API calls is accomplished through an OBJ file or DLL file depending on your development tool. The appropriate interface file will be copied to the sample code directory.

Security Considerations

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

Enhancing Security

The *Check for KEY-LOK* 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 section below that provides additional suggestions regarding this capability.

- ♦ Store client name and or copyright message as text within device for display from within your application.
- ♦ Store critical constants used in calculations within your program.
- ♦ 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 KEY-LOK device to assure that your end user purchases product upgrades.

Reserve one or more addresses within the KEY-LOK 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 KEY-LOK system the memory containing the revision control code can be updated via an email exchange or a phone conversation with your client. *Refer to Remote Update Section for additional information.*
- ♦ One 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

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.

KEY-LOK API Reference

In this chapter we describe the KEY-LOK Programming Interface.

All KEY-LOK 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 KEY-LOK products using the same source code.

The KEY-LOK API has these important advantages:

- ♦ Easy-to-use function calls
- ♦ Local access through LPT, RS232 serial and USB ports
- ♦ Remote access via NetBIOS
- ♦ Hardware access via device drivers for Windows 9x/ME/NT/2000/XP
- ♦ Hardware access via standard RS232 calls for UNIX, Linux, QNX, SCO and Solaris

The KEY-LOK API is Easy, Secure and Portable

Overview

Access to the KEY-LOK API is accomplished by either linking with one of the KEY-LOK object (OBJ) files or utilizing the KEY-LOK DLL (KL2DLL32.DLL). If a DLL is required for your development language the appropriate DLL(s) will be copied to the same directory as your sample source code (e.g. ..Program Files\keylok2\devlang). 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\system directory.

Most KEY-LOK 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.

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	Supported Products
KLCHECK	1	USB, Parallel, Serial
READAUTH	2	USB, Parallel, Serial
GETSN	3	USB, Parallel, Serial
GETVARWORD	4	USB, Parallel, Serial
WRITEAUTH	5	USB, Parallel, Serial
WRITEVARWORD	6	USB, Parallel, Serial
DECMEMORY	7	USB, Parallel, Serial
GETEXPDATE	8	USB, Parallel, Serial
CKLEASEDATE	9	USB, Parallel, Serial
SETEXPDATE	10	USB, Parallel, Serial
SETMAXUSERS	11	Parallel
GETMAXUSERS	12	Parallel
REMOTEUPDUPT1	13	USB, Parallel, Serial
REMOTEUPDUPT2	14	USB, Parallel, Serial
REMOTEUPDUPT3	15	USB, Parallel, Serial
REMOTEUPDCPT1	16	USB, Parallel, Serial
REMOTEUPDCPT2	17	USB, Parallel, Serial
REMOTEUPDCPT3	18	USB, Parallel, Serial
DISABLESNCHECK	19	Parallel
GETNWCOUNTS	20	Parallel
TERMINATE	-1	USB, Parallel, Serial

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.

Check For KEY-LOK

A successful 'check for KEY-LOK' process is a prerequisite to running any other security device task (e.g. reading/writing to memory, etc). Many companies are content to use only the 'check for KEY-LOK' 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 KEY-LOK' event. The check for KEY-LOK 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 KEY-LOK (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 KEY-LOK (Second Call)	
Prerequisite	This call must be immediately preceded by the first call of the 'Check for KEY-LOK' 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. both zero as well as a random number would be acceptable).
ReturnValue1	ReturnValue1 = ClientIDCode1
ReturnValue2	ReturnValue2 = ClientIDCode2

If both parts of the customer identification code are successfully retrieved from the device then you have the option of proceeding with other device operations.

NOTE: Each time a Check for KEY-LOK 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 KEY-LOK*** 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:

- ♦ Write a random value to the device and read it back later to confirm device presence
- ♦ Store a copyright message or the actual name of your client within the device memory and read/display this information from within your program,
- ♦ Store critical constants used in program calculations within the device,
- ♦ Store 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,
- ♦ Store a count of the number of uses available (Counters)

READ AUTHORIZATION

Prior to running any read related task, a successful **check for KEY-LOK** 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 KEY-LOK**.

