KEYLOK®

Software Piracy Prevention System

User Manual

KEYLOK2 SERIAL

Copyright

Trademarks

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

S-LOK™ is a joint trademark of KEYLOK and A.S.M. Inc.

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

KEYLOK

777 S Wadsworth Blvd.
Bldg. 4-220
Lakewood, CO 80226
USA

Phone: (303) 801-0338 or 1-800-4KEYLOK (1-800-453-9365)
Fax: (303) 228-0285 or (720) 294-0275
Email: info@keylok.com
Web: www.keylok.com

Contents

Windows Quick Start

Follow these steps to protect your application with a serial KEYLOK2 in 30 minutes:

Install the software.

1. 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.) **5 Minutes**

Run the demo.

2.
a) Attach your serial dongle.

b) Run *KLSERIAL.EXE* (in the ..\KEYLOK\Serial\Windows directory) to familiarize yourself with the features and functionality of the KEYLOK2 security system. **5 Minutes**

Compile the sample code.

3.
a) Compile the serial demo, using the C/C++ compiler of your choice. We provide Microsoft C/C++ 6 project files (which can be easily converted automatically to newer Visual Studio projects) as well as the source code. **5 Minutes**

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.

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:

KEYLOK2: The KEYLOK security system provides a very high degree of security with an economical hardware key. KEYLOK dongles 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 / Server2012, 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 dongles have programmable memory, remote update, lease control algorithms, and networking capability for controlling multiple users with a single hardware lock.

KEYLOK 3: KEYLOK3 is backwards compatible to the KEYLOK2 features, provides enhanced security over KEYLOK2, and offers the additional convenience of not requiring product unique device drivers, and is currently available for Windows platforms. The KEYLOK security system provides a very high degree of security with an economical hardware key. KEYLOK3 dongles are available for USB ports on computers running WINDOWS 9x / ME / NT / 2000 / XP / Vista / 7 / 8 / Server2003 / Server2008 / Server 2008 R2 / Server 2012. KEYLOK dongles have programmable memory, remote update, lease control algorithms, and networking capability for controlling multiple users with a single hardware lock.

FORTRESS: Backwards compatible to KEYLOK2, 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 (5K - 51K bytes) and provides an added level of tamper resistance, increasing the protection against piracy through reverse engineering.

Feature Comparison

Feature	KEYLOK2	KEYLOK3	Fortress
Hardware Type	Parallel, Serial, USB	USB	USB
Driverless		✓	✓
User Memory (Read/Write) bytes	112	112	5k-55k
Expiration Date	✓	✓	✓
Remote Update	✓	✓	✓
Counters	✓	✓	✓
Anti-Debugger	✓	✓	✓
Network Access	✓	✓	✓
Smart Card			✓
Tamper Resistant			✓
Code Vault (Optional)			✓
Flash Drive (1-4GB Optional)			✓
Real Time Clock (Optional)			✓

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 dongle 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 dongle presence. The KEYLOK 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 dongle it is necessary to successfully complete an exchange of bytes between the host and the dongle that differs during each dongle use (i.e., using an active algorithm). If this sequence is properly executed the dongle will return a customer unique 32-bit identification code which you can use as confirmation that one of your hardware security dongles 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 dongle to authorize memory read operations. Memory write operations require the successful transmission of yet another 48-bit customer unique password to the dongle. The write password must be sent AFTER transmission of the proper READ authorization sequence. If the dongle 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 dongle. 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 dongle for unlimited future use.

At the time of manufacturing each dongle is programmed with a unique dongle serial number, thus providing you the capability of identifying the specific dongle (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 dongle. 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).

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.

