Copy Protection System

Developer Manual

for Microsoft Windows XP to 10 (all 32/64-bit), CE, Linux X86/A64/ARM and Mac OS X
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5</td>
<td>Administrative Functions</td>
<td>31</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Function: SglReadProductId</td>
<td>31</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Function: SglWriteProductId</td>
<td>32</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Function: SglWriteKey</td>
<td>33</td>
</tr>
<tr>
<td>4.5.4</td>
<td>Function: SglReadConfig</td>
<td>34</td>
</tr>
<tr>
<td>4.6</td>
<td>Return Values</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>Encryption, Signing and Key Management</td>
<td>37</td>
</tr>
<tr>
<td>6</td>
<td>Programming Examples</td>
<td>39</td>
</tr>
<tr>
<td>6.1</td>
<td>Function SglAuthent</td>
<td>39</td>
</tr>
<tr>
<td>6.1.1</td>
<td>C/C++</td>
<td>39</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Delphi</td>
<td>39</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Visual Basic</td>
<td>40</td>
</tr>
<tr>
<td>6.2</td>
<td>Function SglSearchLock</td>
<td>42</td>
</tr>
<tr>
<td>6.2.1</td>
<td>C/C++</td>
<td>42</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Delphi</td>
<td>42</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Visual Basic</td>
<td>43</td>
</tr>
<tr>
<td>6.3</td>
<td>Function SglReadSerialNumber</td>
<td>44</td>
</tr>
<tr>
<td>6.3.1</td>
<td>C/C++</td>
<td>44</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Delphi</td>
<td>44</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Visual Basic</td>
<td>45</td>
</tr>
<tr>
<td>6.4</td>
<td>Function SglReadData</td>
<td>46</td>
</tr>
<tr>
<td>6.4.1</td>
<td>C/C++</td>
<td>46</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Delphi</td>
<td>46</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Visual Basic</td>
<td>47</td>
</tr>
<tr>
<td>6.5</td>
<td>Function SglWriteData</td>
<td>49</td>
</tr>
<tr>
<td>6.5.1</td>
<td>C/C++</td>
<td>49</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Delphi</td>
<td>49</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Visual Basic</td>
<td>50</td>
</tr>
<tr>
<td>6.6</td>
<td>Challenge-Response-Authentification of a SG-Lock</td>
<td>52</td>
</tr>
<tr>
<td>7</td>
<td>Technical Data</td>
<td>55</td>
</tr>
<tr>
<td>7.1</td>
<td>SG-Lock U2/U3/U4</td>
<td>55</td>
</tr>
</tbody>
</table>
1 Introduction

SG-Lock is an innovative hardware based copy protection system, that can be used with all 32- and 64-bit Windows, Linux and Mac OS X operating systems. Also Windows CE is supported.

**Outstanding features are:**

- Every SG-Lock has its own unique serial number.
- Up to 1024 bytes SG-Lock internal free useable encrypted memory.
- 128-bit encryption with up to 16 free programmable keys.
- Up to 64 free usable counters for easy logging of countable events.
- The USB SG-Lock can be installed without driver installation and admin rights (Windows XP to 10).

**Special security features:**

- The whole SG-Lock internal memory is transparent for the user encrypted and signed with a unique 128-bit key. Hardware attacks like manipulation of single data values or exchange of the whole memory will be detected and prevented.
- Simple and efficient challenge response authentication mechanism between protected application and SG-Lock API. The SG-Lock API is not like often implemented immediately useable in its full functionality. On principle every application has to authenticate itself before it gets access to the SG-Lock API. This mechanism prevents attacks of not authorized applications. In addition the protected application can itself verify the SG-Lock API and detect a doubtful library. The authentication mechanism is done by calling only one function with one parameter.
Chapter 1: Introduction

Figure 1.1: SG-Lock U-Series

- The SG-Lock API works with a module internal as well as with an application internal 128-bit TEA (Tiny Encryption Algorithm) encryption engine. This symmetrical (key for encryption and decryption are identical) and secure encryption algorithm is the basis for various implementations of data and code protection as well as authentication strategies.

- The single SG-Lock models are offered in two different shapes, which are however functional identical and therefore fully exchangeable. The S-series is because of the small size espacially suitable for notebook and tablet computers. The U-series is approved by its ergonomic form due to its bigger size mainly for desktop computers.
2 Installation and Tools

The installation of SG-Lock is quite simple and transparent structured to ease the integration into the installation of the protected application.

2.1 32/64-bit Operating Systems

The 32-bit SG-Lock libraries are generally used for all 32-bit applications - even when the 32-bit application is running under a 64-bit operating system. The 64-bit versions of the SG-Lock libraries are only used, when a 64-bit application has to be protected.

2.2 Windows XP to 10

To install SG-Lock USB keys process the following 2 steps:

1. Copy the SG-Lock library SGLW32.DLL (32- or 64-bit version) in the Windows system directory (for example C:\WINDOWS\SYSTEM32) or into the installation directory of the protected application.

2. Plug the SG-Lock USB key into the USB port. A small window will show up for a short time giving the information, that the SG-Lock can now be used. Now the installation has successfully finished and the SG-Lock USB is ready for use.

2.3 Linux and Mac OS X

Please take a look onto the SG-Lock SDK CD-ROM for installation instructions.
Figure 2.1: The SG-Lock API (the file SGLW32.DLL) establishes the connection between protected application and SG-Lock hardware.
2.4 Windows CE 4, 5 and 6

Please note, that only the SG-Lock U4 supports Windows CE.

1. Copy the file SGLWCE.DLL in the application or the system directory (e.g. \Windows). This can also be done by a script at system start.

2. Copy the file SGLUSB.DLL in a directory that exists at system start (e.g. \Storage).

3. Adjust both keys with name DLL in the registry script SGLUSB.REG according the path of SGLUSB.DLL (if e.g. your SGLUSB.DLL resides in \Storage, change the values of both keys to Storage\SGLUSB.DLL). Do not use a preceding backslash. Leave the keys PREFIX unchanged.

4. Execute the adjusted registry script and save the registry (e.g. with the AP CONFIG MANAGER in file card APSystem, button STORAGE/REGISTRY/SAVE), to keep the configuration for further system starts.

2.5 Deinstallation

The deinstallation is simply done by deleting the copied files and - when conducted during the installation - also deleting the registry keys.
2.6 Edit SG-Lock with the SG-Lock Manager

The SG-Lock Manager (SGLMGR) is a tool delivered on the SG-Lock CDROM for testing and editing all SG-Lock moduls.

Run the SGLMGR by starting the file SglMgr.exe in the directory Test. The index card Options offers to change the language with Select Language. Additionally the number radix can be changed between decimal and hexadecimal representation. This has also to be taken into account when entering numbers!