Read Authorization (READAUTH = 2)	
Prerequisite	Successful Check for KEY-LOK
Argument1	READAUTH = 2
Argument2	The constant ReadCode1
Argument3	The constant ReadCode2
Argument4	The constant 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 KEY-LOK** 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 a given Customer Identification Code will contain the same serial numbers.

Read Serial Number (GETSN = 3)	
Prerequisite	Successful <i>Read Authorization</i>
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 <i>Read Authorization</i>
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 could 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 KEY-LOK'
- 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 <i>Read Authorization</i>
Argument1	WRITEAUTH = 5
Argument2	The constant WriteCode1
Argument3	The constant WriteCode2
Argument4	The constant WriteCode3
ReturnValue1	Undefined
ReturnValue2	Undefined

WRITE A VARIABLE WORD

This task is used to modify the contents of programmable read/write memory within the KEY-LOK.

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

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

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 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. 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 their system date/time ahead. This often happens to our clients when testing the expiration date features of KEY-LOK.

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.

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 <i>Write Authorization</i>
Argument1	SETEXPDATE = 10
Argument2	Dummy argument
Argument3	<p>The expiration date encoded in the following bit format:</p> <p>YYYYYYMMDDDD (where YYYYYY + Base Year (i.e. 1990) = Year MMMM = Month of Year (1 = January) DDDD = Day of Month (1-31)</p> <p>The sample code shows how to encode the date</p>
Argument4	Dummy argument
ReturnValue1	<p>Status Code:</p> <p>0 = Success</p> <p>4 = Unable to write lease expiration date</p> <p>5 = Unable to write lease expiration support data (i.e. Current system date and time)</p>
ReturnValue2	Undefined

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

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 <i>Read Authorization</i>
Argument1	GETEXPDATE = 8
Argument2	Dummy argument
Argument3	Dummy argument
Argument4	Dummy argument
ReturnValue1	<p>The expiration date encoded in the following bit format:</p> <p>YYYYYYMMMMDDDDD, where</p> <p>YYYYYY + Base Year (i.e. 1990) = Year</p> <p>MMMM = Month of Year (1 = January)</p> <p>DDDDD = Day of Month (1-31)</p> <p>The sample code shows how to encode the date</p>
ReturnValue2	Undefined

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

CHECK LEASE/DEMO EXPIRATION

This task is used to compare the expiration date stored in the KEY-LOK 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 <i>Read Authorization</i>
Argument1	CKLEASEDATE = 9
Argument2	Dummy argument
Argument3	Dummy argument
Argument4	Dummy argument
ReturnValue1	Computer's current System Date in the format YYYYYYMM 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

	<p>date)</p> <p>-5 = Invalid lease expiration date</p> <p>-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 has been 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. This error can be cleared, but usually only by a trusted person. 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.</p> <p>+n = Approximate number of days until lease expires.</p>
--	--

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

Network Control Tasks

(Currently Only Supported with Parallel Port Dongles)

The following tasks are used to set/get the number of simultaneous authorized users of your application installed on a network. We provide a server application that communicates with the KEY-LOK 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 NETBIOS 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 KEY-LOK security with networks.

SET MAX USER COUNT

This task initializes the count of authorized simultaneous network sessions. For most applications, each 'session' corresponds to a 'user', however this is not always the case. NOTE: When using the default (i.e. 2 sessions) Win95/98/ME/NT/ 2000/XP device drivers, the new count will not become effective until the next time the driver is started. When using the 127-user version of Win95/98/ME/NT/ 2000/XP device drivers, the new count will become effective immediately upon completion of this task, without having to restart the driver. See the driver section of this manual for driver start/stop information.

