



SmartKey User Manual v11.0.1

Commercial email: info@eutron.com
Helpdesk email: helpdesk@eutron.com
Commercial phone: +39-035697080
Helpdesk phone: +39-035697055
Fax: +39-035697092

The names of the other products mentioned are trademarks of their respective owners.

COPYRIGHT: The trademarks of the products and software supplied are the property of Eutron spa or its suppliers and are protected by copyright laws in force, by the provisions of international treaties and by any other applicable domestic laws. The user is entitled to make a backup copy of the software supplied. Reverse engineering, alteration, conversion, decoding, decompiling, disassemble or similar operations are illegal and will be prosecuted in conformity with current laws. Documentation, whether on hard copy or electronic medium, cannot be disclosed without Eutron's prior consent in writing. No parts of the document may be reproduced, stored in a retrieval system or transmitted in any form or by any means (electronic, mechanical, photocopying, etc.).

SOFTWARE LICENSE: all the software supplied is the property of Eutron or its suppliers and is granted under license. The software cannot be leased nor rented out, but the rights granted can be assigned finally provided all copies of the software and all the printed matter are assigned and that the licensee accepts the terms and conditions of this Agreement. Disposal of the License is subject to the stipulation that Eutron and its suppliers shall be held harmless and indemnified by the user for any claims or legal action including legal costs arising from the use or distribution of the User's products.

DOCUMENTATION : The documentation supplied has been subject to careful inspection; however neither EUTRON nor its suppliers accept any liability for the consequences of any errors or omissions in the information supplied, or for unrecorded characteristics or variations of its products. The safekeeping of the parts containing information on the operation of both hardware and software products falls under the User's responsibility.

VARIATIONS: EUTRON reserve the right to modify the characteristics of their products, their technical specifications and the prices without advance notice.

LIMITS OF THE GUARANTEE: The term of the guarantee is 5 years counting from the date of purchase. During such period only and in the event of malfunctioning of the hardware or software, Eutron may, at its discretion: a) repair or replace the software or hardware provided the material is sent to Eutron at the User's expense and returned to the User at his own expense and according to his directions; or b) repay the price paid. This guarantee is not applicable if the malfunctioning of the software and hardware is due to accident, misuse or misapplication.

EXCLUSION OF ANY OTHER GUARANTEES : Eutron does not acknowledge any other guarantee, whether expressed or implicit, including, among others guarantees of salability and suitability for a specific purpose, with regard to the software, the hardware and any other associated material.

DISCLAIMER OF RESPONSIBILITY : In no case shall Eutron and its suppliers be held answerable for any damage, including without restrictions damage from loss of profit or failure to secure profit, interruption of the activity, loss of information or other financial losses, direct or indirect, incidental or consequential damage arising from the use or impossibility to use the products supplied for any reason whatever, including failures or breakdowns in the operation of the hardware or the software supplied even should Eutron have received notice of the possibility of such damage. In any case, under the provisions of this agreement, Eutron's liability shall be limited to a sum equivalent to that actually paid for the product. Similarly Eutron shall not be held answerable for damage caused by person using the products illegally or divulging copies of the software or calculation codes or reserved data illegally, breaking the protections by means of hardware and/or software manipulations. Eutron are not answerable for an improper use of its own products and does not guarantee nor authorize their use in applications where they may cause damage to persons or objects either during normal operation or during malfunctioning or breakdowns.

ADDITIONAL SUPPLIES: The clauses of this Agreement are also applicable to the software and hardware of subsequent supplies, as in the case when the User receives or purchases updates, additional licenses or utilities software or other hardware products in addition to the first.

DELIVERY: The goods are consigned ex works and delivered at the customer's expense and according to his instructions. Eutron is therefore answerable only up to the time of delivery of the goods to the carriers and cannot be held liable for losses or damage in transit.

Index

1	INTRODUCTION.....	6
1.1	MANUAL'S OBJECTIVES	6
1.2	TO WHO IS THE MANUAL ADDRESSED	6
1.3	HOW TO CONTACT EUTRON	6
2	WHY PROTECT SOFTWARE?	7
2.1	THE MENACE OF PIRACY FUELS SOFTWARE PROTECTION NEED	7
2.2	BUSINESS REASONS FOR SOFTWARE PROTECTION	7
2.3	PIRACY – A BANE FOR THE SOFTWARE INDUSTRY.....	7
2.4	HISTORY OF SOFTWARE PROTECTION	8
2.5	A NEW-AGE SOFTWARE PROTECTION TECHNIQUE	8
3	INTRODUCTION TO SMARTKEY	9
3.1	WHAT IS SMARTKEY?	9
3.2	WHO CAN USE SMARTKEY?	9
3.3	SALIENT FEATURES OF SMARTKEY	9
3.4	HOW DOES SMARTKEY WORK?	10
3.5	SMARTKEY'S SUPPORTING UTILITIES	10
3.6	SMARTKEY'S SECURITY LEVEL	11
4	SMARTKEY MODELS	12
4.1	FX	12
4.2	PR	12
4.3	EP	13
4.4	SP	13
4.5	NET	14
4.6	A COMPARATIVE SNAPSHOT OF SMARTKEY MODELS.....	14
4.7	SELECTING THE RIGHT SMARTKEY MODEL	15
5	PROTECTING AN APPLICATION WITH SMARTKEY.....	16
5.1	IMPLEMENTING MANUAL PROTECTION	16
5.2	IMPLEMENTING AUTOMATIC PROTECTION.....	16
5.3	MANUAL VERSUS AUTOMATIC PROTECTION	16
6	LOCAL NETWORK PROTECTION.....	18
6.1	THE AUTOMATIC PROTECTION IN LOCAL NETWORKS	18
6.2	THE MANUAL PROTECTION IN LOCAL NETWORKS	19
6.3	PROTECTION OF SEVERAL APPLICATIONS WITH SMARTKEY	19
7	THE INTERNAL STRUCTURE OF SMARTKEY	20
7.1	ID-CODE REGISTER : THE PERSONAL CODE.....	20
7.2	LABEL REGISTER : THE IDENTIFICATION AND ACCESS TAG	20
7.3	PASSWORD REGISTER : THE KEY TO ACCESS DATA	21
7.4	SECURE DATA REGISTER : THE INTERNAL SECURE DATA	21
7.5	FAIL COUNTER REGISTER : WRONG ACCESS ALARM.....	21
8	AUTOMATIC PROTECTION	23
8.1	AUTOMATIC PROTECTION WITH GSS	24
8.2	PROTECTION ACROSS ALL PLATFORMS USING GSS.....	24
8.2.1	Protecting Windows Programs.....	24
8.2.2	Protecting DOS Programs.....	24
8.3	GSS: THE COMMON OPTIONS	24
8.3.1	Checking the Key's Presence.....	24
8.3.2	Programming Error Messages	25
8.3.3	Encrypting the Executable Code	25
8.3.4	Parameter Based Protection.....	25

8.3.5	<i>Message Displayed When the Key is Missing</i>	25
8.3.6	<i>Limiting the Number of Executions and Licenses</i>	25
8.3.7	<i>Encrypting Executable Codes Without Access to the Key</i>	26
8.3.8	<i>Automatic Encryption of Data Files</i>	26
8.3.9	<i>Protecting Applications on Network</i>	26
8.3.10	<i>Protecting Executable Files in Batches</i>	26
8.4	QUICK TAKEAWAYS ON EFFECTIVE APPLICATION PROTECTION	26
9	MANUAL PROTECTION	28
9.1	LOCATING MODE	28
9.1.1	<i>Parameter passing</i>	29
9.2	SCRAMBLING MODE	29
9.2.1	<i>Parameter passing</i>	29
9.3	READING MODE	30
9.3.1	<i>Parameter passing</i>	30
9.4	BLOCK READING MODE	31
9.4.1	<i>Parameter passing</i>	31
9.5	WRITING MODE	31
9.5.1	<i>Parameter passing</i>	32
9.6	BLOCK WRITING MODE	32
9.6.1	<i>Passing parameters</i>	33
9.7	FIXING MODE	33
9.7.1	<i>Passing parameters</i>	34
9.8	ENCRYPTING MODE	34
9.8.1	<i>Passing parameters</i>	35
9.8.2	<i>Definition of the algorithm</i>	35
9.8.3	<i>Example</i>	35
9.9	PROGRAMMING MODE	36
9.9.1	<i>Passing parameters</i>	37
9.10	COMPARING MODE	37
9.10.1	<i>Passing parameters</i>	37
9.11	MODEL READING MODE	38
9.11.1	<i>Passing parameters</i>	38
9.12	SERIAL NUMBER READING MODE	39
9.13	EXT MODEL READING MODE	39
9.14	FIX READING MODE	40
9.15	FAIL COUNTER READING MODE	40
9.16	THE CHANGE OF PARAMETERS WITH SMARTKEY	40
9.17	SOME HINTS ON THE USES OF SMARTKEY FUNCTIONS	41
9.18	AN EXAMPLE	41
10	MANUAL NETWORK PROTECTION	46
10.1	OPEN MODE	46
10.2	ACCESS MODE	46
10.3	USER NUMBER MODE	46
10.4	CLOSE MODE	47
10.5	CLOSE MODE ON TIMEOUT	47
10.6	STAND-ALONE DRIVER OR MULTILAN ?	47
11	PROTECTION OF SEVERAL APPLICATIONS WITH SMARTKEY	48
11.1	THE OPERATING MODES	48
11.2	NUMBERS OF LICENSES AND OF NUMBER OF EXECUTIONS	48
11.3	THE MAP AUTOMATIC PROTECTION	50
11.4	THE MAP AUTOMATIC PROTECTION	50
11.4.1	<i>Open mode</i>	50
11.4.2	<i>Open mode MAP: an example</i>	50
11.5	AN EXAMPLE	50
12	INSTALLATION	56
12.1	WARNINGS FOR THE INSTALLATIONS	56
13	NETWORK INSTALLATION	57

13.1	TCP/IP PROTOCOL	57
13.2	IPX PROTOCOL	58
13.3	ANP PROTOCOL	58
13.4	INSTALLING THE NOVELL SERVER WITH THE IPX PROTOCOL.....	58
13.5	INSTALLING THE NOVELL SERVER WITH THE TCP/IP PROTOCOL	59
13.6	INSTALLING THE SERVER FOR WINDOWS WITH THE TCPIP-ANP PROTOCOLS.....	60
13.7	CONFIGURING THE SERVER FOR WINDOWS MULTIPROTOCOL USING ASKEYADD.....	60
13.8	CONFIGURING THE SERVER FOR WINDOWS MULTIPROTOCOL USING ASKEYSRV.INI	61
13.9	INSTALLING THE SERVICE FOR WINDOWS WITH THE TCPIP-ANP PROTOCOLS.....	61
13.10	CONFIGURING THE SERVICE FOR WINDOWS WITH THE TCP/IP-ANP PROTOCOLS	62
13.11	INSTALLING THE DOS CLIENT WITH THE ANP-IPX PROTOCOLS.....	62
13.12	INSTALLING THE CLIENT FOR WINDOWS WITH THE ANP-IPX-TCPIP PROTOCOLS.....	63
13.13	CONFIGURING THE CLIENT FOR WINDOWS WITH THE ANP-IPX-TCPIP PROTOCOLS USING CSKEYCFG	63
13.14	CONFIGURING THE CLIENT FOR WINDOWS WITH THE ANP-IPX-TCPIP PROTOCOLS USING SKEYLINK.INI	63
13.15	CONFIGURING THE CLIENT FOR WINDOWS WITH THE ANP-IPX-TCPIP PROTOCOLS USING THE OLD SYNTAX OF SKEYLINK.INI	64
13.16	CONFIGURING THE CLIENT FOR WINDOWS WITH THE ANP-IPX-TCPIP PROTOCOLS USING ENVIRONMENTAL VARIABLES	65
14	PROTECTION GUIDELINES	66
14.1	GENERIC GUIDELINES	66
14.1.1	<i>Check the key in many points of your applications</i>	66
14.1.2	<i>Extensive use of the Scrambling operation</i>	66
14.1.3	<i>Hide the Label and Password</i>	67
14.1.4	<i>Use the .OBJ version of the drivers</i>	67
14.1.5	<i>Checksum your Executable and DLLs</i>	67
14.1.6	<i>Do not stop the execution immediately if the key isn't found</i>	67
14.1.7	<i>Encrypt required data with the Scrambling operation</i>	68
14.2	MEMORY GUIDELINES	68
14.2.1	<i>Check the memory functionality</i>	68
14.2.2	<i>Store required data in memory</i>	69
14.3	IMPLEMENTATION EXAMPLES	69
14.3.1	<i>Example 1 – Basic Use</i>	69
14.3.2	<i>Example 2 – Basic Scrambling Use</i>	70
14.3.3	<i>Example 3/4 – Storing and using a C function in the SmartKey memory</i>	71
14.3.4	<i>Example 5 – Check of the DLL checksum</i>	73
14.3.5	<i>Example 6 – Hiding the Label and the Password information</i>	74
14.3.6	<i>Example 7 – Scrambling sensitive data</i>	75
14.3.7	<i>Example 8/9–Generation and use of a big Scrambling table</i>	76
14.3.8	<i>Example 10 – Code Encryption</i>	79
15	TECHNICAL SPECIFICATIONS	80
15.1	FEATURES	80
15.2	SMARTKEY 2 PARALLEL	80
15.3	SMARTKEY 2 USB.....	80
15.4	SMARTKEY 3 USB.....	80

1 Introduction

1.1 *Manual's objectives*

The aim of the following manual is to supply a complete panorama of the application ambits of SmartKey and of the operational potential of the product in general.

The arguments discussed in this manual are:

- the importance in protecting software
- how to use SmartKey in the case of manual protection; the part on detailed programming is dealt with in the Programmers Manual
- how to use SmartKey in the case of automatic protection: this manual exhaustibly deals with the techniques and the instruments for the automatic protection of software and of data
- the different application situations and how to carry out the implementation of protection in the ambit of local networks

1.2 *To who is the manual addressed*

Consulting the present manual can be useful to those who:

- for the first time have gotten close to the software protection techniques and wants a complete panorama on the art and efficient practical indications;
- want to have at disposal the traditional manual protective techniques but at the same time secure and versatile instruments for the automatic protection of software and of data;
- have the need to check the authorization to the execution of software: with few operations, Eutron's technology allows in fact, using the tools supplied in the Developer Kit, to create sophisticated mechanisms for the control of licenses and execution parameters, with the simple usage of a SmartKey-PLUS NET key.

1.3 *How to contact EUTRON*

The best way to contact EUTRON for SmartKey is to send an email message to the helpdesk helpdesk@eutron.com. You may also telephone the Assistance Service at +39-035697055 or send a fax to the number +39-035697092.

For commercial contacts, you may telephone at +39-035697080 or send an e-mail message to info@eutron.com.

2 Why Protect Software?

2.1 The Menace of Piracy Fuels Software Protection Need

In a technologically driven world, software has become critical. The task of developing them demands enormous commitment in terms of time, effort and money. So, software becomes the intellectual property of the developer or the organization that produces it and rightly so. But this right of the developers is often breached, quite often causing crippling loss for the developers. Thus, the need for software protection.

Organizations need to exercise control over their intellectual property to stave off bids by software pirates to alter and distribute them to unlicensed, unregistered users. Besides, decreasing revenues due to piracy has a ripple effect, resulting in lower returns for stakeholders.

Another reason why the scourge of piracy has become so wide spread is because forgery of this type of intellectual property allows production of perfect, functional copies. Add to that the huge profits that can be made by illicitly selling software.

2.2 Business Reasons for Software Protection

Piracy has put software developers on the defensive. Businesses are increasingly factoring in revenue losses due to piracy in their business plan. Alarmed by the enormity of the loss, business reasons for implementing a software protection solution have now gained top-of-mind attention.

Today, the reasons for effective software protection are aplenty and easy to zero in on. For one, software developers stand to increase market share and maximize profits using dynamic licensing and strong application protection from software piracy.

There is an increasing realization among the developer community that there should be a good balance between the protection plan and the financial benefits for that plan.

Additionally, to assess the business feasibility of software protection, developers need to consider factors like:

- Are they operating in a niche market where if the software is pirated, they risk losing the target market
- Where do they intend to sell the application in the world and what are the piracy rates for those world regions
- Consider how cost effective it is to add protection to the application

2.3 Piracy – A Bane For the Software Industry

Software piracy has put the concept of intellectual property in jeopardy. Put simply, software piracy is the practice of copying and using a software product without the permission of its owner or developer. The realization that unauthorized use and duplication of software is illegal is only now beginning to dawn on most computer users.

Although many still show a general disregard for the importance of treating software as valuable intellectual property.

Software piracy takes place in various ways. Some of them are discussed below:

- Soft-lifting: This involves purchasing a single licensed copy of software and loading it onto several computers, contrary to the license terms.
- Uploading and downloading: Stands for making of unauthorized copies of copyrighted software available to end-users connected by modem to online service providers and/or the Internet.
- Software counterfeiting: Illegally duplicating and selling copyrighted software in a form designed to make it appear legitimate.
- OEM unbundling: Selling stand-alone software originally intended to be bundled with specific accompanying hardware.
- Hard disk loading: Installing unauthorized copies of software onto the hard disks of personal computers, often as an incentive for the end user to buy the hardware from that particular hardware dealer.
- Renting: Unauthorized selling of software for temporary use, like you would rent a video.

Software pirates too are of various types. They are:

- Dealers selling hardware pre-loaded with illegal software

- User organizations making unauthorized copies of software for internal use
- Professional software counterfeiters
- Competitors using unauthorized software copies to develop competing products
- Hackers' web sites offering illegal software to users
- Any individual who makes an unauthorized copy of someone else's software program

2.4 History of Software Protection

During the '70s, most software was proprietary to a specific business environment and operated on mainframe systems. Thus, piracy was non-existent. But as computers moved from centralized mainframes to client/server models, installation of software at individual workstations became the norm, resulting in a paradigm shift in software distribution and licensing.

The '80s saw a personal computer explosion, creating the need for software protection as development of a large-scale underground pirate network for illegal software distribution soon followed. The '90s, due to the Internet opening up new channels of software delivery, saw a significant jump in piracy. New technologies like CD burners made duplication and distribution of illegal software copies easy, contributing to the rise in piracy.

The battle to curb piracy had started in the eighties with software protection mechanisms like special disk formatting obtained by driving the floppy drive outside the operating system (additional tracks, unformatted tracks, exchange of disk sectors, variable disk speeds, etc.) for the MSDOS environment or by altering the magnetic support (laser fingerprint, etc.).

Other protection mechanisms, like preventing floppy disks from being copied by standard MSDOS commands to prevent the user (or hinder him considerably) from backing up software for personal use too became popular. As software protection technologies have improved, businesses have started using a combination of hardware-based security and license management applications to eliminate unauthorized use.

With the need for protection growing, ways of keeping software and applications secure have also kept pace. Although, proponents of piracy still survive—in fact, thrive—causing much concern and revenue loss for businesses. The ever-expanding scope of technology has so far remained the biggest enabler for piracy; the biggest motivator being dollops of illegal money. Thus, piracy is not going to die anytime soon, which only serves to amplify the need for credible software protection mechanisms for businesses.

2.5 A New-Age Software Protection Technique

Eutron has created an innovative new technology that represents a departure from the traditional protection paradigm. The Eutron solution effectively resists piracy and the spread of universal cracking methods. Moreover, the focus of software protection strategy has today narrowed from “copy protection” to a much more reliable “controlling the execution of the software”. Thus, curbing software piracy goes beyond merely preventing copies from being made. It has evolved into limiting the use of software to one copy at a time.

Eutron's SmartKey-Plus, a hardware key for software protection, is eminently equipped for this shift in security need. Employing the most advanced anti-hacking mechanisms, which are virtually impossible to crack, SmartKey-Plus is actually a Hardware-based copy protection systems offering the best and the most proactive solution for software vendors.

