



## **Software-Kopierschutzsystem**



## **Entwicklerhandbuch**

**für Microsoft Windows 9x, ME, NT4, 2000, Server 2003,  
XP (32/64-Bit), Vista (32/64-Bit) und Windows CE 4.X, 5.0**



**Software-Kopierschutzsystem**

# **Entwicklerhandbuch**

für Microsoft Windows 9x, ME, NT4, 2000, Server 2003,  
XP (32/64-Bit), Vista (32/64-Bit) und Windows CE 4.X, 5.0

Stand: März 2007

SG Intec Ltd & Co. KG, Schauenburgerstr. 116, D-24118 Kiel  
T ++49 431 97993-00 | F ++49 431 97993-50 | [www.sg-lock.de](http://www.sg-lock.de) | [info@sg-intec.de](mailto:info@sg-intec.de)

SG-Lock unterliegt der EU-Elektrogeräte-Richtlinie. Die entsprechenden Entsorgungsgebühren wurden entrichtet. WEEE-ID:DE 39431724

Alle in diesem Handbuch genannten gesetzlich geschützten Warenzeichen sind Eigentum der entsprechenden Eigentümer. Technische Änderungen vorbehalten. Vervielfältigungen dieses Handbuchs auch nur auszugsweise sind nur mit schriftlicher Genehmigung von SG Intec Ltd & Co. KG erlaubt.

# Inhaltsverzeichnis

<u>1. Einleitung</u> .....	1
<u>2. Installation und Hilfsprogramme</u> .....	2
2.1. Windows 9x/Me/NT4/2000/Server2003/XP/Vista.....	2
2.1.1. SG-Lock USB.....	2
2.1.2. SG-Lock USB und LPT .....	2
Abb. 1: Das SG-Lock API (die Datei SGLW32.DLL) stellt die Verbindung zwischen geschützter Anwendung und der SG-Lock Hardware bereit. ....	3
2.2. Windows CE 4.X und 5.0.....	4
2.2.1. SG-Lock USB.....	4
2.2.2 SG-Lock LPT .....	4
2.3. Deinstallation für alle Systeme.....	4
2.4. SG-Lock bearbeiten mit dem SG-Lock Manager .....	5
<u>3. Software mit SG-Lock schützen</u> .....	9
3.1. Allgemeines.....	9
3.2. Schutzstrategien.....	9
3.3. Die SG-Lock Product ID – wofür brauche ich sie? .....	9
3.4. Verschlüsselung und Challenge-Response-Authentifizierung .....	11
3.5. Einbindung in verschiedene Programmiersprachen .....	13
<u>4. SG-Lock API</u> .....	14
4.1. Funktionsübersicht .....	14
4.2. Basisfunktionen.....	16
4.2.1. Funktion: SglAuthent .....	16
4.2.2. Funktion: SglSearchLock .....	17
4.2.3. Funktion: SglReadSerialNumber.....	18
4.3. Speicherfunktionen.....	19
4.3.1. Funktion: SglReadData .....	19
4.3.2. Funktion: SglWriteData.....	20
4.3.3. Funktion: SglReadCounter .....	21
4.3.4. Funktion: SglWriteCounter .....	22
Rückgabewerte .....	22
4.4. Verschlüsselungs- und Signierungsfunktionen.....	23
4.4.1. Funktion: SglCryptLock.....	23
4.4.2. Funktion: SglSignDataApp .....	25
4.4.3. Funktion: SglSignDataLock .....	27
4.4.4. Funktion: SglSignDataComb.....	29
4.5. Administrative Funktionen .....	31
4.5.1. Funktion: SglReadProductId .....	31
4.5.2. Funktion: SglWriteProductId .....	32
4.5.3. Funktion: SglWriteKey .....	33
4.5.4. Funktion: SglReadConfig .....	34
4.6. Fehlerrückgaben .....	35

<u>5. Verschlüsselung, Signierung und Schlüsselverwaltung</u> .....	36
<u>6. Programmierbeispiele</u> .....	38
6.1. Funktion SglAuthent .....	38
6.2. Funktion SglSearchLock .....	40
6.3. Funktion SglReadSerialNumber.....	41
6.4. Funktion SglReadData.....	43
6.5. Funktion SglWriteData.....	45
6.6. Challenge-Response-Authentifizierung eines SG-Lock Moduls.....	47
<u>7. Technische Daten</u> .....	49
7.1. SG-Lock USB .....	49
7.2. SG-Lock LPT .....	50
Notizen .....	51

# 1. Einleitung

SG-Lock ist ein modernes, flexibles Hardware-basiertes Kopierschutz- und Cryptosystem, das für alle 32-Bit Microsoft Windows-Betriebssysteme zur Anwendung kommen kann. Es wird sowohl für die USB- als auch für die LPT-Schnittstelle angeboten.

## **Herausragende Eigenschaften sind:**

- Jedes SG-Lock besitzt eine individuelle Seriennummer.
- Bis zu 1024 Byte frei nutzbarer SG-Lock-interner Speicher.
- 128-Bit-Verschlüsselung mit bis zu 16 frei wählbaren Schlüsseln.
- Bis zu 64 frei programmierbare Zählerzellen zur einfachen Erfassung zählbarer Ereignisse.
- USB SG-Locks können ohne Treiberinstallation und Admin-Rechte auf dem Zielsystem installiert werden (Windows ME, 2000, Server 2003 und XP).
- USB und LPT SG-Locks können auch nachträglich ohne Änderungen an der geschützten Anwendung gegeneinander ausgetauscht werden.
- LPT SG-Locks sind transparent für daran angeschlossene Drucker und andere Geräte.

## **Besondere Sicherheitsmerkmale:**

- Der gesamte Modul-interne Speicher ist für den Nutzer transparent und mit einem für jedes Modul individuellen 128-Bit-Schlüssel verschlüsselt und zusätzlich signiert. Hardwareangriffe wie mögliche Manipulation einzelner Datenwerte oder Austausch des Speicherbausteins werden vom Modulprozessor erkannt und abgewehrt.
- Einfacher und effektiver Authentifizierungs-Mechanismus zwischen geschützter Anwendung und SG-Lock API. Das SG-Lock API ist nicht wie vielfach implementiert sofort in voller Funktionalität von jeder Anwendung nutzbar. Grundsätzlich muss sich jede Anwendung gegenüber dem SG-Lock API authentifizieren um Zugriff auf SG-Lock-Module zu bekommen. Dies verhindert Angriffe von nicht autorisierten Programmen über die API-Schnittstelle auf SG-Lock Kopierschutzmodule. Zusätzlich hat das geschützte Programm die Möglichkeit das SG-Lock API selbst zu verifizieren und eine gefälschte Bibliothek zu erkennen und abzuwehren. Der gesamte Authentifizierungsmechanismus wird einfach durch den Aufruf einer einzelnen Funktion mit einem einzelnen Parameter durchgeführt.
- Das SG-Lock API arbeitet mit einer sowohl Modul- als auch Applikations-internen TEA- (Tiny Encryption Algorithm) Verschlüsselungs-Engine. Dieser symmetrische (Schlüssel für Ver- und Entschlüsselung sind identisch) und allgemein als sicher anerkannte Verschlüsselungs-Algorithmus bildet die Basis

für die Implementierung vielfältiger Daten- und Codeschutz, sowie Authentifizierungs-Strategien.

## **2. Installation und Hilfsprogramme**

Die Installation von SG-Lock ist einfach und transparent gestaltet, um die Integration in die Installation der geschützten Anwendung zu erleichtern.

Dabei ist zu unterscheiden, ob SG-Lock ausschließlich in der USB-Variante, der LPT-Variante oder in Kombination von USB- und LPT-Modulen zum Einsatz kommen soll.

### **2.1. Windows 9x/Me/NT4/2000/Server2003/XP/Vista**

#### **2.1.1. SG-Lock USB**

Zur Installation von SG-Lock USB-Modulen ohne LPT-Module müssen 2 Schritte durchgeführt werden. Beachten Sie, dass Windows 95 und Windows NT4 USB nicht unterstützen.

1. Kopieren Sie die SG-Lock Bibliothek SGLW32 .DLL in das Installationsverzeichnis der geschützten Anwendung **oder** das Systemverzeichnis (z.B. C:\Windows\System bei Windows 98SE und ME oder C:\WINNT\SYSTEM32 bei Windows 2000, Server 2003, XP und Vista, bei letzteren Schreibrechte berücksichtigen !).
2. Stecken Sie das SG-Lock USB in die USB-Schnittstelle. Unter Windows 2000, Server 2003, XP und Vista wird die Erkennung der Hardware automatisch vollzogen und angezeigt – damit ist die Installation abgeschlossen. Unter Windows 98SE/ME kann die Windows Setup-CDROM benötigt werden, um Standard-USB-Treiber nachzuinstallieren. Hiernach ist auch diese Installation abgeschlossen und das SG-Lock USB ist betriebsbereit.