Set Max User Count (SETMAXUSERS = 11)	
Prerequisite	Successful <i>Write Authorization</i>
Argument1	SETMAXUSERS = 11
Argument2	The desired simultaneous session count (1-127)
Argument3	Dummy argument
Argument4	Dummy argument
ReturnValue1	Undefined
ReturnValue2	Undefined

GET MAX USER COUNT

This task retrieves the authorized simultaneous network session count as recognized by the device driver, as well as the current number of active sessions. Remember that if you are not using the 127-user version of the device driver that the maximum allowed simultaneous sessions is 2. Also, the allowed session count defaults to '1' if the device is programmed with '0'.

Get Max User Count (GETMAXUSERS = 12)	
Prerequisite	Successful <i>Read Authorization</i>
Argument1	GETMAXUSERS = 12
Argument2	Dummy argument
Argument3	Dummy argument
Argument4	Dummy argument
ReturnValue1	The authorized network session count programmed
ReturnValue2	Undefined

GET NETWORK COUNTS

This task retrieves the authorized simultaneous network user count. A similar functionality is available in the form of a utility (LANMON.EXE) that can be run on the KEY-LOK server platform. See section 5.1 of this manual for further details regarding this utility.

Get Network Counts (GETNWCOUNTS = 20)	
Prerequisite	Successful <i>Read Authorization</i>
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 sessions

Remote Update Tasks

Updating memory within an end-user's security device can be accomplished many ways. One obvious alternative is 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 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 KEY-LOK 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:

- ◆ 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 read to the software developer.
- ◆ The software developer inputs the 5 numbers provided by the end user. The application re-computes the checksum and verifies that the data was properly conveyed between the two individuals and keyed

into the computer properly. If the data is 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.

- ♦ 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, as appropriate.

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

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 the configuration of the end user's computer.

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

REMOTE UPDATE USER PART 2

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

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

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 <i>Write Authorization</i>
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	<i>RemoteUpdateTask</i> * 8192 + Address Where 'RemoteUpdateTask' is: <table><tr><td>REMOTEADD</td><td>0</td></tr><tr><td>REMOTEDATEEXTEND</td><td>1</td></tr><tr><td>REMOTEOOR</td><td>2</td></tr><tr><td>REMOTEREPLACE</td><td>3</td></tr><tr><td>REMOTEGETMEMORY</td><td>4</td></tr><tr><td>REMOTESETUSERCT</td><td>5</td></tr><tr><td>REMOTEGETUSERCT</td><td>6</td></tr><tr><td>REMOTEGETDATE</td><td>7</td></tr></table> And address is the target memory address (i.e. 0 through 55).	REMOTEADD	0	REMOTEDATEEXTEND	1	REMOTEOOR	2	REMOTEREPLACE	3	REMOTEGETMEMORY	4	REMOTESETUSERCT	5	REMOTEGETUSERCT	6	REMOTEGETDATE	7
REMOTEADD	0																
REMOTEDATEEXTEND	1																
REMOTEOOR	2																
REMOTEREPLACE	3																
REMOTEGETMEMORY	4																
REMOTESETUSERCT	5																
REMOTEGETUSERCT	6																
REMOTEGETDATE	7																
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	<p>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.</p> <p>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.</p>

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 <i>Write Authorization</i>
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

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' driver operation the serial number of the KEY-LOK 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 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 <i>Read and Write Authorization</i>
Argument1	DISABLESNCHECK = 19
Argument2	Dummy Argument
Argument3	Dummy Argument
Argument4	Dummy Argument
ReturnValue1	Undefined
ReturnValue2	Undefined

Application Termination

This task is applicable when in two separate instances:

1) Application Exit:

When your application exits call ***Terminate*** to stop communications with the dongle.

2) Allowing USB dongles to be swapped

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...

TERMINATE

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

KEYBD

The KEYBD() function is used to activate the Anti-Debugger (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.

USB and Parallel Port

Protecting with S-LOK Wrapper

This easy to use 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 MAI's KEY-LOK 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 PCs running under the DOS, WINDOWS 3.x, WINDOWS 95/98/ME/NT/2000/XP 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.