SmartKey-Plus is a small electronic device, approximately the size of a matchbox, which is plugged into the connector of a computer's parallel or USB interface, or within it. Each key is characterized by a unique personalized digital fingerprint which is recognized by the software that is to be protected. This hardware-based key validates that a user is legally using a software application by verifying whether the key is connected to the user's computer during the application's runtime.

What's more, encryption is the heart of this software protection technology. By encrypting text, it realigns the alphabet or manipulates the text in some consistent pattern. Then, by using a combination of an encryption key, a variable that is combined in some way with the unencrypted text, and an algorithm, which is a formula for combining the key with the text, SmartKey-Plus thwarts violation attempts by pirate copying software. It also overcomes the drawbacks of software key protections, which does not guard against unauthorized proliferation of the software as the end user can generate several back-ups of the protected program.

3 Introduction To Smartkey

3.1 What is SmartKey?

Software protection has truly come of age with SmartKey, a software protection key. It provides you with a full-featured, easy-to-use software protection system. Now you can easily limit the execution of any PC application as well as retain control on the number of computers your software can run on.

Simply put, SmartKey is essentially a piece of hardware which prevents illegal duplication of software. It comprises of a compact sized hardware key which is plugged into the PC parallel port. Each key electronically holds its own unique identity number, which is recognized by the software to be protected and enables it to run properly.

Once the protection key is plugged into the PC's parallel port or the USB port, it communicates with the software application to be protected. During the execution, the software first verifies the key's presence and its internal code. When a correspondence is established, the software is activated. But when the key is not plugged in, the software either stops running or simply proceeds in demo mode.

The software may be protected either by automatically encrypting executable files or integrating specific software drivers. SmartKey comes equipped with several software utilities, making it very versatile. Using the software supplied and the proprietary GSS (Global Security System) technology, an application can be protected in a matter of minutes.

3.2 Who Can Use SmartKey?

The need for software protection spans across industries. Thus, a state-of-the-art software protection system like SmartKey finds use in diverse sets of businesses. Although primarily targeted at software companies as they need to protect their software against illegal circulation and copying, other end users like business and information systems professionals too benefit hugely from its technology.

Discussed below are some of the areas in which SmartKey has been used successfully:

- Software houses use SmartKey to prevent software piracy and illegal circulation of their applications. Only users who own a key can run protected software. Used by over 5,000 software houses the world over, SmartKey is based on ASIC (Application-Specific Integrated Circuit) chips specifically designed for its security functions.
- Sales personnel use SmartKey to deliver fully operative demo copies of their software to prospective buyers, free of the risk of unauthorized copying and use. Using SmartKey, they can even monitor the number of times the software is run.
- Corporate heads and professionals in charge of information systems use SmartKey to avoid civil and criminal liability arising out of software piracy by employees. By using SmartKey site-license agreements, businesses can guard against illegal use of software.
- Teachers as well as labs in schools and universities use SmartKey to guard against liability arising out of software piracy carried out by students. Generally, software used in schools is purchased at special conditions and heavy penalties are involved if they are used outside the campus.

For end users, irrespective of the need and the type of business, using SmartKey is easy. Simply plug it into the parallel port and software protection at its best kicks in, keeping your codes secure even while authorized users continue to run the software. What's more, SmartKey is discreet and easy to install. It protects applications and remains user-transparent at all times.

3.3 Salient Features of SmartKey

SmartKey is a state-of-the-art hardware-based system that prevents unauthorized access to and execution of protected applications. Activating a protected software application is only possible by using its original key. Using SmartKey for protecting your software is easy, yet the level of security it provides is extremely high.

SmartKey is based on microprocessors and proprietary ASIC (Application Specific Integrated Circuit) electronic chips. This enables a high degree of integration in addition to preventing reverse engineering. These and other salient features of SmartKey have been discussed below in detail:

- **High Level of Protection:** As SmartKey does not use standard electronic components but ASIC circuits that are made to order, hardware cloning of the key is impossible.

- **Parallel Port Installation:** SmartKey can be plugged into the parallel port of any IBM PC-compatible personal computer.
- **USB Port Installation:** SmartKey can be plugged into the USB port of any IBM compatible personal computer.
- **Stacking:** Several SmartKeys can be cascaded on the same parallel port with daisy-chain connection.
- **Transparency:** As SmartKey is transparent it does not prevent the use of parallel port. Practically any peripheral meant for parallel port can be daisy-chained to the key including items like the Mains Adapter, SCSI, Portable Hard Disk and other protection keys.
- **Algorithmic Interrogation:** SmartKey uses algorithmic-type interrogation mechanisms to protect software and encode confidential data.
- **Code Customization:** Each SmartKey has a different, internally-set code for each customer. This facilitates customization as unique codes allow businesses to implement their own protection systems.
- **Code Programmability:** SmartKey is equipped with additional user-programmable codes. Thus, it uses the software utilities supplied along with the key and does not require special programming devices.
- **Internal Memory:** SmartKey has a 416-byte read/write non-volatile memory and 16+16 bytes of access codes.
- **Stand-alone and LAN-connected PCs:** A complete range of SmartKey models suitable for protecting stand-alone software as well as local-network software is available.
- **Supports a Wide Range of Operating Systems and Languages:** SmartKey can protect programs written in a wide range of languages using different operating environments. They include: MS-DOS, Windows 3.1, Windows 9x, Windows ME, Windows NT, Windows 2000, Windows XP, Linux, Unix, Novell, and AutoCAD.
- **Protection of Executable Software:** GSS technology allows protection of programs in .EXE format, even if the source codes are not available.
- **No External Power Supply:** SmartKey has no internal batteries nor does it require an external supply.
- **No Tools Needed for Installation:** SmartKey has an ergonomic mechanism with inbuilt knurled through screws. This helps connect the key directly to the PC, doing away with need for any external tool for installing SmartKey.

3.4 How Does SmartKey Work?

SmartKey is a secure and flexible software protection key that works when plugged into the computer's parallel port. Each SmartKey has a unique code that can be customized, allowing businesses to implement their own protection systems.

SmartKey offers protection based on making access to the protected software application dependent on the presence of the correct key. The keys can contain information specific to a company, thus ensuring the uniqueness of the keys.

When a protected software application is run, a query is sent to the PC's parallel or USB port where the SmartKey is plugged in. As each key has its unique personalized digital signature, the query first verifies its presence before the software gets executed.

If the response returned by the SmartKey is incorrect, the software application stops running or takes other defensive action. This could result in the software either not loading or switching over to a demo version. On the other hand, if the key is correctly verified, the software proceeds with its execution, and if necessary, carries out additional checks using the key's other stipulated parameters.

A non-volatile internal memory allows implementation of selective protection criteria. It also allows personalization of keys in such a way that the serial number of the software to be protected and the content of the key match.

SmartKey finds use in multiple business scenarios. A case in point is its widespread use in conducting special marketing initiatives like distribution of demo packages, time-lease of software, measuring the use of software, control of subsequent releases, etc.

3.5 SmartKey's Supporting Utilities

SmartKey comes with a complete range of supporting utilities that facilitate a high level of security. These utilities are bundled with the startup kit and include:

- **SPC utility:** It is a basic tool used to initialize and control the key. Its principal function is to help in personalizing the key by programming its Label and Password. It also assists in programming and verifying the key's memory contents.
- **GSS utility:** The Global Security for SmartKey (GSS) is the fastest way to protect software, as it does not require any source variation. It enables automatic protection of executable files. It builds an .EXE file from another executable file that is fully compatible with the first one, but completely encrypted and impossible to start without a key. And all this in just a couple of seconds.
- **SDI utility:** It is a basic tool for installing the SmartKey drivers.
- **SCC utility:** It is a basic tool for configuring the SmartKey lan servers.
- **Interfacing Software Driver:** Driver libraries are available for most programming languages and operating systems. The integration of SmartKey software driver with source files of the programs to be protected offers the most flexible software protection system and enables personalized protection techniques.

3.6 Smartkey's Security Level

SmartKey provides the widest range of solutions and features in software protection, including memory-, network- and time-based solutions for multiple hardware platforms. In addition, it supports a wide range of programming environments and enables software protection on all major operating systems.

Whether one decides to protect applications by using the software drivers or through the automatic utility GSS, SmartKey continues to offer a very high degree of security.

GSS, another supporting utility bundled with SmartKey, besides protecting the software, also encodes using the values of the registers stored as coding keys. When the software treated with GSS is run, it promptly decodes using the register present in SmartKey. The software can never be decoded without SmartKey. GSS offers other benefits as well. It not only enables SmartKey users to protect their applications, but also helps automatically encrypt any data files being written, read or created by the protected software. GSS thus prevents application data files from being illegally duplicated, which proves extremely critical when they contain reserved information such as corporate accounts, lists of customers, project data, technical or commercial databases.

4 Smartkey Models

The need for software protection is varied. As such, software protection requirements range from simple solutions for low-cost software packages to highly sophisticated network applications where security and flexibility are critical. Which is why SmartKey has a number of models, helping businesses meet the entire gamut of software security needs. The favorable performance/cost ratio of all SmartKey models is an added advantage. In fact, models are available for protecting software packages in any price range and application.

As software protection technology scales new heights, so does SmartKey's sophistication. Not only that, all SmartKey models are hardware and software forward compatible. This means an advanced SmartKey model, besides being technologically superior, also emulates the capabilities of previous models, performing all their functions with ease. Thus, when protection requirements change, one can simply update one SmartKey model with another.

SmartKey models are available in parallel port and USB versions, all with identical functionality. Plus, all SmartKey versions are assigned a unique serial number when they are manufactured. This serial number is accessible through the APIs. The difference between the models lies in their performance and the possibility of installing them on a standalone or a LAN-connected personal computer.

- **SmartKey-FX** Fixed: Has fixed algorithms and security codes
- **SmartKey-PR** Programmable: Consists of programmable security codes and a internal memory of 96 bytes
- **SmartKey-EP** Extended Protection: Same as SmartKey-PR except for its extended security performance
- **SmartKey-SP** Super Protection: Same as SmartKey-EP except for its 448 internal bytes and user-programmable security algorithms
- **SmartKey-NET** Network: Same as SmartKey-SP, but for network applications

4.1 FX

SmartKey-FX is the simplest model from the SmartKey stable. It is also the least expensive model. It uses a protection process based on the assignment of a unique and personal internal code referred to as the Identification Code or Id-Code.

Its algorithm-based protection—a reason why it does not provide fixed answers—utilizes encrypted operations relating to the internal code. The Id-Code is used as the main coding parameter; when a block of data is sent, it is duly returned encrypted in a different form for each Id-Code.

The presence of the key is detected by comparing the returned data with the expected data. If they match, the program runs normally. Otherwise, the software's execution stops, helping prevent its unauthorized use.

Salient Features

- FX is derived from the word 'fixed'. This model stands for security and low cost
- Algorithmic protection is based on a unique, personal code (Id-Code)

4.2 PR

SmartKey-PR is ideal for more flexible applications, where every single software package requires personalization. A case in point: program serialization operations.

In addition to having the algorithmic code and data protection process (which is also applied to SmartKey-FX) SmartKey-PR is also programmable. A built-in 64-byte read/write memory area within the key is accessible only by sending two 16-byte access codes named Label and Password. The data stored inside the key are called Secure Data, as the holder of the password alone can read or write them.

The Label and Password codes are software programmable and do not require the assistance of any external programming device. It is thus possible to assign a different key program to every program that needs to be protected. The protected program can dynamically vary the stored data. This helps generate extremely sophisticated applications such as the use of the data themselves as access counter.

SmartKey-PR uses a double-lock protection mechanism: security is ensured both by the unique factory-assigned Id-Code and by the password and data programmed known only to the software house.

Salient Features

- PR is derived from the word ‘programmable’. This model stands for security and versatility
- Algorithmic protection is based on a unique and personal code (Id-Code)
- 16-byte additional programmable security codes (Label and Password)
- 64-byte internal programmable memory (Secure Data)
- Optional restrictions on the number of executions of the protected program

4.3 EP

SmartKey-EP is equipped with more security features than the previously listed SmartKey models. It is recommended for high security applications such as access control to data banks, confidential programs, etc.

Although SmartKey-EP uses the same protection mechanism as SmartKey-PR, it comes loaded with additional security features as mentioned above. For one, it helps detect access attempts with an erroneous password. A built-in counter (fail counter) returns the number of fraudulent access attempts and enables the software house to use this information in ways it sees fit to defend the application. For example, the company can choose to destroy the key's contents or even the confidential data itself after a preset number of attempts.

Another useful feature that this model of SmartKey is equipped with is the option to freeze the Password and Secure Data once they have been programmed. This means that the codes and data built into the key by the software house can be set irreversibly. The key is prevented from being subsequently manipulated for fraudulent purposes, for instance in order to modify access rights to data banks or the operating limits of the software supplied.

Salient Features

- EP is derived from the words ‘extended protection’. This model offers extended security, using many additional protection techniques
- Algorithmic protection based on a unique and personal code (Id-Code)
- 16-byte additional programmable security codes (Label and Password)
- 64-byte internal programmable memory (Secure Data)
- Fraudulent access attempts counter (Fail Counter)
- Freezing of programmed data
- Optional restrictions on the number of executions of the protected program

4.4 SP

SmartKey-SP is the most sophisticated model from the SmartKey family. It is ideal for top-security applications as well as costly software facing serious risk of piracy.

SmartKey-SP has evolved from SmartKey-EP and expands its inner memory (Secure Data) from 64 to 416 bytes. This is in addition to the Label and Password's 32 bytes (16+16 bytes). When added, the SmartKey-SP has a total internal memory of 448 bytes.

But what makes SmartKey-SP really unique is its encryption function. This helps define user-programmable security algorithms. Not only can the software house customize the key through the Secure Data field (extended to 416 bytes), but also program its own personal security algorithm besides the standard one provided by the Id-Code.

Salient Features

- SP is derived from the words ‘super protection’. This model of SmartKey stands for super protection
- Algorithmic protection based on a unique and personal code (Id-Code)
- 16-byte additional programmable security codes (Label and Password)
- 416-byte internal programmable memory (Secure Data)
- Fraudulent access attempts counter (Fail Counter)
- Freezing of programmed data
- User-programmable security algorithms

- Optional restrictions on the number of executions of the protected program

4.5 NET

SmartKey-NET has been especially developed to protect Local Network Applications (LAN). More precisely, it is used for:

- Protecting local network software against illegal duplication
- Monitoring the number of users who can simultaneously use the protected application (site licenses)

To secure applications on a network, a single SmartKey-NET key is first installed on any computer that forms a part of that network. One SmartKey-NET can handle several network users. In fact, the maximum number of users allowed can be directly programmed into the key itself. Also, the number of users who can utilize the same software package (site license) can be defined. All this is possible because of MAP technology, which allows a single SmartKey-NET to protect different applications and set the number of users and executions for each one of them.

Salient Features

- NET is derived from the word 'network'. This model of SmartKey stands for security in network
- Algorithmic protection based on a unique and personal code (Id-Code)
- 16-byte additional programmable security codes (Label and Password)
- 416-byte internal programmable memory (Secure Data)
- Fraudulent access attempt counter (Fail Counter)
- Freezing of programmed data
- User-programmable security algorithms
- LAN software protection via a single protection key installed either on the server or on the client PCs
- Programming the number of simultaneous users who can access the protected software as well as individual

4.6 A Comparative Snapshot of SmartKey Models

To illustrate how SmartKey models differ from each other, we have used an example of different types of safes. Both SmartKeys and safes are designed to provide security. And, just as there are different types of safes, so are there different SmartKey models. With time, as new models were developed, the level of security too kept pace, progressively offering more advanced security solutions for more complex software applications.

As indicated above, by comparing SmartKey models with different types of safes, one can acquire a clear understanding of how the models differ from each other in the way they work.

First, let's consider SmartKey-FX. This model is like a safe that needs a physical key, which when turned, opens the safe. By extension, safes with different owners will have different keys. Similarly, SmartKey-FX models with different users will have different Id-Codes.

The SmartKey-PR model is like a sophisticated safe. In order to open it, one must have a physical key. But, it is also necessary to set a special combination on a special lock (two turns to the left, 3 to the right, etc.). The PR model has an Id-Code (the physical key) and a password (the combination). So, just as the safe has a programmable combination, so can the PR model, with its password being programmable. Only by knowing the password can one access the Secure Data (the contents of the safe or the software application).

The SmartKey-EP model is a variant of the PR model. To use the example of the safe again, the EP model corresponds to a safe with a key (Id-Code) and a combination (password) with one additional security feature. It also has an internal device that counts the number of wrong access attempts (Fail Counter). In addition, a mechanism, which can be implemented when required, prevents anyone from fraudulently changing the set combination.

The SmartKey-SP model comprises of 416 bytes instead of 64 bytes. So, it's like the previous safe, only bigger. In addition, the user-programmable security algorithm is now analog of a special key with profile that can be defined at will.

Finally, the SmartKey-NET model has the same security mechanisms as the SP model, but it is suitable for PC network applications.

Below, a tabular summary of the characteristics of the five SmartKey models has been provided:

SmartKey	PC/LAN	Id-Code	Password	Memoria	Fail-Counter	Algoritmi Progr.
<i>FX</i>	PC	SI	NO	NO	NO	NO
<i>PR</i>	PC	SI	SI	64 bytes	NO	NO
<i>EP</i>	PC	SI	SI	64 bytes	SI	NO
<i>SP</i>	PC	SI	SI	416 bytes	SI	SI
<i>NET</i>	LAN	SI	SI	416 bytes	SI	SI

4.7 Selecting the Right SmartKey Model

Choosing the right SmartKey model involves assessing your software protection need using a whole host of technical and financial factors. What is more, only a case-by-case analysis based on the variables below would yield the most suitable option:

- The cost of the package
- The environment in which the protected software is placed
- The geographical area in which the product is distributed
- The time available for implementing the protection

Some even believe that software protection is like insurance coverage; the policy must relate to the value of the asset. One would do well to remember that protecting software involves the same sort of issues and choices as theft insurance policies. One needs to consider factors like:

- How much should one insure for?
- What additional guarantees must one include in the insurance package?
- What is the probability of a theft?

Given below are some general considerations that may prove helpful in choosing the most suitable SmartKey model:

SmartKey-FX is simple, fast to implement and inexpensive. It is suitable for protecting low-cost, package-type software, which are standardized and distributed in bulk. It must be remembered that the algorithmic protection mechanism in this case is based on a unique factory-assigned code that is different for each customer (Id-Code) and non-modifiable. Therefore, all FX keys of a single customer have the same code.

SmartKey-PR has the best price/performance ratio. This is because it has an internal memory that can be accessed only by using confidential programmable codes in addition to a unique Id-Code. Therefore, the software company can program the key based on the user's specific requirement. Each key can be set to protect a particular program or even a definite block of modules of the same program. A serial number, customer's name or code, data or any other information that may prove useful for the implemented protection mechanism can be written into the key. It is suitable for protecting software which falls within an average cost range as it offers high security at low cost.

SmartKey-EP is suitable for protecting costly software programs. This model counters not only attempts to copy the software, but also attempts to violate the protection mechanism, which could involve breaking into and altering the contents of the key.

SmartKey-SP is ideal for protection patterns that require high data storage capacity within the key. Its user-programmable algorithm makes it suitable for the most sophisticated and top security applications. Using it demands more time and skill than the other simpler models.

SmartKey-NET is especially designed for protecting local network applications. This model can handle several network users. In fact, the number of users can be directly programmed into the key itself.

5 Protecting An Application With Smartkey

To effectively protect your software applications, you must first have a clear protection strategy in place. This section highlights the different options available to a SmartKey user to ward off piracy and hacking attacks.

It is advisable to first design your protection strategy before you actually start adding protection to your source code. This helps identify, right at the outset, critical protection-related issues like “the type of protection to be used”, “the level and complexity of protection required”, and “the time and resource you are willing to devote to protection implementation”. Here, it is worth noting that the effort required to break into a protected application increases progressively with the number and complexity of the locks put in it.

Protecting an application using SmartKey entails implementing Execution Control. The term “Execution Control” means modifying the application in such a way that its execution is subject to the presence of a software protection key.