#### **2.1.2. SG-Lock USB und LPT**

Zur Installation von SG-Lock zur Benutzung von USB- und LPT- Modulen müssen zunächst alle Schritte des vorherigen Kapitels durchgeführt werden. Falls Sie SG-Lock für LPT ausschließlich benutzen wollen, führen Sie nur den obigen Schritt 1 durch.

Des Weiteren sind zusätzlich folgenden Schritte durchzuführen:

1. Für eine Installation auf Windows NT4, 2000, Server 2003 und XP Systemen loggen Sie sich mit Administrator- oder gleichwertigen Rechten ein und führen die Datei SGLLPT .REG aus, die einen Registry-Eintrag zum Laden des LPT-Treibers erzeugt.

## 2. Installation und Hilfsprogramme

---

2. Kopieren Sie die Datei SGLW32 . INI in das Systemverzeichnis (z.B. C : \WINNT bei Windows 2000, Server 2003 und XP oder C : \WINDOWS\SYSTEM bei Windows 9X und ME).
3. Kopieren Sie die Datei SGLLPT . SYS in das Systemverzeichnis (z.B. C : \WINNT\SYSTEM32\DRIVERS) bei Windows NT4, 2000, Server 2003 und XP oder SGLLPT . VXD (z.B. C : \WINDOWS\SYSTEM) bei Windows 9X und ME. Für Windows 9X und ME ist die Installation damit abgeschlossen.
4. Windows NT4, 2000, Server 2003 und XP Systeme müssen nun heruntergefahren und neu gestartet werden. Damit ist auch diese Installation abgeschlossen.

Die Datei SGLW32 . INI ermöglicht über die Änderung der Schlüsselwerte von SCAN zu NO\_SCAN eine individuelle Anpassung der Schnittstellenabfrage beim Anwender.

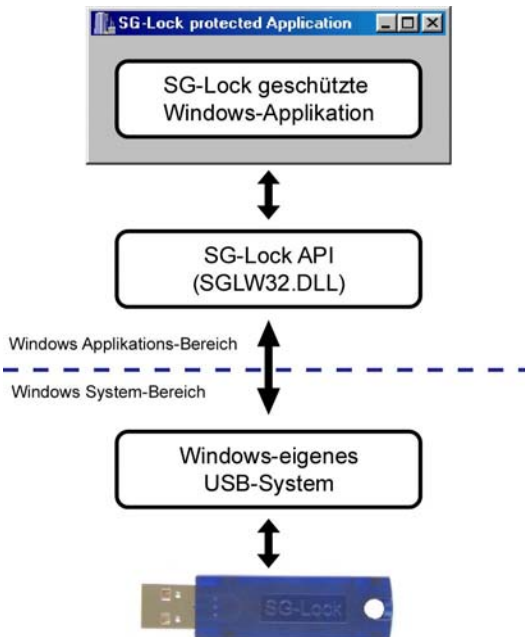


Abb. 1: Das SG-Lock API (die Datei SGLW32.DLL) stellt die Verbindung zwischen geschützter Anwendung und der SG-Lock Hardware bereit.

## 2.2. Windows CE 4.X und 5.0

### 2.2.1. SG-Lock USB

1. Kopieren Sie die Datei SGLWCE.DLL in Ihr Anwendungs- oder in das Systemverzeichnis (z.B. \Windows). Dies kann durch ein Script beim Systemstart passieren.
2. Kopieren Sie die Datei SGLUSB.DLL in ein beim Systemstart existierendes Verzeichnis (z.B. \STORAGE)
3. Passen Sie die beiden Keys mit Namen „DLL“ im Registrierungsscript SGLUSB.REG entsprechend dem Pfad von SGLUSB.DLL an ( wenn z.B. Ihre SGLUSB.DLL in \STORAGE liegt, müssen die Werte der beiden Keys STORAGE\SGLUSB.DLL lauten). Benutzen Sie dabei keinen führenden Backslash. Lassen Sie die Keys PREFIX unverändert.
4. Führen Sie das angepasste Registryscript aus und speichern Sie die Registry (z.B. mit dem AP CONFIG MANAGER in der Karteikarte APSystem , Knopf STORAGE/REGISTRY/SAVE ), damit die Schlüssel für folgende Systemstarts erhalten bleiben.

### 2.2.2 SG-Lock LPT

SG-Lock LPT wird unter Windows CE nicht unterstützt.

## 2.3. Deinstallation für alle Systeme

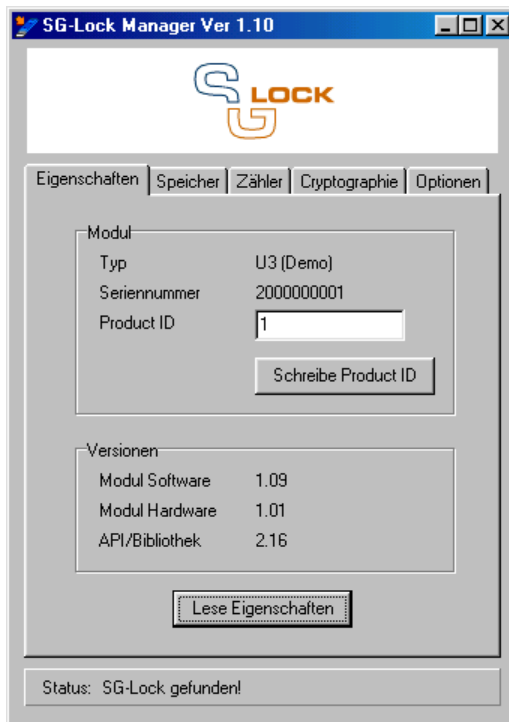
Die Deinstallation erfolgt durch einfaches Löschen der genannten Dateien und - falls diese für die Installation zuvor durchgeführt - das Löschen der in den genannten Registrierungs-Scripten angegebenen Einträge.

### 2.4. SG-Lock bearbeiten mit dem SG-Lock Manager

Der SG-Lock Manager (SGLM) ist ein auf der SG-Lock CDROM mitgeliefertes Hilfsprogramm zum Bearbeiten und Testen von allen SG-Lock Modulen.

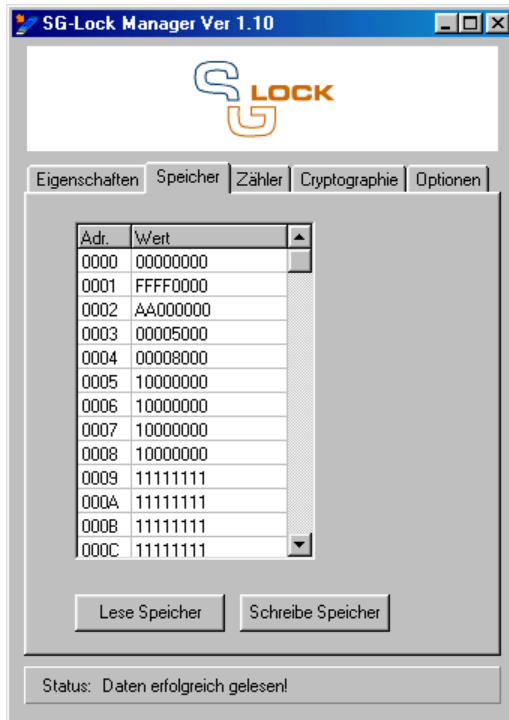
Starten Sie den SGLM indem Sie die Datei **SglMgr.Exe** aus dem Verzeichnis **Test** ausführen. Auf der Karteikarte **Optionen** kann mit dem Wahlfeld **Sprachauswahl** die Sprache angepasst werden. Zusätzlich kann auf dieser Karteikarte auch die Aus- und Eingabedarstellung von Zahlen als dezimale oder hexadezimale Zahlen angepasst werden. Diese muss auch bei der **Eingabe** von Zahlen beachtet werden!

Alle Funktionen, die der SGLM bietet, sind Teil des SG-Lock APIs und können auch von jeder geschützten Applikation genutzt werden. Die Karteikarte **Eigenschaften** bietet über den Druckknopf **Lese Eigenschaften** die Anzeige von wichtigen Informationen wie Typ, Seriennummer, Product ID und Versionsnummern des angeschlossenen SG-Lock.



Mit dem Druckknopf **Schreibe Product ID** kann diese auf Werte zwischen 0 und 65535 (dez.) geändert werden. Die Funktion der Product ID wird in Kapitel 3.3 genauer beschrieben.