All functions, that the SGLMGR offers, are part of the SG-Lock API and can
also be used by protected applications. The index card Properties offers with the push button Read Properties the display of important information like type, serial number, product id and version numbers of the attached SG-Lock.

With the push button Write Product ID the value can be altered between 0 and 65535 (dec.). The function of the product id is described in detail in chapter 3.3.

The index card Memory makes it possible to read and write the module internal memory (when existing). For changing one or more data values they have to be entered into the table and by pushing the push button Write Memory they
are written into the module internal memory. Before writing any value read the memory first.

The index card Counter offers the same functions for the counter memory. They are not located in the data memory area, but use additional memory - which excludes side effects.

Index card Cryptography offers the possibility to use the cryptographic functions of SG-Lock. SG-Lock uses a module internal symmetric (i.e. the keys for encryption and decryption are identical) 128-bit encryption. The data block length is 64-bit and the algorithm is TEA. The SG-Lock series 3 and 4 have multiple key
storage locations. To change a key the key number has to be chosen with *Use Key No.* first, after that enter the key used for encryption. The key will be written to its modul internal memory location by pushing the button *Write Key*. With the push button *Generate Random Key* a 128-bit key can be generated automatically. It can be processed further over the clipboard e.g. for documentation purposes (this is recommended, because the key can not be read for security reasons).

For testing the encryption engine two 32-bit values (which are equivalent to one 64-bit block) can be entered into the fields *Input Data*. By pressing *Encrypt* or *Decrypt Data* the 64 bits input data block will be en- or decrypted. The result is shown in the fields *Output Data*. 

![SG-Lock Manager Ver 1.13](image)

**Authentocode**

|  F574D17B |  A94628EE |  F2857A8F |  69346B4A |
|  4136E8F2 |  89ADC688 |  80C2C1D4 |  A8C6327C |
|  1A72699A |  574B7CA0 |  1E8D3E98 |  D7DEFDC5 |

**Select Language**

- [ ] English

**Number Radix**

- [ ] Decimal
- [x] Hexadecimal

**Status:** Encryption successful!
On the index card *Options* the language and radix of the displayed numbers (except version numbers on index card Properties are always decimal) can be chosen. When choosing hexadecimal numbers, the input is also hexadecimal without leading or trailing special characters.

**Attention:** The input of an authent code (AC) is necessary when not-demo (retail) SG-Lock modules are used. Without entering a authent code only demo-modules will be detected. Every software manufacturer, that uses SG-Lock, gets ones with the first delivery his individual AC. The AC is presented in hexadecimal style - the number radix has to be altered if necessary when entering the AC.
3 Protecting Software with SG-Lock

3.1 Introduction

The operation of SG-Lock as copy protection system is based on the call of certain functions, that establish a connection between an easy to copy software (that therefore has to be protected) and the practically impossible to copy SG-Lock hardware.

The for that used functions are the functions of the SG-Lock API (application programming interface). They are contained in the software library SGLW32.DLL, that is delivered with the SG-Lock hardware. The SG-Lock API provides varying types of functions, of which according to the type of protection different are used. For an effective software protection not all of the functions have to be used.

3.2 Protection Strategies

The most frequently applied type of protection of software against illegal use is the simple "run or run not" copy protection, that avoids that the software runs on more PCs than paid for. In this case the repeatedly test whether the copy protection key is installed on the PC or not is the main task.

Other protection strategies shall allow the software to run only a limited number of starts. For that a counter, that logs the number of starts, is needed additionally beside the existance of the copy protection key. SG-Lock provides counter variables for purposes like that.

A similar limit to run software is the use of an application only up to a certain date. In that case a date is stored in the SG-Lock memory to verify at program start and during software use, that has not been reached.

Another alternative to control software use is "pay per use". The user pays
only for the use of central features of a software, e.g. when chemical analysis software performs one analysis.

### 3.3 The SG-Lock Product ID - what is it good for?

Almost all functions of the SG-Lock API use a parameter called "ProductID". What is that good for? In many cases a software company sells more than one software product. When using a common copy protection key the programmer has to regard that an installed copy protection key is valid for the software just running and not for some of the other products they are also selling. That is additional administrative work and an additional source of error.

SG-Lock solves this with the Product ID. Every software product and the belonging SG-Locks get their own Product ID, e.g. software A gets Product ID 1, software B gets 2 and so on. In the source code of software A the parameter of the SG-Lock API functions is always 1, in B always 2 and so on - the SG-Lock API selects the right SG-Lock for you.
Figure 3.1: The SG-Lock ProductId allows an easy separation of different products of a manufacturer. The Authentcode separates manufactures strictly from each other.
3.4 Encryption and Challenge-Response-Authentication of SG-Lock

SG-Lock provides a special security feature based on an encryption algorithm. It is called a challenge-response-authentication. It gives a maximum of security by verifying the whole path from the protected application through the SG-Lock library, through the operations system and the physical interface (e.g. USB-bus) to the internals of the SG-Lock copy protection key.

The procedure is based on the 128-bit TEA (Tiny Encryption Algorithm) encryption engine implemented in the SG-Lock hardware and big random numbers that have to be encrypted.

A simple example how to implement that feature can be found in the chapter "Programming Examples". The example is written in C/C++, but can also easily be coded in other programming languages (e.g. Delphi, VB, etc.).
Figure 3.2: The SG-Lock Challenge-Response-Authentication provides a secure connection from the protected application (EXE-file) through the whole operating system over the USB-bus to the SG-Lock token itself.
4 SG-Lock API

4.1 Function summary

The SG-Lock API functions can be divided into the 4 groups: basic functions, extended functions, cryptographic and administrative functions. The basic functions, that are essential for most software protection approaches, like e.g. checking if a SG-Lock is actually plugged in a port of the PC. The extended functions with special capabilities provide functionalities for special intended aims, e.g. memory and counters, which can be used to store strings or counters to limit program starts. Functions of that group are used, when certain protection strategies are pursued.

