Copy Protection System

Podręcznik dla programisty

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

Copy Protection System

Podręcznik dla programisty

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

Stan: Październik 2007
SG Intec Ltd & Co. KG, Schauenburgerstr. 116, D - 24118 Kiel, Niemcy
T ++49 431 97993-00 | F ++49 431 97993-50 | www.sg-lock.de |info@sg-intec.de

SG-Lock podlega europejskim prawom dla przyrządów elektrycznych.
Odpowiednie opłaty za likwidację odpadów zostały uiszczone.
WEEE-ID:DE 39431724

Spis rzeczy

1. Wprowadzenie ... 1
2. Instalacja i programy pomocnicze ... 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 i LPT ... 2
  2.2. Windows CE 4.X oraz 5.0 ... 4
    2.2.1. SG-Lock USB ... 4
    2.2.2 SG-Lock LPT ... 4
  2.3. Deinstalacja dla wszystkich systemów ... 4
  2.4. Adaptacja SG-Lock za pomocą SG-Lock Manager ... 5
3. Ochrona oprogramowania z zastosowaniem SG-Lock ... 9
  3.1. Ogólne wprowadzenie ... 9
  3.2. Strategie ochrony ... 9
  3.3. SG-Lock Product ID – do czego potrzebuję? ... 9
  3.4. Szyfrowanie i Challenge-Response-Authentication ... 11
  3.5. Włączenie do różnych języków programowania ... 12
4. SG-Lock API ... 14
  4.1. Przegląd funkcji ... 14
  4.2. Funkcje bazowe ... 16
    4.2.1. Funkcja: SglAuthent ... 16
    4.2.2. Funkcja: SglSearchLock ... 17
    4.2.3. Funktion: SglReadSerialNumber ... 18
  4.3. Funkcje pamięci ... 19
    4.3.1. Funkcja: SglReadData ... 19
    4.3.2. Funkcja: SglWriteData ... 20
    4.3.3. Funkcja: SglReadCounter ... 21
    4.3.4. Funkcja: SglWriteCounter ... 22
  4.4. Funkcje kryptograficzne i sygnujące ... 23
    4.4.1. Funkcja: SglCryptLock ... 23
    4.4.2. Funkcja: SglSignDataApp ... 25
    4.4.3. Funkcja: SglSignDataLock ... 27
    4.4.4. Funkcja: SglSignDataComb ... 29
  4.5. Funkcje administracyjne ... 31
    4.5.1. Funkcja: SglReadProductId ... 31
    4.5.2. Funkcja: SglWriteProductId ... 32
    4.5.3. Funkcja: SglWriteKey ... 33
    4.5.4. Funktion: SglReadConfig ... 34
  4.6. Opis błędów ... 35
5. Szyfrowanie, sygnowanie i administrowanie kluczami ... 36

6. Przykłady programów 38
6.1. Funkcja SglAuthent 38
6.2. Funkcja SglSearchLock 40
6.3. Funkcja SglReadSerialNumber 41
6.4. Funkcja SglReadData 43
6.5. Funkcja SglWriteData 45
6.6. Challenge-Response-Authentication modułu SG-Lock 47
7. Dane techniczne 49
7.1. SG-Lock USB 49
7.2. SG-Lock LPT 50
Notatki 51

1. Wprowadzenie

SG-Lock jest nowoczesnym, elastycznym , hardwarowym systemem ochrony oprogramowania przed nielegalnym kopiowaniem oraz kryptosystemem, który może być zastosowany do wszystkich 32-bitowych systemów operacyjnych firmy Microsoft. Jest wpinany do USB lub do LPT.

Dominującymi właściwościami są:

- Każdy SG-Lock posiada indywidualny numer seryjny.
- 1024 –bajtową, dowolnie programowalną pamięć wewnętrzną.
- 128-bitowy system szyfrowania posiadający 16 wolnych do wyboru kluczy.
- 64 dowolnie programowalnych komórek licznikowych, które mogą służyć do prostego rejestrowania różnych zdażeń.
- USB SG-Locks są instalowane do ścieżki docelowej bez sterowników i bez konieczności praw administratora. (Windows ME, 2000, Server 2003 i XP).
- USB i LPT SG-Locks można stosować wymiennie bez wpływu na ochronę.
- Wpięte LPT SG-Locks są przejrzyste dla podłączonych drukarek i innych urządzeń.

Szczególne cechy bezpieczeństwa:

- Cały moduł pamięci wewnętrznej jest dla użytkownika przejrzysty i zaszyfrowany indywidualnym, 128- bitowym kluczem i dodatkowo zasygnowany. Manipulacje określonych danych hardwaru lub wymiana danych między kostkami pamięci zostaną przez procesor uchwycone i odparte.
- Prosty i efektywny mechanizm autentyfikacyjny użytkowanego programu i SG-Lock API. SG-Lock API nie są natychmiast w pełnej funkcjonalności dostępne do korzystania. Zasadniczo musi każdy użytkowany program poddać sie autentyfikacji przez SG-Lock API, aby otrzymać dostęp do modułów SG-Lock . Ten mechanizm zapobiega atakom nieautoryzowanych programów, poprzez API, na chronione moduły SG-Lock. Chroniony program ma dodatkowo możliwość zweryfikowania SG-Lock API i odeprzeć atak na sfałszowaną biblitekę. Cały mechanizm autentyfikujący jest przeprowadzony dzięki wywołaniu jednej funkcji z jednym jedynym parametrem.
- SG-Lock API współpracuje z modułowym TEA (Tiny Encryption Algorithm), jak również z TEA dla wewnętrznego oprogramowania. Ten symetryczny (kod do szyfrowania i deszyfrowania jest identyczny) i uchodzący jako pewny algorytm kodowania tworzy podstawę do implementacji wielorakich strategii ochronnych jak również autentyfikacji.

2. Instalacja i programy pomocnicze

Instalacja SG-Lock jest prosta i przejrzysta , aby ułatwić włączenie do chronionego oprogramowania. Przy tym ważne jest, czy SG-Lock stosowany będzie wyłącznie w wariancie USB, LPT, czy w kombinacji USB i LPT.

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

2.1.1. SG-Lock USB

Do instalacji modułów SG-Lock USB bez modułów LPT konieczne są 2 kroki do przeprowadzenia. Zauważ, że Windows 95 und Windows NT4 nie wspierają USB.

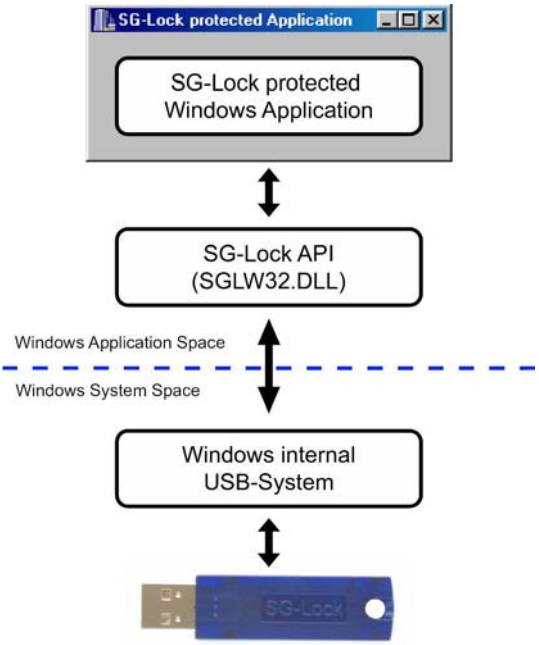
1. Skopiuj bibliotekę `SGLW32.DLL` do ścieżki instalacyjnej **lub** systemowej zabezpieczonego oprogramowania. (Np.
   `C:\Windows\System` dla Windows 98SE i ME lub
   `C:\WINNT\SYSTEM32` dla Windows 2000, Server 2003, XP i Vista, przy ostatnim uwzględnij prawa zapisu !).
2. Wepnij SG-Lock do USB. Dla Windows 2000, Server 2003, XP i Vista instalacja jest zakończona i rozpoznanie hardwaru nastąpi automatycznie.
   Dla Windows 98SE/ME będzie być może potrzebna płytka CD-ROM Windows Setup, aby zainstalować standardowy sterownik dla USB. Tym samym instalacja jest również zakończona i SG-Lock USB jest gotowy do pracy.