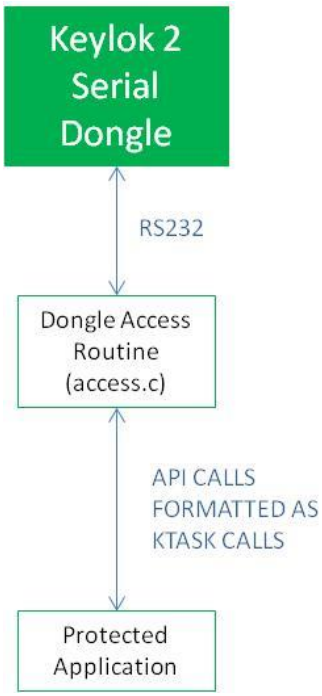

Figure 1: How Dongles Communicate with a Protected Application

Each KEYLOK dongle 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 dongles, 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.

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 dongle, the more effective the total security system will be.

Use of Dongle Memory

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

- The most common use of dongle 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 dongle 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 dongle 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 dongle 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 dongle.

Demo Mode

Many clients place their software in 'demo' mode if the security dongle 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 dongle to assure that your end-user purchases product upgrades.

Reserve one or more addresses within the KEYLOK dongle 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 dongle 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 dongle with the software upgrade. The dongle would be programmed with the appropriate authorization for use with the upgrade.

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
- 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 including (cut and paste) sample code into your target application to be protected.

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.

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	Page
KLCHECK	1	12
READAUTH	2	13
GETSN	3	14
GETVARWORD	4	14
WRITEAUTH	5	15
WRITEVARWORD	6	16
DECMEMORY	7	16
GETEXPDATE	8	18
CKLEASEDATE	9	19
SETEXPDATE	10	20
REMOTEUPDUPT1	13	23
REMOTEUPDUPT2	14	23
REMOTEUPDUPT3	15	24
REMOTEUPDCPT1	16	24
REMOTEUPDCPT2	17	25
REMOTEUPDCPT3	18	25
BLOCKREAD	21	26
BLOCKWRITE	22	26

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 KEYLOK

A successful ***Check for KEYLOK*** process is a prerequisite to running any other security dongle 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 dongle built uniquely for your company is present.

CHECK FOR KEYLOK

All other security dongle 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 dongle 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 dongle then you have the option of proceeding with other dongle 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 dongle serial number (unique to each dongle) and/or the programmable memory within the dongle. Examples of what can be done with the dongle memory include:

- Writing a random value to the dongle and reading it back later to confirm dongle presence
- Storing a copyright message or the actual name of your client within the dongle memory and reading or displaying this information from within your program,
- Storing critical constants used in program calculations within the dongle,
- 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 dongle to perform memory read operations. If the dongle 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 dongle serial number or other memory within the dongle 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 dongle 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 dongle. No two dongles 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 dongle 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 dongle 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 dongle 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 dongle 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 dongle to perform memory write operations. If the dongle does not receive the correct Write password the dongle 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 dongle 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 dongle. 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 dongle 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 dongle, 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 dongle 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 dongle, 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 dongle 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 dongle 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 dongle programming is being done) is written into this memory within the dongle. +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.

Remote Update Tasks

Updating memory within an end-user's security dongle can be accomplished many ways. KEYLOK provides Remote Update utilities that can be used via telephone or via e-mail to update the contents of an end-user's dongle, 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 dongle, 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 dongle 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 dongle (e.g. demo dongle) to test the sequence, the KEYLOK dongle 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 below):

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 dongle 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 dongle 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 dongle), 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 dongle 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.