The group of administrative functions is primarily used for the preparation purposes of the SG-Lock hardware prior delivery to the software users. They are normally not implemented in the source code of the protected application. Instead they are used to build small and simple initialisation applications to prepare SG-Lock devices by the software company before shipment to the users.
<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic functions</strong></td>
<td></td>
</tr>
<tr>
<td>SglAuthent</td>
<td>Authentification of the SG-Lock library</td>
</tr>
<tr>
<td>SglSearchLock</td>
<td>Searches for a SG-Lock device</td>
</tr>
<tr>
<td>SglReadSerialNumber</td>
<td>Reads the serial number of a SG-Lock device</td>
</tr>
<tr>
<td><strong>Extended functions</strong></td>
<td></td>
</tr>
<tr>
<td>SglReadData</td>
<td>Reads data from the memory of a SG-Lock device</td>
</tr>
<tr>
<td>SglWriteData</td>
<td>Writes data to the memory of a SG-Lock device</td>
</tr>
<tr>
<td>SglReadCounter</td>
<td>Reads a counter value from a SG-Lock device</td>
</tr>
<tr>
<td>SglWriteCounter</td>
<td>Writes a counter value to a SG-Lock device</td>
</tr>
<tr>
<td><strong>Cryptographic functions</strong></td>
<td></td>
</tr>
<tr>
<td>SglCryptLock</td>
<td>En- or decrypts one or more data blocks with the SG-Lock device using a SG-Lock device internal key</td>
</tr>
<tr>
<td>SglSignDataApp</td>
<td>Sign or verify data with the PC</td>
</tr>
<tr>
<td>SglSignDataLock</td>
<td>Sign or verify data with a SG-Lock device</td>
</tr>
<tr>
<td>SglSignDataComb</td>
<td>Sign or verify data with a combination of PC and SG-Lock device</td>
</tr>
<tr>
<td><strong>Administrative functions</strong></td>
<td></td>
</tr>
<tr>
<td>SglReadProductId</td>
<td>Reads the ProductID from a SG-Lock device</td>
</tr>
<tr>
<td>SglWriteProductId</td>
<td>Writes a ProductID to a SG-Lock device</td>
</tr>
<tr>
<td>SglWriteKey</td>
<td>Writes a 128-bit key to a SG-Lock device</td>
</tr>
<tr>
<td>SglReadConfig</td>
<td>Reads configuration data from the SG-Lock environment or a SG-Lock device (e.g. type of SG-Lock)</td>
</tr>
</tbody>
</table>
4.2 Basic functions

4.2.1 Function: SglAuthent

Description
Authentication of SG-Lock library to the protected application and vice versa.

Types
U2: ✓ U3: ✓ U4: ✓

Declaration

```c
ULONG SglAuthent(
    ULONG * AuthentCode );
```

Parameters

| AuthentCode | 48 byte sequence, that is unique to every SG-Lock customer |

Return values

| SGL_SUCCESS                  | Authentication successfull |
| SGL_AUTHENTICATION_FAILED   | Authentication failed      |

The full list of return codes is listed in chapter 4.6.

Comments

This function of the SG-Lock API has to be called once first and successfully to enable all other API functions. In the case of dynamic linking the authentication is required after every link procedure (LoadLibrary call).

Every customer gets his unique Authentcode when purchasing SG-Lock (non Demo) the first time. Demo kits have an own authentication code (see example source code files).
4.2.2 Function: SglSearchLock

Description

Searches for a SG-Lock device.

Types

| U2: ✓ | U3: ✓ | U4: ✓ |

Declaration

```c
ULONG SglSearchLock(
    ULONG ProductId);
```

Parameters

| ProductId | Indicates the ProductId of the SG-Lock looked for |

Return values

| SGL_SUCCESS | SG-Lock found |
| SGL_DGL_NOT_FOUND | SG-Lock not found |

The full list of return codes is listed in chapter 4.6.
4.2.3 Function: SglReadSerialNumber

Description
Reads the for every SG-Lock unique serial number.

Types
U2: ✓ U3: ✓ U4: ✓

Declaration

```
ULONG SglReadSerialNumber(
    ULONG ProductId,
    ULONG *SerialNumber);
```

Parameters

<table>
<thead>
<tr>
<th>ProductId</th>
<th>Indicates the ProductId of the SG-Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>SerialNumber</td>
<td>Points to variable, in which the serial number will be given back to the calling application</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>SGL_SUCCESS</th>
<th>SG-Lock serial number successfully read</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGL_DGL_NOT_FOUND</td>
<td>SG-Lock not found</td>
</tr>
</tbody>
</table>

The full list of return codes is listed in chapter 4.6.

Comments
Every SG-Lock has a serial number that is unique, which is also not depending on type and interface.
4.3 Extended Function

4.3.1 Function: SglReadData

Description
Read 32-bit data from the SG-Lock memory.

Types
U2: - U3: ✓ U4: ✓

Declaration

```
ULONG SglReadData(
    ULONG ProductId,
    ULONG Address,
    ULONG Count,
    ULONG * Data);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductId</td>
<td>Indicates the ProductId of the SG-Lock</td>
</tr>
<tr>
<td>Address</td>
<td>Start address of data value block</td>
</tr>
<tr>
<td></td>
<td>0 to 63 - SG-Lock U3</td>
</tr>
<tr>
<td></td>
<td>0 to 255 - SG-Lock U4</td>
</tr>
<tr>
<td>Count</td>
<td>Number of data values</td>
</tr>
<tr>
<td>Data</td>
<td>Pointer to data array, in which the data values will be given back to the calling application. (The developer is responsible to provide an array with a sufficient size).</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGL_SUCCESS</td>
<td>Data values successfully read</td>
</tr>
<tr>
<td>SGL_DGL_NOT_FOUND</td>
<td>SG-Lock not found</td>
</tr>
</tbody>
</table>

The full list of return codes is listed in chapter 4.6.
4.3.2 Function: SglWriteData

Description

Writes 32-bit data values to SG-Lock memory.

Types

U2:  -  U3: ✓  U4: ✓

Declaration

```c
ULONG SglWriteData(
    ULONG ProductId ,
    ULONG Address ,
    ULONG Count ,
    ULONG * Data );
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductId</td>
<td>Indicates the ProductId of the SG-Lock</td>
</tr>
<tr>
<td>Address</td>
<td>Start address of data value block</td>
</tr>
<tr>
<td></td>
<td>0 to 63 - SG-Lock U3</td>
</tr>
<tr>
<td></td>
<td>0 to 255 - SG-Lock U4</td>
</tr>
<tr>
<td>Count</td>
<td>Number of data values</td>
</tr>
<tr>
<td>Data</td>
<td>Pointer to data array, where data values be copied from. (The developer is responsible to provide an array with a sufficient size).</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGL_SUCCESS</td>
<td>Data values successfully written to SG-Lock memory</td>
</tr>
<tr>
<td>SGL_DGL_NOT_FOUND</td>
<td>SG-Lock not found</td>
</tr>
</tbody>
</table>

The full list of return codes is listed in chapter 4.6.
### 4.3.3 Function: 

**SglReadCounter**

**Description**

Reads a 32-bit count value from the SG-Lock memory.