2.1.2. SG-Lock USB i LPT

Przy instalacji SG-Lock do korzystania z modułów USB i LPT konieczne są wszystkie poprzednie kroki. Jeżeli stosujesz SG-Lock wyłącznie dla LPT, przeprowadź tylko w.w. krok pierwszy.
Następnie są do przeprowadzenia dodatkowo następujące kroki:

1. Dla instalacji na Windows NT4, 2000, Server 2003 i Systemów XP zaloguj się jako administrator lub temu podobny i wystartuj plik `SGLLPT.REG`, który stworzy w rejestrze zapis do ładowania sterownika LPT.
2. Skopiuj plik `SGLW32.INI` do ścieżki systemowej (Np.
   `C:\WINNT` dla Windows 2000, Server 2003 i XP lub
   `C:\WINDOWS\SYSTEM` dla Windows 9X i ME).
3. Skopiuj plik `SGLLPT.SYS` do ścieżki systemowej (Np.
   `C:\WINNT\SYSTEM32\DRIVERS`) dla Windows NT4, 2000, Server 2003 i XP lub `SGLLPT.VXD` (Np.
   `C:\WINDOWS\SYSTEM`) dla Windows 9X i ME. Dla Windows 9X i ME instalacja jest tym samym zakończona..
4. Windows NT4, 2000, Server 2003 i systemy XP muszą zostać wyłączone i na nowo wystartowane. Instalacja zakończona.

Plik `SGLW32.INI` umożliwia dzięki zmianom wartości w kluczu ze SCAN na NO_SCAN indywidualne dopasowanie interfejsu dla użytkownika.



Rys. 1: SG-Lock API (plik SGLW32.DLL) przedstawia gotową łączność między chronionym oprogramowaniem i SG-Lock hardware

2.2. Windows CE 4.X oraz 5.0

2.2.1. SG-Lock USB

1. Skopiuj plik SGLWCE.DLL do Twojej ścieżki lub ścieżki systemowej (Np. \Windows). Np. przez wystartowanie skryptu podczas startu systemu
2. Skopiuj plik SGLUSB.DLL do istniejącego już katalogu (Np. \STORAGE)
3. Dopasuj oba klucze odpowiednio do ścieżki SGLUSB.DLL nadając im końcówkę „DLL“ w skrypcie rejestru SGLUSB.REG (jeżeli np. Twój SGLUSB.DLL leży w \STORAGE, muszą obie wartości kluczy mić nazwę STORAGE\SGLUSB.DLL). Początkowy Backshlash jest przy tym niedozwolony. Pozostaw PREFIX klucza niezmieniony.
4. Wystartuj dopasowany skrypt rejestru (Registryscript) i zapisz rejestr (Np. programem AP CONFIG MANAGER do karty APSystem , guzik STORAGE/REGISTRY/SAVE), aby wartość klucza dla nowego wystartowania systemu pozostała zapamiętana.

2.2.2 SG-Lock LPT

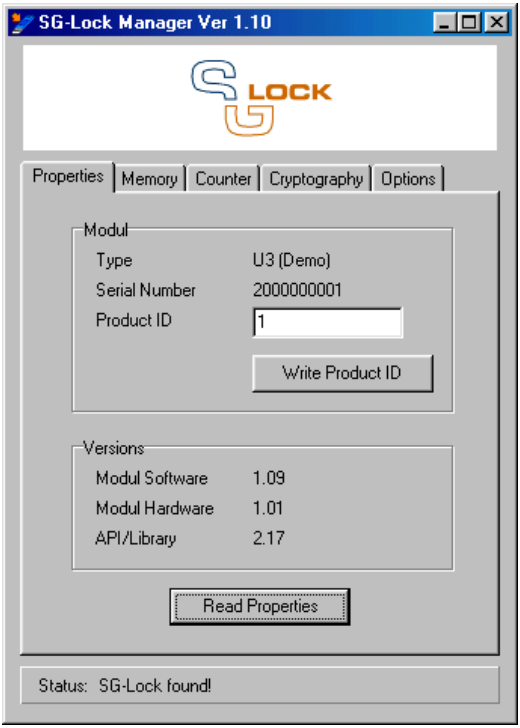
SG-Lock LPT nie jest wspierany pod Windows CE.

2.3. Deinstalacja dla wszystkich systemów

Deinstalację przeprowadza się przez zwykłe skasowanie wszystkich wspomnianych plików. Jeżeli owe zostały wystartowane, konieczne jest również skasowanie zapisów w wymienionych skryptach rejestru.

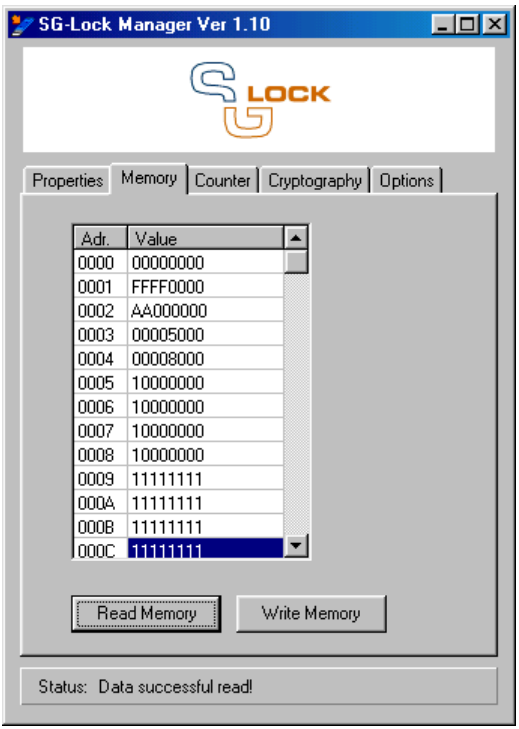
2.4. Adaptacja SG-Lock za pomocą SG-Lock Manager

SG-Lock Manager (SGLM) jest programem pomocniczym do opracowania i testowania wszystkich modułów SG-Lock, dostarczonym na płytce SG-Lock CDROM.
Wystartuj SGLM przez wystartowanie pliku ***SglMgr.Exe*** z katalogu ***Test***.
W karcie ***Options*** możesz dopasować język w polu ***Select language***.
Dodatkowo możesz na tej karcie dopasować system liczbowy jako decymalny albo hexadecymalny. Ten system musisz potem stosowac przy **wprowadzaniu** danych!
Wszystkie funkcje, które SGLM stawia do dyspozycji, są częścią SG-Lock API i mogą być wykorzystane przy każdej chronionej aplikacji. Karta ***Properties*** pokazuje przez naciśnięcie ***Read Properties*** najważniejsze informacje takie jak typ, numer seryjny, ID produktu i numer wersji użytego klucza.