SmartKey presents the following two ways in which you can protect your application:

- **Manual protection:** This type of protection is executed at the level of the source code of the original program and involves use of software drivers.
- **Automatic protection:** As the name suggests, this type of protection is automatic as it directly acts on the .EXE file of the original program.

5.1 Implementing Manual Protection

When manual protection is implemented using SmartKey, software locks are added directly into the application’s source code. These locks are API functions used for verifying the presence of the key.

As the process is manual, you retain control on the amount and location of the locks. Thus, you can determine the frequency of software locks within your application. That is not all. The action performed if no key is found is also left to your discretion.

The difficulty faced by potential hackers while breaking into your application increases with every lock you add to it. The software drivers also introduce sophisticated anti-debugger devices capable of withstanding even the most powerful tools of analysis.

Manual protection is recommended in view of the flexibility it offers in protecting the software. It also allows you to retain control over the protection techniques used to secure the software applications. But, it must be kept in mind that manual protection is a more hands-on approach and takes longer to implement. It requires involvement of a programmer who understands the API calls used and is capable of manually adding them to the code. Thus access to the source code is another precondition for implementing manual protection.

5.2 Implementing Automatic Protection

Automatic protection offers quick and easy protection solution against unauthorized access. It is also relatively simple to implement. Automatic protection facilitated by SmartKey uses Global Security System (GSS), a proprietary technology which wraps a protective layer around the application, preventing it from working unless the appropriate protection key is present.

Use of GSS technology ensures that modification of a software application and access to its source code is not required. It encrypts the original software program. Thus the software runs only when the correct SmartKey is plugged in to decrypt it. Encryption makes access to the software’s source code extremely difficult even for seasoned hackers.

Understanding a GSS-encoded .EXE file is impossible. All ASCII messages in the original .EXE file transforms into a sequence of indecipherable characters. Although the application’s source code is left untouched. The protective layer handles all software locks and communication with the hardware key such as checking and verification.

Preference for automatic protection becomes elevated when access to the application’s source code is not available. It is also preferred when one lacks the time or desire to design a unique protection scheme using the manual protection technique. Another crucial benefit for which automatic protection is used is the facility to include time/date/execution controls for a demo application, as all this can be done without having to manually alter the code.

5.3 Manual Versus Automatic Protection

Which of the two protection types is better, manual or automatic? There is no definitive answer to this question. This is because a number of factors come into play while deciding the protection type that’s best suited for a given protection

need. Factors ranging from availability of qualified resource and time constraints to the need to implement customized or standardized protection, all have a bearing on the protection type one chooses.

To take an informed decision regarding the protection type that best suits your need, follow the metrics provided below:

<i>Type of protection</i>	<i>Necessary Time</i>	<i>Programming knowledge required?</i>	<i>Software sources required?</i>	<i>Security</i>
Manual	Hours	yes	yes	very high
Automatic	minutes	no	no	high

Now, let's discuss in detail the factors involved in choosing the right software security. To start with, you need to decide whether you want to implement quick and easy protection, or the more time-consuming customized protection. This would give you a fair idea about the protection type best suited for you.

In case customized protection is what you are looking at, you would need to implement the protection manually. This protection type allows you to:

- Choose how to use each memory cell in the key
- Select the algorithms used in your protection strategy
- Add data words or counters where appropriate
- Add API function calls to your source code to support your protection strategy
- Create a very secure level of protection for your application

If you are short on time and need to quickly protect your software, you may want to consider using automatic protection. This would allow you to:

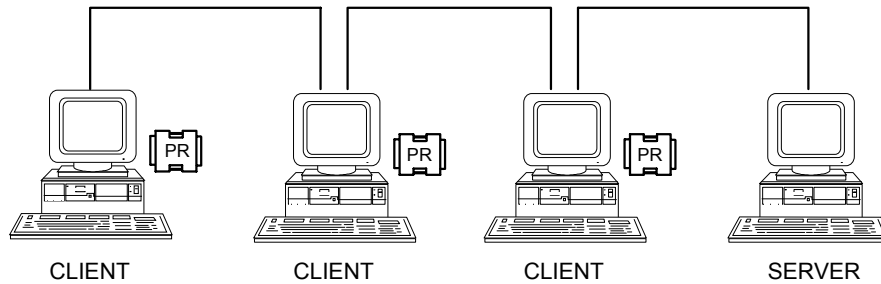
- Implement automatic protection for your application in 30 minutes or less
- Protect applications in .EXE format
- Fix time and date controls and algorithm values
- Implement protection without modifying the application's source code

6 Local Network Protection

When several computers are connected in local network, the network software can be protected in either of two ways:

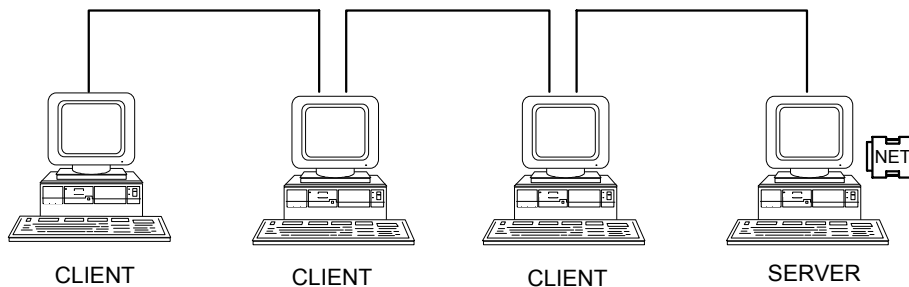
- by inserting a stand-alone type protection key (*FX*, *PR*, *EP* or *SP*) on each of the computers enabled to execute the application. In this case there is no need to modify protected software in stand-alone mode;.

LAN



- by inserting a single NET type protection key with its software drivers

LAN



SmartKey-NET is an extension of the SP model, with which it shares all the main features in addition to a few characteristics which enable protection using a single key installed on any of the computers in the network, or on a non-dedicated server.

The software supplied along with the key enables each computer in the network to interrogate the single SmartKey-NET. It therefore becomes unnecessary to use as many keys as the networked workstations.

By means of the MAP (MultiApplication Protection) proprietary technology, SmartKey-NET also makes it possible:

- to protect a number of different networked applications (up to 116);
- for each protected application, to limit the maximum number of users who may use the application at the same time, so as to keep a check on site licenses; e.g., PROG1 may be enabled for 12 licenses, PROG2 for 27, PROG3 for an unlimited number, etc.;
- to limit the number of executions of each of the protected programs. This feature may prove useful if one wants to create demo versions of the software, or adopt a software leasing policy whereby the user is allowed a preset number of executions.

6.1 The automatic protection in local networks

Besides the functioning applications in stand-alone mode, the GSS is capable of protecting also applications in network.

The use of the protected program is absolutely transparent to the final user, who can use either a local key or network key : GSS will initially research the key on local ports and, if the research were to fail, it will continue trying to communicate throughou the network key.

6.2 The manual protection in local networks

The great diffusion of local networks requires a simple and intuitive approach for interfacing with the protection device. SmartKey follows the **MultiLan** technology which allows software developers to protect applications with a single driver, regardless of the operating environment, whether stand-alone or LAN.

- MultiLan is a single driver for both stand-alone and LAN applications.
- MultiLan allows one to implement LAN protection both in real mode and in protected mode.
- MultiLan identifies the type of network automatically

The end user is required only to install SmartKey-NET on the server if a Novell network is involved, or on any client PC if the network is non-Novell. If the application runs on a stand-alone, the driver seeks the device (SmartKey-FX, PR, EP or SP) on the parallel port of the local PC automatically.

6.3 Protection of several applications with SmartKey

When running in a LAN environment, a single SmartKey-NET key can be used for protecting several applications. The technology used is called MAP - MultiApplication Protection - and allows one:

- To protect several applications running in stand-alone or network In the case of local network it is also possible to define a different number of licenses for each application protected.
- To limit the number of executions of each of the protected programs. This option can prove useful if one needs to create demo versions of the software or intends to lease it out. When the number of executions preset in the counter (and decremented at each program startup) runs out, the program is inhibited from starting.

7 The Internal Structure Of Smartkey

The structure of **SmartKey** is based on a number of internal registers, each with a specific protection function:

- Id-Code register
- Label register (16 bytes)
- Password register (16 bytes)
- Secure Data register (64 / 416 bytes)
- Fail Counter register (2 bytes)

7.1 Id-Code register : the personal code

The Id-Code is a 4-byte register, factory-programmed at the time of testing every single key and no longer modifiable after that. Every **SmartKey**'s customers has a different Id-Code and the total of possible codes is 2^{32} (equal to approx. 4,000,000,000).

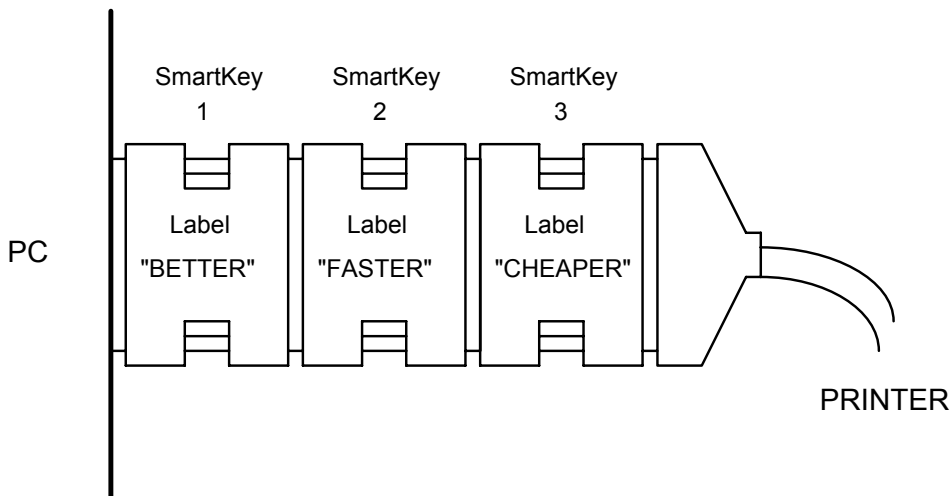
The Id-Code register, which is present in all **SmartKey** models for parallel port, makes sure that keys of different customers are really different from each other. In fact, it assigns a personal and unique code to the holder of a protection key. Your own Id-Code is shown on the front page of this manual.

For the sake of security the Identification Code entered in the Id-Code register cannot be read directly, but its value affects the result of the algorithmic interrogation of the key. Different users have different Id-Codes and their keys therefore give different replies to the algorithm questioning by the protected software

7.2 Label register : the identification and access tag

The Label register contains one of the key's two access codes and serves to identify the correct key for the particular software being run. In fact the Label is a sort of electronic identification tag inside the key.

The function of the Label is of particular importance when the **SmartKey** keys are daisy-chained to the same parallel port. In this case the Label is a sort of address which allows the protected software to query the correct key. In order to make sure that the sought for key is present, the protected software sends the value of the Label to the parallel port; only the key with the matching label will return a response.



An example is provided above where three **SmartKey** keys are daisy-chained on the same parallel port. They are programmed with different Labels and each key protects a different program ("BETTER", "FASTER", "CHEAPER").

It is therefore important to assign a different label for each of one's applications so that several **SmartKey** keys can be installed at the same time.

The size of the Label register is 16 bytes (2^{128} combinations, equivalent to $3 * 10^{38}$); therefore since the number of combination is huge, it is practically impossible that two different software houses should decide to assign the same label to their applications).

With programmable type keys (**PR**, **EP**, **SP** and **NET**), the Label can be programmed off-line using the *SmartPrg* utility, selecting the PROGRAMMING mode, or on-line via the software drivers installed.

With **FX** keys the Label register is fixed and coincides with the Id-Code

FX	Non programmable Label	Id-Code
PR EP SP NET	Programmable Label	16 bytes

When trying out the *SmartKey*, one might happen to forget the value of the Label programmed. In this case it is no longer possible to access the contents of key.

7.3 Password register : the key to access data

The Password register is highly important in the protection mechanism as only by knowing how it has been programmed can one access data contained in the key's nonvolatile memory. Like the combination of a safe, a knowledge of the correct password allows one to turn the key and access the contents.

The password register, whose size is 16 byte, may be programmed off-line by the *SmartPrg* utility, selecting the PROGRAMMING

It is never possible to reread directly the contents of the Password set in the key; access to the Password is write only during the programming phase.

It is also possible to reprogram the Password without knowing the previous one; in this case, however, the contents of the data memory are reset automatically.

7.4 Secure Data register : the internal secure data

The Secure Data register is equipped with 64 bytes (**PR** and **EP** keys) or 416 bytes (**SP** and **NET** key) of non-volatile memory inside the key, which can be accessed only by knowing the password. By verifying the contents of the register it is possible to find out whether an attempt to break in was made or legitimate software installation.

The action of the Secure Data in the protection mechanism takes place in different ways, but mainly through a comparison between the expected contents and those actually read by the key: the result of the comparison makes it possible to decide whether to go on executing the program or not.

With **SP** and **NET** keys, the register can also be used for storing the user-programmable security algorithm. Finally, with the **NET** key, some of the bytes are used to define the highest number of simultaneous users of the protected software (site license) and even the limitation of the number of the execution.

For reasons of compatibility with 64-byte models, the Secure Data register of the **SP** and **NET** keys actually consists of two separate registers: a 64-byte basic one and a 352-byte extended one

PR EP SP NET	Secure Data Base	64 bytes
SP NET	Expanded Data Base	352 bytes

The Secure Data may be read or written by software provided the correct password has been transferred. If the password was wrong, the read operations do not return the contents of the secure Data but a pseudo-random series of bits; also write attempts using a wrong password are ineffectual so that the valid contents of data programmed are not altered.

7.5 Fail Counter register : wrong access alarm

The Password Fail Counter Register, available only in the most highly developed versions **EP**, **SP** and **NET**, provides automatic counting of the attempts to access the key with a wrong password.

Each time a read or write access attempt is made using a wrong Password, the contents of the register are automatically incremented by 1.

It is thus a self-incrementing read-only counter, which allows one to identify attempts to break in by seeking the password. For security reasons, the counter is not reset by any of the key's write or program functions.

The counting range is from 0 to 10 000. The register is read in reading mode after learning the password. If the password is wrong, the number returned is generated randomly.

The software to be protected may use the register to check any attempts at illicit access. If the key is used, for instance, to allow access to data banks or confidential data, the software may disable it from working permanently by altering the contents of the Secure Data after a set number of wrong accesses, even if they are not consecutive.

8 Automatic Protection

SmartKey offers a quick and easy protection implementation technique commonly referred to as automatic protection. Based on Global Security System (GSS), a proprietary technology, automatic protection helps shield applications in .EXE format from unauthorized access. It also works on applications running with internal and external overlays as well as those running on protected mode.

The most noteworthy aspect about automatic protection is that it can be implemented without modifying the source code of the application. From the original .EXE standard file, the GSS technology creates a new .EXE file, which works only when SmartKey is connected to the computer.

SmartKey's automatic protection works at two levels. First, the key's presence is essential to use the application. Second, the application is encrypted to prevent debugging.

Besides the .EXE files, encryption can be carried out on data files managed by the application software. This ensures a high overall security standard.

Thanks to the GSS technology, SmartKey is capable of offering a couple of other benefits. It can be effectively used to limit the number of executions when programs are distributed for demonstration purposes. Once the set limit is breached, the software gets de-activated. Protecting several applications on a LAN environment is also now easy with SmartKey. It is also possible to set the maximum number of licenses for the protected application.

All SmartKey models are based on the proprietary GSS technology. If SmartKey-FX is used, the protection is based on the Identification Code. In case a programmable key is used, other protection elements like Password and Secure Data too help safeguard the application against unauthorized access.

As we have seen above, both SmartKey and the GSS software combine to offer several robust security functions. They have been summarized below:

- Automatic protection of .EXE files
- Regular checking of the key's presence
- Protection of the application against alteration of the object code through debugging
- Optional encryption of data files associated to a protected program
- Optional maintaining the CRC of encrypted file on the key.
- Protection of different applications with the same key

The screenshot shows the GSS configuration window with the following settings:

- Target environment:** Windows 32 (Unicode)
- Protection dependents:** Label, Id Code, Password, Data
- Key parameters:** Label: SMARTKEY, Password: EUTRON, Hex Format: unchecked
- Protection options:** Crc on file, Crc on key, Map, Periodic
- Files:** Executable file, Icon file, Data files (one for line)
- Destination directory:** dest_dir
- Messages:** No key, Program corrupted, No licenses, No executions

8.1 Automatic Protection With GSS

Protection offered by SmartKey is powered by the GSS technology, which comes packed in the GSS.EXE utility supplied with the SmartKey Kit. Some of the benefits that this utility offers have been outlined below:

Selects the set of files to be protected

- Saves the original file before protecting it
- Activates the protection dependent on Id-Code, Password and Secure Data
- Activates the periodic control of key
- Activates the maintenance of CRC on key
- Chose the message to be displayed
- Executes the protection of the file

8.2 Protection Across All Platforms Using GSS

The Global Security System is platform independent, acting as a common protection instrument for all platforms. Thus you can protect executable files optimized for platforms ranging from DOS, Windows 9x, Windows ME, Windows NT, Windows 2000 & Windows XP.

GSS comes equipped with a special mechanism that allows independence from an operating system. This mechanism is based on the use of an appropriate run-time module, making it mandatory that the directory of the application contains the run-time module that corresponds to the hosting operating system.

GSS is also available in the GSSLINE command line form. This too makes use of the same mechanism as the run-time module of GSS.

8.2.1 Protecting Windows Programs

To run protected Windows programs, presence of run-time modules MODW9X.EXE and MODWNT.EXE and the device GSSDRV10.VXD is necessary in the same directory as the executable. Under Windows NT, Windows 2000 and Windows XP, it's also necessary to install the GSS device driver specific for these operation systems. To correctly install the device driver, the user has to have administration rights.

The installation of the SmartKey and GSS device drivers can be done manually with the SDI utility, or automatically integrating the SDI library with your installation procedure.

For detailed and updated information on the use of the product please refer to the README.TXT file contained in the developer kit.

8.2.2 Protecting DOS Programs

SmartKey, backed by the GSS technology, is capable of protecting DOS programs for both DOS and Windows 9x. While implementing protection, you need to select the target operating system on which the program will be run. Select "DOS" for a pure DOS environment or "Windows 16/32" for a Windows 9x environment. The protection of DOS applications is not supported in Windows NT, Windows 2000 and Windows XP.

To carry out the program on the desired platform it is sufficient to just rename the file in the extension .EXE. Therefore, in the case of DOS programs, the GSS run-time module is not required.

8.3 GSS: The Common Options

The various versions of GSS have an identical user interface. It helps safeguard the integrity of the protected application's object code. It proves useful when a hacker, in an attempt to alter a part of the code or the numerical parameters or ASCII strings, fraudulently changes the object code.

The smallest of changes, even if it is by only one bit, is detected by GSS, which displays a warning message on startup and immediately stops the program. GSS also provides the option to enter a text to be shown in case the key is missing or the program file is corrupted.

8.3.1 Checking the Key's Presence

SmartKey users can send frequent queries to keep a tab on the key's presence by using the Periodic Check option. This assures users of the key's presence during the complete execution of the protected program.

Used generally as a protection mechanism against unauthorized access by skilled hackers who find it relatively easy to isolate the single call made at the outset, the Periodic Check option also prevents users from removing the key after the application starts. In its absence, the protected application would continue to run as no queries are being performed to check for the key's continued presence.

On the flip side, though, periodic check of the key's presence instead of a continuous check helps run multiple copies of the same program. This is because in the absence of a continuous check you can first run the protected program and then remove the key for the given period to run copies of the same application.

It must be kept in mind that this facility is not uniform across all operating platforms. In the DOS environment, the protected application stops running until the key is reinserted. In the Windows environment, though, the key's removal temporarily stops not only the protected software but also all active applications.