The S-LOK security system also includes the use of optional automatic counters, expiration dates, and serial number verification.

A counter is 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 (i.e. the payment check clears), 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 end user) should you desire 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. Each side keys these values 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 the end-user's computer that is used to generate the initially displayed numeric sequence.

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.

S-LOK's price to feature ratio is unparalleled in the industry.

Security Device Installation

- 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 NT/2000/XP systems.

How to Protect An Executable Program

S-LOK is extremely easy to use. In order to protect a program simply follow these steps.

- 1) Run INSTALL.EXE to install KEY-LOK driver files
- 2) Run SHRLOK32.EXE (Ensure the Client.h and SHRINK32.DLL are located in the same directory)
- 3) Click the Input File button and select the application to be protected
- 4) Click the Output File button and select the name of the protected application (it is recommended make a backup copy of the Unprotected application if the same file name will be used for the protected application)
- 5) Click the Protect button.
- 6) The Output file produced will require a dongle to be attached to run

Protecting Microsoft Office Products

Several batch files are provided to assist clients who are primarily interested in protecting Microsoft Office products. These batch files allow you to issue a single command to protect all Office 95/97/2000 applications (i.e. WinWord, Access, PowerPoint, and Outlook (Office 97) installed on a Windows 95/98 platform.

LOCKO95.BAT - Lock Office 95 applications
LOCKO97.BAT - Lock Office 97 applications
LOCKO2K.BAT - Lock Office 2000 applications

Each of these files has as its first line a 'SET OFFICEDIR="path"' statement. If the default path is incorrect for your environment then simply edit the file with your correct target path, run the batch file, and all Office applications are automatically protected. The batch file also copies the appropriate device driver files to the proper location on the target system. Please remember that the CLIENT.H file corresponding to the desired run-time security device must be present at protection time. These batch files use the following options during the protection process:

/b - Batch mode
/on - Original file not preserved as .BAK
/oo - Original file overwritten by protected version of file

Activating S-LOK Optional Features

The various optional features are activated by preprogramming the security device memory using the CONFIG utility described in section 4. 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 REQUIREDSDN 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 Counter

This feature allows you to program the security device with a preestablished 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.

Remote Update

A remote update utility is provided to permit extensions to expiration dates, counters and product licenses without having to acquire the security device from your end-user. A very secure algorithm is implemented to assure that the update process is used only a single time, and only with the intended recipient. The update process is activated by an end-user calling you requesting some enhancement. Once you have arranged for payment you ask the end-user to run his half of the exchange algorithm (the utility named ENDUSER.EXE). The utility will display five numbers to the end-user, which he reads to you. The software developer's half of this algorithm is embedded within the CONFIG.EXE utility. Selecting menu option 'Remote Update' accesses it. You enter these five numbers into your half of the exchange algorithm. You then input the desired activity (e.g. extend the expiration date by 3 months, add an additional 100 program activations, activate the device to work with product 'B', increase the number of simultaneous network sessions to 10, etc.). You are then provided with four numbers, which must be entered by the end-user. Finally, the end-user will read you three numbers displayed on his system, which when decoded by the developer's utility will display the newly extended expiration date, counter, etc.

This utility can also be used to acquire current device settings (e.g. remaining counts, current expiration date, etc.) without having to acquire the device from the end-user.

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.

Device Access Over a Network

Overview

Parallel Port

Support for parallel port dongles allows monitoring the number of concurrent users. The SDK dongle allows up to 2 concurrent licenses. To increase this to 127 please contact our order desk.

USB

Support for USB dongles over a network allows unlimited users to use a single dongle over the network.

General Information

The KEY-LOK security system can be used to share the device among various applications running on a network. The advantage of this technique is that multiple nodes can be controlled through use of a single security device.

A server application is provided that communicates with the KEY-LOK device installed on the machine on which the server application is running. This technique should work on any network operating system running the NETBIOS 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 through the server application, up to the established maximum simultaneous user count.