**Types**

<table>
<thead>
<tr>
<th></th>
<th>U2:</th>
<th>U3:</th>
<th>U4:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Declaration**

```c
ULONG SglReadCounter ( 
    ULONG ProductId ,
    ULONG CntNum ,
    ULONG * Data );
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductId</td>
<td>Indicates the ProductId of the SG-Lock</td>
</tr>
<tr>
<td>CntNum</td>
<td>Number of counter</td>
</tr>
<tr>
<td></td>
<td>0 to 15 - SG-Lock U3</td>
</tr>
<tr>
<td></td>
<td>0 to 63 - SG-Lock U4</td>
</tr>
<tr>
<td>Data</td>
<td>Pointer to variable, that the counter value is assigned to</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGL_SUCCESS</td>
<td>Count value successfully read</td>
</tr>
<tr>
<td>SGL_DGL_NOT_FOUND</td>
<td>SG-Lock not found</td>
</tr>
</tbody>
</table>

The full list of return codes is listed in chapter 4.6.

**Comments**

Counters are simple 32 bit data values in the SG-Lock memory. If desired, they can also be used for everything a 32 bit read/write variable is suitable for. By doing so you can extend the general purpose memory over the indicated size.
4.3.4 Function: SglWriteCounter

Description
Writes a 32-bit count value to the SG-Lock memory.

Types
U2: ✓  U3: ✓  U4: ✓

Declaration

```c
ULONG SglWriteCounter(
    ULONG ProductId,
    ULONG CntNum,
    ULONG Data);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductId</td>
<td>Indicates the ProductId of the SG-Lock</td>
</tr>
<tr>
<td>CntNum</td>
<td>Number of counter</td>
</tr>
<tr>
<td></td>
<td>0 to 63 - SG-Lock U3</td>
</tr>
<tr>
<td></td>
<td>0 to 255 - SG-Lock U4</td>
</tr>
<tr>
<td>Data</td>
<td>Counter value to be written</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGL_SUCCESS</td>
<td>Count value successfully written</td>
</tr>
<tr>
<td>SGL_DGL_NOT_FOUND</td>
<td>SG-Lock not found</td>
</tr>
</tbody>
</table>

The full list of return codes is listed in chapter 4.6.

Comments
See SglReadCounter.
4.4 Cryptographic and Signing functions

4.4.1 Function: SglCryptLock

Description
En- or decrypts one or more 64-bit data blocks with 128-bit key. The cryptographic algorithm is TEA.

Types
U2: ✓  U3: ✓  U4: ✓

Declaration

```
ULONG SglCryptLock(
    ULONG ProductId,
    ULONG KeyNum,
    ULONG CryptMode,
    ULONG BlockCnt,
    ULONG * Data);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductId</td>
<td>Indicates the ProductId of the SG-Lock</td>
</tr>
<tr>
<td>KeyNum</td>
<td>Number of key to use</td>
</tr>
<tr>
<td></td>
<td>0 to 1 - SG-Lock U3</td>
</tr>
<tr>
<td></td>
<td>0 to 15 - SG-Lock U4</td>
</tr>
<tr>
<td>CryptMode</td>
<td>Working mode</td>
</tr>
<tr>
<td></td>
<td>0 - Encrypt</td>
</tr>
<tr>
<td></td>
<td>1 - Decrypt</td>
</tr>
<tr>
<td>BlockCnt</td>
<td>Number of data blocks to en- or decrypt</td>
</tr>
<tr>
<td>Data</td>
<td>Pointer to data array, where values shall be copied to. (The developer is responsible to provide an array with a sufficient size).</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGL_SUCCESS</td>
<td>En-/Decryption successfully finished</td>
</tr>
<tr>
<td>SGL_DGL_NOT_FOUND</td>
<td>SG-Lock not found</td>
</tr>
</tbody>
</table>

The full list of return codes is listed in chapter 4.6.

Comments

The function uses destructive data processing mode. That means the input of the parameter Data will be overwritten during execution of the function.
4.4.2 Function: SglSignData

Description

Signs or verifies the signature of a data array. The task will be processed by the SG-Lock and if desired also by the application (PC-CPU) to accelerate the signing process (combined mode). **Important condition for combined mode:** Both keys (application and SG-Lock internal) have to be different, to ensure highest security! The signature is 64-bit long.

Types

U2: ✓  U3: ✓  U4: ✓

Declaration

```c
ULONG SglSignData( 
    ULONG ProductId, 
    ULONG *AppSignKey, 
    ULONG LockSignKeyNum, 
    ULONG Mode, 
    ULONG LockSignInterval, 
    ULONG DataLen, 
    ULONG *Data, 
    ULONG *Signature );
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductId</td>
<td>Indicates the ProductId of the SG-Lock</td>
</tr>
<tr>
<td>AppSignKey</td>
<td>The 128-bit key used by the application for signing and verification. Pointer to an array of 4 integer values of 32-bit. Only needed in combined mode otherwise set 0.</td>
</tr>
<tr>
<td>LockSignKeyNum</td>
<td>Number of 128-bit key in SG-Lock for signing and verification.</td>
</tr>
</tbody>
</table>
| Mode             | working mode  
|                  | 0 - create signature  
|                  | 1 - verify signature  |
| LockSignInterval | Indicates the partitioning of computing power between SG-Lock and application (PC-CPU). 0 is SG-Lock only. If >0 the value is used as the power of 2, where the result of that determines which block index is signed or verified by the SG-Lock. E.g. value= 8, $2^8=256$, that means the first and after that every 256th block is processed by the SG-Lock and all others by the application (PC-CPU). That means $1/256= 0.4\%$ of the task is done by the SG-Lock and $99.6\%$ by the PC-CPU. The result is a very high acceleration of the process. |
| DataLen          | Count of 32-bit values to process in data array.                                                                                                                                                           |
| Data             | Pointer to the array of 32-bit values to sign or verify.                                                                                                                                                   |
| Signature        | Pointer to an array of 2 integers of 32-bit where the created signature is given back (signing mode) or where the signature is given to the function for verification (verify mode).                                |
Return values

<table>
<thead>
<tr>
<th>SGL_SUCCESS</th>
<th>Signature successfully created or signature valid (depending on used mode).</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGL_SIGNATURE_INVALID</td>
<td>Signature invalid</td>
</tr>
</tbody>
</table>

The full list of return codes is listed in chapter 4.6.

Comments

The same value for the parameter LockSignInterval has to be used for signing and verifying a certain data block, when using different 128-bit keys in the SG-Lock and application (highly recommended for highest security). The reasons is, that the 128-bit key in the application is less safe than the 128-bit key in the SG-Lock. When a hacker succeeds to investigate the 128-bit key in the application (which is in principle possible), then he will try that also first for the SG-Lock. That will fail, if a different 128-bit key is used in the SG-Lock. For a deeper understanding of the function, please take a look into the SG-Lock include/header file for your programming language. You will find the source code of the function in it.
4.5 Administrative Functions

4.5.1 Function: SglReadProductId

Description

Reads the 16-bit ProductId from the SG-Lock.

Types

<table>
<thead>
<tr>
<th>Types</th>
<th>✓</th>
<th>✓</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2:</td>
<td>✓</td>
<td>U3:</td>
<td>✓</td>
</tr>
</tbody>
</table>

Declaration

```c
ULONG SglReadProductId ( 
    ULONG* ProductId );
```

Parameters

| ProductId | Pointer to variable, that the ProductId assigned to |

Return values

<table>
<thead>
<tr>
<th>SGL_SUCCESS</th>
<th>ProductId successfully read</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGL_DGL_NOT_FOUND</td>
<td>SG-Lock not found</td>
</tr>
</tbody>
</table>

The full list of return codes is listed in chapter 4.6.