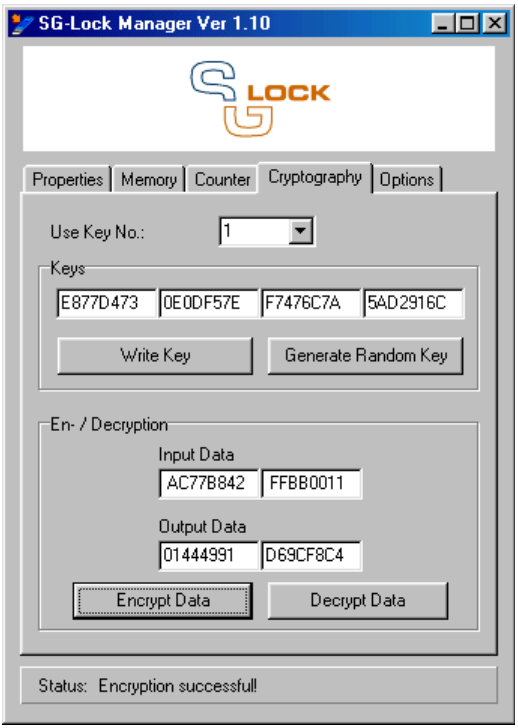
Przez naciśnięcie guzika **Write Product ID** można zmieniać wartości ID od 0 do 65535 (dez.). Opis tej funkcji (Product ID) został dokładnie przedstawiony w rozdziale 3.3.

Karta ***Memory*** umożliwia czytanie i opracowanie wewnętrznego modułu pamięci (jeżeli taką posiada). Do zmiany jednego lub więcej pól pamięci zostaną zażyczone wartości naniesione do tabeli i przez naciśniecie guzika ***write Memory*** zapisane w wewnętrznej pamięci modułu.



Karta ***Counter*** posiada podobne możliwpści dla pól licznika pamięci. Nie są one wyświetlone w normalnej pamięci tylko korzystają z dodatkowej pamięci, co w efekcie wyklucza interferencje między pamięcią danych i pamięcią licznika.
Karta ***Cryptography*** dysponuje możliwością wystartowania funkcji kryptograficznych z SG-Lock. SG-Lock korzysta z wewnątrzmodułowego symetrycznego (tzn. klucze do szyfrowania modułu i deszyfrowania są identyczne), 64-bitowego, blokowego szyfrowania. Długość klucza wynosi 128-bitów. Algorytmem szyfrowania jest TEA. Typy SG-Lock serii 3 i 4 mają więcej pamięci do dyspozycji.
Najpierw wybiera się pod ***Use Key No*** klucz, który ma zostać zmieniony lub służyć do szyfrowania.
Za pomocą guzika ***Generate Random Kay*** zostanie wygenerowany 128-bitowy przypadkowy klucz. Ten klucz moze służyć potem np. do dokumen-

tacji (zalecane, ponieważ klucz ten ze względów bezpieczeństwa może być **tylko** zapisany a **nie** czytany). Można go zapisać za pomocą ***Write kay*** do modułu wewnętrznej pamięci kluczca.

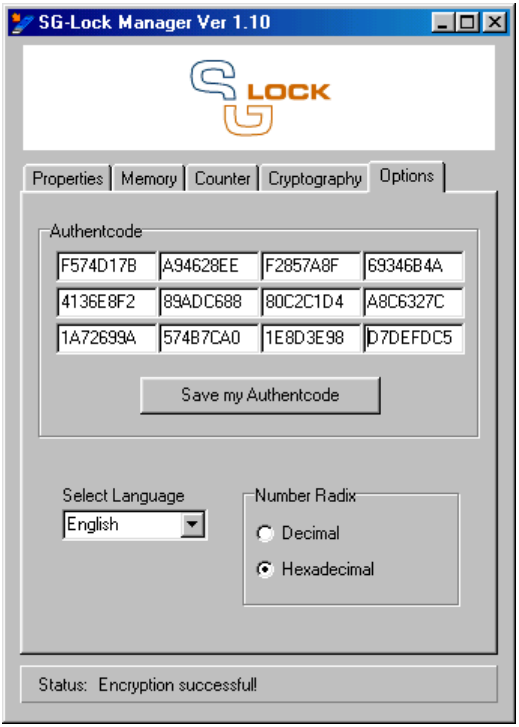


Aby zaszyfrować lub zdeszyfrować dane testujące, konieczne jest wpisanie do ***Input Data*** wartości do dwóch 32-bitowych pól (odpowiada jednemu blokowi 64-bitowemu) . Przez naciśnięcie ***Encrypt Data*** lub ***Descrypt Data*** 64-bitowy blok zostanie zaszyfrowany lub zdeszyfrowany. Rezultat można zobaczyć w ***Output Data.***

Pod kartą ***Options*** można wybrać dowoly język i system liczbowy. Wszystkie liczby z wyjątkiem numeru wersji będą pokazane w ***Properties*** w systemie hexadecymalnym. Wpisywanie danych musi przebiegać również w systemie hexadecymalnym z wyjątkiem znaków specjalnych.

Uwaga: Wprowadzenie kodu autentyfikującego (AC) jest wtedy konieczne, gdy nie mamy do czynienia z modułami demo (retail) SG-Lock. Bez podania kodu AC zostaną rozpoznane tylko moduły demo. Każdy producent oprogramowania, który zastosuje SG-Lock , otrzyma jednorazowo z pierwszą dostawą swój indywidualny

AC, który mu gwarantuje wyłączny dostęp do modułów SG-Lock . Wszystkie następne dostawy SG-Lock będą zainicjowane tym samym AC.
Aby otrzymać dostęp do modułów retail, trzeba podać jednorazowo AC i zapisać w pamięci.
Uwaga: AC zostanie dostarczone w kodzie hexadecymalnym. Również przy wprowadzaniu AC trzeba uwzględnić system liczbowy.



3. Ochrona oprogramowania z zastosowaniem SG-Lock

3.1. Ogólne wprowadzenie

Sposób funkcjonowania SG-Lock jako ochrony przed kopiowaniem polega na wywołaniu określonych funkcji, które tworzą połączenie między wymagającym ochrony oprogramowaniem i praktycznie nie do skopiowania SG-Lock Hardware. Te funkcje są funkcjami SG-Lock APIs (Application Programming Interface). One zostaną również dostarczone jako software i znajdują sie w bibliotece `SGLW32.DLL`.
SG-Lock API przedstawia różne rodzaje funkcji, które zostaną wywołane według rodzaju użytego zabezpieczenia. Dla efektywnej ochrony nie jest konieczne wywołanie wszystkich funkcji.