Networking Components

The following components are required to implement the network security system:

Server Application: This is the program that each copy of your protected application communicates with in order to acquire contact with the security device. This program acts as the interface between your protected application and the device driver, which actually talks to the security device. The name of the server application is PARCLASS.EXE. A maximum of three copies of this server can be run on any network.

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 is PARCLASS.SYS for Windows NT/2000/XP and PARCLASS.VXD for Windows 95/98/ME.

KEY-LOK security device: The security device required for network operation is identical to that used for standalone operation. The device must be physically installed on the network node that is running the 'server application'. The device may be optionally programmed with user limits. If more than one key is designed to respond to the same set of client unique codes are 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.

Network: The computers on which the protected application and server application is running must be physically connected via a network with NETBIOS protocol supported on each platform. The underlying transport

layer below NETBIOS is unimportant (e.g. NETBEUI, TCP/IP, IPX/SPX, etc.), as long as there is at least one transport layer that support NETBIOS. Please note that it is critical on a TCP/IP network that the 'subnet mask' be set the same for both the server and clients. Network 'properties' can be used to acquire the subnet mask. Another method to acquire the subnet mask and other platform related data is to issue the command 'ipconfig –all'. One solution for resolving networking issues is to add an entry in the WINS server identifying the 'IP address' of the platform with the dongle server and the name of the dongle server service (typically 'PARCLASSSRV1'). Another similar solution involves adding the name/IP information to an LMHOSTS file on the client platform.

Utilities: Various utility applications are provided to assist in implementing and monitoring networked dongles. A brief description of each utility follows:

The **DEMO** program contains the required functions to read and write to device memory. This can be used to write the user count into the device in order to control the number of simultaneous users (i.e. client applications) that are allowed to run.

LANMON.EXE is a utility that can be run on the KEY-LOK server platform. The utility reports the maximum number of allowed sessions as programmed into the security device, as well as the current number of active sessions. This is a device specific utility (i.e. the 'DEMO' device version will not work with company unique devices, and vice versa).

Please note that when using network security features that your application must notify the security system when it is terminating so that it can release its usage allocation to another user/application. This is accomplished by sending a command code of TERMINATE.

Network Protected Program Preparation

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

- If a 32-bit application is linked with KFUNC32.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. KFUNC32N.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. Successful linking of 32-bit applications with KFUNC32N.OBJ also requires the WIN32 SDK supplied library of NETAPI32.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).

A second difference is notification to the security system when the application terminates so that the usage count allocated to the application can be freed up for use by another user. This is done automatically for 32-bit applications by the operating system. However, 16-bit applications require a physical termination call to the security system to let it know it is OK to free the count allocated to the application. This is done by calling KTASK with a command code of TERMINATE (i.e. -1).

It is always possible that a program may terminate abnormally, (e.g. CTRL-BREAK out of program, power failure, etc.). This will ordinarily result in the continued consumption of the allocated usage count. In order to prevent this from happening the security system will automatically free any node that has had no activity for a period of nine hours. Once a node is disconnected the application must be restarted in order to establish a connection with the security device.

If the default nine hour period is inappropriate for your application you can set it to whatever value you wish by creating a text file named PARCLASS.DAT that contains the desired number of minutes you would like to have elapse without device activity before a node is disconnected. This optional file should be placed in the same directory as the PARCLASS.EXE file as described in the paragraph below entitled, 'Installing Server Application'. The easiest way to create the text file is as follows:

At a DOS prompt enter the command:

```
copy con PARCLASS.DAT
```

Enter the desired number of minutes without any delimiters (e.g. commas):

```
1200
```

Then depress the function key F6 and ENTER key, which terminates and writes the file to disk.