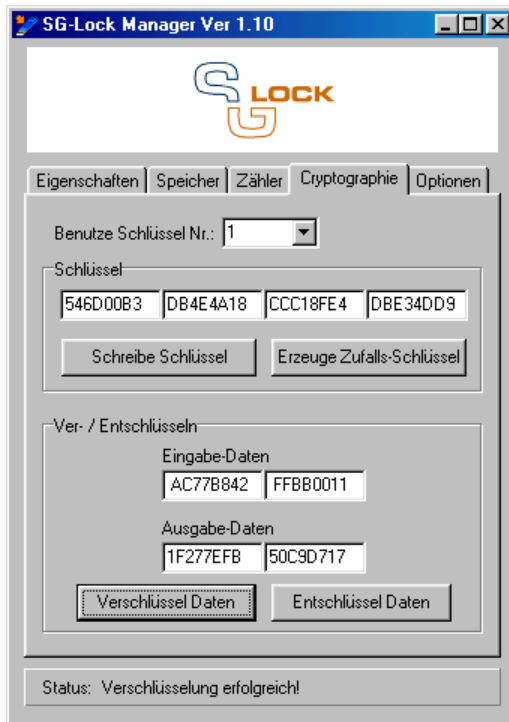
Die Karteikarte **Speicher** ermöglicht das Lesen und Beschreiben des Modul-internen Speichers (soweit vorhanden). Zum Ändern einer oder mehrerer Speicherstellen werden die gewünschten Werte in die Tabelle eingetragen und mit dem Druckknopf **Schreibe Speicher** in den Modul-internen Speicher geschrieben.



Die Karteikarte **Zähler** bietet die gleichen Möglichkeiten für Zählerspeicherstellen. Diese sind nicht in den normalen Speicher eingeblendet sondern benutzen einen zusätzlichen Speicher, was Interferenzen zwischen Datenspeicher und Zählerspeicher ausschließt.

Die Karteikarte **Cryptographie** bietet die Möglichkeit die cryptografischen Funktionen von SG-Lock auszuführen. SG-Lock benutzt eine Modul-interne symmetrische (d.h. die Schlüssel zum Ver- bzw. Entschlüsseln sind identisch) 64-Bit-Blockverschlüsselung. Die Schlüssellänge beträgt 128-Bit. Der Verschlüsselungsalgorithmus ist TEA. Die SG-Lock Typen der Serie 3 und 4 haben mehrere

Schlüsselspeicher. Zunächst wählt man unter **Benutze Schlüsselnummer** den Schlüssel aus, der geändert oder zur Verschlüsselung genutzt werden soll. Mithilfe des Druckknopfes **Erzeuge Zufalls-Schlüssel** wird ein 128-Bit Zufallsschlüssel generiert. Dieser kann über die Zwischenablage z.B. zu Dokumentationszwecken weiter verarbeitet werden (dies ist zu empfehlen, weil die Schlüssel aus Sicherheitsgründen **nur** geschrieben und **nicht** gelesen werden können). Dieser kann mit **Schreibe Schlüssel** dann in den Modul-internen Schlüsselspeicher geschrieben werden.



Um Testdaten zu ver- bzw. entschlüsseln, müssen zwei 32-Bit-Werte (entspricht einem 64-Bit-Block) in die zwei Felder **Eingabe-Daten** eingegeben werden. Durch Drücken von **Verschlüssel** oder **Entschlüssel Daten** werden diese mit dem vorgewählten Schlüssel Modul-intern ver- bzw. entschlüsselten Daten in den Feldern **Ausgabe-Daten** angezeigt.

Auf der Karteikarte **Optionen** kann die gewünschte Sprache und Zahlendarstellung festgelegt werden. Bei hexadezimaler Zahlendarstellung werden alle Zahlen mit Ausnahme der Versionsnummern auf der Karteikarte **Eigenschaften** ent-

sprechend angezeigt. Die Eingabe erfolgt dann ebenfalls in hexadezimaler Schreibweise ohne führende oder folgende Sonderzeichen.

**Achtung:** Die Eingabe eines Authentcodes (AC) ist dann erforderlich, wenn nicht-Demo (Retail) SG-Lock Module verwendet werden sollen. Ohne Eingabe eines AC werden nur Demo-Module erkannt. Jeder Software-Hersteller, der SG-Lock einsetzt, bekommt einmalig einen individuellen AC mit der Erstlieferung zugeteilt, der ihm den exklusiven Zugriff auf seine SG-Lock Module sichert. Alle nachfolgend gelieferten Module sind mit demselben AC initialisiert.

Um mit dem SGLM ebenfalls Zugriff auf Retail-Module zu erhalten, muss einmalig der AC eingeben und gespeichert werden.

**Achtung:** Der AC wird in hexadezimaler Schreibweise mitgeteilt – die Zahlendarstellung muss gegebenenfalls bei der Eingabe des AC umgestellt werden.



## **3. Software mit SG-Lock schützen**

### **3.1. Allgemeines**

Die Funktionsweise von SG-Lock als Kopierschutz beruht auf einer durch Aufruf von bestimmten Funktionen hergestellten Verbindung zwischen der leicht zu kopierenden und deshalb zu schützenden Software und der praktisch nicht zu kopierenden SG-Lock Hardware.

Die hierzu benutzten Funktionen sind die Funktionen des SG-Lock APIs (Application Programming Interface). Sie sind enthalten in der mit der SG-Lock Hardware mitgelieferten Software – in der Softwarebibliothek SGLW32.DLL.

Das SG-Lock API stellt verschiedene Arten von Funktionen bereit, von denen je nach Art des gewünschten Schutzes andere zum Einsatz kommen. Für einen effektiven Kopierschutz müssen nicht alle dieser Funktionen benutzt werden.

### **3.2. Schutzstrategien**

Die am häufigsten anzutreffende Variante, Software vor nicht vertragsgemäßer Nutzung zu schützen, ist der einfache Kopierschutz, der verhindern soll, dass die Software auf mehr Rechnern zum Ablauf gebracht wird als vertraglich vereinbart. Hier steht für die Software beim Ablauf der wiederholte Test, ob ein SG-Lock am Rechner installiert ist, im Vordergrund.

Andere Schutzstrategien sollen ermöglichen, dass eine Software nur eine begrenzte Anzahl von Starts zulässt. Hier muss neben dem Vorhandensein des SG-Locks zusätzlich ein Zähler bzw. eine Zählervariable im SG-Lock überwacht werden, um bei Überschreitung einer bestimmten Anzahl die weitere Ausführung der Software zu verhindern.

Eine ähnliche Ablaufbegrenzung kann die Ausführmöglichkeit eines Programms nur bis zu einem festgelegten Datum sein. Hier muss im SG-Lock das entsprechende Datum im Datenspeicher abgelegt werden und bei jedem Ablauf überprüft werden. Gegebenenfalls muss der Nutzer für die weitere Benutzung zahlen, so dass ein neues Datum gespeichert wird und der Vorgang von vorn beginnt (pay per use). Eine weitere Variante ist die Möglichkeit das Programm unbegrenzt ablaufen zu lassen, aber die Benutzung einzelner Funktion mitzuzählen und in regelmäßigen Abständen kostenpflichtig abzurechnen.

### **3.3. Die SG-Lock Product ID – wofür brauche ich sie?**

Häufig bietet ein Hersteller verschiedene Softwarepakete und/oder Ausbaustufen hiervon an. Wenn mehrere dieser unterschiedlichen Applikationen einen Kopierschutz eines Anbieters verwenden entsteht programmiertechnischer Verwaltungsaufwand, da bei einer Abfrage zunächst überprüft werden muß, ob das betreffende Kopierschutzmodul überhaupt zu dem laufenden Softwarepaket gehört bzw. die Ausbaustufe unterstützt.

### 3. Software mit SG-Lock schützen

Dieser Verwaltungsaufwand mit den möglichen Überschneidungen und Fehlerquellen kann mit SG-Lock einfach durch der Vergabe einer Product ID gelöst werden. Das SG-Lock API entscheidet anhand der Product ID, ob das zur Applikation gehörige Kopierschutz-Modul angesteckt ist oder nicht.

Beispiel: Firma X hat drei Softwarepakete A, B und C im Programm. Dem Produkt A wird die Product ID 1, dem Produkt B die 2 und C die 3 zugeordnet. Beim Nutzer wird die Software B gestartet und es sind drei SG-Lock Module für alle drei angebotenen Pakete am Rechner angesteckt.

Ohne die Möglichkeit der Nutzung einer Product ID müsste nun jedes der drei Module bei Aufruf einer API-Funktion nacheinander angesprochen und überprüft werden, ob es das der Software zugehörige Kopierschutzmodul ist, erst danach kann z.B. die Seriennummer ausgelesen werden.