8.3.2 Programming Error Messages

SmartKey allows users to customize error messages that are issued under a given set of conditions. By using this feature, you can define error messages based on how you would like the users to respond. The conditions for which error messages can be programmed are:

- SmartKey is not present
- The program code has been altered

The list of messages that can be set depends either on the model of SmartKey or on the Protection Configuration character. The GSS utility too suggests some default messages for each error situation listed above. There is also a facility to add new messages as well as modify them.

8.3.3 Encrypting the Executable Code

The GSS technology on which SmartKey is based completely encrypts original files by default. This helps ward off hacking attempts, as it is virtually impossible to disassemble the original application using its encrypted .EXE file.

Any protection strategy devoid of encryption technique results in a low security level. Which is why applications are sent to the field with encrypted code. The relevant decryption can only be run if the hardware key is attached. Most encryption algorithms depend on a key value to transform the data.

Speaking specifically about Windows based programs, encryption helps avoid illicit exporting and copying of its resources. This is necessary as there is wide availability of Windows based programs that allow you to extract its resources (icons, cursors, dialogues, menus, bitmaps, toolbars, etc.) as well as copy and reuse them.

8.3.4 Parameter Based Protection

SmartKey users get to use multiple parameters based protection apart from the protection offered by encryption. This ensures that the protected program runs only when the key used is the same as the one originally used while creating the file. This is because each key has its own unique set of parameters that include the Id-Code, Label, Password and Secure Data.

However, parameter based protection is optional as it can easily be deactivated except for the Label. Use of the Label is mandatory. When a protected program runs, and if the optional key parameters are deactivated, only the Label is checked leaving out the other parameters. On a successful match of only the Label, the program is executed successfully.

8.3.5 Message Displayed When the Key is Missing

If the protected program is run without the presence of the relevant SmartKey, the program stops and the following default message is displayed: **No Key**

The message provided above is not fixed. As with messages associated with other features, the GSS.EXE utility also allows you to replace the standard message with a personalized one.

8.3.6 Limiting the Number of Executions and Licenses

Use of SmartKey is not limited to just protecting applications against unauthorized access. It also helps set the number of executions and licenses allowed for the protected application. This feature proves useful when releasing demo versions of the application. This gives users the option to allow only limited use of the application. On expiry of the number of preset executions, the application ceases to run.

On the other hand, the facility to limit the number of licenses ensures the application's use is in accordance with the licensing policy. This feature is available with the SmartKey-NET version.

8.3.7 Encrypting Executable Codes Without Access to the Key

No such demo mode option. Users can avail of this benefit by disabling dependence on the ID-Code using GSS.EXE. When this is done, all other options are ignored like dependence on Password, Regular Check, etc.

8.3.8 Automatic Encryption of Data Files

To completely secure an application from unauthorized access, you need to not only protect the executable application but also all relevant data/databases (.DBF, .DAT, etc.) associated with it. Thanks to the GSS technology, SmartKey now allows you to do exactly that.

When a data string is sent to the key, a preprogrammed algorithm is used to encrypt it. The GSS utility is primarily used for this purpose. At run time, your protected application automatically decrypts the encrypted files. When not in use, these files are re-encrypted.

8.3.9 Protecting Applications on Network

In addition to protecting applications that run in stand-alone mode, SmartKey also secures applications on Local Area Network (LAN). As the use of a protected application is absolutely transparent to end user, they can use either a local key or a network key. The GSS technology first looks for the device on the local ports. If its search is unsuccessful, it starts communicating with a LAN device.

SmartKey-NET has been developed especially for this purpose. The software supplied along with this key helps computers in a network to send queries to a single SmartKey-NET. Therefore, it becomes unnecessary to use as many keys as the networked workstations.

8.3.10 Protecting Executable Files in Batches

Sometimes, you might need to automatically protect several executable files in one go. In such situations, the GSSLINE.EXE, which is the DOS command line version of the GSS.EXE utility, would prove useful. As it can call several executable files into a batch, the GSSLINE.EXE helps implement automatic protection for the entire batch.

The syntax for the execution of GSSLINE is:

```
gssline CFG_FILE EXE_FILE [DATA_FILE...] DESTINATION_DIR
```

CFG_FILE	is the configuration file
EXE_FILE or DATA_FILE	is the name of the .EXE or data file to be protected
DESTINATION_DIR	is path where the final protected .EXE file will reside.

While entering the command line, do keep the following in mind:

- The path of the protected file (DESTINATION_DIR) to be produced by GSSLINE must be different from the path where original file(s) (EXE_FILE|DATA_FILE) is placed.
- Indicate clearly the extension of each file (.CFG and .EXE)

8.4 Quick Takeaways on Effective Application Protection

So far, this chapter has covered automatic software protection in detail. Here, you will find a wrap up of the key steps you need to take to quickly implement protection for your application.

- Choose the Target Environment depending on your application
- Activate the protection dependent on Id-Code, Password and Secure Data
- Insert Label and Password of your SmartKey
- Select the Protection Options: CRC on File, CRC on Key, Periodic
- Check the SmartKey; if you are using SmartKey-NET, you can activate MAP protection to control licenses and executions of your application
- Choose the Application file
- Choose an Icon file, if present
- Choose possible file data related with your application

- Select the destination where you want to save the protected file; it would be advisable to change the folder to avoid overwriting the original file
- Change the Error Message (if you so desire) depending on the error situation - No Key, Bad CRC, No License, and No Executions

9 Manual Protection

The present chapter proposes to supply an introduction to the techniques to manual protection of software with SmartKey. For details on procedures and examples of the use of software modes available, we advice you to make reference to the Programmer's Manual.

Manual protection is based on the use of software drivers, which make it possible to activate *SmartKey*'s different operating modes.

The program to be protected is modified through appropriate calls to the software drivers and recompiled. The security functions that deal with key management are built into the link phase.

The software drivers can interpret a range of commands, each of them implementing one of the following operative protection modes:

- Locating mode: identifies if and where the key is present on one of the parallel or USB ports.
- Scrambling mode: it verifies algorithmically whether the Id-Code is correct.
- Reading mode: it reads the Secure Data.
- Block Reading mode: it reads the Secure Data.
- Writing mode: it writes the Secure Data.
- Block Writing mode: it writes the Secure Data.
- Fixing mode: it fixes the contents of the key in such a way that it can no longer be modified.
- Encrypting mode: it activates the user-programmed coding algorithm.
- Programming mode: it reprograms the contents of the key.

9.1 Locating mode

A personal computer can have up to 3 parallel ports (LPT1, LPT2, LPT3) and the same number of USB ports. A protection key can be plugged into any of them.

The LOCATING mode **automatically seeks the *SmartKey* with a present Label on all the parallel or USB ports of the system** allowing the protected software to identify the parallel door on which the key is located.

The result of the LOCATING operation is the identifier of the parallel or USB port (Lpt parameter) on which the key is installed. The identifier may be used for subsequent read / write operations on the key. It can take on a value of either 1, 2 or 3 for the parallel ports or 101,102 or 103 for USB ports.

For example caring out a LOCATING call on a PC that has only one parallel port positioned on the 378 hex address, the presence of SmartKey will be shown by the Lpt parameter equal to 2.

We'd like to point out that the Label is not programmable with *FX* keys and coincides with the Id-Code, while it can be programmed with the programmable keys with any sequence of 16 bytes. In both cases the LOCATING mode is fully operative.

The exchange of information is organized as follows:

Models	ALL	
Input	COMMAND LABEL	'L' Label
Output	LPT STATUS	Port Status ==0 SmartKey Found !=0 SmartKey Not Found

We warmly recommend the use of the LOCATING function even if the users' PC is expected to be fitted with just one parallel port.

Use the LOCATING function makes the protected software independent of the parallel or USB port on which the user installs the key.

The Lpt port field is automatically filled with the appropriate parallel or USB port identifier on which the key is present and the programmer thus avoids direct assigning via an instruction into the software.

9.1.1 Parameter passing

Parameters to transfer to carry out a LOCATING operation, with a research on all parallel or USB ports, of a key having the “SMARTKEY” label.

COMMAND	4C 00	Locating ("L")
LABEL	53 4D 41 52 54 4B 45 59 00 00 00 00 00 00 00 00	Label ("SMARTKEY")

If a key is present on the parallel or USB port containing the passed Label, the LPT field will count its logical value (1, 2 or 3 for parallel or 101,102 or 103 for USB).

9.2 Scrambling mode

The SCRAMBLING mode ("encoding"), supported by all the models of *SmartKey* keys, is based on the unique customization for each Customer of the Id-Code register. **Its function is to identify the correctness of the Id-Code algorithmically.**

The Identification Code contained in the Id-Code register is used as the main parameter of a mathematical encoding function (SCRAMBLING): a block of data is sent, processed and returned after encryption in unique way for each Customer. Testing the presence of the key is carried out by comparing the data returned with those expected.

Keys with different Id-Codes have been encoded differently, so for equal data sent to the key, different data will be returned. To every different Id-Code one can thus associate a table of correspondence between data sent to the key (Original Data) and data returned (Scrambled Data).

The algorithm used by *SmartKey* carries out scrambling on 64 bits at a time, therefore the table provides 8 coded bytes every 8 original data bytes passed each time.

The algorithm used is a high-security, non-linear type algorithm.

The exchange of information is organized as follows:

Models	ALL	
Input	COMMAND	'S'
	LPT	Port
	LABEL	Label
	DATA[0..7]	Original data (8 bytes)
Output	DATA[0..7]	Scrambled data (8 bytes)
	STATUS	Status
		==0 Success !=0 Error

If a LOCATING operation has been previously carried out, the correct value is automatically assigned to the LPT filed; it is therefore not necessary that the programmer worries on the assignment of a value.

The SCRAMBLING function does not alter the contents of Secure Data as a supportive variable for the exchange of Original and Scramble Data.

9.2.1 Parameter passing

Parameters to be transferred to carry out a SCRAMBLING operation, with a key research on the LPT 1 parallel port

LPT	01 00	Port
COMMAND	53 00	Scrambling ("S")
LABEL	53 4D 41 52 54 4B 45 59	Label ("SMARTKEY")

	00 00 00 00 00 00 00 00	
DATA	XX XX XX XX XX XX XX XX	Data to scramble (8 bytes)

At the end of the operation the first 8 bytes of the DATA field are substituted with the Scrambled Data that depend of the Original Data sent as well as of the Id-Code of the key.

In this case an initializing of the LPT field is necessary in function of the port itself. Such operation can be avoided by using the LOCATING command for the automatic research on all the ports inserted on the calculator.

9.3 Reading mode

The programmable models of the *SmartKey* family (**PR**, **EP**, **SP**, **NET**), in addition to the SCRAMBLING mechanism, are equipped with a protection mechanism based on the selective read/write access to the Secure Data register with software programmable password.

Every software house can code personally the keys with the help of the software utilities supplied and without any need for external programming devices. The management of the access codes and of the non volatile memory contents is handled directly by the software house, which becomes the only repository of the identification code.

As regards to reading, the READING function allows one to access the Secure Data to verify their contents and compare them with the expected ones. The Label and the Password must both be known. With **EP** keys also the value of the FAIL COUNTER register is available for READING

The exchange of information is organized as follows:

Models	PR, EP, SP, NET	
Input	COMMAND LPT LABEL PASSWORD	'R' Port Label Password
Output	DATA EXP_DATA FAIL_CTR STATUS	Data read Exp data read Fail Counter Status ==0 Success !=0 Error

With the key present but a wrong password, DATA and FAIL_CTR will contain fortuitously generated values.

9.3.1 Parameter passing

Parameters to transfer to carry out a READING operation on a key present on the LPT1 parallel port:

LPT	01 00	Port
COMMAND	52 00	Reading ("R")
LABEL	53 4D 41 52 54 4B 45 59 00 00 00 00 00 00 00 00	Label ("SMARTKEY")
PASSWORD	45 55 54 52 4F 4E 00 00 00 00 00 00 00 00 00 00	Password("EUTRON")
DATA		not used
FAIL_CTR		not used
STATUS		not used
EXP_DATA		not used

If the key is found on the indicated parallel port with the Label values and the correct Passwords the DATA field will contain the first 64 bytes of the Secure Data read and in the case of the **SP** and **NET** key, the EXP DATA field will contain the remaining 352 bytes.

9.4 Block Reading mode

The BLOCK READING function allows one to read portions of Secure Data, e.g. one, two or a few words instead of the complete field. This allows you to save a few fractions of a second with respect to the complete reading in READING MODE.

The exchange of information is organized as follows.

Models	PR, EP, SP, NET	
Input	COMMAND	'BR'
	LPT	Port
	LABEL	Label
	PASSWORD	Password
	DATA[0,1]	Pointer at the first word to read (from 0 to 31 for PR+ and EP+, form 0 to 207 for SP+ and NET+) (2 bytes)
	DATA[2,3]	Number of words to read (from 1 to 16) (2 bytes)
Output	DATA[4,...]	Values to read in the indicated area by the previous two parameters. (2 - 32 bytes)
	STATUS	Status ==0 Success !=0 Error

If the Password passed to the key is not correct, the Secure Data and the Fail Counter are generated in a pseudofortitious manner

9.4.1 Parameter passing

Parameters to transfer to carry out an BLOCK READING operation on a key present on the LPT1 parallel port; the lecture of 15 words is required (000F hex equal to 30 bytes) as from the twelfth word (address 11 = 000B hex).

We remind that in the structure of parameter passing the first two bytes of the DATA field are reserved to the address of the first word to be read, the following two bytes contain a number of words to be read, as from the fifth byte onwards there will be contained the data block read at the end of the operation.

LPT	01 00	Port
COMMAND	52 42	Block Reading ("BR")
LABEL	53 4D 41 52 54 4B 45 59 00 00 00 00 00 00 00 00	Label ("SMARTKEY")
PASSWORD	45 55 54 52 4F 4E 00 00 00 00 00 00 00 00 00 00	Password("EUTRON")
DATA	0B 00 0F 00	Address and Number of Words

9.5 Writing mode

If you know the Password, Secure Data writing can be carried out in the same way as described in the previous paragraph for reading. The WRITING function modifies the contents of the Secure Data on line.

The exchange of information is organized as follows:

Models	PR, EP, SP, NET	
Input	COMMAND	'W'
	LPT	Port

	LABEL	Label
	PASSWORD	Password
	DATA	Data to write
	EXP_DATA	Exp data to write (only for SP and NET)
Output	STATUS	Status ==0 Success !=0 Error

If the Password passed on to the key is not correct, **the present data in the Secure Data register will not be altered.**

9.5.1 Parameter passing

Parameters to transfer to carry out a WRITING operation on a key inserted into LPT1 parallel port:

LPT	01 00	Port
COMMAND	57 00	Reading ("W")
LABEL	53 4D 41 52 54 4B 45 59 00 00 00 00 00 00 00 00	Label ("SMARTKEY")
PASSWORD	45 55 54 52 4F 4E 00 00 00 00 00 00 00 00 00 00	Password ("EUTRON")
DATA	XX XX XX XX XX XX XX XX ...	Data
EXP_DATA	XX XX XX XX XX XX XX XX ...	Exp Data

If the key is found on the indicated port and with the correct Label and Password values, the DATA field (and eventually the EXP.DATA field for the **SP** and **NET** key) will be transferred in the Secure Data of the key.

With the key present but a wrong Password, the Secure Data will not be modified.

9.6 Block Writing mode

The BLOCK WRITING function allows writing of portions of Secure Data, e.g. one, two or a few words instead of the complete field. This allows you to save a few fractions of a second with respect to the complete writing in WRITING MODE.

The exchange of information is organized as follows:

Models	PR, EP, SP, NET	
Input	COMMAND	'BW'
	LPT	Port
	LABEL	Label
	PASSWORD	Password
	DATA[0,1]	Pointer at the first word to be written (from 0 to 31 for PR+ and EP+, from 0 to 207 for SP+ and NET+) (2 bytes)
	DATA[2,3]	Number of words to be written (from 1 to 16) (2 bytes)
	DATA[4,...]	Values to be written in the area indicated by the two previous parameters (2 - 32 bytes)
Output	STATUS	Status ==0 Success !=0 Error

If the Password passed on to the key is not correct, the present data in the Secure Data register will not be altered.

9.6.1 Passing parameters

Parameters to transfer to carry out a BLOCK WRITING operation on a key present on the LPT1 parallel port, the writing of 10 words is required (000A hex equal to a 20 bytes) as from the first word (address 0 = 0000 hex).

We remind that within the structure of parameter passing the first two bytes of the DATA field are reserved to the address of the first word to be read, the following two bytes contain a number of words to be read, as from the fifth byte onwards there will be contained the data block read at the end of the operation.

LPT	01 00	Port
COMMAND	57 42	Block Reading ("BW")
LABEL	53 4D 41 52 54 4B 45 59 00 00 00 00 00 00 00 00	Label ("SMARTKEY")
PASSWORD	45 55 54 52 4F 4E 00 00 00 00 00 00 00 00 00 00	Password("EUTRON")
DATA	0B 00 0F 00 XX	Address, Number of Words, and Data

9.7 Fixing mode

Data fixing means that the Label, Password and Secure Data registers can not be reprogrammed, i.e., it is possible to freeze data input earlier. After executing the FIXING mode it is no longer possible to change in any way the contents of the **SmartKey**. The data fixing mode is available together with the Fail Counter Register only on models **EP**, **SP** and **NET**.

Data fixing enables the software-house to generate fully personalized and unchangeable keys, foiling any attempts to alter the contents for the purpose of modifying, for instance, data bank access priority or enabling unscheduled software modules.

We advice that the fixing operation be not carried out during the testing phase of the first key supplied with the Startup-Kit, because otherwise you will not be able to reprogram the key.

We therefore advice that only in the final phase of definite implementation of the protection should the opportunity of fixing the contents programmed in the key be evaluated or not, keeping in mind that the additional functionality's that involve the memory of the key will not be available.

The exchange of information is organized as follows:

Models	EP, SP, NET	
Input	COMMAND LPT LABEL PASSWORD DATA EXP_DATA	'F' Port Label Password Data Exp Data
Output	STATUS	Status ==0 Success !=0 Error

The fixing of the memory is **only carried out if the sent Label, Password and Secure Data coincide with the content of the respective registers.**

9.7.1 Passing parameters

Parameters to transfer to carry out a FIXING operation with a key present on the LPT1 parallel port, with Label, Password and Data sending and comparison with the contents expected (respectively "SMARTKEY", "EUTRON" "SECURITY DATA"). If the parameters coincide, they will be fixed.

LPT	01 00	Port
COMMAND	46 00	Fixing Mode("F")
LABEL	53 4D 41 52 54 4B 45 59 00 00 00 00 00 00 00 00	Label ("SMARTKEY")
PASSWORD	45 55 54 52 4F 4E 00 00 00 00 00 00 00 00 00 00	Password ("EUTRON")
DATA	53 45 43 55 52 49 54 59 20 44 41 54 41 00	Data ("SECURITY DATA")
EXP_DATA	00 00 00 00 00 00 00 00 ...	Exp Data

9.8 Encrypting mode

The **SmartKey** key security functions are incremented in the most advanced **SP** and **NET** models through the ENCRYPTING modes. This is an algorithmic interrogation mode conceptually similar to the SCRAMBLING operation, but in which the security algorithm is not fixed but programmable.

The user is enabled to define his own security algorithms, using a series of logic operators. Just like in SCRAMBLING mode, the algorithm operates on the first 8 bytes of the Secure Data (original data) field and the result of the processing is again a set of 8 bytes placed on top of the original data (Encrypted Data).

Unlike the SCRAMBLING mode, the algorithm is programmed into the key through **SmartKey**'s ENCRYPTING function and fills the Expanded Secure Data register in reverse order starting from the last byte in the field.

The ENCRYPTING mode is without doubt the most versatile and secure of **SmartKey** keys. It is however also its most complex function and requires a good knowledge of the logic operators. We therefore suggest that you firstly start off by implementing the other modes which assure top security. Consider the ENCRYPTING function as a shield to be used in situations where danger of attack by high-tech hackers is high.

Using the ENCRYPTING function consists in comparing the Encrypted Data obtained by the key with those calculated inside the software applying the algorithm to the Original Data.