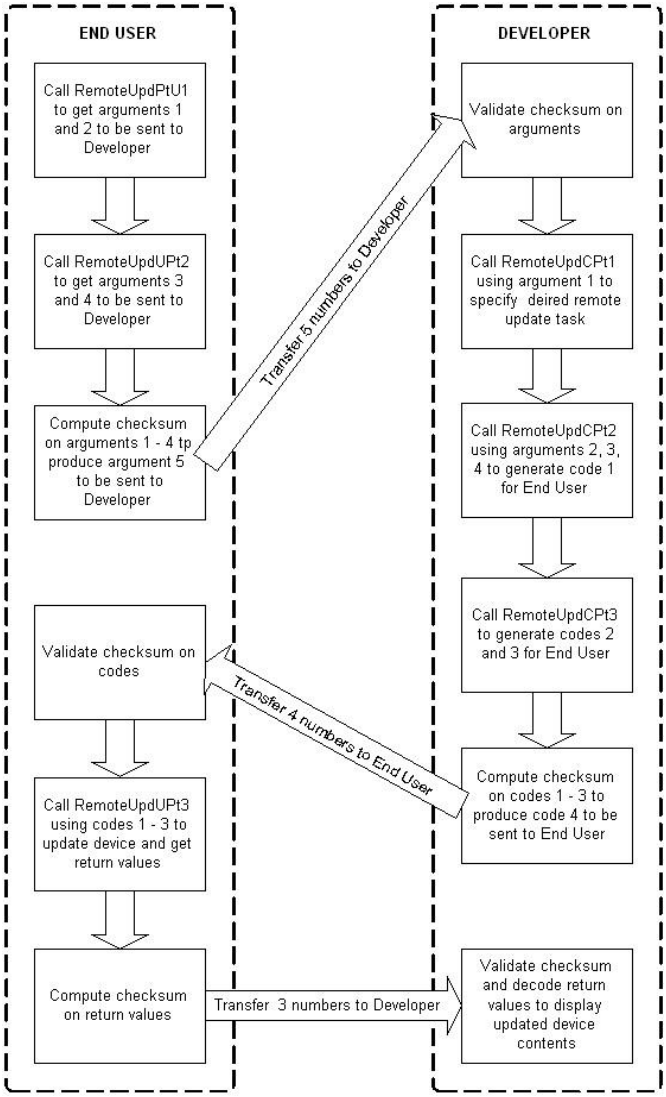

Figure 2: Remote Update Process Using API Calls

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 dongle 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 dongle 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.

Block Memory Read

This task is used to activate a block memory read from the dongle.

Block Memory Read **(BLOCKREAD=21)**	
Prerequisite	Successful ***Read Authorization***
Argument1	**BLOCKREAD = 21**
Argument2	number of words to be read
Argument3	starting address to begin reading from
Argument4	Dummy argument*
ReturnValue1	Undefined
ReturnValue2	Undefined

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

The words are returned in low byte/high byte sequence. 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

This task is used to activate a block memory write to the dongle.

Block Memory Write **(BLOCKWRITE=22)**	
Prerequisite	Successful ***Write Authorization***
Argument1	**BLOCKWRITE = 22**
Argument2	number of words to write
Argument3	starting address to begin writing
Argument4	Dummy argument*
ReturnValue1	Undefined
ReturnValue2	Undefined

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

The words are sent in high byte/low byte sequence.

The serial port dongle does not require the use of any external device driver to communicate with the dongle. API functions are passed directly to the dongle via RS232 (see Fig. 3). Serial port dongle communication is based on the use of standard RS232 protocols.

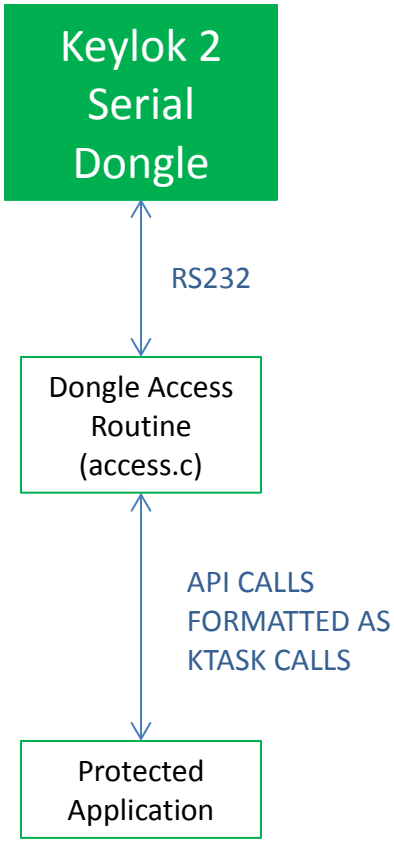

Figure 3: Serial Dongle Access

Dongle 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 dongles 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 dongles, and data from downstream dongles 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 dongle 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 dongle 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 dongle reset and any attempt to write to the dongle 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.