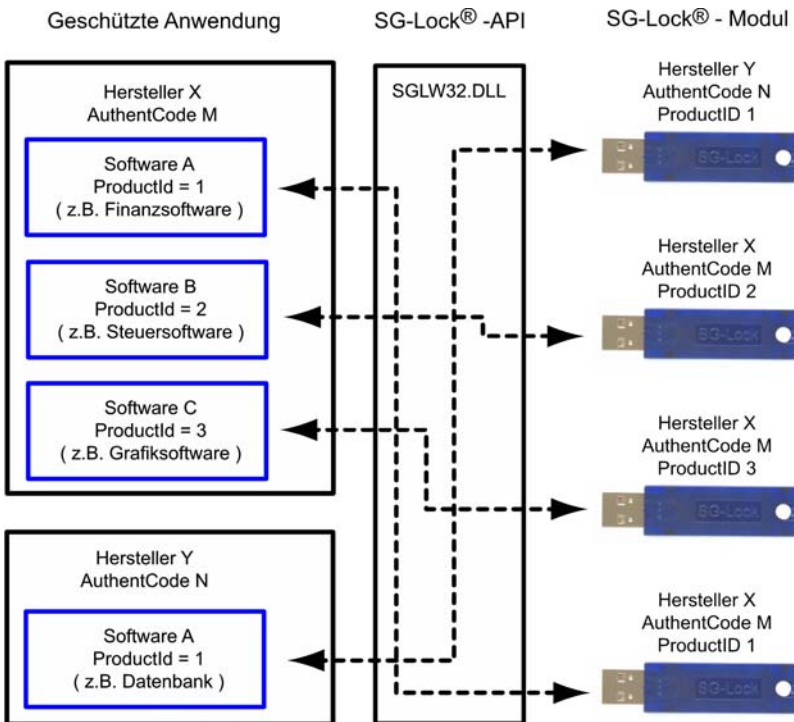


Abb. 2: Die SG-Lock ProductId ermöglicht eine einfache Abgrenzung unterschiedlicher Produkte eines Herstellers. Der AuthentCode trennt Hersteller strikt voneinander.

Unter Nutzung der Product ID des SG-Lock APIs ist der Abfragevorgang deutlich vereinfacht und weniger fehlerträchtig. Bei der Abfrage des Kopierschutzmoduls der Software B erwartet das SG-Lock-API die zugehörige Product ID als Parameter und bekommt entsprechend den Wert 2 übergeben. Resultat: Das Modul der Anwendung B wird gefunden und die beiden anderen sind vom SG-Lock-API ausgeblendet. Die Anwendung B arbeitet virtuell auf einem Rechner an dem immer nur maximal 1 Kopierschutzmodul angesteckt ist.

## 3.4. Verschlüsselung und Challenge-Response-Authentifizierung

Die von SG-Lock verwendete TEA- (Tiny Encryption Algorithm) Verschlüsselung ist gut zur Verschlüsselung wichtiger Daten geeignet – doch nicht nur dafür alleine. Jeder – auch einfache - Softwareschutz mit SG-Lock kann nocheinmal verbessert werden, wenn die integrierte TEA-Verschlüsselung angewendet wird.

Das dahinter stehende Sicherheitskonzept ist eine Authentifizierung des gefundenen Kopierschutzmoduls mit Hilfe der korrekten Verschlüsselung einer Zufallszahl und einem beiden Seiten bekannten aber ansonsten geheimen Schlüssels.

Dieses Verfahren ist in der Kryptographie als Challenge-Response-Authentication (Herausforderung-Antwort) bekannt und sehr verbreitet. Der 128-Bit Schlüssel dient hierbei als Password, das dem zu authentifizierenden SG-Lock bekannt sein bzw. im Schlüsselspeicher stehen muss. Beim Authentifizierungsvorgang wird das Password allerdings selbst nicht übergeben, weil es damit im Übertragungskanal abfangbar und damit nicht mehr geheim wäre, sondern lediglich das Wissen um ein Password bzw. die Existenz des richtigen Schlüssels wird überprüft.

Der Ablauf gestaltet sich wie folgt (Programmierbeispiel siehe Abschnitt 6.6):

1. Sie erzeugen (z.B. mit dem SG-Lock Manager) einen zufälligen 128-Bit Schlüssel, programmieren diesen in das SG-Lock Modul ( API-Funktion `SglWriteKey` ) und deklarieren diesen zusätzlich als Konstante in Ihrem zu schützenden Programm. Damit ist der Schlüssel beiden Seiten (geschütztes Programm und SG-Lock-Modul) bekannt. Achtung: Dieser Schritt entfällt bei Modulen der Serie 2, da die werksseitig programmierten Schlüssel dieser Serie nicht überschreibbar sind. Nutzen Sie den werksseitigen Schlüssel, er wird Ihnen bei Erstlieferung gesondert mitgeteilt. Demo-Module haben eigene Schlüssel, die in Kapitel 5 angegeben ist.
2. Erzeugen Sie in Ihrem zu schützenden Programm eine 64-Bit Zufallszahl (in der Praxis zwei 32-Bit Zufallszahlen) mit einer Zufallszahlgeneratorfunktion Ihrer Programmiersprache. Benutzen Sie dabei – wenn möglich – eine weitere Funktion, die gewährleistet, dass nach dem Programmstart immer wieder mit anderen Zahlen begonnen wird (Stichwort „seed“), weil andernfalls immer gleiche Sequenzen an Zufallszahlen auftreten werden, was den Schutz vermindert.

3. Verschlüsseln Sie nun die 64-Bit Zufallszahl mit der SG-Lock API-Funktion `SglCryptLock` und speichern Sie das Ergebnis. Diese Verschlüsselung wird **in dem SG-Lock Kopierschutzmodul** durchgeführt.
4. Verschlüsseln Sie nun dieselbe 64-Bit Zufallszahl (nicht das vorherige Ergebnis) mit der in der `SglW32-Include-Datei` enthaltenen Funktion `SglTeaEncrypt` ebenfalls mit Hilfe des zuvor erzeugten 128-Bit Schlüssels. Speichern Sie das Ergebnis ebenso, um es im nächsten Schritt vergleichen zu können. Diese Verschlüsselung wird **in der geschützten Anwendung** durchgeführt werden.
5. Vergleichen Sie nun, ob das Verschlüsselungsergebnis des Moduls dem (richtigen) Ergebnis der Verschlüsselung der Anwendung entspricht. Die Authentizität ist nur bei zwei identischen Ergebnissen erfüllt, da beide Verschlüsselungen offensichtlich unter Anwendung des richtigen 128-Bit-Schlüssel durchgeführt wurde, der außerhalb der Anwendung nur in dem gesuchten SG-Lock-Modul vorliegt.

Der Vergleich der beiden Verschlüsselungsergebnisse kann auch auf mehrere Programmteile verteilt werden, indem Sie das gesamte oder nur Einzelteile des Ergebnisses (z.B. die erste 16-Bit Sequenz) an einer Programmstelle vergleichen und das gesamte Ergebnis bzw. die verbleibenden Teile später (nocheinam) überprüfen. Ein entfernen des Vergleichs und damit des Kopierschutzes im Maschinencode der geschützten Anwendung wird dadurch erschwert. Alternativ können auch mehrere 64-Bit-Blöcke in einem Schritt verschlüsselt und verteilt verglichen werden.

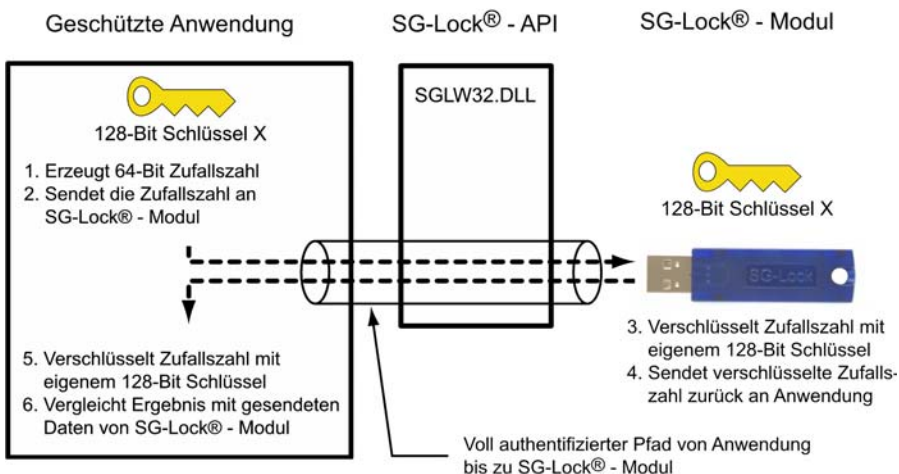


Abb. 3: Die SG-Lock Challenge-Response-Authentifizierung stellt eine sichere Verbindung zwischen der geschützten Anwendung (EXE-Datei) über das gesamte Betriebssystem und den USB-Bus zum SG-Lock Modul selbst zur Verfügung.

## 3.5. Einbindung in verschiedene Programmiersprachen

Die Funktionen des SG-Lock APIs können direkt im geschützten Programm aufgerufen werden. Hierzu müssen entsprechend der eingesetzten Programmiersprache die auf der SG-Lock CD-ROM mitgelieferten Include-Dateien in den Programmtext eingebunden werden. Bei C#, Visual Basic, Delphi und einigen anderen ist darin bereits der Hinweis für den Linker enthalten, dass die SG-Lock Funktionen in der externen Bibliothek (auch manchmal „third party DLL“ genannt) `SGLW32.DLL` zu finden sind.