If the comparison is successful, you can be sure that your key is connected with the PC and therefore the program can continue to run.

The ENCRYPTING function does not alter the contents of the Secure Data, but uses the Data software parameter as a leaning variable for the exchange of Original and Encrypted Data.

The exchange of information is organized as follows:

Models	SP, NET	
Input	COMMAND	'E'
	LPT	Port
	LABEL	Label
	PASSWORD	Password

	DATA[0..7]	Original Data (8 bytes)
Output	DATA[0..7] STATUS	Encrypted Data (8 bytes) Status ==0 Success !=0 Error

The memorization of the algorithm takes place backwards in the Expanded Secure Data field starting from the last byte in the field itself. It is opportune to check its occupation to avoid the accidental over writing during the WRITING operation.

9.8.1 Passing parameters

Parameters to transfer to carry out an ENCRYPTING operation, with the key on the LPT1 parallel port:

LPT	01 00	Port
COMMAND	45 00	Encrypting ("E")
LABEL	53 4D 41 52 54 4B 45 59 00 00 00 00 00 00 00 00	Label ("SMARTKEY")
PASSWORD	45 55 54 52 4F 4E 00 00 00 00 00 00 00 00 00 00	Password ("EUTRON")
DATA	XX XX XX XX XX XX XX XX	Data to encrypt (8 bytes)

At the end of the operation, the first 8 bytes of the DATA field are substituted with the Encrypted Data that depend on the Original Data as well as on the previously programmed algorithm in the key.

9.8.2 Definition of the algorithm

The logical **operators** that can be used for the construction of a personal security algorithm are the following:

OPERATOR	DEFINITION	SYMBOL
Left Rotate	Rotation to the left	<
Right Rotate	Rotation to the right	>
NOT	Logical negation	~
AND	Logical product	&
OR	Logical addition	
XOR	Exclusive logical addition	^
Add	256 module addition	+
Move	Movement	M
Assign	Assignment	A
End	End of elaboration	E

The **operands** on which the operators can act are each of the 8 bytes of the Original Data and two temporary variables named X and Y.

OPERAND	DEFINITION
N	Byte n of the Original Data ($1 \leq n \leq 8$)
X	Temporary variable X
Y	Temporary variable Y

9.8.3 Example

Here is an example of a user programmable algorithm.

^	1	2	X	&	X	6	7	~	4	4	E
---	---	---	---	---	---	---	---	---	---	---	---

The meaning is the following :

1. It carries out the exclusive addition (^ / XOR) of the first byte of the Secure Data field with the second and it assigns the temporary variable X to the result.
2. It carries out the logical product (& / AND) of the X variable with the sixth byte of the Secure Data field and assigns the result to the seventh byte.
3. Carries out the negation (~ / NOT) of the fourth byte of the Secure Data field.

We see how the algorithm acts on the following values of the Original Data :

byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8
A3	45	C7	83	34	33	43	40

1. A3 XOR 45 = E6 $\Rightarrow$ X
2. X AND 33 = 22 $\Rightarrow$ byte 7
3. NOT 83 = 7C $\Rightarrow$ byte 4

At the end of the ENCRYPTING operation, the 8 bytes will be consequently transformed into the following Encrypted Data :

byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8
A3	45	C7	7C	34	33	22	40

9.9 Programming mode

The PROGRAMMING function allows complete reprogramming of the key and especially of the Label and Password registers (remember that the WRITING and READING functions can involve only the Secure Data field).

The availability of the PROGRAMMING function allows one to set up one's own dedicated program for the configuration of the keys, usually connected to a data base, in order to associate the contents of each key with a list of customers and/or products.

Programming new Labels and Passwords does not require one to know the previous Password (while one must know the correct label, because the operation automatically resets the existing Secure Data field (it is filled with 00 hex)). For obvious reasons connected with security, the Fail Counter is not reset, instead.

If you forgot the current Label value while testing, you can revert to default value through the SPC utility.

The PROGRAMMING mode is necessary for the off-line programming of the key. We advice though not to use on-line in the software to be protected, as well as for security reasons as to avoid that a programming error puts into act the function on a different SmartKey present in the system. Furthermore, normally it is sufficient to act on the key through the READING and WRITING functions that limit the own intervention to the Secure Data field.

The exchange of information is organized as follows:

Models	PR, EP, SP, NET	
Input	COMMAND LPT LABEL PASSWORD DATA[0..15]	'P' Port New Label New Password Current Label (16 bytes)
Output	STATUS	Status ==0 Success !=0 Error

9.9.1 Passing parameters

Parameters to be transferred to carry out a PROGRAMMING operation with access to the key on the LPT1 parallel port, programming Label and Password. It is supposed that the memorized LABEL before the operation be "LABELOLD".

LPT	01 00	Port
COMMAND	50 00	Programming("P")
LABEL	53 4D 41 52 54 4B 45 59 00 00 00 00 00 00 00 00	Label ("SMARTKEY")
PASSWORD	45 55 54 52 4F 4E 00 00 00 00 00 00 00 00 00 00	Password ("EUTRON")
DATA	4C 41 42 45 4C 4F 4C 44 00 00 00 00 00 00 00 00	Current Label ("LABELOLD")

If the key is found on the indicated port, the values of Label and Password are transferred to the key whilst the Secure Data in the key are automatically set back to zero. The current LABEL uses the first 16 bytes of the Data field, usually used for the transfer of the Secure Data.

9.10 Comparing mode

The COMPARING mode has been introduced to make it easier to use the keys for someone who is dealing with software protection technique for the first time.

It is the simplest way to use programmable type stand-alone SmartKey keys, i.e. models **PR**, **EP** and **SP**. It provides an automatic search for the protection key with the assigned Label, Password and Secure Data on all the system ports, obtaining information that shows whether a key is present and, if so, on which port.

The COMPARING mode is therefore an extension of the LOCATING mode, usable only for programmable keys. It can be useful for applications where it is required only to verify the presence of the key, without any need to update the contents by rewriting the Secure Data.

Models	PR, EP, SP	
Input	COMMAND LABEL PASSWORD DATA	'C' Label Password Data
Output	FAIL_CTR STATUS	Fail Counter Status >=0 Success, number of port <0 Error

The status indicates whether the operation was successful returning the number of port, or a value less than 0 on error.

9.10.1 Passing parameters

Parameters to be transferred to carry out a COMPARING operation with a key research on all the parallel ports present:

COMMAND	50 00	Comparing ("C")
LABEL	53 4D 41 52 54 4B 45 59 00 00 00 00 00 00 00 00	Label ("SMARTKEY")
PASSWORD	45 55 54 52 4F 4E 00 00 00 00 00 00 00 00 00 00	Password ("EUTRON")
DATA	53 45 43 55 52 49 54 59 20 44 41 54 41 00 00 00	Data("SECURITY DATA")

	00 00 00 00 00 00 00 00	
	00 00 00 00 00 00 00 00	
	00 00 00 00 00 00 00 00	
	00 00 00 00 00 00 00 00	
	00 00 00 00 00 00 00 00	
	00 00 00 00 00 00 00 00	

If the key is found on any of the ports present in the system and with the correct values of Label, Password and Secure Data , the status variable will assume at the end of the operation the value of 1, 2 or 3 in function of the LPT.

9.11 Model reading mode

The MODEL READING MODE function allows identification of the *SmartKey* model plugged-in (*FX, PR, EP, SP, NET*). This is an accessory function, that you can use in your package to activate a different behavior for software packages working both in stand-alone version and in network mode.

The exchange of information is organized as follows:

Models	ALL	
Input	COMMAND LPT LABEL	'M' Port Label
Output	DATA[0] DATA[1] STATUS	SmartKey model = '0', none = '1', FX = '2', PR = '3', EP = '9', SP = 'A', NET Available memory on the key = '0', 0 byte = '1', 64 byte = '3', 416 byte Status ==0 Success !=0 Error

9.11.1 Passing parameters

Parameters to be transferred to carry out a MODEL READING operation, with access to the key on the LPT1 parallel port.

LPT	01 00	Port
COMMAND	4D 00	Model Reading ("M")
LABEL	53 4D 41 52 54 4B 45 59 00 00 00 00 00 00 00 00	Label ("SMARTKEY")

If the key is found on the indicated port, the key model is available on the first byte of the Secure Data field according to the chart brought in the descriptive paragraph of this form.

9.12 Serial number reading mode

This command read the SerialNumber of the SmartKey. The SerialNumber is a 32-bit integer unique for any SmartKey.

The exchange of information is organized as follows:

Models	ALL	
Input	COMMAND LPT LABEL	'n' Port Label
Output	DATA[0,1,2,3] STATUS	Serial Number Status ==0 Success !=0 Error

9.13 Ext model reading mode

Read the Extended Model register value. With this value, you can get extensive information of the SmartKey hardware model.

The exchange of information is organized as follows:

Models	ALL	
Input	COMMAND LPT LABEL	'h' Port Label
Output	DATA[0] DATA[1] DATA[2] DATA[3] STATUS	Model = '0', none = '1', FX = '2', PR = '3', EP = '9', SP = 'A', NET Available memory on the key = '0', 0 byte = '1', 64 byte = '3', 416 byte Hardware model = 2, Parallel = 3, USB Zero value = 0 Status ==0 Success !=0 Error

9.14 Fix reading mode

Read the Fix register value.

The exchange of information is organized as follows:

Models	EP, SP, NET	
Input	COMMAND LPT LABEL PASSWORD	'x' Port Label Password
Output	DATA[0] STATUS	= 1, The SmartKey is Fixed = 0, The SmartKey is not Fixed Status ==0 Success !=0 Error

9.15 Fail counter reading mode

Read the FailCounter register value. This is the same value obtained with the READING_MODE command. With this command, you can read it without reading all the memory.

The exchange of information is organized as follows:

Models	EP, SP, NET	
Input	COMMAND LPT LABEL PASSWORD	'a' Port Label Password
Output	FAIL_COUNTER STATUS	Fail counter value Status ==0 Success !=0 Error

9.16 The change of parameters with SmartKey

The software drivers allow the management of SmartKey through an exchange of parameters with a protected application.

The parameters that are to be exchanged depend only of the protection strategy adopted and of the key model, whereas the syntax for the passing parameters is a function of the language used.

For whatever language used it is necessary to define a structure that includes the parameters now defined. The structure is then passed to the interface driver for the particular language.

Although they have the same name, the physical registers present in the key and the relative descriptive parameters are different entities. For example the contents of the Secure Data register of 64 bytes is described by the Data field of the software driver during the Reading and Writing operation; but the same Data field is also used in the SCRAMBLING function for the passing of the Original Data and of the Scrambled Data, that have nothing to do with the register of Secure Data. The SCRAMBLING function in fact modifies the Secure Data, but uses the Data field as a supportive variable for the development of its own function.

The parameters actually used by the type of key and the type of form as indicated in the chart that follows, where for every functioning form are indicated the parameters used.

Command	LPT	CMD	LABEL	PASSWORD	DATA	FAIL_CNT
LOCATING	No	" L "	Yes	No	No	No
SCRAMBLING	Yes	" S "	Yes	No	Yes	No
READING	Yes	" R "	Yes	Yes	Yes	Yes
BLOCK READING	Yes	" BR "	Yes	Yes	Yes	Yes
WRITING	Yes	" W "	Yes	Yes	Yes	Yes
BLOCK WRITING	Yes	" BW "	Yes	Yes	Yes	Yes
FIXING	Yes	" F "	Yes	Yes	Yes	Yes
ENCRYPTING	Yes	" E "	Yes	Yes	Yes	No
PROGRAMMING	Yes	" P "	Yes	Yes	No	No
COMPARING	No	" C "	Yes	Yes	Yes	Yes
MODEL READ	Yes	" M "	Yes	No	Yes	No
SERIAL READ	Yes	" n "	Yes	No	Yes	No
EXT MODEL READ	Yes	" h "	Yes	No	Yes	No
FIX READ	Yes	" x "	Yes	Yes	Yes	No
FAIL READ	Yes	" a "	Yes	Yes	Yes	Yes

9.17 Some hints on the uses of smartkey functions

A joint use of the functions listed allows the possibility of setting up high-security and flexible protection criteria. In any case here are just a few hints for a secure implementation of the protection.

- Always use the LOCATING mode to identify the presence of the key; this will enable you to make the software independent on the parallel port on which *SmartKey* is installed.
- Use SCRAMBLING mode at different points of the program with different Original Data values; if the result of subsequent comparisons with Scrambled Data matches, you may be confident that a key with your Id-Code has been connected to the PC.
- If you use programmable keys, carry out a READING operation at different points of the program; you can compare the values read with the expected ones.

9.18 An example

Here follows a demo program in C that highlights the use of software.

```

/*
 * SmartKey Demo Program
 * Copyright EUTRON 1993-2002
 */

#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#include "skeydrv.h"

```

```

static void print_error(int err)
{
    switch (err) {
        case ST_NONE_KEY :
            printf("ST_NONE_KEY : Key Not Found\n");
            break;
        case ST_SYNT_ERR :
            printf("ST_SYNT_ERR : Syntax Error\n");
            break;
        case ST_LABEL_FAILED :
            printf("ST_LABEL_FAILED : Wrong Label\n");
            break;
        case ST_PW_DATA_FAILED :
            printf("ST_PW_DATA_FAILED : Wrong Password\n");
            break;
        case ST_HW_FAILURE :
            printf("ST_HW_FAILURE : Hardware failure\n");
            break;
        default:
            printf("%04x : Unrecognizable Status\n", err);
            break;
    }
}

int main(void)
{
    SKEY_DATA skey;
    int i;
    unsigned model;

    printf("SmartKey Demo Program\n\n");

    /* Initialize */
    memset(&skey, 0, sizeof(skey));

    /* Set the Label */
    strncpy(skey.label, "SMARTKEY", LABEL_LENGTH);

    /* Locating */
    printf("Locating : ");
    skey.command = LOCATING_MODE;

    msclink(&skey);
    if (skey.status != ST_OK) {
        print_error(skey.status);
        exit(EXIT_FAILURE);
    }

    printf("SmartKey found on port %d\n", skey.lpt);

    /* Scrambling */
    printf("Scrambling ");

    memcpy(skey.data, "12345678", SCRAMBLE_LENGTH);
    skey.command = SCRAMBLING_MODE;

    for(i=0; i<SCRAMBLE_LENGTH; ++i)
        printf("%02x", (unsigned)skey.data[i]);

    printf(" : ");

    msclink(&skey);

```

```

    if (skey.status != ST_OK) {
        print_error(skey.status);
        exit(EXIT_FAILURE);
    }

    for(i=0;i<SCRAMBLE_LENGTH;++i)
        printf("%02x", (unsigned)skey.data[i]);

    printf("\n");

    /* Serial */
    printf("Serial : ");
    skey.command = SERIAL_NUMBER_READING_MODE;

    msclink(&skey);
    if (skey.status != ST_OK) {
        print_error(skey.status);
        exit(EXIT_FAILURE);
    }

    printf("%02x%02x%02x%02x\n", (unsigned)skey.data[3],
(unsigned)skey.data[2], (unsigned)skey.data[1], (unsigned)skey.data[0]);

    /* Model */
    printf("Model : ");
    skey.command = MODEL_READING_MODE;

    msclink(&skey);
    if (skey.status != ST_OK) {
        print_error(skey.status);
        exit(EXIT_FAILURE);
    }

    switch (skey.data[0]) {
        case SKEY_FX : printf("model FX"); break;
        case SKEY_PR : printf("model PR"); break;
        case SKEY_EP : printf("model EP"); break;
        case SKEY_SP : printf("model SP"); break;
        case SKEY_NET : printf("model NET"); break;
        default : printf("model unknown"); break;
    }
    switch (skey.data[1]) {
        case '0' : printf(", memory 0 byte"); break;
        case '1' : printf(", memory 64 byte"); break;
        case '3' : printf(", memory 416 byte"); break;
        default : printf(", memory unknown"); break;
    }
    printf("\n");

    /* Ext Model */
    printf("Ext Model : ");
    skey.command = EXT_MODEL_READING_MODE;

    msclink(&skey);
    if (skey.status != ST_OK) {
        print_error(skey.status);
        exit(EXIT_FAILURE);
    }

    model = skey.data[0];

    switch (skey.data[0]) {
        case SKEY_FX : printf("model FX"); break;

```

```

        case SKEY_PR : printf("model PR"); break;
        case SKEY_EP : printf("model EP"); break;
        case SKEY_SP : printf("model SP"); break;
        case SKEY_NET : printf("model NET"); break;
        default : printf("model unknown"); break;
    }
    switch (skey.data[1]) {
        case '0' : printf(", memory 0 byte"); break;
        case '1' : printf(", memory 64 byte"); break;
        case '2' : printf(", memory 192 byte"); break;
        case '3' : printf(", memory 416 byte"); break;
        default : printf(", memory unknown"); break;
    }
    switch (skey.data[2]) {
        case 2 : printf(", type ASIC"); break;
        case 3 : printf(", type USB"); break;
        default : printf(", type unknown"); break;
    }
    printf("\n");

    /* set the password */
    strncpy(skey.password, "EUTRON", PASSWORD_LENGTH);

    if (model == SKEY_EP || model == SKEY_SP || model == SKEY_NET) {

        /* Fix */
        printf("Fix : ");
        skey.command = FIX_READING_MODE;

        msclink(&skey);
        if (skey.status != ST_OK) {
            print_error(skey.status);
            exit(EXIT_FAILURE);
        }

        printf("%s\n", skey.data[0] != 0 ? "yes" : "no" );
    }

    if (model == SKEY_EP || model == SKEY_SP || model == SKEY_NET) {

        /* Fail Counter */
        printf("Fail Counter : ");
        skey.command = FAIL_COUNTER_READING_MODE;

        msclink(&skey);
        if (skey.status != ST_OK) {
            print_error(skey.status);
            exit(EXIT_FAILURE);
        }

        printf("%d\n", skey.fail_counter);
    }

    if (model == SKEY_PR || model == SKEY_EP || model == SKEY_SP || model ==
SKEY_NET) {

        /* Reading */
        printf("Reading: ");
        skey.command = READING_MODE;

        msclink(&skey);
        if (skey.status != ST_OK) {
            print_error(skey.status);

```

```

        exit(EXIT_FAILURE);
    }

    for(i=0;i<DATA_LENGTH;++i)
        if (skey.data[i] >= 32 && skey.data[i] <= 127)
            printf("%c", (char)skey.data[i]);
        else
            printf(" ");

    printf("\n");

    /* Writing */
    printf("Writing: ");
    skey.command = WRITING_MODE;
    memset(skey.data,0,DATA_LENGTH);
    strncpy(skey.data,"SECURITY DATA",DATA_LENGTH);

    msclink(&skey);
    if (skey.status != ST_OK) {
        print_error(skey.status);
        exit(EXIT_FAILURE);
    }

    for(i=0;i<DATA_LENGTH;++i)
        if (skey.data[i] >= 32 && skey.data[i] <= 127)
            printf("%c", (char)skey.data[i]);
        else
            printf(" ");

    printf("\n");
}

return EXIT_SUCCESS;
}

```

10 Manual Network Protection

SmartKey-NET supports some of the standard commands of the SP model. In addition to these, a set of commands slaved to the networks, i.e., OPEN, ACCESS, CLOSE and USERS.

10.1 Open Mode

The OPEN mode is used by the user to activate communication (LOGIN request) with SmartKey. The OPEN command must be given before any access operation to the memory of the protection device and possibly upon startup of the protected application..

The OPEN command generates a special password called NET-Password, to be used in subsequent commands (ACCESS, CLOSE). This is a further increment of protection procedures aimed at avoiding possible attempts at breaching the interrogation procedure of the network protection device.

The exchange of information is organized as follows:

Models	NET	
Input	NET_COMMAND LABEL PASSWORD	'O' Label Password
Output	NET_PASS STATUS	Net Password. Status ==0 Success !=0 Error

10.2 Access Mode

The ACCESS mode allows actual access to the protection device, selected via the label which must always be passed before effecting any standard interrogation operation with the protection device. This mode requires the NET-Password which identifies the client who requests access and the label which identifies the protection device to which access is requested.