Comments

The ProductId is an identifier that eases to distinguish between different protected applications of SG-Lock users. For example company X protects its application A and B with SG-Lock and gives all keys for application A the ProductId 1 and the keys for application B the ProductId 2, then all keys of application B are ”hidden“ for application A and vice versa. This simple mechanism offers an effective way to prevent confusion between keys of different applications (see also chapter 3.3.). Setting of the ProductId should be integrated into the initialization process of the SG-Locks before distributing with the protected software.
4.5.2 Function: SglWriteProductId

Description

Writes a new 16-bit ProductId to the SG-Lock.

Types

| U2: ✓ | U3: ✓ | U4: ✓ |

Declaration

```
ULONG SglWriteProductId(  
    ULONG OldProductId,   
    ULONG NewProductId );
```

Parameters

<table>
<thead>
<tr>
<th>OldProductId</th>
<th>Indicates the actual ProductId of the SG-Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewProductId</td>
<td>Indicates the new ProductId of the SG-Lock</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>SGL_SUCCESS</th>
<th>ProductId successfully written</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGL_DGL_NOT_FOUND</td>
<td>SG-Lock not found</td>
</tr>
</tbody>
</table>

The full list of return codes is listed in chapter 4.6.

Comments

See SglReadProductId.
4.5.3 Function: SglWriteKey

Description

Writes a 128-bit key to the SG-Lock key memory.

Types

U2: - U3: ✓ U4: ✓

Declaration

ULONG SglWriteKey(
    ULONG ProductId,
    ULONG KeyNum,
    ULONG *Key);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductId</td>
<td>Indicates the ProductId of the SG-Lock</td>
</tr>
<tr>
<td>KeyNum</td>
<td>Number of the key to be written</td>
</tr>
<tr>
<td></td>
<td>0 to 1 - SG-Lock U3</td>
</tr>
<tr>
<td></td>
<td>0 to 15 - SG-Lock U4</td>
</tr>
<tr>
<td>Key</td>
<td>128-bit Key to be written. Pointer to data array of 4 integer values of 32-bit, that form the 128-bit key</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGL_SUCCESS</td>
<td>Key successfully written to SG-Lock</td>
</tr>
<tr>
<td>SGL_DGL_NOT_FOUND</td>
<td>SG-Lock not found</td>
</tr>
</tbody>
</table>

The full list of return codes is listed in chapter 4.6.

Comments

The 128-bit key of the U2 is read only and not changeable.
4.5.4 Function: SglReadConfig

Description

Reads configuration information about SG-Lock.

Types

| U2 | ✓ |
| U3 | ✓ |
| U4 | ✓ |

Declaration

```c
ULONG SglReadConfig(
    ULONG ProductId,
    ULONG Category,
    ULONG * Data);
```

Parameters

<table>
<thead>
<tr>
<th>ProductId</th>
<th>Indicates the ProductId of the SG-Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Type of requested information</td>
</tr>
<tr>
<td></td>
<td>0: Information about SG-Lock modul</td>
</tr>
<tr>
<td>Data</td>
<td>Pointer to Data array of 8 integers of 32-bit</td>
</tr>
<tr>
<td></td>
<td>The meaning of the single values are:</td>
</tr>
<tr>
<td></td>
<td>Index 0: Type</td>
</tr>
<tr>
<td></td>
<td>Index 1: Interface</td>
</tr>
<tr>
<td></td>
<td>Index 2: Software Version</td>
</tr>
<tr>
<td></td>
<td>Index 3: Hardware Version</td>
</tr>
<tr>
<td></td>
<td>Index 4: Serial number</td>
</tr>
<tr>
<td></td>
<td>Index 5: Memory size in Dwords</td>
</tr>
<tr>
<td></td>
<td>Index 6: Number of counters</td>
</tr>
<tr>
<td></td>
<td>Index 7: Number of 128-Bit Keys</td>
</tr>
</tbody>
</table>

Return values

| SGL_SUCCESS | Information successfully read |
| SGL_DGL_NOT_FOUND | SG-Lock not found |

The full list of return codes is listed in chapter 4.6.
Comments

Further information to certain values can be found in the include and include/-header files of the SG-Lock API.
4.6 Return Values

Every SG-Lock API function gives a return value back to the caller, to check if the function was executed without errors. In the case that an error occurred, the return value is unequal to 0. A detailed explanation of the error can be obtained from the table below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGL_SUCCESS</td>
<td>0</td>
<td>Execution without errors</td>
</tr>
<tr>
<td>SGL_DGL_NOT_FOUND</td>
<td>1</td>
<td>SG-Lock not found</td>
</tr>
<tr>
<td>SGL_AUTHENTICATION_REQUIRED</td>
<td>5</td>
<td>Authentication with <code>SglAuthent()</code> not or not errorfree processed</td>
</tr>
<tr>
<td>SGL_AUTHENTICATION_FAILED</td>
<td>6</td>
<td>Authentication with <code>SglAuthent</code> failed</td>
</tr>
<tr>
<td>SGL_FUNCTION_NOT_SUPPORTED</td>
<td>7</td>
<td>The called function is not supported by the found SG-Lock.</td>
</tr>
<tr>
<td>SGL_PARAMETERS_INVALID</td>
<td>8</td>
<td>Parameter of the called function is out of the valid value range.</td>
</tr>
<tr>
<td>SGL_SIGNATURE_INVALID</td>
<td>9</td>
<td>Signature invalid</td>
</tr>
<tr>
<td>SGL_USB_BUSY</td>
<td>10</td>
<td>Another application is claiming the SG-Lock for more than 5 seconds.</td>
</tr>
</tbody>
</table>

Table 4.1: Return values and description of the SG-Lock-API.
5 Encryption, Signing and Key Management

SG-Lock comes with a modular internal symmetric (that is to say encryption and decryption key being the same) block cipher. The block size is 64 Bit or rather 2 double words. The key length is 128 Bit or rather 4 double words.

The cryptographic functions can be used for de- and encryption, and also for data signing of, e.g. configurations and confidential data. The de- and encryption can be realized directly in SG-Lock, which provides a high level of security due to secret keys that can not be read from the memory. Although by this functionality achieved security, this procedure has the constraint of a low data rate of 100 blocks per second, meaning 0.8 kb/sec, conditioned by the limited processing power of SG-Lock. Practically this function is limited to small data volume as far as only internal encryption by SG-Lock will be used. An encryption by PC-CPU is unsecure due to the appearance and possibility of reading the key in the memory with help of special tools. The advantage of this method is a fast encryption rate of more than 10MB/sec.