Bei C/C++ muss bei der üblichen statischen Bindung noch dem Linker mit Hilfe einer sogen. Importbibliothek mitgeteilt werden, dass die Funktionen des SG-Lock APIs in einer externen Bibliothek, nämlich der `SGLW32.DLL`, zu finden sind. Dies geschieht, indem man in die Liste der vom Linker zu berücksichtigenden Bibliotheken die Importbibliothek `SGLW32.Lib` in das Projekt mit aufnimmt. Geschieht dies nicht, wird der Linker mit der Fehlermeldung die benutzten SG-Lock Funktionen nicht gefunden zu haben, den Linkvorgang abbrechen. Ungünstigerweise ist das Dateiformat von Importbibliotheken von Compiler zu Compiler unterschiedlich. Importbibliotheken der gängigsten Compiler sind auf der SG-Lock CD-ROM zu finden. Für den Fall, dass für einen Compiler keine passende Bibliothek zu finden ist, kann man diese i.d.R. selbst erzeugen, da ein entsprechendes Tool jedem Compiler beiliegt. Schlagen Sie hierzu in der Dokumentation des Compilers nach (mögliches Stichwort: „Erzeugen einer Importbibliothek“).

Alternativ hierzu kann die SG-Lock Bibliothek auch dynamisch, d.h. zur Laufzeit mit der Win32-Funktionen `LoadLibrary()` und `GetProcAddress()` gebunden werden. Genaueres hierzu finden Sie in der Microsoft Windows SDK-Dokumentation.

## **4. SG-Lock API**

### **4.1. Funktionsübersicht**

Die Funktionen des SG-Lock APIs lassen sich in die vier Gruppen Basisfunktionen, erweiterte und kryptographische Funktionen sowie Verwaltungsfunktionen einteilen. Die Basisfunktionen, die von ihrer Arbeitsweise grundlegend sind, wie z.B. die Abfrage ob ein SG-Lock an einen PC tatsächlich angeschlossen ist, kommen praktisch in jeder Art von Softwareschutz zum Einsatz.

Die Erweiterten Funktionen mit speziellen Möglichkeiten, stellen Funktionalitäten zur Verfügung, die für besondere Einsatzzwecke vorgesehen sind, wie z.B. Speicher oder Aufrufzähler, mit denen z.B. Strings gespeichert oder die Anzahl der Aufrufe eines Programms mitgezählt und begrenzt werden können. Die kryptographischen Funktionen erlauben das Verschlüsseln und Signieren von beliebigen Daten, wie Texten, Bildern, Emails, Filmen, usw.. Funktionen dieser Gruppe werden nur entsprechend der Schutz- und Vermarktungsstrategie in den Programmcode aufgenommen.

Der Funktionssatz der Verwaltungsfunktionen ist in erster Linie ergänzend für die Vorbereitung der SG-Lock Module zur Auslieferung mit der geschützten Software vorgesehen und wird in der Regel nicht in den Programmcode der geschützten Anwendung aufgenommen. Hiermit lassen sich vielmehr schnell einfache Initialisierungsprogramme individuell für diesen Zweck erstellen.

<b>Funktionsname</b>	<b>Beschreibung</b>
<b>Basisfunktionen</b>	
SglAuthent	Authentifizierung der SG-Lock Bibliothek
SglSearchLock	Sucht ein SG-Lock Gerät
SglReadSerialNumber	Liest die Seriennummer eines SG-Lock Gerätes
<b>Erweiterte Funktionen</b>	
SglReadData	Liest Daten aus dem Speicher eines SG-Lock Gerätes
SglWriteData	Schreibt Daten in den Speicher eines SG-Lock Gerätes
SglReadCounter	Liest einen Zählwert aus einem SG-Lock Gerät
SglWriteCounter	Schreibt einen Zählwert in ein SG-Lock Gerät
<b>Kryptographische Funktionen</b>	
SglCryptLock	Ver- und entschlüsselt ein oder mehrere Datenblöcke mit einem SG-Lock Gerät mit einem Geräte-internen 128-Bit Schlüssel
SglSignDataApp	Signiert Daten mit dem PC
SglSignDataLock	Signiert Daten mit einem SG-Lock Gerät
SglSignDataComb	Signiert Daten kombiniert mit PC und SG-Lock Gerät

### **Verwaltungsfunktionen**

SglReadProductId	Liest die ProductID aus einem SG-Lock Gerät
SglWriteProductId	Schreibt eine ProductID in ein SG-Lock Gerät
SglWriteKey	Schreibt einen 128-Bit Schlüssel in ein SG-Lock Gerät
SglReadConfig	Liest Konfigurationsdaten aus der SG-Lock Umgebung oder einem SG-Lock Gerät (z.B. den Typ eines SG-Lock Gerätes)

## 4.2. Basisfunktionen

### 4.2.1. Funktion: SglAuthent

#### Beschreibung

Authentifizierung der SG-Lock Bibliothek gegenüber der geschützten Anwendung und umgekehrt.

#### Modelle

U2: ✓    U3: ✓    U4: ✓    L3: ✓    L4: ✓

#### Deklaration

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

#### Parameter

Authent-Code	48 Byte-Folge, die individuell jedem Nutzer von SG-Lock zugeteilt wird.
--------------	---

#### Rückgabewerte

SGL_SUCCESS	Authentifizierung erfolgreich
SGL_AUTHENTICATION_FAILED	Authentifizierung fehlgeschlagen

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

#### Kommentar

Diese Funktion muss als erste des SG-Lock APIs einmalig erfolgreich aufgerufen werden, da alle anderen Funktionen des SG-Lock APIs vorher nicht freigeschaltet werden. Bei dynamischer Bindung ist dies zu jedem Bindungsvorgang zu tun (LoadLibrary Aufruf). DEMO-Kits haben einen eigenen Authentifizierungscode, der in den Programmierbeispielen abgedruckt ist.

**Achtung:** Eine Abfrage, ob ein SG-Lock Kopierschutzmodul angesteckt ist, erfolgt durch diese Funktion nicht!

## 4.2.2. Funktion: SglSearchLock

### Beschreibung

Sucht ein SG-Lock.

### Modelle

U2: ✓    U3: ✓    U4: ✓    L3: ✓    L4: ✓

### Deklaration

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

### Parameter

ProductID	Gibt die ProductId des gesuchten SG-Locks an.
-----------	---

### Rückgabewerte

SGL_SUCCESS	SG-Lock gefunden
SGL_DGL_NOT_FOUND	SG-Lock nicht gefunden

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

### Kommentar

Standard-Funktion zum Testen, ob ein SG-Lock Modul angesteckt ist.

### 4.2.3. Funktion: SglReadSerialNumber

#### Beschreibung

Liest die jedem SG-Lock individuelle Seriennummer aus.

#### Modelle

U2: ✓    U3: ✓    U4: ✓    L3: ✓    L4: ✓

#### Deklaration

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

#### Parameter

ProductId	Gibt die ProductId des SG-Locks an.
SerialNumber	Zeiger auf die Variable, über die die Seriennummer des SG-Locks zurückgegeben wird

#### Rückgabewerte

SGL_SUCCESS	SG-Lock Seriennummer erfolgreich gelesen
SGL_DGL_NOT_FOUND	SG-Lock nicht gefunden

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

#### Kommentar

Jedes SG-Lock-Modul hat unabhängig von der Schnittstellenvariante USB oder LPT eine individuelle Seriennummer (trifft nicht auf DEMO-Module zu).

Diese individuelle Seriennummer kann nicht nur der eindeutigen Identifizierung eines SG-Lock-Moduls dienen, sondern auch als „Mastervalue“ um gesichert individuelle Nummern, Schlüssel, Codes jeglicher Art mit geeigneten Umrechnungsfunktionen abzuleiten.

## 4.3. Speicherfunktionen

### 4.3.1. Funktion: SglReadData

#### Beschreibung

Liest 32-Bit-Datenwerte aus dem SG-Lock Speicher.

#### Modelle

U2: ✗    U3: ✓    U4: ✓    L3: ✓    L4: ✓

#### Deklaration

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

#### Parameter

ProductId	Gibt die ProductId des gesuchten SG-Locks an.
Address	Startadresse des Blocks aus Datenwerten 0 bis 63 – SG-Lock U3, L3 0 bis 255 – SG-Lock U4, L4
Count	Anzahl der Datenwerte
Data	Zeiger auf Datenbereich in den die Datenwerte kopiert werden sollen (der Entwickler muß die ausreichende Größe des Datenbereiches sicherstellen).

#### Rückgabewerte

SGL_SUCCESS	Datenwerte erfolgreich aus SG-Lock gelesen
SGL_DGL_NOT_FOUND	SG-Lock nicht gefunden

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

#### Kommentar

Der Modul-interne Datenspeicher kann zur sicheren Speicherung jeglicher Daten, wie Codes, Nummer, Schlüssel, Passwörtern, Lizenzen etc. benutzt werden

### 4.3.2. Funktion: SglWriteData

#### Beschreibung

Schreibt 32-Bit-Datenwerte in den SG-Lock Speicher.

