



# HAK2

**KLUCZ USB**

**DO ZABEZPIECZENIA OPROGRAMOWANIA  
I SZYFROWANIA DANYCH**

**Instrukcja 2.03.01**

Copyright © 2004 by **MicroMade**

All rights reserved

Wszelkie prawa zastrzeżone

# **MicroMade**

**Gałka i Drożdż sp. j.**

**64-920 PIŁA, ul. Wieniawskiego 16**

**Tel./fax: (67) 213.24.14**

**E-mail: [mm@micromade.pl](mailto:mm@micromade.pl)**

**Internet: [www.micromade.pl](http://www.micromade.pl)**

Wszystkie nazwy i znaki towarowe użyte w niniejszej publikacji są własnością odpowiednich firm.

## Spis treści

<b>1. Możliwości i zasoby klucza HAK2.....</b>	<b>4</b>
1.1 Możliwości klucza HAK2.....	4
1.2 Stałe w kluczu HAK2.....	4
1.2.1 Stałe wpisane przez MicroMade.....	4
1.2.2 Stałe wpisywane przez programistę.....	5
1.2.3 Stałe wpisywane przez administratora.....	5
1.3 Pamięć klucza HAK2.....	6
1.3.1 Pamięć haseł DH.....	6
1.3.2 Pamięć nazw użytkowników DN.....	6
1.3.3 Strefy pamięci danych D0, D1 i D2.....	6
<b>2. Używanie klucza HAK2.....</b>	<b>8</b>
2.1 Przygotowanie klucza HAK2.....	8
2.1.1 Etap 1 - przez programistę.....	8
2.1.2 Etap 2 - przez administratora.....	8
2.2 Korzystanie z kluczy HAK2.....	9
2.2.1 Znajdowanie klucza HAK2.....	9
2.2.2 Sprawdzanie obecności klucza HAK2.....	9
2.2.3 Szyfrowanie danych.....	9
2.2.4 Podpisywanie danych.....	9
2.2.5 Sesja.....	10
2.2.6 Użytkownicy.....	11
Użytkownik programisty (PU).....	11
Użytkownik.....	11
Użytkownik systemowy (SU).....	11
Właściciel.....	12
Administrator.....	12
<b>3. Oprogramowanie.....</b>	<b>13</b>
3.1 Funkcje szyfrujące.....	13
3.2 Inicjalizacja biblioteki.....	13
3.3 Mechanizm PNP (Plug&Play).....	14
3.4 Znajdowanie kluczy HAK2.....	14
3.5 Funkcje wykonujące rozkazy klucza HAK2.....	15
3.5.1 Konfiguracja klucza HAK2.....	15
3.5.2 Utworzenie sesji użytkownika (patrz 1.3.2 Sesja).....	15
3.5.3 Sprawdzanie podpisu programisty (patrz 2.2.4 Podpisywanie danych).....	15
3.5.4 Szyfrowanie.....	16
3.5.5 Funkcje administratora.....	16
3.5.6 Szyfrowane wiadomości adresowalne (SAM).....	17
3.6 Kompilacja biblioteki.....	17
3.7 Wielozadaniowość.....	18
3.8 Wielowątkowość.....	18
<b>4. Dane i rozkazy klucza HAK2.....</b>	<b>20</b>
4.1 Tablica powiązań danych i rozkazów klucza HAK2.....	20
4.2 Lista rozkazów klucza HAK2.....	21
4.3 Opis rozkazów klucza HAK2 (kolejność alfabetyczna).....	22
4.4 Obszary pamięci klucza HAK2.....	30
<b>5. Dodatki.....</b>	<b>31</b>
5.1 Plik DES.h.....	31
5.2 Plik Crypt_utils.h.....	33
5.3 Plik HAK2.h.....	35