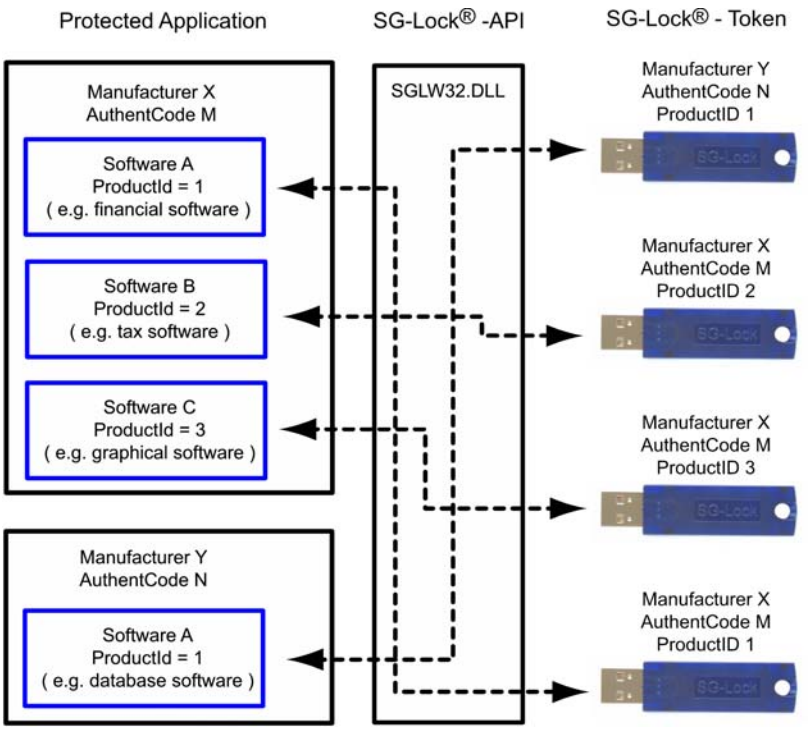
3.2. Strategie ochrony

Najczęstszym wariantem do ochrony oprogramowania przed nielegalnym korzystaniem przez inne PC, jest zwykła ochrona przed kopiowaniem. Tutaj najważniejsze jest powtarzanie testowania, czy SG-Lock jest w komputerze zainstalowany.
Inne strategie dopuszczją dla określonego oprogramowania ograniczoną liczbę startów. W tym wypadku trzeba dodatkowo kontrolować licznik lub zmienną licznika modułu SG-Lock, aby uniemożliwić następne, niedopuszczalne starty programu.
Innym ograniczeniem startów programu może być określenie końcowej daty użytkowania programu. Tutaj konieczne jest zapisanie w pamięci SG-Lock odpowiedniej daty. Ta data będzie testowana przy każdym wywołaniu programu. Użytkownik musi ponownie zapłacić za dalszą możliwość korzystania z programu (pay per use), przy czym w pamięci zostanie zapisana nowa data. (wygodne przy dzierżawie oprogramowania)
Inną możliwością jest nieograniczone korzystanie z programu przy ograniczeniu ilości wywołań poszczególnych funkcji. W regularnych odstępach, na podstawie tychże wywołań, użytkownik dostanie rachunek do zapłacenia.

3.3. SG-Lock Product ID – do czego potrzebuję?

Często producent oprogramowania oferuje różne pakiety z możliwościami rozbudowy. Jeżeli więcej z tych aplikacji korzysta z ochrony pewnego dostawcy, powstaje problem, który wymaga wiele wysiłku, aby działający moduł zidentyfikawać i czy nowy pakiet bedzie współpracować z istniejącym.
SG-Lock rozwiązuje problem ewentualnego, dodatkowego nakładu pracy, przez identyfikację Productu ID. SG-Lock API decyduje na podstawie ID, czy należący do aplikacji SG-Lock jest wpięty czy nie.

Przykład: Firma X ma trzy pakiety softwarowe A, B i C w programie. Produktowi A jest przyporządkowany Product ID 1, produktowi B Product ID 2 i produktowi C Product ID 3. Pakiet software B wystartował i pozostałe trzy moduły SG-Lock są wpięte do USB komputera.
Bez tego rozwiązania (Product ID), musiałby każdy z tych trzech modułów SG-Lock, przy wywołaniu funkcji API, zostać po kolei zapytany i sprawdzony, czy odpowiedni software odpowiada modułowi zabezpieczającemu SG-Lock. Dopiero wtedy mógłby zostać odczytany numer seryjny.



Rys. 2: SG-Lock ProductID umożliwia proste rozgraniczenie różnych produktów jednego producenta. AuthentCode ściśle oddziela producentów od siebie.

Przy korzystaniu z możliwości funkcji API, które daje Product ID, proces kontroli jest prostszy i naraża na mniej błędów. Przy komunikacji SG-Lock z software B , SG-Lock-API oczekuje pokrewny Product ID jako parametr i otrzymuje z powrotem odpowiednio wartość 2. Rezultat: Moduł SG-Lock softwaru B został zidentyfikowany i pozostałe klucze przez SG-Lock-API wyłączone. Program B

pracuje wirtualnie w komputerze, do którego zawsze jest wpięty maksymalnie 1 klucz SG-Lock.

3.4. Szyfrowanie i Challenge-Response-Authentication

Wykorzystany przez SG-Lock algorytm TEA- (Tiny Encryption Algorithm) nadaje się do zaszyfrowania ważnych danych, ale nie tylko. Każde zabezpieczenie softwaru, jest pewniejsze, gdy jest zastosowany szyfr TEA.
Konceptem ochrony jest tutaj autentyfikacja rozpoznanego SG-Lock za pomocą zaszyfrowanej liczby z generatora przypadku i tajnego, tylko dla obu stron znanego klucza.
Ta procedura jest w kryptografii znana jako Challenge-Response-Authentication (Pytanie-Odpowiedź) i szeroko rozpowszechniona. 128-bitowy klucz służy jako pasword, który musi być rozpoznany przez autentyfikujący SG-Lock lub być zapisany w pamięci wewnetrznej modułu. W procesie identyfikacji pasword nie będzie przekazany z powrotem przez kanał transmisyjny, aby nie został przechwycony przez osoby niepowołane. Będzie tylko testowana egzystencja właściwego klucza.

Przebieg jest następujący (Przykład programowy w rozdziale 6.6):

1. Utwórz (np. Przy pomocy SG-Lock Manager) przypadkowy 128-bitowy klucz, zaprogramuj go do pamięci wewnętrznej modułu SG-Lock (funkcja API: `SglWriteKey`) i zadeklaruj go dodatkowo jako constant w Twoim programie, który chcesz chronić. Tym samym klucz jest znany obu stronom (chroniony program i moduł SG-Lock). Uwaga: Ten krok odpada przy modułach serii 2, ponieważ one są już warsztatowo zaprogramowane i nie można ich ponownie zapisać. Skorzystaj z gotowego klucza, który zostanie osobno dostarczony z pierwszą dostawą. Moduły demo mają własne klucze, są one podane w rozdziale 5.
2. Utwórz w Twoim chronionym programie za pomocą funkcji random, w Twoim języku programowania, 64-bitową przypadkową liczbę (czyli dwie 32-bitowe liczby przypadkowe). Skorzystaj przy tym – o ile to możliwe – z następnej funkcji, która zapewni, że przy każdym starcie programu zawsze otrzymasz inną liczbę (hasło „seed“). W innym przypadku wystąpią zawsze te same sekwencje liczbowe, co pomniejszy ochronę.
3. Zaszyfruj następnie tę 64-bitową liczbę przypadkową za pomocą funkcji SG-Lock API: `SglCryptLock` i zapisz wynik. To szyfrowanie zostanie przeprowadzone w module **SG-Lock** .
4. Zaszyfruj teraz tą samą 64-bitową liczbę przypadkową (nie poprzedni wynik) za pomocą funkcji zawartej w pliku SglW32-Include pod nazwą `SglTeaEncipher` również za pomocą tego samego, przed chwilą utworzonego 128-bitowego klucza. Zapisz wynik również do pamięci, aby w następnym kroku móc porównać. To szyfrowanie zostanie przeprowadzone w chronionym programie Twojego komputera.

5. Porównaj teraz, czy wynik szyfrowania modułu odpowiada (właściwemu) wynikowi szyfrowania w programie. Autentyfikacja jest spełniona tylko w przypadku, gdy oba wyniki są identyczne. Czyli obydwa procesy szyfrowania korzystały z tego samego, właściwego 128-bitowego klucza, który jest ukryty w Twoim module SG-Lock.