#### Modelle

U2: ✗    U3: ✓    U4: ✓    L3: ✓    L4: ✓

#### Deklaration

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

#### Parameter

ProductId	Gibt die ProductId des gesuchten SG-Locks an.
Address	Startadresse des Blocks aus Datenwerten 0 bis 63 – SG-Lock U3, L3 0 bis 255 – SG-Lock U4, L4
Count	Anzahl der Datenwerte
Data	Zeiger auf Datenbereich aus dem die Datenwerte übernommen werden sollen (der Entwickler muß die ausreichende Größe des Datenbereiches sicher- stellen).

#### Rückgabewerte

SGL_SUCCESS	Datenwerte erfolgreich in SG-Lock Speicher geschrieben
SGL_DGL_NOT_FOUND	SG-Lock nicht gefunden

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

#### Kommentar

siehe SglReadData

### 4.3.3. Funktion: SglReadCounter

#### Beschreibung

Liest einen 32-Bit-Zählwert aus dem SG-Lock Speicher.

#### Modelle

U2: ✗    U3: ✓    U4: ✓    L3: ✓    L4: ✓

#### Deklaration

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

#### Parameter

ProductId	Gibt die ProductId des gesuchten SG-Locks an.
CntNum	Nummer des Zählers 0 bis 15 – SG-Lock U3, L3 0 bis 63 – SG-Lock U4, L4
Data	Zeiger auf Variable, die den Zählwert aufnehmen soll.

#### Rückgabewerte

SGL_SUCCESS	Zählwert erfolgreich aus SG-Lock gelesen
SGL_DGL_NOT_FOUND	SG-Lock nicht gefunden

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

#### Kommentar

Counter sind 32 Bit-Datenbereiche im SG-Lock Speicher, die neben der Nutzung als Zähler auch für alle anderen Anwendungen, die 32 Bit-Lese/Schreibvariablen zulassen, benutzt werden können. Der Speicher des Moduls kann so weiter vergrößert werden.

### 4.3.4. Funktion: SglWriteCounter

#### Beschreibung

Schreibt einen 32-Bit-Zählwert in den SG-Lock Speicher.

#### Modelle

U2: ✗    U3: ✓    U4: ✓    L3: ✓    L4: ✓

#### Deklaration

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

#### Parameter

ProductId	Gibt die ProductId des gesuchten SG-Locks an.
CntNum	Nummer des Zählers 0 bis 15 – SG-Lock U3, L3 0 bis 63 – SG-Lock U4, L4
Data	Zu schreibender Zählwert

#### Rückgabewerte

SGL_SUCCESS	Zählwert erfolgreich in SG-Lock geschrieben
SGL_DGL_NOT_FOUND	SG-Lock nicht gefunden

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

#### Kommentar

siehe SglReadCounter

## 4.4. Verschlüsselungs- und Signierungsfunktionen

### 4.4.1. Funktion: SglCryptLock

#### Beschreibung

Ver- oder entschlüsselt einen oder mehrere 64-Bit Datenblöcke mit 128-Bit Schlüssel. Algorithmus ist TEA.

#### Modelle

U2: ✓    U3: ✓    U4: ✓    L3: ✓    L4: ✓

#### Deklaration

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

#### Parameter

ProductId	Gibt die ProductId des gesuchten SG-Locks an.
KeyNum	Nummer des zu verwendenden Schlüssels 0 bis 1 – SG-Lock U3, L3 0 bis 15 – SG-Lock U4, L4
CryptMode	Arbeitsmodus 0 – Verschlüsseln 1 – Entschlüsseln
BlockCnt	Anzahl der zu bearbeitenden 64-Bit-Datenblöcke
Data	Zeiger auf Datenbereich in dem die zu bearbeitenden Datenblöcke stehen (der Entwickler muß die ausreichende Größe hinsichtlich des Parameters BlockCnt sicherstellen).

#### Rückgabewerte

SGL_SUCCESS	Verschlüsselung erfolgreich durchgeführt
SGL_DGL_NOT_FOUND	SG-Lock nicht gefunden

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

### **Kommentar**

Die Funktion überschreibt die über den Parameter `Data` übergebenen Eingangswerte. Falls diese anderweitig weiterverarbeitet werden sollen, müssen diese vor Aufruf der Funktion gesichert werden.

## 4.4.2. Funktion: SglSignDataApp

### Beschreibung

Signiert oder überprüft die Signatur eines Datenfeldes. Läuft vollständig und ohne Zutun eines SG-Locks in der geschützten Applikation ab. Die Signatur ist 64-Bit lang.

### Modelle

U2: ✓    U3: ✓    U4: ✓    L3: ✓    L4: ✓

### Deklaration

```

ULONG SglSignDataApp(
    ULONG *AppSignKey,
    ULONG Mode,
    ULONG DataLen,
    ULONG *Data,
    ULONG *Signature );

```

### Parameter

AppSignKey	Gibt die ProductId des gesuchten SG-Locks an.
Mode	Arbeitsmodus 0 – Signatur erzeugen 1 – Signatur überprüfen
DataLen	Anzahl der 32-Bit-Werte des Datenfeldes.
Data	Zeiger auf Datenfeld (Array) mit zu signierenden Daten.
Signature	Zeiger auf Datenbereich in dem entweder die erzeugte Signatur zurückgegeben oder die Signatur zur Überprüfung übergeben wird. Zeiger auf ein Datenfeld mit zwei 32-Bit-Werten (der Entwickler muß die ausreichende Größe hinsichtlich des Parameters DataLen sicherstellen).

### Rückgabewerte

SGL_SUCCESS	Signatur erfolgreich erzeugt, Signatur gültig
SGL_SIGNATURE_INVALID	Signatur ungültig

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

**Kommentar**

Vorteil: Da die Funktion innerhalb der Applikation abläuft, können schnell große Datenmengen signiert und geprüft werden.

Nachteil: Der Schlüssel ist in der Applikation enthalten, d.h. er könnte prinzipiell im Maschinencode gefunden werden.

### 4.4.3. Funktion: SglSignDataLock

#### Beschreibung

Signiert oder überprüft die Signatur eines Datenfeldes. Beide Vorgänge werden mit den angeschlossenen SG-Lock durchgeführt. Die Signatur ist 64-Bit lang.

#### Modelle

U2: ✓    U3: ✓    U4: ✓    L3: ✓    L4: ✓

#### Deklaration

```
ULONG SglSignDataLock(  
    ULONG ProductId,  
    ULONG LockSignKeyNum,  
    ULONG Mode,  
    ULONG DataLen,  
    ULONG *Data,  
    ULONG *Signature );
```

#### Parameter

ProductId	Gibt die ProductId des zu verwendenden SG-Locks an.
LockignKeyNum	Die Nummer des zur Erzeugung oder Überprüfung zu verwendender 128-Bit-Schlüssel im SG-Lock.
Mode	Arbeitsmodus 0 – Signatur erzeugen 1 – Signatur überprüfen
DataLen	Anzahl der 32-Bit-Werte des Datenfeldes.
Data	Zeiger auf Datenfeld (Array) mit zu signierenden Daten.
Signature	Zeiger auf Datenbereich in dem entweder die erzeugte Signatur zurückgegeben oder die Signatur zur Überprüfung übergeben wird. Zeiger auf ein Datenfeld mit zwei 32-Bit-Werten (der Entwickler muß die ausreichende Größe hinsichtlich des Parameters DataLen sicherstellen).

#### Rückgabewerte

SGL_SUCCESS	Signatur erfolgreich erzeugt, Signatur gültig
SGL_SIGNATURE_INVALID	Signatur ungültig

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

### **Kommentar**

Vorteil: Da die Funktion innerhalb des SG-Locks abläuft ist der Schlüssel zur Signierung im SG-Lock geschützt.

Nachteil: Aufgrund der begrenzten Rechenleistung des SG-Locks können nur kleine Datenmengen signiert werden.

## 4.4.4. Funktion: SglSignDataComb

### Beschreibung

Signiert oder überprüft die Signatur eines Datenfeldes. Die Bearbeitung wird dabei sowohl von der Applikation (Rechner-CPU) als auch des angeschlossenen SG-Locks vorgenommen. Diese Funktion kombiniert die Vorteile beider vorangehenden Funktionen – **Wichtige Bedingung:** beide Schlüssel (Applikations- und SG-Lock Modul-intener) müssen unterschiedlich sein! Die Signatur ist 64-Bit lang.