# 1. Możliwości i zasoby klucza HAK2

## 1.1 MOŻLIWOŚCI KLUCZA HAK2

### Uwaga!

*W kryptologii wyróżnia się algorytmy (ogólnie znane) i klucze (tajne). W niniejszym opisie słowo klucz występuje w dwóch znaczeniach. W połączeniu z symbolem HAK2 oznacza urządzenie jakim jest klucz HAK2 przeznaczony do zabezpieczenia oprogramowania. W każdym innym wypadku oznacza tajny klucz (ciąg danych, zwykle małych rozmiarów, np. klucz do algorytmu DES ma 56 bitów) przeznaczony do szyfrowania danych za pomocą określonego algorytmu.*

Klucz HAK2 jest wykonany w formie wtyczki włączanej bezpośrednio w złącze USB. Do komunikacji z komputerem wykorzystuje najniższą ze zdefiniowanych w standardzie USB prędkości transmisji (Low Speed = 1,5Mb/s). W związku z tym wystarczy aby komputer posiadał złącze zgodne ze starszą specyfikacją USB 1.1.

HAK2 nie wymaga instalowania specyficznego dla siebie drivera. Korzysta ze standardowego drivera HID (human interface device), który jest dostępny w systemie Windows od wersji Windows'98. Driver ten musi być dostępny również w każdym innym systemie operacyjnym, który umożliwia podłączenie do komputera klawiatury USB.

Program może współpracować z wieloma kluczami HAK2 jednocześnie włączonymi w gniazda USB. Możliwa jest również bezkonfliktowa praca kilku aplikacji, z których każda komunikuje się ze swoim kluczem HAK2.

Transmisja danych między programem a kluczem HAK2 jest szyfrowana algorytmem DES. Dla każdej sesji tworzony jest inny klucz transmisyjny w oparciu o hasło użytkownika i liczby losowe.

Klucz HAK2 zawiera:

- stałe wpisane przez MicroMade,
- stałe wpisywane przez programistę (autora zabezpieczanego programu),
- stałe wpisywane przez administratora danego egzemplarza programu,
- pamięć haseł i uprawnień użytkowników (w tym administratorów) **DH**,
- pamięć nazw użytkowników **DN**,
- strefy pamięci danych **D0**, **D1** i **D2** o niezależnie ustalonym dostępie.

Klucz HAK2 umożliwia:

- szyfrowanie/desyfrowanie danych algorytmem DES i DES3 z kluczem ustalonym przez programistę,
- sprawdzanie autentyczności podpisu programisty pod dowolnym ciągiem danych,
- szyfrowanie/desyfrowanie danych algorytmem DES3 z kluczem ustalonym przez administratora,
- szyfrowanie wiadomości adresowalnych SAM,
- przechowywanie danych o użytkownikach (haseł, uprawnień, nazw) i danych o konfigurowalnym dostępie.

## 1.2 STAŁE W KLUCZU HAK2

### 1.2.1 Stałe wpisane przez MicroMade

- **VID** - identyfikator producenta = 0x13AB (nadany MicroMade przez organizację USB-IF)
- **PID** - identyfikator urządzenia = 0x0001
- **SN** - niepowtarzalny 4 bajtowy numer fabryczny danego egzemplarza HAK2
- **VN** - wersja oprogramowania (2 bajty)