The exchange of information is organized as follows:

Models	NET	
Input	NET_COMMAND NET_PASS COMMAND ...	'A' Net Password SmartKey Command
Output	... STATUS	Status ==0 Success !=0 Error

Otherwise one proceeds as in the modes already analyzed for manual protection with stand-alone applications.

10.3 User number mode

With this command one obtains information on the number of users connected with the protection device specified in the label field.

The exchange of information is organized as follows:

Models	NET	
Input	NET_COMMAND NET_PASS COMMAND	'A' Net Password 'U'
Output	STATUS	Status >=0 Number of user connected <0 Error

10.4 Close mode

The CLOSE mode enables “closing” (LOGOUT request) of communication with SmartKey of the program connected via an earlier OPEN command.

If owing to an error in the application software the program should terminate without first carrying out CLOSE of the open protection device, problems with the client’s timeout management may occur and may at time involve rebooting. It is essential to carry out the login and logout procedures carefully to avoid uselessly inhibiting other users from using the key.

The exchange of information is organized as follows:

Models	NET	
Input	NET_COMMAND NET_PASS	'C' Net Password
Output	STATUS	Status ==0 Success !=0 Error

10.5 Close mode on timeout

In order to avoid keeping licenses engaged due to abnormal termination of the program (e.g. because the PC was switched off or the <CTRL><ALT> was given), a transparent timeout refresh function has been implemented, i.e. the automatic disconnection of users who did not run the CLOSE mode correctly. The timeout check function is fully transparent to the user and developer and is managed by the key’s software drivers; it is therefore unnecessary to carry out regular key access operations for timeout refresh.

10.6 Stand-alone driver or MultiLan ?

The use of the driver relative to the stand-alone application was illustrated in chapter Manual Protection with Software drivers . In the actual chapter however MultiLan technology has been discussed, which allows the use of the driver either for stand-alone and network. Here is some advise on which of the two types of driver to use:

- **Only stand-alone applications:** use a stand-alone driver because the dimension is less than the Multilan driver (about half of it, because the support of the network is not included)
- **Stand-alone and network applications:** use a driver of the MultiLan type, because it makes the application independent of the functional mode (stand-alone or network).

11 Protection of several applications with SmartKey

When running in a LAN environment, a single SmartKey-NET key can be used for protecting several applications. The technology used is called MAP - MultiApplication Protection - and allows one:

- **To protect several** applications running in stand-alone or network In the case of local network it is also possible to define a different number of licenses for each application protected.
- To limit the number of executions of each of the protected programs. This option can prove useful if one needs to create demo versions of the software or intends to lease it out. When the number of executions preset in the counter (and decremented at each program startup) runs out, the program is inhibited from starting.

11.1 The operating modes

Implementing MAP involves only very few changes against the operating modes already described, i.e.:

- The optional limitation of the number of executions and, in the case of SmartKey-NET inputting the number of licenses of each single application
- A different call mode for the management driver as regards OPEN mode and USER NUMBER mode.

11.2 Numbers of licenses and of number of executions

The top number of applications which can be protected with the same device is 116. The top number of executions of each application is in the range of 1 to 65533; the value 65535 disables the control for an unlimited number of executions.

The number of licenses and executions of each program can be set by writing the value into the first bytes of the Secure Data Field and, if required, in the key's Extended Data field as tabulated below:

Secure Data byte n.	Value	Meanings
0	4D hex	MAP fixed identif.code
1	41 hex	MAP fixed identif.code
2	01-32 hex (1-50 dec)	Max. Number of users for application 1
3-4	0000-FFFF hex (0-65535 dec)	Max. Number of executions for application 1
5	01-32 hex (1-50 dec)	Max. Number of users for application 2
6-7	0000-FFFF hex (0-65535 dec)	Max. Number of executions for application 2
8	01-32 hex (1-50 dec)	Max. Number of users for application 3
9-10	0000-FFFF hex (0-65535 dec)	Max. Number of executions for application 3
.	.	.
.	.	.
.	.	.
62	01-32 hex (1-50 dec)	Max. Number of users for application 20

63- 0ext_data	0000-FFFF hex (0-65535 dec)	Max. Number of executions for application 20
1 ext_data	01-32 hex (1-50 dec)	Max. Number of users for application 21
2-3 ext_data	0000-FFFF hex (0-65535 dec)	Max. Number of executions for application 21
.	.	.
.	.	.
285 ext_data	01-32 hex (1-50 dec)	Max. Number of users for application 116
287-288 ext_data	0000-FFFF hex (0-65535 dec)	Max. Number of executions for application 116
.	.	.
.	.	.
.	.	.

For instance, if one wanted to protect 3 different applications with the following number of licenses and executions:

- First program: 23 licenses and unlimited execution
- Second program: 4 licenses and 4000 executions
- Third program: 12 licenses and 100 executions

the memory of the key would need to be set as follows:

Secure Data byte n.	Value	Meanings
0	4D hex	MAP fixed identif. code
1	41 hex	MAP fixed identif. code
2	17 hex (23 dec)	Max. 23 users for application 1
3-4	FFFF hex 65535 dec)	No limits to the no. of executions of the program
5	4 hex (4 dec)	Max. 4 users for application 1
6-7	FA0 hex (4000 dec)	Active limit to 4000 executions
8	0C hex (12 dec)	Max. 12 users for application 3
9-10	64 hex (100 dec)	Active limit to 100 executions
11	XX	Value not used by MAP
12	XX	Value not used by MAP
.	.	.

.	.	.
---	---	---

11.3 The MAP automatic protection

If automatic protection via the GSS utility is implemented with a network key, there is a button to associate a code to every application. This code can be used to find out the number of licenses for which the program is enabled as well as any limit to the number of executions of the software.

11.4 The MAP automatic protection

11.4.1 Open mode

Since several applications may be protected at the same time, just in the case of the OPEN command one has to specify the program to which one is referring (identified by a number from 1 to 116). In passing the parameters relevant to each of the two commands, the DATA field of the relevant data structure must therefore be initialized as follows:

DATA byte n.	Value	Meanings
0	4D hex	MAP fixed identification code
1	41 hex	MAP fixed identification code
2	01-74 hex (1-116 dec)	Reference number of the application

If one wants to use the MAP technology with the Global Security System automatic protection utility, it is important to remember to deselect the Protection depends on Secure Data option at the time of protecting the application. This is due to the need for MAP to decrement the counter by the number of executions (if enabled) input in the Secure Data field.ds on Secure Data.

11.4.2 Open mode MAP: an example

Parameters to be transferred to execute an OPEN operation using the MAP multiple protection on a network key: opening of application 2 is required. If the key was programmed as in the preceding programming table, not more than 4 users may use the software at the same time.

NET_COMMAND	4F 00	Open ("=")
LABEL	53 4D 41 52 54 4B 45 59 00 00 00 00 00 00 00 00	Label ("SMARTKEY")
PASSWORD	45 55 54 52 4F 4E 00 00 00 00 00 00 00 00 00 00	Password ("EUTRON")
DATA	4D 41 02	Map, 2 application

11.5 An Example

Here follows a demo program in C that highlights the use of software.

```

/*
 * SmartKey Demo Program
 * Copyright EUTRON 1993-2002
 */

#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#include "skeylink.h"

```

```

static void print_error(int err)
{
    switch (err) {
        case ST_NONE_KEY :
            printf("ST_NONE_KEY : Key Not Found\n");
            break;
        case ST_SYNT_ERR :
            printf("ST_SYNT_ERR : Syntax Error\n");
            break;
        case ST_LABEL_FAILED :
            printf("ST_LABEL_FAILED : Wrong Label\n");
            break;
        case ST_PW_DATA_FAILED :
            printf("ST_PW_DATA_FAILED : Wrong Password\n");
            break;
        case ST_HW_FAILURE :
            printf("ST_HW_FAILURE : Hardware failure\n");
            break;
        case ST_NET_ERROR :
            printf("ST_NET_ERROR : Net error\n");
            break;
        case ST_CLOSE_ERROR :
            printf("ST_CLOSE_ERROR : Attempting to CLOSE without
OPENing\n");
            break;
        case ST_ACCESS_ERROR :
            printf("ST_ACCESS_ERROR : Attempting to ACCESS without
OPENing\n");
            break;
        case ST_USER_ERROR :
            printf("ST_USER_ERROR : Max user reached\n");
            break;
        case ST_NET_PWD_ERR :
            printf("ST_NET_PWD_ERR : Net password wrong\n");
            break;
        case ST_TOO_MANY_OPEN_KEY :
            printf("ST_TOO_MANY_OPEN_KEY : Too many open SmartKey\n");
            break;
        case ST_NET_PASS_INVALID :
            printf("ST_NET_PASS_INVALID : Invalid net password\n");
            break;
        case ST_NET_CONF_ERROR :
            printf("ST_NET_CONF_ERROR : Configuration error in
INI/Registry/Environment\n");
            break;
        case ST_NET_ANP_INIT_ERROR :
            printf("ST_NET_ANP_INIT_ERROR : Error initializing the ANP
protocol\n");
            break;
        case ST_NET_TCPIP_INIT_ERROR :
            printf("ST_NET_TCPIP_INIT_ERROR : Error initializing the TCPIP
protocol\n");
            break;
        case ST_NET_NOVELL_INIT_ERROR :
            printf("ST_NET_NOVELL_INIT_ERROR : Error initializing the
Novell protocol\n");
            break;
        case ST_NET_LOCAL_INIT_ERROR :
            printf("ST_NET_LOCAL_INIT_ERROR : Error initializing the Local
protocol\n");
            break;
        case ST_NET_KEY_NOT_MAP :
            printf("ST_NET_KEY_NOT_MAP : Not MAP programmed key found\n");
    }
}

```

```

        break;
    default:
        printf("%04x : Unrecognizable Status\n", err);
        break;
    }
}

int main(void)
{
    KEY_NET skey;
    int i;
    unsigned model;

    printf("SmartKey Demo Program\n\n");

    /* Initialize */
    memset(&skey, 0, sizeof(skey));

    /* Set the Label */
    strncpy(skey.label, "SMARTKEY", LABEL_LENGTH);

    /* set the password */
    strncpy(skey.password, "EUTRON", PASSWORD_LENGTH);

    /* Opening with MAP */
    printf("Opening MAP : ");
    skey.net_command = NET_KEY_OPEN;
    skey.data[0] = 'M';
    skey.data[1] = 'A';
    skey.data[2] = 1;
    smartlink(&skey);
    if (skey.status != ST_OK) {
        print_error(skey.status);

        /* Opening without MAP */
        printf("Retrying without MAP : ");
        skey.net_command = NET_KEY_OPEN;
        skey.data[0] = 0;
        skey.data[1] = 0;
        skey.data[2] = 0;

        smartlink(&skey);
        if (skey.status != ST_OK) {
            print_error(skey.status);
            exit(EXIT_FAILURE);
        }
    }

    printf("SmartKey found with protocol ");
    switch (skey.data[0]) {
    case NET_TYPE_LOCAL : printf("LOCAL"); break;
    case NET_TYPE_IPX : printf("IPX"); break;
    case NET_TYPE_ANP : printf("ANP"); break;
    case NET_TYPE_TCPIP : printf("TCPIP"); break;
    default : printf("unknown"); break;
    }
    printf("\n");

    /* Scrambling */
    printf("Scrambling ");

    memcpy(skey.data, "12345678", SCRAMBLE_LENGTH);
    skey.net_command = NET_KEY_ACCESS;

```

```

skey.command = SCRAMBLING_MODE;

for(i=0;i<SCRAMBLE_LENGTH;++i)
    printf("%02x", (unsigned)skey.data[i]);

printf(" : ");

smartlink(&skey);
if (skey.status != ST_OK) {
    print_error(skey.status);
    exit(EXIT_FAILURE);
}

for(i=0;i<SCRAMBLE_LENGTH;++i)
    printf("%02x", (unsigned)skey.data[i]);

printf("\n");

/* Serial */
printf("Serial : ");
skey.net_command = NET_KEY_ACCESS;
skey.command = SERIAL_NUMBER_READING_MODE;

smartlink(&skey);
if (skey.status != ST_OK) {
    print_error(skey.status);
    exit(EXIT_FAILURE);
}

printf("%02x%02x%02x%02x\n", (unsigned)skey.data[3],
(unsigned)skey.data[2], (unsigned)skey.data[1], (unsigned)skey.data[0]);

/* Model */
printf("Model : ");
skey.net_command = NET_KEY_ACCESS;
skey.command = MODEL_READING_MODE;

smartlink(&skey);
if (skey.status != ST_OK) {
    print_error(skey.status);
    exit(EXIT_FAILURE);
}

switch (skey.data[0]) {
    case SKEY_FX : printf("model FX"); break;
    case SKEY_PR : printf("model PR"); break;
    case SKEY_EP : printf("model EP"); break;
    case SKEY_SP : printf("model SP"); break;
    case SKEY_NET : printf("model NET"); break;
    default : printf("model unknown"); break;
}

switch (skey.data[1]) {
    case '0' : printf(", memory 0 byte"); break;
    case '1' : printf(", memory 64 byte"); break;
    case '2' : printf(", memory 192 byte"); break;
    case '3' : printf(", memory 416 byte"); break;
    default : printf(", memory unknown"); break;
}

printf("\n");

/* Ext Model */
printf("Ext Model : ");
skey.net_command = NET_KEY_ACCESS;

```

```

skey.command = EXT_MODEL_READING_MODE;

smartlink(&skey);
if (skey.status != ST_OK) {
    print_error(skey.status);
    exit(EXIT_FAILURE);
}

model = skey.data[0];

switch (skey.data[0]) {
    case SKEY_FX : printf("model FX"); break;
    case SKEY_PR : printf("model PR"); break;
    case SKEY_EP : printf("model EP"); break;
    case SKEY_SP : printf("model SP"); break;
    case SKEY_NET : printf("model NET"); break;
    default : printf("model unknown"); break;
}
switch (skey.data[1]) {
    case '0' : printf(", memory 0 byte"); break;
    case '1' : printf(", memory 64 byte"); break;
    case '2' : printf(", memory 192 byte"); break;
    case '3' : printf(", memory 416 byte"); break;
    default : printf(", memory unknown"); break;
}
switch (skey.data[2]) {
    case 0 : printf(", type A"); break;
    case 1 : printf(", type B"); break;
    case 2 : printf(", type ASIC"); break;
    case 3 : printf(", type USB"); break;
    default : printf(", type unknown"); break;
}
printf("\n");

if (model == SKEY_EP || model == SKEY_SP || model == SKEY_NET) {

    /* Fix */
    printf("Fix : ");
    skey.net_command = NET_KEY_ACCESS;
    skey.command = FIX_READING_MODE;

    smartlink(&skey);
    if (skey.status != ST_OK) {
        print_error(skey.status);
        exit(EXIT_FAILURE);
    }

    printf("%s\n", skey.data[0] != 0 ? "yes" : "no" );
}

if (model == SKEY_EP || model == SKEY_SP || model == SKEY_NET) {

    /* Fail Counter */
    printf("Fail Counter : ");
    skey.net_command = NET_KEY_ACCESS;
    skey.command = FAIL_COUNTER_READING_MODE;

    smartlink(&skey);
    if (skey.status != ST_OK) {
        print_error(skey.status);
        exit(EXIT_FAILURE);
    }
}

```

```

        printf("%d\n",skey.fail_counter);
    }

    if (model == SKEY_PR || model == SKEY_EP || model == SKEY_SP || model ==
SKEY_NET) {

        /* Reading */
        printf("Reading: ");
        skey.net_command = NET_KEY_ACCESS;
        skey.command = READING_MODE;

        smartlink(&skey);
        if (skey.status != ST_OK) {
            print_error(skey.status);
            exit(EXIT_FAILURE);
        }

        for(i=0;i<DATA_LENGTH;++i)
            if (skey.data[i] >= 32 && skey.data[i] <= 127)
                printf("%c", (char)skey.data[i]);
            else
                printf(" ");

        printf("\n");

        /* Writing */
        printf("Writing: ");
        skey.net_command = NET_KEY_ACCESS;
        skey.command = WRITING_MODE;
        memset(skey.data,0,DATA_LENGTH);
        strncpy(skey.data,"SECURITY DATA",DATA_LENGTH);

        smartlink(&skey);
        if (skey.status != ST_OK) {
            print_error(skey.status);
            exit(EXIT_FAILURE);
        }

        for(i=0;i<DATA_LENGTH;++i)
            if (skey.data[i] >= 32 && skey.data[i] <= 127)
                printf("%c", (char)skey.data[i]);
            else
                printf(" ");

        printf("\n");
    }

    /* Closing */
    printf("Closing : ");
    skey.net_command = NET_KEY_CLOSE;

    smartlink(&skey);
    if (skey.status != ST_OK) {
        print_error(skey.status);
        exit(EXIT_FAILURE);
    }

    printf("Closed\n");

    return EXIT_SUCCESS;
}

```

12 Installation

The installation of the SmartKey drivers in Windows is completely done by the *SmartKey Driver Installer* application (SDI).

The installer contains a visual interface called SDI to manually install and uninstall all the SmartKey drivers. It also contains a DLL library which allow to integrate the SmartKey driver installation in the install process of your application.

12.1 Warnings for the Installations

Further on are indicated some precautions and some advises that must be commonly adopted during the installation and the use to avoid damages and/or functional anomalies of the key:

- Plug and unplug SmartKey between PC and Printer only when both are off.
- Do not insert SmartKey in the serial interface connector (D-type 25 male pin) because the presence of negative tensions can alter or damage the contents of the key.
- Do not manipulate SmartKey in the presence of electrostatic charges (for example due to carpets or other). Avoid touching unnecessarily the pins on the key's connector so as not to inject parasitic electrostatic charges.
- Ensure that the calculator and the printer are connected, as by norm, to the floor implant.
- Do not submit the key to high temperatures or to high changes of temperature.
- Even with a correctly installed SmartKey, extreme events, such as the breakdown of the communication circuits of the personal computer, electrical discharges on the calculators behalf or from the exterior can provoke damages or functional anomalies of the key. Therefore take the necessary provisions to avoid that such damages or functional anomalies can cause damages to people or materials.

13 Network Installation

SmartKey-NET can be used with any kind of local area network thanks to the supporting software supplied. To be more precise, the software takes into consideration two different kinds of network:

- Novell Netware 3.x, 4.x, 5.x networks. In this case SmartKey-NET is placed on the network server on which a SmartKey NLM (Novell Loadable Module) driver will be installed.
- Non Novell Netware networks. Independently of the kind of server or network SmartKey-NET can be used by connecting it to any PC in the network.

Moreover, you can choose from three possible communications protocols:

- TCPIP: Protocol normally used for non-Novell networks.
- IPX: Protocol normally used for Novell networks.
- ANP: Communications protocol which provides the greatest flexibility of use and compatibility with different kinds of network.

Depending on the protocol you choose, the key can be used remotely on a PC with the following operating system as in the table:

SmartKey Client	IPX	TCPIP	ANP
DOS	✓		✓
Windows 3.x	✓	✓	✓
Windows 9x/Nt/2000	✓	✓	✓

The key will be physically installed on a PC which shall be called the KeyServer. The KeyServer could be either the network server or a generic client PC. The KeyServer can use any of the following operating systems, depending on the protocol used:

SmartKey Server	IPX	TCPIP	ANP
DOS			✓
Windows 3.x			✓
Windows 9x		✓	✓
Windows Nt 3.x			✓
Windows Nt 4.x/2000		✓	✓
OS/2			✓
Novell 3.x/4.x	✓		
Novell 5.x	✓	✓	

13.1 TCP/IP Protocol

The TCP/IP protocol can be used on any kind of network which supports it, this excludes Novell 3.x/4.x networks but not 5.x.

The protocol doesn't put any constraints on the choice of the SmartKey server which can either be the network server or a generic client.

To use the protocol you have to:

- Identify the network address of the PC on which the key and the SmartKey server will be installed.