Controlling Authorized User Count

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

- Driver Limited: The driver (PARCLASS.SYS/VXD) used with the server application contains a constraint on the maximum number of allowable end users. The default driver is capable of supporting two simultaneous users. This is the maximum number of simultaneous users that you are licensed to authorize with a single KEY-LOK device. If you wish each of your network installations to be restricted to two authorized users then no further action is required. Support for more than two simultaneous users requires a replacement driver with a maximum supported session limit of

127. The SDK dongle allows up to 2 concurrent licenses. To increase this to 127 please contact our order desk.

- Device Limited (End User Licenses): The KEY-LOK device can be programmed with the maximum number of sessions you wish to authorize with the device (SETMAXUSERS). This count further constrains the session count at the end user's facility within the limit established by the device driver.

The maximum number of sessions that can be started with a network enabled application is equal to the sum of: 1) the number of sessions programmed into a local security device, plus 2) the sum of the number of sessions programmed into one, two or three remote devices located 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.

A network enabled protected program will only operate in an environment in which sessions can be counted. The application should run on any workstation (e.g. Win 3.11, Win95/98/ME/NT/2000/XP, etc) that has NETBIOS connection to a Win95/98/ME/NT/2000/XP platform on which a security device is installed and the Parclass.Exe networking service running. However, the same protected program will only work with a local security device if it running under Win95/98/ME/NT/2000/XP with the KEY-LOK driver installed. It will not work on Win3.11 or other workstations with only a local security device because they are incapable of tracking the sessions.

Installing Server Application

The server application PARCLASS.EXE must be installed and started on the platform on which the KEY-LOK security device is attached. This is accomplished by running the INSTALL program with the /N option. Please review the detailed instructions for running KEYSETUP in the section of this manual entitled, *Distribution of Protected Application*.

Device Search Sequence

The following is the search sequence executed during the first call to check for the presence of the security device from each 'client' station running your protected application:

- 1) Search of local LPT Ports (parallel printer ports).
- 2) Search for dongle over the Network

Serial Port

The serial port dongle developed uses the same API functions and call structures for accessing dongle memory and authenticating the presence of a unique dongle as documented in the *KEY-LOK 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. Serial port dongle communications is based on the use of standard RS232 protocols.

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 gets the attention of the dongle connected to the platform issuing commands. 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.

Balance of 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 KEY-LOK II User's 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 user's manual delineates the use of task arguments.

The character sequence is encoded utilizing a client-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.

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 User's Manual. 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 attached 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 a 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 KEY-LOK 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.

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

- Clients receiving the software upgrade are required to call into a central location. Using the remote update capabilities of the KEY-LOK system the memory containing the revision control code is updated to correspond with the new product. This requires the end user to run either a utility or simply a menu selection from within your application that would trigger the remote update process. Basically a series of codes are presented to the end-user that are read to your representative. Your representative enters the codes into an application at your facility, tells the application what type of memory modification is required at the end user's facility, and is given a special code to read to the end user. The end user types the code into his computer, which upgrades the device memory, and presents him with a verification code. When the verification code is entered into your representative's system it confirms that the proper update has occurred. This same technique can be used to add licenses, extend counters or lease expiration dates, add additional network licenses, etc. The primary advantage of this technique is that the same distribution package can be sent to all recipients regardless of their current authorized licenses (i.e. no customization is required of the distribution media).
- The second approach is to prepare a utility to distribute with the software upgrade that instructs the device to advance the version control code. The primary advantage of this approach is that there is no telephone call required. The disadvantage of this approach is that the diskette containing the utility has to be customized to only work with a predefined serial number device in order to prevent its use on multiple end user platforms. One technique for doing this is to store the serial number in an encrypted form within a data file on the distributed diskette. The upgrade utility reads the value from the data file, decodes it, checks for the proper device, and only performs the memory upgrade within the KEY-LOK device if the serial numbers matched.

Distributing Your Application

Using the KEY-LOK Install Utility

The easiest and recommend method to install the required KEY-LOK files to your end-user's computer is to use the KEY-LOK Install Utility. This utility may be invoked from your existing installation program such as Install Shield or WISE.

It determines which operating system is running, and then copies the appropriate individual driver and KEY-LOK files to the proper directories.

NOTE: Modification of the NT/2000/XP system registry requires that a person with Administrator permission run the utility.