### 1.2.2 Stałe wpisywane przez programistę

- Identyfikacja programu
  - ◆ **SID** - identyfikator programu (4 bajty)
  - ◆ **SOP** - opis programu (26 bajtów)
- Klucze
  - ◆ **S0** - klucz do sprawdzania podpisu programisty (DES3 - 168 bitów)
  - ◆ **S1** - klucz do szyfrowania/desyfrowania (DES3 - 168 bitów)
- Domyślny podział pamięci
  - ◆ **DRNGS** - patrz **RNGS** w **Stałe wpisywane przez administratora**
- Domyślna konfiguracja dostępu do stref pamięci
  - ◆ **DZAR** - patrz **ZAR** w **Stałe wpisywane przez administratora**
- Domyślne dane administratora
  - ◆ **DAID** - domyślny identyfikator egzemplarza/instalacji klucza HAK2 (4 bajty)
  - ◆ **DAOP** - domyślny opis egzemplarza/instalacji klucza HAK2 (18 znaków)
  - ◆ **DKA** - domyślny klucz administratora (168 bitów)
- Dane domyślnego użytkownika (jest on administratorem)
  - ◆ **DUID** - identyfikator (2 bajty), warunki: różny od **PUID** i **SUID** (patrz strona 11 rozdział 2.2.6 Użytkownicy)
  - ◆ **DPASS** - domyślne hasło (13 bajtów)
- Inne
  - ◆ **MST** - maksymalny czas sesji (1 bajt) - czas, po którym zostanie ona zerwana przez klucz HAK2 (patrz strona 10 rozdział 2.2.5 Sesja)
  - ◆ **RSVD** - zarezerwowane pole flagowe (1 bajt), w obecnej wersji wykorzystany tylko bit 0 (patrz strona 9 rozdział 2.2.2 Sprawdzanie obecności klucza HAK2), pozostałe bity należy ustawić na "1"
  - ◆ **RAC** - awaryjny kod kasujący (8 bajtów)

Gdy klucz jest resetowany (przez administratora bądź przez **RAC**) domyślne dane administratora są przepisywane do obszaru danych administratora (opis **DAOP** jest uzupełniany zapisanym heksalnie numerem fabrycznym klucza w rezultacie tworząc **AOP**), a dane domyślnego użytkownika są wpisywane do pierwszej strony pamięci hasel.

### 1.2.3 Stałe wpisywane przez administratora

- **AID** - identyfikator egzemplarza/instalacji klucza HAK2 (4 bajty)
- **AOP** - opis egzemplarza/instalacji klucza HAK2 (26 znaków)
- **A** - klucz administratora - do szyfrowania/desyfrowania (DES3 - 168 bitów)
- **RNGS** - podział pamięci (5 bajtów)
  - ◆ **MH** - rozmiar pamięci hasel
  - ◆ **MN** - rozmiar pamięci nazw użytkowników
  - ◆ **M0, M1, M2** - rozmiary stref pamięci danych
 

Stałe określają liczbę 16-bajtowych stron przypisanych dla każdego obszaru. Warunki:  $MH \geq 1$ ,  $MH \geq MN$ ,  $MH+MN+M0+M1+M2 \leq 250$
- **ZAR** - konfiguracja dostępu do stref pamięci: **Z0, Z1, Z2** (3 bajty), patrz strona 6 rozdział 1.3.3 Strefy pamięci danych D0, D1 i D2
- **OID** - identyfikator (**UID**) użytkownika-właściciela egzemplarza klucza HAK2
- **PAD0** - 32 bajty stałych do dowolnego wykorzystania przez aplikację
- **PAD1** - 32 bajty stałych do dowolnego wykorzystania przez aplikację

- **PAD2** - 10 bajtów stałych do dowolnego wykorzystania przez aplikację

## 1.3 PAMIĘĆ KLUCZA HAK2

Klucz HAK2 posiada 250 stron nieulotnej pamięci danych. Każda strona zawiera 16 bajtów. Administrator wpisując stałe **MH**, **MN**, **M0**, **M1** i **M2 (RNGS)** oraz **Z0**, **Z1** i **Z2 (ZAR)** podczas kasowania klucza ustala podział całej pamięci na pięć obszarów o określonych rozmiarach i prawach dostępu.

### 1.3.1 Pamięć haseł DH

Pamięć **DH** służy do przechowywania haseł użytkowników programu, a tym samym użytkowników klucza HAK2. Hasła umożliwiają zalogowanie się użytkownika (patrz strona 10 rozdział 2.2.5 Sesja). W kluczu HAK2 nie muszą być zapisywane bezpośrednio hasła wybrane przez użytkowników. Wskazane jest, aby były to raczej hasła przetworzone wybraną przez programistę funkcją jednokierunkową. Operacja ta nie wpływa na funkcjonowanie klucza HAK2, dla którego przesyłane dane są po prostu hasłami użytkowników.