Also exists a possibility of combine the internal encryption by SG-Lock and encryption by PC-CPU to gain both advantages - high security provided by SG-Lock and fast encryption by PC-CPU - in a single procedure. The encryption rate is nearly as high as the internal encryption by PC-CPU only.

This encryption method encrypts all 64-Bit data blocks and links them. The first and in a periodic interval more blocks will be internally encrypted by SG-Lock, all the rest by PC-CPU. Very important for this method is the usage of unequal 128-Bit keys by the internal encryption by SG-Lock and by PC-CPU. In this case the key used by PC-CPU can be lurked, but data is due to internal encryption by SG-Lock and linking of data blocks already encrypted by a different and secret 128-Bit key.

Factory-provided all key storage (1, 2 or 16 keys, depending on module type) will be initialized with keys. Each SG-Lock user gets his own set of secret keys, that can be overwritten by self-generated keys (keys of module type U2 are static
and can not be overwritten). All SG-Lock modules of a user include an identical set of keys. This set of keys will be delivered with the first purchase. All demo modules have an own individual set of keys, printed below.

<table>
<thead>
<tr>
<th>No</th>
<th>Type</th>
<th>128-Bit key hexadecimal</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2,3,4</td>
<td>D94B6C2B 17E88CEF DADBCF1D 202161A2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3,4</td>
<td>21815888C 3798A2BB 36CAB86B 051040C1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>BBDBF022 D0D85396 9B6EFB5F 41354633</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>97CCAFDC 1EB606E7 5CB83119 9F7F457C</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>F8BA5A4D 1C1BCBD0 61140A39 49507A3F</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>326FD7E8 E6C39F3A CBA04A4B 37804850</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>554E5BA7 81665744 8F747F62 E0EE72F9</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>BAD58985 238BF49B C97B1173 D3A28313</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>98940499 D20EDC71 68388EB6 B5DF3D1C</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>0FC6EC5F EBD20065 093984EF F52F415F</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>8DC071AA 668477BE 095C0CBE 3545E855</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>CBC15944 155BF5E3 88D9C8D3 E7142A18</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>F0D76719 43A48195 7AA26332 D3B2E83C</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>8A467F11 789CD8E2 030FE272 A4750E6B</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>18FD8C08 B29157D7 F160F6A2 9E2FA426</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>90EC452F 04C30099 4B5102A9 4D942D78</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Factory-provided 128-Bit keys of demo modules
6 Programming Examples

6.1 Function SglAuthent

6.1.1 C/C++

```c
#include "SGLW32.h"

unsigned int ReturnCode;

// This is the DEMO authentication code,
// every regular SG-Lock user gets its
// own unique authentication code.
unsigned int MyAuthentCode[12] = {
    0xF574D17B, 0xA94628EE,
    0xF2857A8F, 0x69346B4A,
    0x4136E8F2, 0x89ADC688,
    0x80C2C1D4, 0xA8C6327C,
    0x1A72699A, 0x574B7CA0,
    0x1E8D3E98, 0xD7DEFDC5
};

// do authentication of SGLW32.Dll
ReturnCode = SglAuthent( MyAuthentCode );
if( ReturnCode != SGL_SUCCESS ) {
    // authentication failed!!
    printf( "SglAuthent: Error! (code: 0x%X)\n",
            ReturnCode );
}

// authentication succeeded... do the next regular thing...
```

6.1.2 Delphi

```delphi
interface
uses
    {$INCLUDE 'SGLW32IF.PAS'}
implementation
    {$INCLUDE 'SGLW32IP.PAS'}
```
{ This is the DEMO authentication code, every regular SG–Lock user gets its own unique authentication code. }

MyAuthentCode : Array[0..11] of LongWord =
($F574D17B, $A94628EE, $F2857A8F, $69346B4A, $4136E8F2, $89ADC688, $80C2C1D4, $A8C6327C, $1A72699A, $574B7CA0, $1E8D3E98, $D7DEFDC5);

procedure TForm1.Button1Click(Sender: TObject);
var ReturnCode: LongWord;

{ do authentication of SGLW32.Dll }
ReturnCode := SglAuthent(MyAuthentCode);
if (ReturnCode <> SGL_SUCCESS) then
begin
{ authentication failed !!!! }
Memo1.Text := 'SglAuthent: Error! ' + chr($0D) + chr($0A);
end;

{ authentication succeeded .. do the next regular thing ... }
end;

6.1.3 Visual Basic

' The file SGLW32.BAS has to be included in the project
' to ensure that all SG–Lock functions and constants
' are declared !!!

' This is the DEMO authentication code, every regular
' SG–Lock user gets its own unique authentication code.
Dim MyAuthentCode(0 To 11) As Long

MyAuthentCode(0) = &HF574D17B
MyAuthentCode(1) = &HA94628EE
MyAuthentCode(2) = &HF2857A8F
MyAuthentCode(3) = &H69346B4A
MyAuthentCode(4) = &H4136E8F2
MyAuthentCode(5) = &H89ADC688
MyAuthentCode(6) = &H80C2C1D4
MyAuthentCode(7) = &HA8C6327C
MyAuthentCode(8) = &H1A72699A
MyAuthentCode(9) = &H574B7CA0
MyAuthentCode(10) = &H1E8D3E98
MyAuthentCode(11) = &H7DEFDC5

Private Sub ButtonSearchSGLock_Click ()
Dim Rc As Long  ' ReturnCode

' do authentication of SGLW32.Dll
Rc = SglAuthent(AuthentCode())

If Rc = SGL_SUCCESS Then
    Text1.Caption = "SglAuthent succeeded!"
Else
    Text1.Caption = "SglAuthent failed!"
    Exit Sub
End If

' SG-Lock found .. do the next regular thing ...

End Sub
6.2 Function SglSearchLock

6.2.1 C/C++

```c
#include "SGLW32.h"

// In the case a SG-Lock user protects more than 1 application/product, he should give each of it a unique product ID. Then its very easy to distinguish the SG-Locks for each product
#define MY_PRODUCT_ABC_ID 1
#define MY_PRODUCT_XYZ_ID 2

unsigned int ReturnCode;

// Search SG-Lock with product ABC
ReturnCode = SglSearchLock( MY_PRODUCT_ABC_ID );
if( ReturnCode != SGL_SUCCESS ) {
    // no SG-Lock found!!
    printf("SglSearchLock : Error!(code: 0x%0X)\n", ReturnCode);
}

// SG-Lock found! ... do the next regular thing ...
```

6.2.2 Delphi

```delphi
interface
uses
{$INCLUDE 'SGLW32IF.PAS'}
implementation
{$INCLUDE 'SGLW32IP.PAS'}

{In the case a SG-Lock user protects more than 1 application/product, he should give each of it a unique product ID. Then its very easy to distinguish the SG-Locks for each product}
const MY_PRODUCT_ABC_ID = 1;
    MY_PRODUCT_XYZ_ID = 2;

procedure TForm1.Button1Click(Sender: TObject);
var ReturnCode: LongWord;

    { Search SG-Lock for product ABC }
ReturnCode := SglSearchLock( MY_PRODUCT_ABC_ID );
if( ReturnCode <> SGL_SUCCESS ) then
begin
    { no SG-Lock found!! }
```
Chapter 6: Programming Examples

6.2.3 Visual Basic

The file SGLW32.BAS has to be included in the project to ensure that all SG−Lock functions and constants are declared !!!