Porównanie obu wyników szyfrowania można podzielić na wiele części. Można porównać całość lub tylko części wyniku (np. pierwszą 16-bitową sekwencję) w jednej części programu i cały wynik lub jego pozostałe części później (jeszcze raz) sprawdzić. Pominięcie porównania i tym samym ochrony kopiowania w kodzie maszynowym programu będzie utrudnione. Alternatywnie można zaszyfrować więcej 64-bitowych bloków w jednym kroku i w częściach porównać.

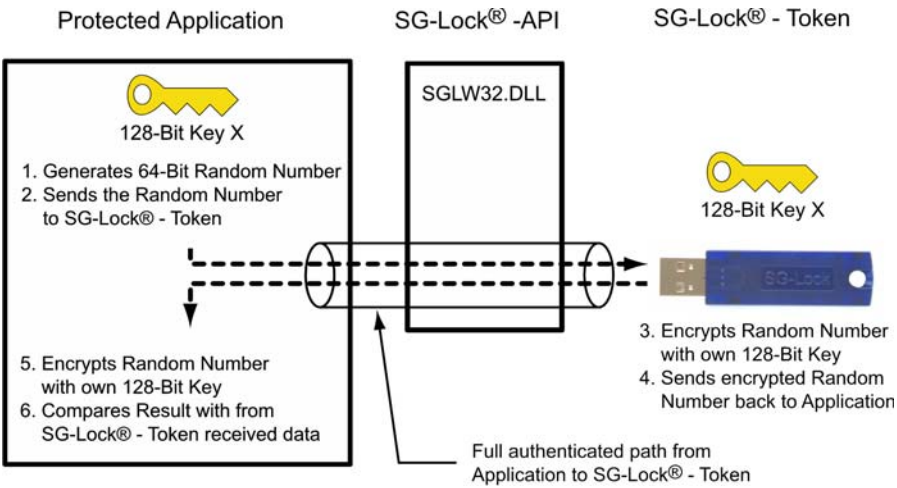


Abb. 3: SG-Lock Challenge-Response-Authentication przedstawia bezpieczne połączenie między chronionym programem (plik-EXE) od systemu operacyjnego począwszy, przez bus USB na module SG-Lock skończywszy.

3.5. Włączenie do różnych języków programowania

Funkcje API modułu SG-Lock mogą zostać bezpośrednio wywołane w chronionym programie. Tutaj trzeba włączyć bezpośrednio do programu dostarczone na CD-ROM pliki *Include*, odpowiednio do stosowanych językówe. Przy C#, Visual Basic, Delphi i kilku innych linker wie, że funkcje SG-Lock są zawarte w zewnętrznej bibliotece (niekiedy nazwanej też „third party DLL“) `SGLW32.Dll`. Przy C/C++ trzeba w zwykłym statycznym włączeniu wskazać linkerowi, za pomocą tzw. biblioteki importowanej `SGLW32.Lib`, że funkcje API znajdują się w biblitece zewnętrznej czyli `SGLW32.Dll`. A więc włączyć bibliotekę `SGLW32.Lib` do listy linkera, aby uwzględnił ją w projekcie. Jeżeli to zostanie

pominięte, kompilacja zostanie przerwana i zostanie wysłany meldunek, że funkcje SG-Lock nie zostały znalezione. Niestety format importowanych bibliotek jest inny dla różnych compilerów. Większość bibliotek często stosowanych compilerów jest dostarczona na płytce CD-ROM SG-Lock. W przypadku, gdy żadna biblioteka nie zostanie znaleziona dla danego compilera, można stworzyć ją samemu. Odpowiedni tool stoi do dyspozycji. (Szukaj w dokumentacji compilera pojęcia: „Creating an Import Library“).
Alternatywnie może biblioteka SG-Lock zostać też dynamicznie włączona, tzn. Przez funkcje systemu Win32 `LoadLibrary()` i `GetProAddress()`.
Więcej znajdziesz w dokumentacji Microsoft Windows SDK.

4. SG-Lock API

4.1. Przegląd funkcji

Funkcje SG-Lock API można podzielić na cztery grupy: funkcje bazowe, funkcje pamięci, funkcje kryptograficzne i administracyjne. Funkcje bazowe występują w każdym rodzaju ochrony oprogramowania. Np. sprawdzenie, czy SG-Lock rzeczywiście jest wpięty do USB komputera.
Funkcje pamięci ze specjalnymi możliwościami takimi jak: wywołanie licznika, zapis danych do pamięci wewnętrznej modułu np. strings lub ilość wywołań (zliczanie lub ograniczenie) określonego programu.
Funkcje kryptograficzne umożliwiają szyfrowanie i sygnowanie dowolnych danych takich jak: teksty, obrazy, emails, filmy itp. W zależności od tego, do czego ma służyć ochrona, przyjmujemy pewną strategię i korzystamy z odpowiedniej funkcji.
Grupa funkcji administracyjnych służy w pierwszym rzędzie do przygotowania modułu SG-Lock do dystrybucji z ochroną i nie będzie z reguły przewidziana do implementowania w chronionym oprogramowaniu. Do tego celu można zastosować prostsze programy inicjalizujące.

Nazwa funkcji	**Opis**
Funkcje bazowe	
SglAuthent	Autentyfikacja bibliteki SG-Lock
SglSearchLock	Szukanie SG-Lock
SglReadSerialNumber	Czyta numer seryjny SG-Lock
Funkcje pamięci	
SglReadData	Czytanie danych z pamięci wewnętrznej SG-Lock
SglWriteData	Zapis danych do pamięci SG-Lock
SglReadCounter	Odczyt wartości licznika w SG-Lock
SglWriteCounter	Zapis wartości licznika do pamięci SG-Lock
Funkcje kryptograficzne	
SglCryptLock	Szyfrowanie i deszyfrowanie jednego lub więcej bloku danych 128-bitowym kluczem za pomocą SG-Lock
SglSignDataApp	Sygnowanie danych za pomocą PC
SglSignDataLock	Sygnowanie danych za pomocą SG-Lock
SglSignDataComb	Sygnowanie danych za pomocą kombinacji PC i SG-Lock
Funkcje administracyjne	
SglReadProductId	Czytanie ProductID SG-Lock

SglWriteProductId	Zapis ProductID do pamięci SG-Lock
SglWriteKey	Zapis 128-bitowego klucza do pamięci SG-Lock
SglReadConfig	Czytanie konfiguracji danych z otoczenia SG-Lock lub SG-Lock (np. typ SG-Lock)

4.2. Funkcje bazowe

4.2.1. Funkcja: SglAuthent

Opis
Autentyfikacja biblioteki SG-Lock z chronionym oprogramowaniem i odwrotnie.

Modele
U2: ✓ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

AuthentCode	48 byte ciąg, który zostaje indywidualnie przydzielony każdemu użytkownikowi SG-Lock.

Wartość zwrotna funkcji

SGL_SUCCESS	autentyfikacja dokonana
SGL_AUTHENTICATION_FAILED	autentyfikacja niedokonana (błąd)

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz
Ta funkcja musi zostać wywołana jako pierwsza z funkcji API SG-Lock , jednorazowo i z sukcesem, ponieważ pozostałe funkcje API nie zostaną wcześniej dopuszczone. Przy połączeniu dynamicznym funkcja ta jest wywoływana do każdego procesu łączenia (LoadLibrary). DEMO-Kit mają własny kod autentyfikacyjny, który jest zawarty w programach przykładowych.
Uwaga: Zadaniem tej funkcji nie jest sprawdzenie, czy moduł SG-Lock jest wpięty do komputera czy nie!