### Modelle

U2: ✓    U3: ✓    U4: ✓    L3: ✓    L4: ✓

### Deklaration

```

ULONG SglSignDataComb(
    ULONG ProductId,
    ULONG *AppSignKey,
    ULONG LockSignKeyNum,
    ULONG Mode,
    ULONG LockSignInterval,
    ULONG DataLen,
    ULONG *Data,
    ULONG *Signature );

```

### Parameter

ProductId	Gibt die ProductId des zu verwendenden SG-Locks an.
AppSignKey	bei der Erzeugung oder Überprüfung zu verwendender Applikations-128-Bit-Schlüssel. Zeiger auf Datenfeld von vier 32-Bit-Werten, die den 128-Bit-Schlüssel bilden
LockSignKeyNum	Die Nummer des zur Erzeugung oder Überprüfung zu verwendender 128-Bit-Schlüssel im SG-Lock.
Mode	Arbeitsmodus 0 – Signatur erzeugen 1 – Signatur überprüfen

LockSignInterval	gibt an, wie die Rechenleistung zur Signierung zwischen Applikation und SG-Lock aufgeteilt wird. Der Wert bestimmt als Potenz von 2, welcher Datenblock vom SG-Lock bearbeitet wird. Beispiel: Wert= 8, $2^8 = 256$ , d.h. bei jedem 256-zigsten 32-Bit-Datenwert bearbeitet das SG-Lock die Daten, alle anderen bearbeitet die Applikation (Rechner-CPU).
Data	Zeiger auf Datenfeld (Array) mit zu ignierenden Daten.
Signature	Zeiger auf Datenbereich in dem entweder die erzeugte Signatur zurückgegeben oder die Signatur zur Überprüfung übergeben wird. Zeiger auf ein Datenfeld mit zwei 32-Bit-Werten (der Entwickler muß die ausreichende Größe hinsichtlich des Parameters DataLen sicherstellen).

**Rückgabewerte**

SGL_SUCCESS	Signatur erfolgreich erzeugt, Signatur gültig
SGL_SIGNATURE_INVALID	Signatur ungültig

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

**Kommentar**

Die Funktion vereint die Vorteile der beiden vorangehend beschriebenen Funktionen: Da sowohl der Schlüssel der geschützten Applikation als auch der des verwendeten SG-Locks zur Erzeugung der Signatur zwingend notwendig sind, wird die hohe Rechenleistung der Applikation (Rechner-CPU) mit der hohen Sicherheit des SG-Locks kombiniert. Das SG-Lock bearbeitet immer den ersten Datenblock und durch die bei der Erzeugung der Signatur benutzte Verkettung der Datenwerte sind alle nachfolgenden davon abhängig.

**Wichtig: Beide verwendeten 128-Bit Schlüssel (Applikations- und SG-Lock Modul-interner) müssen unterschiedlich und sollten möglichst nicht ähnlich sein, um die Vorteile der Funktion auszuschöpfen.**

## 4.5. Administrative Funktionen

### 4.5.1. Funktion: SglReadProductId

#### Beschreibung

Liest die 16-Bit-ProductId aus dem SG-Lock

#### Modelle

U2: ✓    U3: ✓    U4: ✓    L3: ✓    L4: ✓

#### Deklaration

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

#### Parameter

ProductId	Zeiger auf 32-Bit-Variable, die die ProductId aufnehmen soll.
-----------	---

#### Rückgabewerte

SGL_SUCCESS	ProductId erfolgreich aus SG-Lock gelesen
SGL_DGL_NOT_FOUND	SG-Lock nicht gefunden

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

#### Kommentar

Die Product ID dient der Vereinfachung der Verwaltung mehrerer geschützter Anwendung bzw. Ausbaustufen einer Anwendung. Achtung: Nur die unteren 16 Bit bzw. Product IDs von 0- 65535 können verwendet werden. Eine genauere Beschreibung über die Verwendung der Product ID finden Sie im Kapitel 3.3.

## 4.5.2. Funktion: SglWriteProductId

### Beschreibung

Schreibt eine neue 16-Bit-ProductId in das SG-Lock.

### Modelle

U2: ✓    U3: ✓    U4: ✓    L3: ✓    L4: ✓

### Deklaration

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

### Parameter

OldProductId	Gibt die aktuelle ProductId des SG-Locks an.
NewProductId	Gibt die neue ProductId des SG-Locks an.

### Rückgabewerte

SGL_SUCCESS	ProductId erfolgreich in SG-Lock geschrieben.
SGL_DGL_NOT_FOUND	SG-Lock nicht gefunden

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

### Kommentar

Siehe *SglReadProductId*

### 4.5.3. Funktion: SglWriteKey

#### Beschreibung

Schreibt einen 128-Bit-Schlüssel in den SG-Lock Schlüsselspeicher.

#### Modelle

U2: ✗    U3: ✓    U4: ✓    L3: ✓    L4: ✓

#### Deklaration

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

#### Parameter

ProductId	Gibt die ProductId des SG-Locks an.
KeyNum	Nummer des zu schreibenden Schlüssels 0 bis 1 – SG-Lock U3, L3 0 bis 15 – SG-Lock U4, L4
Key	zu schreibender 128-Bit-Schlüssel Zeiger auf Datenfeld von vier 32-Bit-Werten, die den 128-Bit Schlüssel bilden.

#### Rückgabewerte

SGL_SUCCESS	ProductId erfolgreich in SG-Lock geschrieben.
SGL_DGL_NOT_FOUND	SG-Lock nicht gefunden

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

#### Kommentar

## 4.5.4. Funktion: SglReadConfig

### Beschreibung

Liest Konfigurationen und Informationen über SG-Lock aus.

### Modelle

U2: ✓    U3: ✓    U4: ✓    L3: ✓    L4: ✓

### Deklaration

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

### Parameter

ProductId	Gibt die ProductId des SG-Locks an.
Category	Art der angeforderten Information 0: Information über SG-Lock Modul
Data	Zeiger auf Datenfeld von 8 Ganzzahlwerten mit 32 Bit Breite. Die Bedeutungen der einzelnen Werte sind: Index 0: Typ Index 1: Interface Index 2: Software Version Index 3: Hardware Version Index 4: Seriennummer Index 5: Speichergröße in Dwords Index 6: Zähleranzahl Index 7: Anzahl 128-Bit Schlüssel

### Rückgabewerte

SGL_SUCCESS	ProductId erfolgreich aus SG-Lock gelesen
SGL_DGL_NOT_FOUND	SG-Lock nicht gefunden

Die vollständige Liste der Rückgabewerte befindet sich im Abschnitt 4.6.

### Kommentar

Weitergehende Informationen zu einzelnen Werten sind in Include- und Headerdateien des SG-Lock APIs aufgeführt.

## 4.6. Fehlerrückgaben

Jede SG-Lock API-Funktion gibt einen Rückgabewert zurück, anhand dessen die fehlerfreie Ausführung der Funktion überprüft werden kann. Tritt ein Fehler während der Ausführung auf, so wird ein Wert ungleich 0 zurückgegeben.

Der zurückgegebene Wert beschreibt den Fehler genauer.

Rückgabe	Wert	Beschreibung
SGL_SUCCESS	0	Fehlerfreie Ausführung
SGL_DGL_NOT_FOUND	1	SG-Lock nicht gefunden
SGL_LPT_BUSY	2	Mindestens ein LPT-Port wird von einem anderen Programm dauerhaft benutzt, auf anderen Ports kein SG-Lock gefunden.
SGL_LPT_OPEN_ERROR	3	Der SG-Lock LPT-Treiber wurde nicht gefunden, obwohl die LPT-Unterstützung für SG-Lock installiert wurde.
SGL_NO_LPT_PORT_FOUND	4	Es ist kein LPT-Port im PC installiert, obwohl die LPT-Unterstützung für SG-Lock installiert wurde.
SGL_AUTHENTICATION_REQUIRED	5	Authentifizierung mit SglAuthent wurde zuvor nicht oder nicht fehlerfrei durchgeführt.
SGL_AUTHENTICATION_FAILED	6	Authentifizierung mit SglAuthent fehlgeschlagen.
SGL_FUNCTION_NOT_SUPPORTED	7	Die aufgerufene Funktion wird von dem gefundenen SG-Lock-Modul nicht unterstützt (z.B. SglReadData bei Modul U2).
SGL_PARAMETER_INVALID	8	Ein Parameter der aufgerufenen Funktion liegt nicht im zulässigen Wertebereich (z. B. Adresse 5000 bei Funktion SglReadData).
SGL_SIGNATURE_INVALID	9	Signatur ungültig

Tab.1: Rückgabewerte des SG-Lock-APIs und zugehörige Beschreibungen.

## 5. Verschlüsselung, Signierung und Schlüsselverwaltung

SG-Lock verfügt über eine modulinterne symmetrische ( d.h. der Schlüssel für Ver- und Entschlüsselung ist gleich) Blockverschlüsselungsfunktion. Die Datenblockgröße beträgt 64 Bit bzw. 2 Doppelworte. Die Schlüsselbreite beträgt 128 Bit bzw. 4 Doppelworte.

Die Verschlüsselungsfunktion kann zur Ver- und Entschlüsselung, sowie zur Signierung beliebiger Daten, wie z.B. Konfigurationsinformationen, vertraulicher Daten usw. benutzt werden. Die Verschlüsselung kann dabei direkt in der SG-Lock Hardware durchgeführt werden, was eine sehr hohe Sicherheit darstellt, weil der Schlüssel nicht außerhalb des SG-Locks in Erscheinung tritt und somit nicht belauscht werden kann.