In the case a SG−Lock user protects more than 1 application/product, he should give each of it a unique product ID. Then it's very easy to distinguish the SG−Locks for each product.

Public Const MY_PRODUCT_ABC_ID As Long = 1
Public Const MY_PRODUCT_XYZ_ID As Long = 2

Private Sub ButtonSearchSGLock_Click()

Dim Rc As Long ' ReturnCode

' Search SG−Lock for product ABC
Rc = SglSearchLock( MY_PRODUCT_ABC_ID )

Select Case Rc
Case SGL_SUCCESS
    Text1.Caption = "SG−Lock found!"
Case SGL_DGL_NOT_FOUND
    Text1.Caption = "SG−Lock not found!"
Exit Sub
Case Else
    Text1.Caption = "Error: " & Rc & " occurred!"
Exit Sub
End Select

' SG−Lock found .. do the next regular thing ...

End Sub
6.3 Function SglReadSerialNumber

6.3.1 C/C++

```c
#include "SGLW32.h"
#define PROD_ABC_ID 1

unsigned int ReturnCode;
unsigned int SerialNumber;

// Read serial number of SG-Lock with product ABC
ReturnCode = SglReadSerialNumber( PROD_ABC_ID, &SerialNumber );
if( ReturnCode != SGL_SUCCESS ) {
    // no SG-Lock found!!
    printf("SglReadSerialNumber: Error! (code: %d)\n", ReturnCode);
}

// SG-Lock serial number read! ...do the next regular thing...
```

6.3.2 Delphi

```delphi
interface
uses
{$INCLUDE 'SGLW32IF.PAS'}
implementation
{$INCLUDE 'SGLW32IP.PAS'}

procedure TForm1.Button1Click(Sender: TObject);
const PROD_ABC_ID = 1;
var ReturnCode : LongWord;
    SerialNumber : LongWord;

    { Read serial number of SG-Lock with product ABC }
ReturnCode := SglReadSerialNumber( PROD_ABC_ID,
                                        Addr(SerialNumber);
if( ReturnCode <> SGL_SUCCESS ) then
begin
    { no SG-Lock found!! }
    Memo1.Text := 'SglReadSerialNumber: Error! ' +
                    char($0D) + char($0A);
end;

    { SG-Lock serial number read! ...do the next regular thing.. }
end;
```
6.3.3 Visual Basic

' The file SGLW32.BAS has to be included in the project
' to ensure that all SGLock functions and constants
' are declared !!!

' In the case a SGLock user protects more than 1
' application/product, he should give each of it a
' unique product ID. Then its very easy to distinguish
' the SGLocks for each product.
Public Const PROD_ABC_ID As Long = 1

Private Sub ButtonSearchSGLock_Click()

  Dim Rc As Long   ' ReturnCode
  Dim SerialNumber As Long

  ' Read serial number of SGLock for product ABC
  Rc = SglReadSerialNumber(PROD_ABC_ID, SerialNumber)

  Select Case Rc
    Case SGL_SUCCESS
      Text1.Caption = SerialNumber
    Case SGL_DGL_NOT_FOUND
      Text1.Caption = "SGLock not found!"
      Exit Sub
    Case Else
      Text1.Caption = "Error: " & Rc & " occurred!"
      Exit Sub
  End Select

  ' SGLock serial number read ..do the next regular thing...
End Sub
6.4 Function SglReadData

6.4.1 C/C++

```cpp
#include "SGLW32.h"
define PROD_ABC_ID 1
// address where date is stored in SG-Lock:
define RUN_DATE_ADR 10
// date stored as year/month/day (3 DWords):
define RUN_DATE_CNT 3

unsigned int RC;
unsigned int RunDate[3];
// date storage for compare

// Read date to run of SG-Lock with product ABC
RC = SglReadData(PROD_ABC_ID,
    RUN_DATE_ADR,
    RUN_DATE_CNT,
    RunDate);
if (RC != SGL_SUCCESS) {
    // no SG-Lock found!!
    printf("SglReadData : Error! (code:%d)\n", ReturnCode);
}

// read date from system, compare with RunDate
// and decide what to do
```

6.4.2 Delphi

```pascal
interface
uses
{ $INCLUDE 'SGLW32IF.PAS' }
implementation
{ $INCLUDE 'SGLW32IP.PAS' }

procedure TForm1.Button1Click(Sender: TObject);
const PROD_ABC_ID = 1;
    RUN_DATE_ADR = 10; // address where date is stored in SG-Lock
    RUN_DATE_CNT = 3; // date stored as year/month/day (3 DWords)

var RC : LongWord;
    RunDate : Array [0..2] of LongWord; // date storage for compare

{ Read date to run of SG-Lock with product ABC }
```
Chapter 6: Programming Examples

6.4.3 Visual Basic

' The file SGLW32.BAS has to be included in the project
to ensure that all SG-Lock functions and constantsare declared !!!

' In the case a SG-Lock user protects more than 1 application/product, he should give each of it a
' unique product ID. Then it's very easy to distinguish
' the SG-Locks for each product.
Public Const PROD_ABC_ID As Long = 1
' address where date is stored in SG-Lock
Public Const RUN_DATE_ADR As Long = 10
' date stored as year/month/day (3 DWords)
Public Const RUN_DATE_CNT As Long = 3

Private Sub ButtonSearchSGLock_Click()
    Dim Rc As Long ' ReturnCode
    Dim RunDate (0 to 2) As Long ' date storage for compare

    ' Read date to run of SG-Lock with product ABC
    Rc = SglReadData( PROD_ABC_ID, RUN_DATE_ADR,
                      RUN_DATE_CNT, RunDate () )

    Select Case Rc
        Case SGL_SUCCESS
            Text1.Caption = RunDate(0)&" / "&RunDate(1)&" / "&RunDate(2)
        Case SGL_DGL_NOT_FOUND
            Text1.Caption = "SG-Lock not found !"
        Exit Sub
    End Select
End Sub
Case Else
    Text1.Caption = "Error: " & Rc & " occurred!"
Exit Sub
End Select