4.2.2. Funkcja: SglSearchLock

Opis
Szuka SG-Lock

Modele
U2: ✓ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

ProductId	Podaje ProductID szukanego SG-Lock.

Wartość zwrotna funkcji

SGL_SUCCESS	SG-Lock znaleziony
SGL_DGL_NOT_FOUND	SG-Lock nie został znaleziony

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz
Funkcja standardowa do testowania, czy moduł SG-Lock jest wpięty do komputera.

4.2.3. Funktion: SglReadSerialNumber

Opis
Czyta indywidualny numer seryjny SG-Lock.

Modele
U2: ✓ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

ProductId	Podaje ProductID szukanego SG-Lock.
SerialNummer	Wskaźnik zmiennej (pointer), który podaje wartość zwrotną funkcji.

Wartość zwrotna funkcji

SGL_SUCCESS	Numer seryjny SG-Lock odczytany z sukcesem
SGL_DGL_NOT_FOUND	SG-Lock nie został znaleziony

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz

Każdy moduł SG-Lock, niezależnie od rodzaju interfejsu USB czy LPT, ma indywidualny numer seryjny (oprócz modułów DEMO). Ten numer może służyć nie tylko do identyfikacji modułu SG-Lock lecz również jako „Mastervalue“, aby wywieść funkcję pochodną, która zabezpiecza numer indywidualny, klucz lub kod innego rodzaju do własnych potrzeb.

4.3. Funkcje pamięci

4.3.1. Funkcja: SglReadData

Opis
Czyta 32-bitowe wartości danych z pamięci modułu SG-Lock.

Modele
U2: ✗ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

ProductId	Podaje ProductId szukanego SG-Lock.
Address	Adres standardowy bloku : 0 do 63 – SG-Lock U3, L3 0 do 255 – SG-Lock U4, L4
Count	Ilość wartości danych
Data	Wskaźnik (pointer) do pola danych, do którego można wpisać skopiowane wartości. (producent musi odpowiednie pole danych wskazać do dyspozycji).

Wartość zwrotna funkcji

SGL_SUCCESS	Czytanie z SG-Lock dokonane
SGL_DGL_NOT_FOUND	SG-Lock nie został znaleziony

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz
Pamięć wewnętrzna modułu może służyć do bezpiecznego zapisu różnych danych takich jak: kod, numer, klucz, pasword, licencje itp.

4.3.2. Funkcja: SglWriteData

Opis
Zapisuje 32-bitowe wartości danych do pamięci SG-Lock.

Modele
U2: ✗ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

ProductId	Podaje ProductId szukanego SG-Lock.
Address	Adres standardowy bloku : 0 do 63 – SG-Lock U3, L3 0 do 255 – SG-Lock U4, L4
Count	Ilość wartości danych
Data	Wskaźnik (pointer) do pola danych, z którego można przejąć dane (producent musi odpowiednie pole danych wskazać do dyspozycji).

Wartość zwrotna funkcji

SGL_SUCCESS	Dane zapisane w pamięci
SGL_DGL_NOT_FOUND	SG-Lock nie został znaleziony

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz
zobacz *SglReadData*

4.3.3. Funkcja: SglReadCounter

Opis
Czyta 32-bitową wartość licznika z pamięci SG-Lock.

Modele
U2: ✗ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

ProductId	Podaje ProductId szukanego SG-Lock.
CntNum	Numer licznika 0 do 15 – SG-Lock U3, L3 0 do 63 – SG-Lock U4, L4
Data	Wskaźnik (pointer) na zmienną, która ma przejąć wartość licznika.

Wartość zwrotna funkcji

SGL_SUCCESS	Warość licznika odczytana
SGL_DGL_NOT_FOUND	SG-Lock nie został znaleziony

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz
Counter są 32-bitowe pola pamięci w SG-Lock, które służą jako licznik ale też do innych potrzeb, w przypadku, gdy będą dopuszczone 32-bitowe zmienne z możliwością odczytu i zapisu. Pamięć modułu może zostać do tych celów rozszerzona.

4.3.4. Funkcja: SglWriteCounter

Opis
Zapisuje 32-bitową wartość licznika do pamięci SG-Lock.

Modele
U2: ✗ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

ProductId	Podaje ProductId szukanego SG-Lock.
CntNum	Numer licznika 0 do 15 – SG-Lock U3, L3 0 do 63 – SG-Lock U4, L4
Data	Wartość licznika do zapisu

Wartość zwrotna funkcji

SGL_SUCCESS	Wartość licznika zapisana
SGL_DGL_NOT_FOUND	SG-Lock nie został znaleziony

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz
Zobacz *SglReadCounter*

4.4. Funkcje kryptograficzne i sygnujące

4.4.1. Funkcja: SglCryptLock

Opis
Szyfrowanie i deszyfrowanie jednego lub więcej 64-bitowych bloków danych 128-bitowym kluczem. Stosowany algorytm: TEA.

Modele
U2: ✓ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

ProductId	Podaje ProductId szukanego SG-Lock.
KeyNum	Numer klucza do zastosowania 0 do 1 – SG-Lock U3, L3 0 do 15 – SG-Lock U4, L4
CryptMode	Tryb pracy 0 – Szyfrowanie 1 – Deszyfrowanie
BlockCnt	Liczba bloków 64-bitowych do zaszyfrowania/deszyfrowania
Data	Wskaźnik (pointer) do pola danych, w którym stoją 64-bitowe bloki (producent musi odpowiednie pole danych, z uwzględnieniem parametru BlockCnt, wskazać do dyspozycji).

Wartość zwrotna funkcji

SGL_SUCCESS	Szyfrowanie dokonane.
SGL_DGL_NOT_FOUND	SG-Lock nie został znaleziony

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz
Funkcja ta zapisuje, kasując poprzedni zapis, przekazane za pomocą parametru Data dane wejściowe. Jeżeli owe mają być wykorzystane w innym miejscu, muszą zostać zabezpieczone przed wywołaniem tej funkcji.

4.4.2. Funkcja: SglSignDataApp

Opis
Sygnuje lub sprawdza sygnaturę określonego pola danych. Przebiega bez ingerencji SG-Lock w chronionej aplikacji. Sygnatura ma długość 64 bity.

Modele
U2: ✓ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

AppSignKey	służy do tworzenia lub sprawdzenia 128-bitowego klucza. Jest wskaźnikiem (pointer) do pola czterech 32-bitowych danych , które tworzą ten klucz.
Mode	Tryb pracy 0 – Tworzenie sygnatury 1 – Sprawdzanie sygnatury
DataLen	Ilość 32-bitowych pól danych.
Data	Ilość 32-bitowych pól danych.
Signature	Wskaźnik (pointer) do pola danych, w którymsygnatura będzie przekazana jako wartość zwrotna lub zostanie przekazana do sprawdzenia. Jest wskaźnikiem do dwóch 32-bitowych pól danych (producent musi odpowiednie pole danych, z uwzględnieniem parametru DataLen, wskazać do dyspozycji).

Wartość zwrotna funkcji

SGL_SUCCESS	Sygnatura utworzona, sygnatura ważna.
`SGL_SIGNATURE_INVALID`	Sygnatura nieważna

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz
Zaleta: Ponieważ ta funkcja przebiega wewnątrz aplikacji,
mogą zostać szybko zasygnowane i sprawdzone duże pola danych.
Wada: Klucz jest zawarty w aplikacji, tzn. w zasadzie może zostać wyśledzony w kodzie maszynowym.