- Check that the SmartKey PC server can be reached by all the other PCs over the network. You can use the PING utility supplied with most operating systems.
- Identify a free TCP/UDP port which will be used for communications. The number of the port has to be between 1024 and 49151 in order to avoid any conflict with other protocols. For example, 13527.
- Install and configure correctly both the SmartKey client and server for the use of the TCP/IP protocol specifying the address of the SmartKey server and the TCP/IP port to use.

13.2 IPX Protocol

The IPX protocol can only be used on Novell networks and requires that the SmartKey server also be the network server.

To use the protocol you don't have to do anything special, you only have to:

- Install and correctly configure both the server and the SmartKey client for the use of the Novell IPX protocol.

13.3 ANP Protocol

The ANP protocol (Algorithmic Network Protection) makes use of shared filesystems for communications between client and server. This means that the ANP protocol is one of the most general since shared filesystems are available on any kind of network.

The only condition required for it to work is the support for file locking by the shared filesystem.

The protocol doesn't place any constraints on the choice of the SmartKey server which can either be the network server or any PC client.

In order to use the protocol you have to:

- Identify the network drive shared by all the client PCs (it could be the one on which the network application usually runs) for example N :
- On the network drive, create a working directory for the SmartKey server program. The directory can be called anything, for example:

```
MD N:\ANP
```

- Grant read and write access to the directory from all the computers in the network which will need to use the protected program. It's important that all computers can create and modify files in this directory. If you are not sure if this is the case, before proceeding with the next steps, try copying some files to the directory from every client computer that has to use the protected program.
- Install and correctly configure both the server and the SmartKey client for the use of the ANP protocol and the shared directory you created.

With some network operating systems, the level of file locking is not automatically enabled following installation: please refer to the supplied manuals for details regarding the availability and enabling of file locking.

It could be that the complete name of the shared directory necessary for the ANP protocol has different values from the KeyServer and the client PCs. The value which is used on loading the server program identifies the shared directory so that it's seen by the KeyServer which is not necessarily visible to the PC clients in the same way or with the same name. The KeyServer could see the shared directory as C:\MYDIR while the client PCs might see the same directory as F:\MYDIR. In this case the SmartKey server configuration has to be C:\MYDIR and the SmartKey client has to be F:\MYDIR.

13.4 Installing the Novell server with the IPX protocol

The Novell IPX server can be used on a Novell Netware 3.x/4.x/5.x network.

The installation procedure is as follows:

- Connect the **SmartKey-NET** key to any parallel port of the network server.
- From any client computer in the network perform a user login as the SUPERVISOR. If for security reason you don't have the password to gain access as the SUPERVISOR, contact your system administrator and ask him to install the software.

- Copy the program `NSKEYSRV.NLM` supplied with the key to the `SYSTEM` directory of the volume `SYS:` of the server. This is an NLM file (Netware Loadable Module) which must stay resident in the memory of the server.

For example:

```
COPY A:\NSKEYSRV.NLM X:\SYSTEM
```

- From the server console, edit the `AUTOEXEC.NCF` file using the command:

```
LOAD EDIT AUTOEXEC.NCF
```

or by using the system utility

```
LOAD INSTALL
```

- At the end of the file, add the instruction for loading the NLM module:

```
LOAD NSKEYSRV
```

and save the changes when you exit the editor. This will ensure that the NLM module will automatically be loaded into memory every time the server is started.

- In order to avoid switching the server off and on again, the first time you run the NLM module type the following command from the console:

```
LOAD NSKEYSRV
```

When the NLM module has been loaded, a screen which is reserved for the viewing of statistics relative to the use of the SmartKey keys will be activated. It will also show the name of the NLM as it has been declared to all the Client workstations.

In the Load instruction for loading the server you can set a few parameters for its configuration:

<code>-t=<TIMEOUT></code>	Specifies the time-out for the protocol in seconds.
	If no parameter is given, a default time-out of 10 seconds will be used. If you specify a value of 0 then the time-out will be disabled.

Please remember that to use the network key properly in a Novell Netware environment the IPX protocol has to be version 3.10 or higher. If it isn't, ask your Novell dealer to supply you with a free update.

13.5 Installing the Novell server with the TCP/IP protocol

The TCP/IP server for Novell can be used on a Novell Netware 5.x system.

The installation procedure is as follows:

- Connect the SmartKey key to any parallel port on the network server.
- From any client computer in the network perform a user login as the `SUPERVISOR`. If for security reason you don't have the password to gain access as the `SUPERVISOR`, contact your system administrator and ask him to install the software.
- Copy the program `NSKTCPIP.NLM` supplied with the key to the `SYSTEM` directory of the volume `SYS:` of the server. This is an NLM file (Netware Loadable Module) which must stay resident in the memory of the server.

For example:

```
COPY A:\NSKTCPIP.NLM X:\SYSTEM
```

- From the server console, edit the AUTOEXEC.NCF file with the command:

```
LOAD EDIT AUTOEXEC.NCF
```

or by using the system utility

```
LOAD INSTALL
```

- At the end of the file, add the instruction for loading the NLM module:

```
LOAD NSKTCP/IP -p:<PORT>
```

And save the changes when you exit the editor. This will ensure that the NLM module is automatically loaded every time the server is started.

Replace the <PORT> string with the number of the TCP/IP port to be used.

- Make sure that the NSPDNS.NLM module supplied with the operating system is loaded. If not, insert the automatic loading of it in the file AUTOEXEC.NCF.
- To avoid switching off and on the server the first time you run the NLM module, type the following command from the console:

```
LOAD NSKTCP/IP -p:<PORT>
```

- When the NLM module has been loaded, a screen which is reserved for the viewing of statistics relative to the use of the SmartKey keys will be activated. It will also show the name of the NLM as it has been declared to all the Client workstations.

In the LOAD instruction for loading the server you can specify a few parameters for its configuration:

-p:<PORT>	Specifies the listening port of the TCP/IP protocol
-t:<TIMEOUT>	Specifies the time-out in seconds for the protocol. If it is not specified the time-out will not be activated

13.6 Installing the server for Windows with the TCP/IP-ANP protocols

This server can be used on any client or server PC in the network using the Windows 9x, NT 4 or 2000 operating system. ANP and TCP/IP communications protocols are supported.

The installation procedure is as follows:

- Configuring the server with the ASKEYADD.EXE command or manually editing the ASKEYSRV.INI configuration file.
- Manually start the server which shows the current configuration with the communications protocols used.
- To obtain statistics on the use of the SmartKey key start the Monitor utility from the server menu.

13.7 Configuring the server for Windows multiprotocol using ASKEYADD

The ASKEYADD utility carries out the configuration of the network protocols supported by the SmartKey server.

The following settings are accepted:

-a:<DIRECTORY>	Specifies the absolute path of the shared directory for the temporary ANP files. The path specified has to be in the UNC format (for example: \\SERVER1\DIRECTORY and not F:\DIRECTORY). Each directory can support only one service. In other words, if two services have been installed on the network, there must be two clearly distinct and separate directories.
-p:<PORT>	Specifies the listening port of the TCP/IP protocol

-t:<TIMEOUT>	Specifies the time-out for the ANP protocol. If nothing is specified the default value will be 30 seconds.
-h	Automatically hides the server window 5 seconds after start-up
-c	Cancels the current configuration

The configuration is saved in the Windows registry in the path:

"HKEY_LOCAL_MACHINE\Software\Eutron\SmartKey Multilan Server"

The ASKEYADD utility always displays the current configuration even if it has been defined using different methods.

13.8 Configuring the server for Windows multiprotocol using ASKEYSRV.INI

The ASKEYSRV.INI configuration file allows you to specify the configuration of the network protocols of the SmartKey server.

The file must be in the Windows directory.

[Protocol]

Anp = <YESNO> Enables or not the ANP protocol (the default is No)

TcpIp = <YESNO> Enables or not the TCP/IP protocol (the default is No)

[Server]

ANPShared = <DIRECTORY> Directory to use for the ANP protocol

IpServer = <SERVER> Host to use for the TCP/IP protocol (the default is the server on which the SmartKey server is running)

Port = <PORT> Port to use for the TCP/IP protocol

Timeout = <TIMEOUT> Timeout in seconds for disconnecting clients, (0 = no timeout)

Example:

[Protocol]

Anp=Yes

TcpIp=Yes

[Server]

Port=13527

ANPShared=F:\SKEYANP

Timeout=1000

13.9 Installing the service for Windows with the TCPIP-ANP protocols

The service for Windows can be used on any client or server PC in the network with the Windows NT 4.x/2000 operating system. ANP and TCP/IP communications protocols are supported.

Installation of the service is carried out as follows:

- To work properly the service requires that the KEYP.SYS device be loaded and running. Please refer to the relative paragraph in chapter 2 regarding the subject.
- Copy the file SKEYSRVC.EXE to the directory <windows>\System32
- Run the command ASKEYADD.EXE to configure the server.

- Start the Services program from the Control Panel, select the ASKEYSRV module and press the Start button.
- During both normal working and in the case of errors, the service inserts messages in system log reserved for applications. Use the Event Viewer utility available in the menu Windows Nt Programs/Administrative Tools/Event Viewer to view it.
- To obtain statistics on the use of the SmartKey key updated in real time, run the SKEYMON utility.

The service can be uninstalled manually from the Control Panel or by using the ASKEYRM utility supplied with the SmartKey kit. ASKEYRM also uninstalls the system service by carrying out the de-register functions.

13.10 Configuring the service for Windows with the TCP/IP-ANP protocols

The ASKEYADD utility carries out the configuration of the network protocols supported by the SmartKey server.

It accepts the following parameters:

-a : <DIRECTORY>	Specifies the absolute path of the shared directory for the temporary ANP files. The path specified must be in the UNC format, (for example: \\SERVER1\DIRECTORY and not F:\DIRECTORY). Each directory can support only one service. In other words, if there are two services installed on the network there must be two separate and distinct directories
-p : <PORT>	Specifies the listening port of the TCP/IP protocol
-t : <TIMEOUT>	Specifies the time-out for the ANP protocol. If nothing is specified it will default to 30 seconds. Using the value 0 disables the time-out.
-a : <ACCOUNT>	Specifies the logon account
-w : <PASSWORD>	Specifies the password which corresponds to the specified account
-C	Cancels the current configuration

The account specified must have access rights to the directory specified. The account must be configured correctly in order to ensure access to the directory if it is located on a network resource and therefore not local to the workstation on which the installation was carried out.

In any case it is advisable to locate the directory for the sharing of temporary files directly on the workstation on which the service is installed.

The configuration is saved in the Windows registry in the path:

"HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\SmartKey Multilan Service"

13.11 Installing the DOS client with the ANP-IPX protocols

The client for DOS allows you to use the SmartKey key remotely with the MS-DOS operating system.

The installation procedure for the SmartKey client consists of configuring it using the environmental variables TYPENET, DIRNET, MESSNET.

SET TYPENET=<PROTOCOL>	Specifies the protocol to use
SET DIRNET=<DIRECTORY>	Specifies the directory to use in the ANP protocol. It is only used if the ANP protocol is active.
SET MESSNET=<MESSAGE>	Specifies the key not found message

Where <PROTOCOL> defines one of the following protocols:

LOCAL	Activate the Local protocol
ANP	Activate the ANP protocol

Some drivers require the SKEYTSR utility to be loaded before starting the program which uses the SmartKey. If this is necessary the SKEYTSR utility has to be distributed with the protected program.

13.12 Installing the client for Windows with the ANP-IPX-TCPIP protocols

The client for Windows allows you to use the SmartKey remotely with the Windows 3.x, 9x, NT 3.x/4x operating systems.

For proper installation the client has to be configured using the CSKEYCFG utility or by using the SKEYLINK.INI configuration file or environmental variables.

If the client is unable to find a network key with the protocols configured, it will always try to find a local one. This means that for local use no configuration process is necessary.

13.13 Configuring the client for Windows with the ANP-IPX-TCPIP protocols using CSKEYCFG

The CSKEYCFG utility carries out the configuration of the network protocols used by the client.

It accepts the following parameters:

-a: <DIRECTORY>	Activates the ANP protocol in the directory specified.
-t: <SERVER>: <PORT>	Activates the TCP/IP protocol towards the server and port specified
-n	Activates the Novell IPX protocol
-l	Activates the local protocol
-c	Cancels the current configuration
-r	Doesn't use the Local protocol as a last resource if all the other protocols specified fail
-d	Activates Debug mode. Every error in the use of the key will be displayed with a message box.

The specification order of the parameters defines the priority of activating the protocols. If the option -r is not specified and all the protocols fail, the local protocol is tried.

The configuration is saved in the Windows registry in the path

"HKEY_LOCAL_MACHINE\Software\Eutron\SmartKey Multilan Client".

The CSKEYCFG utility always displays the current configuration, even if it was set-up using different methods.

Configuration with the CSKEYCFG utility uses the Windows Registry of Windows 95 only and so it can't be used with Windows 3.x or by 16 bit Windows programs.

13.14 Configuring the client for Windows with the ANP-IPX-TCPIP protocols using SKEYLINK.INI

The SKEYLINK.INI configuration file allows you to manually configure the SmartKey Client.

The file SKEYLINK.INI must be in the Windows directory, usually C:\WINDOWS.

The syntax is:

[Protocol]

Protocol <NUMBER>=<PROTOCOL>

Specifies the protocol to use

Where <NUMBER> is a number in digits from 1 to 16 which indicate the priority of the protocol (the protocols with the lowest numbers are considered preferential) and <PROTOCOL> defines one of the following protocols:

LOCAL	Activates the Local protocol
TCPIP <SERVER> <PORT>	Activates the TCP/IP protocol towards the server and port specified
ANP <DIRECTORY>	Activates the ANP protocol in the directory specified
NOVELL	Activates the Novell IPX protocol

[Message]

MsgNotFound = <MESSAGGIO>	Specifies the key not found message
---------------------------	-------------------------------------

Example:

```
[Protocol]
Protocol1 = Local
Protocol2 = Tcpiip smartkey.server 13527
Protocol3 = Anp F:\SKEYANP
Protocol4 = Tcpiip smartkey2.server 13527
Protocol5 = Novell
```

[Message]

```
MsgNotFound = Key not found
```

13.15 Configuring the client for Windows with the ANP-IPX-TCPIP protocols using the old syntax of SKEYLINK.INI

The SKEYLINK.INI configuration file allows you to manually configure the SmartKey Client even using the syntax of the previous version of SmartKey.

The file SKEYLINK.INI must be in the Windows directory, usually C:\WINDOWS.

The syntax is:

[Protocol]	
Anp = <YESNO>	Activates or not the ANP protocol (the default is No)
TcpIp = <YESNO>	Activates or not the TCP/IP protocol (the default is No)
Novell = <YESNO>	Activates or not the Novell IPX protocol (the default is No)
Local = <YESNO>	Activates or not the protocol for the local key (the default is No)

[Server]

ANPShared = <DIRECTORY>	Specifies the directory to use for the ANP protocol
IpServer = <SERVER>	Specifies the server to use for the TCP/IP protocol
Port = <PORT>	Specifies the port to use for the TCP/IP protocol

[Message]

MsgNotFound = <MESSAGGIO> Specifies the key not found message

Using this syntax you can't specify the order of priority of the protocols which is fixed as follows: Local, TCPIP, IPX Novell, ANP.

Example:

[Protocol]

TcpIp=Yes

Anp=Yes

Novell=Yes

Local=Yes

[Server]

Port=13527

IPServer=smartkey.server

ANPShared=F:\SKEYANP

[Message]

MsgNotFound = Key not found

13.16 Configuring the client for Windows with the ANP-IPX-TCPIP protocols using environmental variables

The SmartKey Client can be configured using the three environmental variables TYPENET, DIRNET, MESSNET.

SET TYPENET=<PROTOCOL>	Specifies the protocol to use
SET DIRNET=<DIRECTORY>	Specifies the directory to use in the ANP protocol. This is only used if the ANP protocol is active.
SET MESSNET=<MESSAGE>	Specifies the key not found message

Where <PROTOCOL> defines one of the following protocols:

LOCAL	Activates the Local protocol
ANP	Activates the ANP protocol
NOVELL	Activates the Novell IPX protocol

You can't use the TCP/IP protocol if you configure the client using environmental variables.

For the sake of convenience it is worth while inserting the SET commands in the AUTOEXEC.BAT system file. You will have to re-start the computer for the changes to become effective.

Be careful not to use empty spaces before or after the “=” symbol. The only space you can type is the one that separates the word SET from the rest of the line.

14 Protection Guidelines

By using *SmartKey*, you have set up a very strong deterrent to software piracy attempts; however, you must remember that the common principle of all security systems applies to software protection as well:

A security system is only as strong as its weakest link.

It is therefore necessary to focus not only on the type of protection, but also on the way the software is implemented.

In other words, using a hardware protection without implementing it accurately in the software means keeping out hackers of average skill (such as occasional hackers, college students challenging the system, etc.) but not the organized criminal, with time, money and skill enough to pirate software or steal confidential data.

Based on a wide experience in the field of software and data protection, the following list of techniques and practical hints could prove helpful. The choice of the technique to be used depends on the type of application, on the cost and on the degree of secrecy of the software and/or data protected.

Generally speaking, to make life harder for software thieves the following suggestions and considerations should be kept in mind:

- Use more than one of the protection techniques set out below.
- Distribute the protection measures throughout the program.
- If a protective measure fails, all the others should go on running in such a way that the program appears to be in order for some time, after which it stops at random.

Finally, it must always be remembered that the psychological factor is highly important, too: the hacker can never be sure he has cracked all protections; if each time he thinks he has found one, the protected software raises another problem, perhaps hours or days later, he will probably be baffled enough to give up his attempts.

When you protect your application you should consider the typical attacks and try at least to make hard these most common cases. These cases are:

- Reverse engineering of the application and removal of any calls at the SmartKey API
- Use of a High Level (User Level) emulator that intercepts and registers any SmartKey API calls and tries to simulate the SmartKey API behavior. These emulators potentially know the semantic of the API calls and are able to emulate a generic SmartKey but with one exception: the *Scrambling* operation.
- Use of a Low Level (Kernel or Hardware) emulator that intercepts and registers the physical communications on the Parallel or USB port and tries to simulate the physical SmartKey behavior. Usually these emulators completely ignore the semantic of the communications.

The following guidelines can greatly enhance the protection against all these attacks.

14.1 Generic Guidelines

The following guidelines are valid for any SmartKey models.

14.1.1 Check the key in many points of your applications

The application should not check for the SmartKey presence in only one point of execution. The SmartKey check should be duplicated in many points of the program. You can do the *Locate/Open* operation only at the startup, but you must do the *Scrambling* operation in many other locations.

14.1.2 Extensive use of the Scrambling operation

The *Scrambling* operation must be used to check the presence of a real SmartKey token.

The check should be done precomputing a set of pairs of the input and of the output strings of the *Scrambling* operation. When the application runs, some pairs should be compared with the result of the same operation on the current SmartKey.

At runtime you must chose the pairs to check using a combination of both random and deterministic elements. For example, the choice should depend on: a random value, the current system time, the point of execution of your program, and any other variable that may change from an execution and another.

Example

Suppose that you identified three important points of execution where you want to check the SmartKey presence: the program startup, the save function, and the print function. You want also that the check is month based, in such a way to have 12 different time entries. Finally, you want 100 different random checks.

Therefore you need a table of $3 \times 12 \times 100 = 3600$ pairs.

14.1.3 Hide the Label and Password

The *Label* and *Password* strings should not be stored in plain text in your application. Otherwise, a simple analysis of the resulting binary may reveal these information.

A good approach is to generate a random string and compute the XOR of the original information and this random value. The original information can be reconstructed at runtime with another XOR with the original random string and the previous result.

You can also store these information in many location of your program and compare them during the program execution.

Example

With these precomputation steps:

```
LABEL = "SMARTKEY"  
RANDOM = "01234567"  
CRYPT = LABEL XOR RANDOM
```

in your source you can do:

```
CRYPT = "?????????" (previously computed)  
RANDOM = "01234567"  
LABEL = CRYPT XOR RANDOM  
SmartKeyCheckWithLabel(LABEL)
```

14.1.4 Use the .OBJ version of the drivers

If available always prefer the .OBJ version rather the .DLL version of the driver.