Communications Parameters

KEYLOK Serial dongles 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 **Remote Update** API calls that will allow you to securely read, write, or modify any of the programmable features of KEYLOK dongles remotely. The operation uses exchanges of codes between the developer and the end-user.

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. 4):

1) The end-user runs *RemoteUpdateUser.exe* on a machine that has the dongle to be updated attached (and the device drivers installed) and clicks “Update Security Dongle”. 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.).

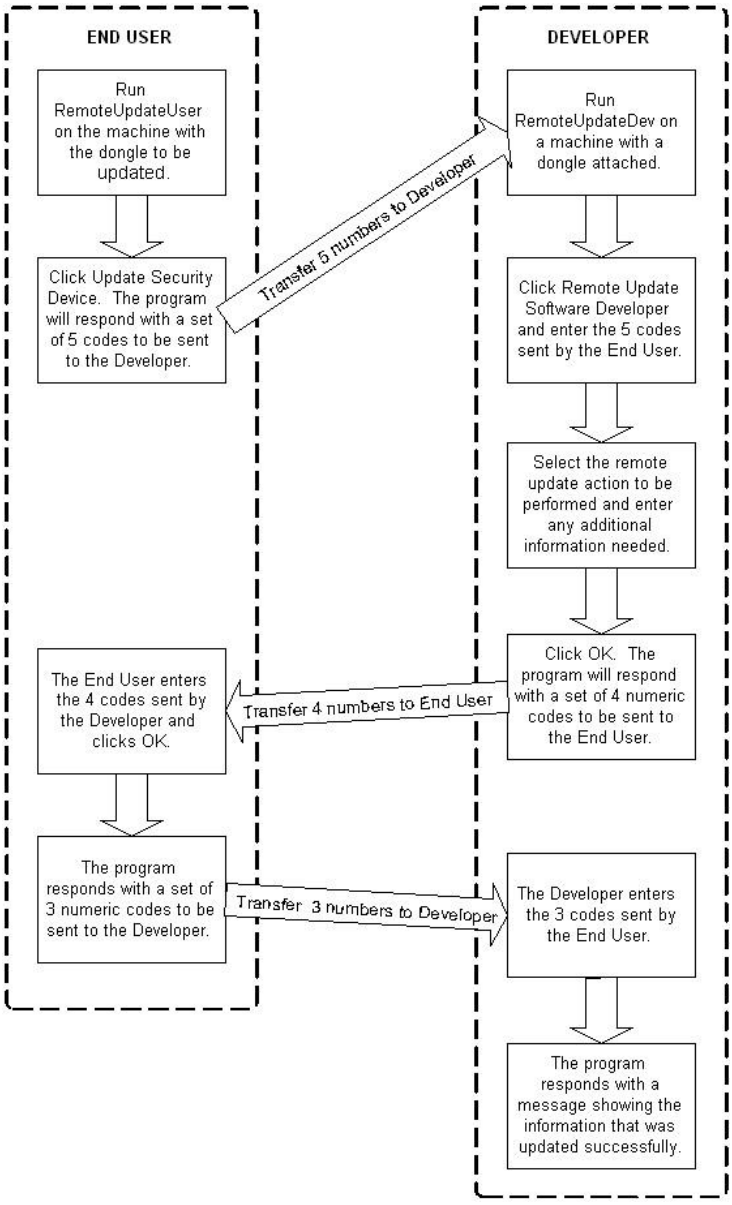

Figure 4: Telephone Remote Update Process

Appendices

A. 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	
Serial Port	5/8" x 1 ¼" x 2 3/8", Body 2 1/16" (16 x 33 x 63mm, Body 53mm) DB9 Socket: Connects to computer DB9 Plug: Connect to peripheral
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

B. Troubleshooting

For up-to-date troubleshooting guides and frequently asked questions please visit our support website: http://www.keylok.com/support-center/overview

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 303.801.0338 x788
Mon - Fri 8am – 5pm MST

Index