4.4.3. Funkcja: SglSignDataLock

Opis
Sygnuje lub sprawdza sygnaturę określonego pola danych. Obydwa procesy są przeprowadzane w obecności wpiętego modułu SG-Lock. Sygnatura ma długość 64 bity.

Modele
U2: ✓ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

ProductId	Podaje ProductId do użytego SG-Lock.
LockSignKeyNum	Numer 128-bitowego tworzonego lub sprawdzanego 128-bitowego klucza SG-Lock.
Mode	Tryb pracy 0 – Tworzenie sygnatury 1 – Sprawdzanie sygnatury
DataLen	Ilość 32-bitowych pól danych.
Data	Pointer do pola danych (Array), do zasygnownia.
Signature	Wskaźnik (pointer) do pola danych, w którym- sygnatura będzie przekazana jako wartość zwrotna lub zostanie przekazana do sprawdzenia. Jest wskaźnikiem do dwóch 32-bitowych pól danych (producent musi odpowiednie pole danych, z uwzględnieniem parametru DataLen, wskazać do dyspozycji).

Wartość zwrotna funkcji

SGL_SUCCESS	Sygnatura utworzona, sygnatura ważna.
SGL_SIGNATURE_INVALID	Sygnatura nieważna

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz

Zaleta: Ponieważ ta funkcja przebiega wewnątrz SG-Lock, klucz do sygnowania jest chroniony w module SG-Lock.
Wada: Możliwości SG-Lock dokonania obliczeń są ograniczone, dlatego tylko małe ilości danych mogą być sygnowane.

4.4.4. Funkcja: SglSignDataComb

Opis
Sygnuje lub sprawdza sygnaturę określonego pola danych. Obydwa procesy będą przeprowadzone jednocześnie w aplikacji (CPU) oraz we wpiętym SG-Lock. Ta funkcja kumuluje obie zalety poprzednich funkcji. **Warunek:** obydwa klucze (aplikacji i modułu wewnętrznego SG-Lock) muszą być różne! Sygnatura ma długość 64 bity.

Modele
U2: ✓ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

ProductId	Podaje ProductId do użytego SG-Lock.
AppSignKey	służy do tworzenia lub sprawdzenia 128-itowego klucza aplikacji. Jest wskaźnikiem (pointer) do pola czterech 32-bitowych danych , które tworzą ten klucz.
LockSignKeyNum	Numer 128-bitowego tworzonego lub sprawdzanego 128-bitowego klucza SG-Lock.
Mode	Tryb pracy 0 – Tworzenie sygnatury 1 – Sprawdzanie sygnatury

LockSignInterval	podaje, jak może być podzielona praca w procesie sygnowania między aplikacją i SG-Lock. Zmienna określa dane, które będą obrabiane przez SG-Lock. Przykład: Zmienna = 8, $2^8 = 256$, tzn. każde następne, 256 –te z kolei 32-bitowe słowo będzie zasygnowane przez SG-Lock , a reszta w aplikacji (CPU).
DataLen	Ilość 32-bitowych pól danych.
Data	Pointer do pola danych (Array) do zasygnownia.
Signature	Wskaźnik (pointer) do pola danych, w którym sygnatura będzie przekazana jako wartość zwrotna lub zostanie przekazana do sprawdzenia. Jest wskaźnikiem do dwóch 32-bitowych pól danych (producent musi odpowiednie pole danych, z uwzględnieniem parametru DataLen, wskazać do dyspozycji).

Wartość zwrotna funkcji

SGL_SUCCESS	Sygnatura utworzona, sygnatura ważna.
`SGL_SIGNATURE_INVALID`	Sygnatura nieważna

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz

Funkcja jednoczy zalety poprzednich funkcji.: Ponieważ klucz chronionej aplikacji jak również klucz zastosowanego modułu SG-Lock są konieczne do stworzenia sygnatury , kombinacja ta zapewnia dużą wydajność (CPU) i pewną ochronę przez SG-Lock. SG-Lock zaczyna pracę zawsze od pierwszego bloku i splot bloków danych następujących przy tworzeniu sygnatury będzie od tego zależny.

Ważne: Obydwa zastosowane 128-bitowe klucze (aplikacji i modułu SG-Lock) muszą być różne i w miarę niepodobne, aby wykorzystać maksymalnie zalety tej funkcji.

4.5. Funkcje administracyjne

4.5.1. Funkcja: SglReadProductId

Opis
Czyta 16-bitowy ProductId z SG-Lock.

Modele
U2: ✓ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

ProductId	Wskaźnik (pointer) na 32-bitową zmienną, która ma przejąć ProductID.

Wartość zwrotna funkcji

SGL_SUCCESS	ProductId odczytany z SG-Lock
SGL_DGL_NOT_FOUND	SG-Lock nie został znaleziony

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz
Product ID służy do ułatwienia w administrowaniu wielu chronionych aplikacji lub poszerzeń tej aplikacji użytkownika SG-Lock.
Uwaga: Tylko poniższe 16 bitów tzn. Product ID od 0- 65535 mogą zostać zastosowane. Dokładny opis zastosowania Product ID znajdziesz w rozdziale 3.3.

4.5.2. Funkcja: SglWriteProductId

Opis
Zapisuje nowy 16-bitowy ProductID do SG-Lock.

Modele
U2: ✓ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

OldProductID	Pokazuje aktualną wartość ProductId SG-Locks
NewProductID	Oddaje nową wartość ProductID SG-Lock.

Wartość zwrotna funkcji

SGL_SUCCESS	ProductID zapisany do SG-Lock
SGL_DGL_NOT_FOUND	SG-Lock nie został znaleziony

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz
Zobacz *SglReadProductId*

4.5.3. Funkcja: SglWriteKey

Opis
Zapisuje 128-bitowy klucz do pamięci kluczowej SG-Lock.

Modele
U2: ✗ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

ProductId	Stwierdza numer ProductID SG-Lock.
KeyNum	Numer klucza, który ma być użyty 0 do 1 – SG-Lock U3, L3 0 do 15 – SG-Lock U4, L4
Key	128-bitowy klucz do zapisania w pamięci. Pointer na pole czterech 32-bitowych słów danych, które tworzą 128-bitowy klucz.

Wartość zwrotna funkcji

SGL_SUCCESS	Klucz zapisany w SG-Lock.
SGL_DGL_NOT_FOUND	SG-Lock nie został znaleziony

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz

4.5.4. Funktion: SglReadConfig

Opis
Czyta konfiguracje i informacje o SG-Lock.

Modele
U2: ✓ U3: ✓ U4: ✓ L3: ✓ L4: ✓

Deklaracja funkcji

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

Parametr

ProductId	Stwierdza numer ProductID SG-Lock.
Category	Rodzaj zażądanej informacji 0: informacja o module SG-Lock
Data	Pointer na pole 8 liczb całkowitych, 32-bitowych. Znaczenie poszczególnych wartości: Index 0: Typ Index 1: Interface Index 2: Wersja softwaru Index 3: Wersja hardwaru Index 4: Numer seryjny Index 5: Obszar pamięci w in Dwords Index 6: Wielkość na liczniku Index 7: Ilość 128-bitowych kluczy

Wartość zwrotna funkcji

SGL_SUCCESS	ProductID odczytany z SG-Lock
SGL_DGL_NOT_FOUND	SG-Lock nie został znaleziony

Lista błędów jest wydrukowana w rozdziale 4.6.

Komentarz
Dalsze informacje o poszczególnych wartościach znajdziesz w plikach Include i Header funkcji SG-Lock API.