The .DLL driver exposes a simple and known entry point. With this entry point, it is simple to monitor and to filter all the driver calls made by your application.

For this reason the correct use of the *Scrambling* operation is extremely important to ensure that a real token is present.

14.1.5 CheckSum your Executable and DLLs

You can compute and check the CRC/Checksum of your application and DLLs. This step is very important if you want to use one of the SmartKey DLL drivers. In this way, you can ensure that your program is using the original DLL and not a faked version.

You should not use a widely known CRC algorithm. For example, the CRC32 of a file can be changed to an arbitrary value modifying only 3/4 bytes of the file. A home made algorithm or a strong cryptographic HASH function like MD5 are surely better.

14.1.6 Do not stop the execution immediately if the key isn't found

If the SmartKey check fails, the correct behavior is to not stop abruptly the program execution, but postpone the termination on another region of your code. This prevents the exposure of the point where the key is checked.

Example

This example uses the `KeyPresent` variable to store the result of the SmartKey check. Note that the variable is accessed only when the key is found. This partially prevents the use of the debugger feature that can control any access of a variable.

```

variable KeyPresent = False;

DoSomething();

if (SmartKeyPresent())
    KeyPresent = True;

DoSomethingOther();

If (KeyPresent == False)
    Abort();

```

All the access at the `KeyPresent` variable should be done at different levels of function calls.

14.1.7 Encrypt required data with the Scrambling operation

You can encrypt some values required for the correct execution of your application with the *Scrambling* operation. The Scrambling operation is only a mono directional function. However, you can use it to encrypt data using the XOR operator. You should only chose an initialization value, stored in your application, as input of the Scrambling. You can encrypt and decrypt your data with a XOR with the *Scrambling* output.

Example

With these precomputation steps:

```

DATA0 = "FIRSTDATA"
DATA1 = "OTHERDATA"
SEED = "01234567"
MASK = Scramble(SEED)
CYPHER0 = MASK XOR DATA0
MASK = Scramble(MASK)
CYPHER1 = MASK XOR DATA1
...

```

in your source you can do:

```

CYPHER0 = "?????????" (previously computed)
CYPHER1 = "?????????" (previously computed)
SEED = "01234567"
MASK = Scramble(SEED)
DATA0 = MASK XOR CYPHER0
MASK = Scramble(MASK)
DATA1 = MASK XOR CYPHER1
...

```

14.2 Memory Guidelines

The following guidelines are valid for any SmartKeys with a memory.

14.2.1 Check the memory functionality

A step against the possible use of *Hardware* and *Kernel Driver* emulators is to test the memory functionality of the SmartKey. You should simply write a random value in a memory cell and read it. If the values are different, you are in presence of an emulator of the SmartKey.

This trick works because these emulators operate at a very low level and they usually ignore the semantic of the operations but are able only to record the communication.

On the other hand, the *User Level* emulators that intercepts the SmartKey API calls can easily simulate the memory functionality. Against these emulators, you can only use the *Scrambling* operation as described previously.

14.2.2 Store required data in memory

A typical approach consists in storing some known values on the SmartKey and checking them to verify the SmartKey presence. This approach is subject of easy attacks, because you need to store a correct copy of these values in your application, and it is relatively simple to remove these controls.

Instead, you can save in the SmartKey memory some values required for the correct execution of your application. You can check their validity indirectly, i.e. using them and expecting errors. You do not need to compare them with a correct copy.

Example

Usually there are many constant values in a Windows application. You do not have any limitation on the choice of these values. You can use everything that can be easily converted in bytes.

Typical examples are:

- All the numerical constants.
- All the content of your tables.
- Any Windows define/constant: dialog IDs, string IDs, message codes, and constant function arguments.
- Names of data files, DLLs.
- Names of functions called on other DLLs.

14.3 Implementation Examples

This chapter contains some C implementation examples of the guidelines described previously. You can also find them in the archive `SmartKeyProtectionGuidelinesExample.zip`.

All the examples suppose to work with a *Demo Smartkey* with the default “SMARTKEY” *Label* and the default “EUTRON” *Password*.

14.3.1 Example 1 – Basic Use

This example shows the basic use of the SmartKey. This example is very simple and MUST NOT BE USED IN A REAL PROGRAM.

```
#include "skeylink.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    KEY_NET k;

    memset(&k, 0, sizeof(k));
    strncpy(k.label, "SMARTKEY", LABEL_LENGTH);
    strncpy(k.password, "EUTRON", PASSWORD_LENGTH);

    /* Open */
    k.net_command = NET_KEY_OPEN;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_OPEN\n");
        exit(EXIT_FAILURE);
    }

    /* Close */
    k.net_command = NET_KEY_CLOSE;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_CLOSE\n");
        exit(EXIT_FAILURE);
    }
}
```

```

        return EXIT_SUCCESS;
}

```

14.3.2 Example 2 – Basic Scrambling Use

This example shows the basic use of the scrambling operation. This example is very simple and MUST NOT BE USED IN A REAL PROGRAM.

```

#include "skeylink.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Scrambling input/output */
unsigned char scrambling_in[SCRAMBLE_LENGTH] = { 0x45, 0x34, 0x67, 0x23, 0xa5,
0x8f, 0x2c, 0x6d };
unsigned char scrambling_out[SCRAMBLE_LENGTH] = { 0x98, 0xab, 0x22, 0x24, 0xbb,
0xe6, 0x61, 0x8f };

int main() {
    KEY_NET k;

    memset(&k,0,sizeof(k));
    strncpy(k.label,"SMARTKEY",LABEL_LENGTH);
    strncpy(k.password,"EUTRON",PASSWORD_LENGTH);

    /* Open */
    k.net_command = NET_KEY_OPEN;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_OPEN\n");
        exit(EXIT_FAILURE);
    }

    /* Scrambling */
    k.net_command = NET_KEY_ACCESS;
    k.command = SCRAMBLING_MODE;
    memcpy(k.data,scrambling_in,SCRAMBLE_LENGTH);
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in SCRAMBLING_MODE\n");
        exit(EXIT_FAILURE);
    }

    if (memcmp(k.data,scrambling_out,SCRAMBLE_LENGTH)!=0) {
        printf("Wrong SCRAMBLING\n");
        exit(EXIT_FAILURE);
    }
    printf("Scramble ok\n");

    /* Close */
    k.net_command = NET_KEY_CLOSE;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_CLOSE\n");
        exit(EXIT_FAILURE);
    }

    return EXIT_SUCCESS;
}

```

14.3.3 Example 3/4 – Storing and using a C function in the SmartKey memory

This example shows how to store and to use the binary code of a C function in the SmartKey memory. You have some limitations:

- The function size must be lower or equal than the size of the SmartKey memory (64 or 416 bytes)
- You can't call directly any external functions, but you can call them indirectly passing as argument a function pointer.
- You can't use directly any external variables, but you can use them indirectly passing as argument a pointer.
- You must link your projects with the /FIXED option to prevent any relocation in your code.

Store of the function

This example stores the function `my_func()` in the SmartKey memory.

```
#include "skeylink.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Function type */
typedef int my_func_t(int m, int n);

/* Buffer used to store the function */
char my_func_data[DATA_LENGTH + EXTENDED_DATA_LENGTH];

/* Function */
static int my_func(int m, int n) {
    return m * n;
}

/* Marker of the end of the function */
static int my_func_end(void) {
    return 0;
}

int main() {
    KEY_NET k;
    unsigned size;

    /* Call the functions to prevent some optimization */
    my_func(1,1);
    my_func_end();

    memset(&k,0,sizeof(k));
    strncpy(k.label,"SMARTKEY",LABEL_LENGTH);
    strncpy(k.password,"EUTRON",PASSWORD_LENGTH);

    /* Open */
    k.net_command = NET_KEY_OPEN;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_OPEN\n");
        exit(EXIT_FAILURE);
    }

    /* Compute the function size */
    size = (char*)my_func_end - (char*)my_func;

    printf("Function size %d\n", size);
    if (size > DATA_LENGTH + EXTENDED_DATA_LENGTH) {
        printf("Function size %d too big\n", size);
    }
}
```

```

        exit(EXIT_FAILURE);
    }

    /* Copy the function on the key */
    if (size > DATA_LENGTH) {
        memcpy(k.data, ((char*)my_func), DATA_LENGTH);
        memcpy(k.ext_data, ((char*)my_func) + DATA_LENGTH, size -
DATA_LENGTH);
    } else {
        memcpy(k.data, ((char*)my_func), size);
    }

    /* Write the key */
    k.net_command = NET_KEY_ACCESS;
    k.command = WRITING_MODE;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in WRITING_MODE\n");
        exit(EXIT_FAILURE);
    }

    printf("Function written on the key\n");

    /* Close */
    k.net_command = NET_KEY_CLOSE;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_CLOSE\n");
        exit(EXIT_FAILURE);
    }

    return EXIT_SUCCESS;
}

```

Use of the function

This example read the function `my_func()` from the SmartKey memory and execute it.

```

#include "skeylink.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Function type */
typedef int my_func_t(int m, int n);

/* Buffer used to store the function */
char my_func_data[DATA_LENGTH + EXTENDED_DATA_LENGTH];

int main() {
    KEY_NET k;
    int result;
    my_func_t* my_func_ptr;

    memset(&k, 0, sizeof(k));
    strncpy(k.label, "SMARTKEY", LABEL_LENGTH);
    strncpy(k.password, "EUTRON", PASSWORD_LENGTH);

    /* Open */
    k.net_command = NET_KEY_OPEN;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_OPEN\n");
    }
}

```

```

        exit(EXIT_FAILURE);
    }

    /* Read the key */
    k.net_command = NET_KEY_ACCESS;
    k.command = READING_MODE;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in READING_MODE\n");
        exit(EXIT_FAILURE);
    }

    /* Copy the data in the buffer */
    memcpy(my_func_data, k.data, DATA_LENGTH);
    memcpy(my_func_data + DATA_LENGTH, k.ext_data, EXTENDED_DATA_LENGTH);

    /* Set the function pointer */
    my_func_ptr = (my_func_t*)my_func_data;

    /* Call the function */
    result = my_func_ptr(2,3);
    if (result != 6) {
        printf("Error in function result\n");
        exit(EXIT_FAILURE);
    }

    printf("Result of the stored function %d\n", result);

    /* Close */
    k.net_command = NET_KEY_CLOSE;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_CLOSE\n");
        exit(EXIT_FAILURE);
    }

    return EXIT_SUCCESS;
}

```

14.3.4 Example 5 – Check of the DLL checksum

This example shows how to compute and check a simple checksum of the *SMARTLINK.DLL* file.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    FILE* f;
    int c;
    unsigned checksum;

    /* Initialize the checksum */
    checksum = 0;

    /* Compute the checksum of the DLL */
    f = fopen("skeylink.dll", "rb");
    if (!f) {
        printf("Error opening the DLL\n");
        exit(EXIT_FAILURE);
    }
    c = fgetc(f);
    while (c != EOF) {
        checksum += c;
    }
}

```

```

        c = fgetc(f);
    }
    fclose(f);

    printf("DLL checksum %08X\n",checksum);

    if (checksum != 0x007ffcfl) {
        printf("Error invalid checksum\n");
        exit(EXIT_FAILURE);
    }

    return EXIT_SUCCESS;
}

```

14.3.5 Example 6 – Hiding the Label and the Password information

This example shows how to hide the *Label* and *Password* information using a simple masking algorithm.

```

#include "skeylink.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Hidden values */
static unsigned char hidden_label[LABEL_LENGTH] = {
    0x09, 0x2f, 0x2d, 0x2a, 0xd2, 0xdd, 0xed, 0xe5,
    0xd2, 0xea, 0x04, 0x20, 0x3e, 0x5e, 0x80, 0xa4
};

static unsigned char hidden_password[PASSWORD_LENGTH] = {
    0xe0, 0x91, 0xb1, 0x5a, 0x62, 0x1a, 0x7d, 0xa8,
    0xd5, 0x04, 0x35, 0x68, 0x9d, 0xd4, 0x0d, 0x48
};

int main() {
    KEY_NET k;
    unsigned i;

    memset(&k,0,sizeof(k));

    /* Compute the correct label and password */
    for(i=0;i<LABEL_LENGTH;++i)
        k.label[i] = hidden_label[i] ^ (i*(i+0x7)+0x5a);
    for(i=0;i<PASSWORD_LENGTH;++i)
        k.password[i] = hidden_password[i] ^ (i*(i+0x1e)+0xa5);

    /* Open */
    k.net_command = NET_KEY_OPEN;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_OPEN\n");
        exit(EXIT_FAILURE);
    }

    printf("Net password %d\n",k.net_password);

    /* Close */
    k.net_command = NET_KEY_CLOSE;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_CLOSE\n");
        exit(EXIT_FAILURE);
    }
}

```

```

    }

    return EXIT_SUCCESS;
}

```

14.3.6 Example 7 – Scrambling sensitive data

This example shows how to hide sensitive data with the *Scrambling* operation.

In the example the value of *pigreco* is stored in this manner.

```

#include "skeylink.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

/* Scrambled data */
unsigned char scrambled_data[SCRAMBLE_LENGTH] = {
    0x0c, 0xd8, 0xb3, 0xf6, 0x57, 0x6f, 0x4d, 0xe5
};

/* Scrambled input */
unsigned char scrambling_in[SCRAMBLE_LENGTH] = {
    0x45, 0x34, 0x67, 0x23, 0xa5, 0x8f, 0x2c, 0x6d
};

int main() {
    KEY_NET k;
    unsigned i;
    double pi;

    memset(&k, 0, sizeof(k));
    strncpy(k.label, "SMARTKEY", LABEL_LENGTH);
    strncpy(k.password, "EUTRON", PASSWORD_LENGTH);

    /* Open */
    k.net_command = NET_KEY_OPEN;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_OPEN\n");
        exit(EXIT_FAILURE);
    }

    /* Scrambling */
    k.net_command = NET_KEY_ACCESS;
    k.command = SCRAMBLING_MODE;
    memcpy(k.data, scrambling_in, SCRAMBLE_LENGTH);
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in SCRAMBLING_MODE\n");
        exit(EXIT_FAILURE);
    }

    for(i=0; i<sizeof(pi); ++i)
        ((unsigned char*)&pi)[i] = k.data[i] ^ scrambled_data[i];

    printf("Pi greco is %g\n", pi);

    /* Close */
    k.net_command = NET_KEY_CLOSE;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_CLOSE\n");
    }
}

```



```

    unsigned j;
    k.net_command = NET_KEY_ACCESS;
    k.command = SCRAMBLING_MODE;

    scramble_in(k.data,i);

    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in SCRAMBLING_MODE\n");
        exit(EXIT_FAILURE);
    }

    fprintf(f,"{ ");
    for(j=0;j<SCRAMBLE_LENGTH;++j) {
        unsigned v = k.data[j];
        if (j)
            fprintf(f,", ");
        fprintf(f,"0x%02x",v);
    }
    fprintf(f," }");
    if (i+1!=SCRAMBLE_MAX)
        fprintf(f,",");
    fprintf(f,"\n");
}
fprintf(f,"};\n");
fclose(f);

printf("Scrambling table written\n");

/* Close */
k.net_command = NET_KEY_CLOSE;
smartlink(&k);
if (k.status != ST_OK) {
    printf("Error in NET_KEY_CLOSE\n");
    exit(EXIT_FAILURE);
}

return EXIT_SUCCESS;
}

```

Use of the Scrambling table

This example uses the generated file to check the presence of the SmartKey.

```

#include "skeylink.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "table.h"

/* Return a random index in the table */
unsigned get_scrambling_index(void) {
    time_t t;
    struct tm* ptm;
    unsigned i;

    time(&t);
    ptm = localtime(&t);

    i = (rand() % (SCRAMBLE_MAX / 31)) * 31;
    i += ptm->tm_mday;
}

```

```

        i = i % SCRAMBLE_MAX;

        return i;
    }

int main() {
    KEY_NET k;
    unsigned i;

    /* Initialized the random number generator */
    srand(time(0));

    memset(&k,0,sizeof(k));
    strncpy(k.label,"SMARTKEY",LABEL_LENGTH);
    strncpy(k.password,"EUTRON",PASSWORD_LENGTH);

    /* Open */
    k.net_command = NET_KEY_OPEN;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_OPEN\n");
        exit(EXIT_FAILURE);
    }

    /* Get the random index in the table */
    i = get_scrambling_index();
    printf("Scramble index %d\n",i);

    /* Do the scrambling */
    k.net_command = NET_KEY_ACCESS;
    k.command = SCRAMBLING_MODE;
    scramble_in(k.data,i);
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in SCRAMBLING_MODE\n");
        exit(EXIT_FAILURE);
    }

    /* Check the scramble */
    if (memcmp(k.data,SCRAMBLE[i],SCRAMBLE_LENGTH)!=0) {
        printf("Wrong SCRAMBLING\n");
        exit(EXIT_FAILURE);
    }
    printf("Scramble ok\n");

    /* Close */
    k.net_command = NET_KEY_CLOSE;
    smartlink(&k);
    if (k.status != ST_OK) {
        printf("Error in NET_KEY_CLOSE\n");
        exit(EXIT_FAILURE);
    }

    return EXIT_SUCCESS;
}

```

The file “table.h” generated

This is the generated file table.h

```

void scramble_in(unsigned char* dst, unsigned src) {
    unsigned i;
    for(i=0;i<SCRAMBLE_LENGTH;++i)
        dst[i] = ((src + 0x5a) >> i) * (i + 0x13) ^ 0x3e;
}

```

```

}

#define SCRAMBLE_MAX 1024

unsigned char SCRAMBLE[SCRAMBLE_MAX][SCRAMBLE_LENGTH] = {
{ 0xa2, 0x43, 0x2d, 0xdc, 0xf0, 0x49, 0x4b, 0x5c },
{ 0x00, 0x24, 0x9f, 0x6e, 0x51, 0x10, 0x9c, 0x1a },
...stripped...
{ 0x10, 0x0f, 0x5e, 0x8e, 0x5b, 0x44, 0x67, 0x11 },
{ 0xcd, 0x0a, 0x74, 0xed, 0x78, 0xc0, 0x0a, 0x97 }
};

```

14.3.8 Example 10 – Code Encryption

You can also use the SmartKey to encrypt the code of your .EXE or .DLL file if they are developed in VisualC. You can find an extensive and complete example in the archive `SmartKeyEncryptionGuidelinesExample.zip`. For major details please refer at the included `README.txt` (English) and `LEGGIMI.txt` (Italian) file in the .zip.

15 TECHNICAL SPECIFICATIONS

15.1 Features

Protection mechanism: algorithmic and/or password

Access codes: fixed or programmable 16+16 bytes

Secure data memory: 64/416 bytes Data

Programming: on-line and/or off-line

Capability to detect fraudulent access attempts

Capability to *freeze* data stored

15.2 SmartKey 2 Parallel

Number of write operations: 100000 (typically)

Data storage: 10 years (typically)

Module dimensions: 48 x 52 x 15 mm

Interface: standard Centronics parallel port

Connector computer side: D-type 25 pins male

Connector printer side: D-type 25 pins female

Power feed: No external supply need

Expected MTBF: 2.800.000 hours

Temperature: 0-70°C

Humidity: 20-80 % relative humidity

15.3 SmartKey 2 USB

Number of write operations: 100000 (typically)

Data storage: 10 years (typically)

Interface: USB 1.1 LowSpeed

Connector: USB Male Type A

Power feed: No external supply need

Expected MTBF: 2.000.000 hours

Temperature: 0-70°C

Humidity: 20-80 % relative humidity

15.4 SmartKey 3 USB

Number of write operations: 100000 (typically)

Data storage: 10 years (typically)

Interface: USB 2.0 LowSpeed

Connector: Male Type A

Power feed: No external supply need

Expected MTBF: 3.000.000 hours

Operating Temperature: -20 - +80°C

Humidity: 20-95% relative humidity