' Read date from system, compare with RunDate
' and decide what to do

End Sub
6.5 Function SglWriteData

6.5.1 C/C++

```c
#include "SGLW32.h"
#define PROD_ABC_ID 1
// address where date is stored in SG-Lock:
#define RUN_DATE_ADR 10
// date stored as year/month/day (3 DWords):
#define RUN_DATE_CNT 3
unsigned int RC;
unsigned int RunDate[3]; // date storage
RunDate[0] = 2005; // new run date
RunDate[1] = 12;
RunDate[2] = 24;

// Write new date to run to SG-Lock with product ABC
RC = SglWriteData(PROD_ABC_ID,
                  RUN_DATE_ADR,
                  RUN_DATE_CNT,
                  RunDate);
if( RC != SGL_SUCCESS ) {
    // no SG-Lock found!!
    printf("SglWriteData: Error! (code:%d)\n", RC);
}
// new date successfully written, lets do the next thing...
```

6.5.2 Delphi

```delphi
interface
uses
  {$INCLUDE 'SGLW32IF.PAS'}
implementation
  {$INCLUDE 'SGLW32IP.PAS'}

procedure TForm1.Button1Click(Sender: TObject);
const
  PROD_ABC_ID = 1;
  RUN_DATE_ADR= 10; { addresse where date is stored in SG-Lock }
  RUN_DATE_CNT= 3; { date stored as year/month/day (3 DWords) }

var
  RC : LongWord;
  RunDate : Array [0..2] of LongWord; { date storage }

RunDate[0]:= 2005; // new run date
```
RunDate[1] := 12;
RunDate[2] := 24;

\{ Write new date to run to SG-Lock with product ABC \}
RC := SglWriteData( PROD_ABC_ID, RUN_DATE_ADR,
                    RUN_DATE_CNT, Addr(RunDate);
if ( RC <> SGL_SUCCESS ) then
begin
\{ no SG-Lock found !! \}
  Memo1.Text := 'SglWriteData : Error' +
                 char($0D) + char($0A);
end;

\{ new date successfully written, lets do the next thing ... \}
end;

6.5.3 Visual Basic

' The file SGLW32.BAS has to be included in the project
' to ensure that all SG-Lock functions and constants
' are declared !!!

' In the case a SG-Lock user protects more than 1
' application/product, he should give each of it a
' unique product ID. Then it's very easy to distinguish
' the SG-Locks for each product.
Public Const PROD_ABC_ID As Long = 1
  ' address where date is stored in SG-Lock
Public Const RUN_DATE_ADR As Long = 10
  ' date stored as year/month/day (3 DWords)
Public Const RUN_DATE_CNT As Long = 3

Private Sub ButtonSearchSGLock_Click()
  Dim Rc As Long   \' ReturnCode
  Dim RunDate (0 to 2) As Long \' date storage

  RunDate (0) = 2005  \' new run date
  RunDate (1) = 12
  RunDate (2) = 24

  \' Write new date to run to SG-Lock with product ABC
  Rc = SglWriteData( PROD_ABC_ID, RUN_DATE_ADR,
                     RUN_DATE_CNT, RunDate() )

  Select Case Rc
Case SGL_SUCCESS
    Text1.Caption = RunDate(0) & "/" & RunDate(1) & "/" & RunDate(2)
Case SGL_DGL_NOT_FOUND
    Text1.Caption = "SG-Lock not found!"
    Exit Sub
Case Else
    Text1.Caption = "Error " & Rc & " occurred!"
    Exit Sub
End Select
'
new date successfully written, lets do the next thing...
End Sub
6.6 Challenge-Response-Authentication of a SG-Lock

6.6.1 C/C++

```c
#include <time.h>
#include <stdlib.h>
#include "SGLW32.h"
#define PROD_ABC_ID 1
#define TEA_KEY_NUM 1
#define CRYPT_MODE_ENCRYPT 0

unsigned int RC;
unsigned long int RandomNumber[2]; // test random number
unsigned long int RanSglResult[2]; // encryption result
    // of SG-Lock
unsigned long int RanAppResult[2]; // encryption result
    // of application

// 1. step: generate a 128-bit key
// (NOT for U2/L2 - fixed key!)
unsigned long int TEA_Key[4]={ 0x238A3F10,
    0x61EAB67A,
    0x092E1CD2,
    0x832FAEC3 };

// ATTENTION !  ATTENTION !  ATTENTION !  ATTENTION !
// Do this only once when initialising the key prior to
// delivery of the dongle and NOT in the protected application!
// Writing the key into the SG-Lock modul
RC = SglWriteKey( PROD_ABC_ID, TEA_KEY_NUM, TEA_Key );
if ( RC != SGL_SUCCESS ) {
    printf("SglWriteKey : Error! (code : %d)\n", ReturnCode);
}
// ATTENTION END

// 2. step: generate two 32-bit (= one 64-bit) random numbers
srand( clock() ); // force every time different
    // start of sequence
RandomNumber[0] = rand() << 16 | rand();
RandomNumber[1] = rand() << 16 | rand();

// 3. step: encrypt the random number in the SG-Lock modul
RanSglResult[0]= RandomNumber[0];
RanSglResult[1]= RandomNumber[1];
```
RC = SglCryptLock( PROD_ABC_ID,
    TEA_KEY_NUM,       // number of key
    CRYPTO_MODE_ENCRYPT, // encrypt
    1,                  // block count
    RanSglResult );

// 4. step: encrypt the random number in the protected
// application
SglTeaEncipher( RandomNumber, RanAppResult, TEA_key );

// 5. step: compare both results
if (((RanSglResult[0] != RanAppResult[0]) ||
    (RanSglResult[1] != RanAppResult[1]))) {

    // authentication failed !
    printf( "SG-Lock Modul authentication: Error!\n" );
}

// authentication successful ...

More programming examples and the necessary include files can be found on
the SG-Lock CD-ROM.
# 7 Technical Data

## 7.1 SG-Lock U2/U3/U4

<table>
<thead>
<tr>
<th>Typ</th>
<th>U2</th>
<th>U3</th>
<th>U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
<td>USB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Type</td>
<td>non volatile RAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>no Memory</td>
<td>256 Bytes</td>
<td>1024 Bytes</td>
</tr>
<tr>
<td>32-Bit-Counter</td>
<td>no Counter</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>128-Bit-Key</td>
<td>1 (fixed)</td>
<td>2 (free writable)</td>
<td>16 (free writable)</td>
</tr>
<tr>
<td>Algorithm</td>
<td>TEA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read Cycles</td>
<td>unlimited</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Cycles</td>
<td>&gt; 1.000.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Storage</td>
<td>128-Bit encrypted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Retention</td>
<td>&gt; 20 Years</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Consump.</td>
<td>&lt; 50 mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Working Temp.</td>
<td>0 to 70°C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storage Temp.</td>
<td>-30 to 70°C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>47 × 16 × 8 mm (L×B×H)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>5 g</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard Color</td>
<td>blue</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>