4.6. Opis błędów

Każda funkcja API dostarcza wartość zwrotną, na podstawie której można sprawdzić, czy nastąpiło bezbłędne wyprowadzenie funkcji. Jeżeli wystąpił błąd, otrzymamy z powrotem wartość różną od 0.
Dokładny opis numeru błędów:

Nazwa	Wartość	Opis
SGL_SUCCESS	0	Bezbłędne wyprowadzenie funkcji
SGL_DGL_NOT_FOUND	1	SG-Lock nie został znaleziony
SGL_LPT_BUSY	2	Przynajmniej jeden port LPT jest wykorzystywane przez inny program, w innych portach nie znaleziono SG-Lock.
SGL_LPT_OPEN_ERROR	3	Sterownik (driver)do LPT SG-Lock nie został znaleziony , mimo, że wsparcie LPT dla SG-Lock zostało zainstalowane.
SGL_NO_LPT_PORT_FOUND	4	Port LPT nie jest zainstalowany w komputerze, mimo, że wsparcie LPT dla SG-Lock zostało zainstalowane.
SGL_AUTHENTICATION_REQUIRED	5	Autentyfikacja za pomocą funkcji `SglAuthent` nie została przeprowadzona lub przeprowadzona z błędami.
SGL_AUTHENTICATION_FAILED	6	Nieudana autentyfikacja za pomocą funkcji `SglAuthent`.
SGL_FUNCTION_NOT_SUPPORTED	7	Wywołana funkcja nie jest wspierana przez zidentyfikowany SG-Lock (np.: `SglReadData` dla U2).
SGL_PARAMETER_INVALID	8	Parametr wywołanej funkcji nie leży w dozwolonym obszarze danych (np.: Adrese 5000 dla funkcji `SglReadData`).
SGL_SIGNATURE_INVALID	9	Sygnatura nieważna

Tab.1: Wartości zwrotne funkcji API SG-Lock i odpowiednie opisy.

5. Szyfrowanie, sygnowanie i administrowanie kluczami

SG-Lock dysponuje wewnątrzmodułową, symetryczną (tzn. kodowanie i dekodowanie tym samym kluczem) funkcją do szyfrowania 64-bitowych bloków (lub 2 podwóje słowa). Wielkość klucza wynosi 128 bitów lub 4 słowa podwójne.

Funkcja szyfrująca może być zastosowana do szyfrowania i deszyfrowania oraz do sygnowania dowolnych danych , takich jak konfiguracje, tajne dane itd. Szyfrowanie może przebiegać bezpośrednio w SG-Lock , co daje gwarancję, że klucz nie zostanie wyśledzony.
Ta wysoko jakościowo ochrona ogranicza jednak wydajność szyfrowania do ok. 100 bloków na sekundę, tzn. ca. 0,8 KB/sec, które jest spowodowane małą wydajnością rachunkową SG-Lock. W praktyce wykorzystuje się tę możliwość do szyfrowania małych ilości danych.
Szyfrowanie PC-CPU nie jest pewną ochroną, ponieważ klucz jest zapisany w systemie i może być wyśledzony. Zaletą tego wariantu jest szybkość szyfrowania do ponad 10 MB/sec.
Jest możliwa jednak kombinacja obu w.w. wariantów, tzn.: szyfrowanie wewnątrz SG-Lock i wewnatrz PC i wykorzystanie obu zalet, czyli wysoką wydajność szyfrowania i wysoką jakość. Wydajność jest w tym przypadku tak samo wysoka jak wydajność szyfrowania w samym PC.
W tej kombinacji szyfrowania, wszystkie 64-bitowe bloki danych zostaną osobno zaszyfrowane i ze sobą połączone. Pierwszy block i w rególarnych odstępach następne, będą zaszyfrowane przez SG-Lock-Hardware, wszystkie pozostałe przez PC-CPU. Szczególnie ważne dla bezpieczeństwa tej metody jest to, że, obydwa zastosowane, 128-bitowe klucze (wewnątrz SG-Lock i wewnątrz PC) są **różne**. W ten sposób wewnętrzny klucz PC jest w zasadzie do wyśledzenia, ale przez dodatkowe zaszyfrowanie **innym** (nieznanym) kluczem w SG-Lock i splecienie tych bloków ze sobą powstaje “powtórne“szyfrowanie i dododatkowe zabezpieczenie dla danych.

Przed dostawą SG-Lock wszystkie pamięci kluczy (1, 2 lub 16 kluczy, w zaleności od modułu) zostaną zainicjowane. Każdy użytkownik SG-Lock otrzyma własne tajne klucze, z których może korzystać lub w miarę potrzeby wygenerować nowe i zapisać w miejsce starych. (Kluczy typu U2/L2 nie można na nowo zapisać). Wszystkie moduły SG-Lock danego użytkownika zawierają jednakowy zestaw kluczy. Ten zestaw zostanie nowemu użytkownikowi dostarczony z pierwszą dostawą.

Wszystkie moduły demo (USB i LPT) mają swój własny, indywidualny zestaw kluczy, który przedstawia poniższa tabela:

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: Warsztatowo zaprogramowane 128-bitowe klucze modułów demo.

6. Przykłady programów

6.1. Funkcja 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. Funkcja 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!! }
    Memo1.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 & " occured !"
        Exit Sub
  End Select

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


End Sub
```

6.3. Funkcja 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);
}
```

```
// 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 !"
        Exit Sub
```

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

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

End Sub
```

6.4. Funkcja 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! ' +
```

```
                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
' addresse 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. Funkcja SglWriteData

C/C++:

```
#include "SGLW32.h"
#define PROD_ABC_ID 1
#define RUN_DATE_ADR 10   // addresse 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; { 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;
```

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
' addresse 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-Authentication modułu SG-Lock

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 appliction

// 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] )) {
```

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

}

// authentication successful ...
```

Inne przykłady programowania i konieczne pliki Include są do znalezienia na płytce CD-ROM .

7. Dane techniczne

7.1. SG-Lock USB

Interfejs	USB
Typ pamięci	RAM (zachowuje dane bez podtrzymania prądowego)
Pamięć	U2: brak pamięci U3: 256 Bytes U4: 1024 Bytes
Licznik 32-bitowy	U2: brak licznika U3: 16 U4: 64
128-bitowy klucz	U2: 1 (stały) U3: 2 (dowolny, do zaprogramowania) U4: 16 (dowolny, do zaprogramowania)
Algorytm	TEA
Cykle czytania	nieograniczone
Cykle zapisu	> 1.000.000
Zapis danych	Zaszyfrowane 128-bitowym kluczem
Zachowanie danych	> 20 lat
Zużycie prądu	< 50 mA
Rozmiary	63 x 16 x 8 mm (DxSxW)
Ciężar	8 g

7.2. SG-Lock LPT

Interfejs	LPT
Typ pamięci	RAM (zachowuje dane bez podtrzymania prądowego)
Pamięć	L2: brak pamięci L3: 256 Bytes L4: 1024 Bytes
32-bitowy licznik	L2: brak licznika L3: 16 L4: 64
128-bitowy klucz	L2: 1 (stały) L3: 2 (dowolny, do zaprogramowania) L4: 16 (dowolny, do zaprogramowania)
Algorytm	TEA
Cykle czytania	nieograniczone
Cykle zapisu	> 1.000.000
Zapis danych	Zaszyfrowane 128-bitowym kluczem
Zachowanie danych	> 20 lat
Zużycie prądu	< 50 mA
Rozmiary	50 x 54 x 17 mm (DxSxW)
Ciężar	42 g

Notatki