Die durch diese Funktionsweise erreichte hohe Sicherheit geht mit der Einschränkung einer geringen Verschlüsselungsleistung von ca. 100 Blöcken pro Sekunde, d.h. ca. 0,8 kb/sec einher, die durch die begrenzte Rechenleistung der SG-Lock-Hardware entsteht. In der Praxis bleibt diese Funktion auf kleine Datenmengen beschränkt, soweit eine rein SG-Lock-interne Verschlüsselung durchgeführt wird. Eine Verschlüsselung mit der PC-CPU ist als unsicherer anzusehen, da der Schlüssel irgendwo im System gespeichert ist und deshalb mit entsprechenden Tools auffindbar wäre. Der Vorteil dieser Variante ist eine sehr hohe Verschlüsselungsleistung von weit über 10 MB/sec.

Es ist aber auch möglich eine Kombination aus SG-Lock-interner und PC-interner Verschlüsselung durchzuführen und dabei beide Vorteile - hohe Verschlüsselungsleistung der PC-CPU und hohe Sicherheit der SG-Lock-internen Verschlüsselung - in einem Verfahren zu vereinen. Die Verschlüsselungsleistung ist praktisch genauso hoch, wie bei der rein PC-internen Verschlüsselung.

Bei dieser Verschlüsselungsvariante werden alle 64-Bit-Datenblöcke einzeln verschlüsselt und miteinander verknüpft. Der erste und in regelmäßigen Abständen wenige weitere Blöcke werden dabei von der SG-Lock-Hardware verschlüsselt, alle anderen von der PC-CPU. Besonders wichtig für die Sicherheit dieses Verfahrens ist, dass die beiden verwendeten 128-Bit-Schlüssel (SG-Lock-interner und PC-interner) **unterschiedlich** sind. So ist der PC-interne Schlüssel prinzipiell immernoch belauschbar aber die Daten sind durch die SG-Lock-interne Verschlüsselung und der Verknüpfung der Datenblöcke untereinander bereits mit einem **anderen** (und damit unbekanntem) 128-Bit-Schlüssel „vorverschlüsselt“.

Werksseitig werden alle Schlüsselspeicher (1, 2 oder 16 Stück, je nach Modultyp) vor der Auslieferung mit Schlüsseln initialisiert. Jeder SG-Lock Nutzer erhält dabei eigene geheime Schlüssel, die er bei Bedarf mit selbst generierten Schlüsseln überschreiben kann (die Schlüssel der Modultypen U2/L2 sind nicht überschreib-

bar). Alle SG-Lock Module eines Nutzers enthalten dabei einen identischen Satz von Schlüsseln. Dieser Schlüsselsatz wird dem Nutzer bei der Erstbestellung mitgeteilt.

Alle Demo-Module (USB- und LPT) haben einen eigenen individuellen Satz von Schlüsseln, der im Folgenden abgedruckt ist.

Schlüsselnr.	Modultyp	128-Bit Schlüssel (hexadezimal)
0	2,3,4	D94B6C2B 17E88CEF DADBCF1D 202161A2
1	3,4	2181588C 3798A2BB 36CAB86B 051040C1
2	4	BBDBF022 D0D85396 9B6EFB5F 41354633
3	4	97CCAFDC 1EB606E7 5CB83119 9F7F457C
4	4	F8BA5A4D 1C1BCBD0 61140A39 49507A3F
5	4	326FD7E8 E6C39F3A CBA04A4B 37804850
6	4	554E5BA7 81665744 8F747F62 E0EE72F9
7	4	BAD58985 238BF49B C97B1173 D3A28313
8	4	98940499 D20EDC71 68388EB6 B5DF3D1C
9	4	0FC6EC5F EBD20065 093984EF F52F415F
10	4	8DC071AA 668477BE 095C0CBE 3545E855
11	4	CBC15944 155BF5E3 88D9C8D3 E7142A18
12	4	F0D76719 43A48195 7AA26332 D3B2E83C
13	4	8A467F11 789CD8E2 030FE272 A4750E6B
14	4	18FD8C08 B29157D7 F160F6A2 9E2FA426
15	4	90EC452F 04C30099 4B5102A9 4D942D78

Tab. 2: Werksseitig programmierte 128-Bit-Schlüssel von Demo-Modulen

## 6. Programmierbeispiele

### 6.1. Funktion SglAuthent

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 ...
```

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! ' + char($0D) + char($0A);
end;

{ authentication succeeded .. do the next regular thing ... }
```

```
end;
```

### 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) = &HD7DEFDC5
```

```
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. Funktion SglSearchLock

**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%X)\n", ReturnCode );
}

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

**Delphi:**

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

{ In the case a SG-Lock user protects more than 1 applicati-
on/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!! }
        Memol.Text:= 'SglSearchLock: Error! ' + char($0D) + char($0A);
    end;

    { SG-Lock found .. do the next regular thing ... }

end;
```

### 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 its 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 & " ocured !"
    Exit Sub
End Select

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

End Sub
```

## 6.3. Funktion SglReadSerialNumber

### 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);
}
```

## 6. Programmierbeispiele

---

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

### 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;
```

### 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 its very easy to distinguish the SG-Locks 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 SG-Lock 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 = "SG-Lock not found !"
    End Select
End Sub
```

```
Case Else
    Text1.Caption = "Error " & Rc & " occurred !"
Exit Sub
End Select

' SG-Lock serial number read .. do the next regular thing ...

End Sub
```

### 6.4. Funktion SglReadData

#### C/C++:

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

#### Delphi:

```
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 }
RC:= SglReadData( PROD_ABC_ID, RUN_DATE_ADR,
                 RUN_DATE_CNT, Addr(RunDate));
if( RC <> SGL_SUCCESS ) then
begin
    { no SG-Lock found!! }
    Memo1.Text:= 'SglReadData: Error! ' +
```

## 6. Programmierbeispiele

---

```
                char($0D) + char($0A);
end;

{read date from system, compare with RunDate and decide what to do}
end;
```

### 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 its very easy to distinguish the SG-Locks for each product.
Public Const PROD_ABC_ID As Long = 1
' adresse 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
        Case Else
            Text1.Caption = "Error " & Rc & " occured !"
            Exit Sub
    End Select

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

End Sub
```

## 6.5. Funktion SglWriteData

C/C++:

```
#include "SGLW32.h"
#define PROD_ABC_ID 1
#define RUN_DATE_ADR 10 // adresse where date is stored in SG-Lock
#define RUN_DATE_CNT 3 // date stored as year/month/day (3 DWords)
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", ReturnCode);
}

// new date successfully written, lets do the next thing ...
```

Delphi:

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

procedure TForm1.Button1Click(Sender: TObject);
const PROD_ABC_ID = 1;
      RUN_DATE_ADR= 10; { adresse 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;

### 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 its very easy to distinguish the SG-Locks for each product.
Public Const PROD_ABC_ID As Long = 1
' adresse 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 & " occured !"
            Exit Sub
    End Select

    ' new date successfully written, lets do the next thing ...

End Sub
```

## 6.6. Challenge-Response-Authentifizierung eines SG-Lock Moduls

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
                  CRYPT_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] )) {
```

## 6. Programmierbeispiele

---

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

Weitere Programmierbeispiele und die benötigten Include-Dateien sind auf der SG-Lock CD-ROM zu finden.

## 7. Technische Daten

### 7.1. SG-Lock USB

Anschluß	USB
Speichertyp	nicht flüchtiger RAM
Speicher	U2: kein Speicher U3: 256 Bytes U4: 1024 Bytes
32-Bit-Zähler	U2: kein Zähler U3: 16 U4: 64
128-Bit-Schlüssel	U2: 1 (fest) U3: 2 (frei programmierbar) U4: 16 (frei programmierbar)
Algorithmus	TEA
Lesezyklen	unbegrenzt
Schreibzyklen	> 1.000.000
Datenspeicherung	128-Bit verschlüsselt
Datenerhalt	> 20 Jahre
Stromverbrauch	< 50 mA
Abmessungen	63 x 16 x 8 mm (LxBxH)
Gewicht	8 g

## 7.2. SG-Lock LPT

Anschluß	LPT
Speichertyp	nicht flüchtiger RAM
Speicher	L2: kein Speicher L3: 256 Bytes L4: 1024 Bytes
32-Bit-Zähler	L2: kein Zähler L3: 16 L4: 64
128-Bit-Schlüssel	L2: 1 (fest) L3: 2 (frei programmierbar) L4: 16 (frei programmierbar)
Algorithmus	TEA
Lesezyklen	unbegrenzt
Schreibzyklen	> 1.000.000
Datenspeicherung	128-Bit verschlüsselt
Datenerhalt	> 20 Jahre
Stromverbrauch	< 5mA
Abmessungen	50 x 54 x 17 mm (LxBxH)
Gewicht	42 g

## **Notizen**