USB Installations

It is very important that the USB dongle NOT be attached to the computer until after the INSTALL utility has been run.

Refer to <http://www.keylok.com/fag/FAQ0017.htm> to resolve any possible issues if the dongle was inserted prior to running the INSTALL.

Parallel Port Installations

Begin by attaching a KEY-LOK device to the parallel port on the computer. The presence of the KEY-LOK is required in order to 'start' the device driver, but isn't actually required to 'install' the device driver files.

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 (NT, 2000, 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

This utility can be launched with one or more optional command line parameters, as follows:

Install.exe Utility	
Delimiters	Valid command line delimiters '/' 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	All drivers (parallel and USB) Local Dongle Display install messages
/B	Install USB files only Display install messages
/C	Client Allow Local USB Dongle Remote Parallel Dongle
/N	Network Dongle Install files on the network server – Parallel dongle
/P	Install Parallel files only Display install messages
/Q	Quiet mode install Install messages NOT displayed
/U	Uninstall Remove previously installed files

Appendix

File Descriptions

The following files provide support for device communication over a NETBIOS protocol on a network. These files are extracted into the ..\KEYLOK2\SENDUSER directory during installation by our SETUP utility.

Linked OBJ files and DLL's	
KFUNC32N.OBJ	Network object file to be linked into 32-bit applications. Checks for the dongle will first be made locally, if it is not found an attempt is made to find the device on the network. Successful linking of 32-bit applications with KFUNC32N.OBJ also requires the WIN32 SDK supplied library of NETAPI32.LIB.
KL2.OBJ	Some versions of KFUNC32.OBJ require linking with KL2.OBJ for resolution of external calls. 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.
KL2N.DLL	<i>16-bit DLL interface for local/remote (network) dongle checking</i>
NWKL2_32.DLL	32-bit DLL interface for local/remote (network) dongle checking
Networking	
PARCLASS.EXE	LAN KEY-LOK networking service that interfaces between your protected program and the device driver that actually communicates with the KEY-LOK device.
LANMON.EXE	A utility that can be run on the KEY-LOK server platform. The utility reports the maximum number of allowed sessions as programmed into the security device, as well as the current number of active

	sessions. This is a device specific utility (i.e. the 'DEMO' device version will not work with company unique devices, and vice versa).
ONLYLANA.DAT	A file developed to compensate for a bug in Windows 98 NETBIOS support that can cause the operating system to hang See section 5.6.2 LANA Searching – One or All for more details.
Utilities	
LANMON.EXE	Utility to run on KEY-LOK server platform to report the number of allowed sessions and current number of active sessions.

Windows NT/2000/XP Device Driver Files	
PARCLASS.SYS	WinNT KEY-LOK kernel driver. Must be in the following directory on the server: %SystemRoot%\System32\Drivers
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. %SystemRoot%\System32
Win95/98/ME	
PARCLASS.VXD	KEY-LOK kernel VXD driver. Used to communicate with the KEY-LOK. %windir%\system

Protecting 16-bit Applications

Applies to Parallel Port Dongles Only

KEYBD Function

KEYBD is a special function that 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 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 KEY-LOK 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 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.

Protecting DOS Applications

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

32-bit DOS extended (not true 32-bit) applications that run under Win95/98/ME/NT/2000/XP must be linked in such a way that the KEY-LOK data resides within the first 64K of the data space and the KEY-LOK 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.

Special Language Support

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

Parallel Port Dongles with 16-bit applications running under MS DOS.

The first is 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).

Parallel Port and USB Dongles

The other approach is to acquire a security routine in .EXE form from Microcomputer Applications. 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/written, by both programs.

Selecting Particular Parallel Port

The following are the rules for creating this port search criteria 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 KEY-LOK device drivers.

There is a known problem associated with the use of Watcom C to generate AutoCAD addins. 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.

Troubleshooting

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

<http://www.keylok.com/Support.htm>

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

(720) 904-2252 (Mon - Fri 7am – 5pm MST)