Liczba dostępnych stron pamięci haseł określona jest stałą **MH**.

Zapis na stronie pamięci haseł zawiera:

- **UID** - identyfikator użytkownika (2 bajty),
- **PASS** - hasło (13 bajtów),
- **UPR** - uprawnienia (1 bajt).

Przy kasowaniu klucza HAK2 (przez administratora) kasowane są wszystkie zapisy w pamięci haseł i wpisywany jest jeden użytkownik o uprawnieniach administratora (z domyślnym identyfikatorem **DUID** i hasłem **DPASS** określonymi przez programistę).

Administrator może:

- wpisać nowego użytkownika (zwykły użytkownik może jedynie zmienić swoje hasło),
- odczytać dane o użytkowniku (z wyjątkiem hasła),
- usunąć użytkownika,
- zmienić uprawnienia użytkownika.

Użytkownicy są jednoznacznie rozróżniani poprzez identyfikator **UID**: ponowne wpisanie użytkownika z tym samym **UID** nadpisuje poprzedniego. Identyfikator **PUID** (0xFFFF) jest zarezerwowany do logowania się użytkownika programisty (**PU**) i w pozostałych operacjach dotyczących użytkowników jest nieprawidłowym parametrem.

### 1.3.2 Pamięć nazw użytkowników DN

Pamięć **DN** służy do przechowywania nazw (**NAME**) użytkowników wpisanych do pamięci haseł. Wewnętrznie w aplikacjach użytkownicy rozróżniani są przez unikalny identyfikator (w kluczu HAK2 jest to **UID**). Natomiast użytkownik posługuje się swoją nazwą - na jej podstawie aplikacja musi wyszukać identyfikator na liście użytkowników. Istnieją sytuacje, gdy aplikacja nie posiada listy użytkowników lub nie ma do niej dostępu. Klucz HAK2 umożliwia logowanie się poprzez nazwę tym użytkownikom, którym administrator taką nazwę wpisał.

Nazwa użytkownika składa się z 15 bajtów, 16-ty bajt służy do powiązania nazwy z użytkownikiem. Aby zapewnić poprawność identyfikacji programista musi zapewnić unikalność nazw - klucz HAK2 jej nie kontroluje. Podobnie jak w przypadku haseł dane **DN** nie muszą być wprost nazwami użytkowników.

Liczba dostępnych stron pamięci nazw użytkowników określona jest stałą **MN**. Administrator może dodawać, odczytywać i usuwać nazwy (indywidualnie dla każdego użytkownika).

### 1.3.3 Strefy pamięci danych D0, D1 i D2

Na każdej stronie można wpisywać dowolne dane (16 bajtów). Liczbę dostępnych stron pamięci danych w strefach **D0**, **D1** i **D2** określają stałe (odpowiednio) **M0**, **M1**, **M2** (w **RNGS**), a prawa dostępu stałe **Z0**, **Z1** i **Z2** (w **ZAR**). Dostęp do zapisu/kasowania stref przez określonych użytkowników określają bity 0..3 w **Zx** (1 = dozwolony):

- bit 0 - programista,

- bit 1 - administratorzy,
- bit 2 - właściciel,
- bit 3 - pozostali użytkownicy.

Bity 4..7 w analogiczny sposób określają uprawnienia do odczytu.

Strefa zablokowana do zapisu (bity 0..3 w Zx wyzerowane) jest strefą programisty: możliwy jest tylko zapis adresowalny. Programista ustala:

- dane (16 bajtów),
- numer strony (1 bajt) i strefy (1 bajt),
- numer fabryczny klucza HAK2, dla którego te dane są przeznaczone (4 bajty).

Następnie wytwarza cyfrowy podpis (8 bajtów) całego takiego zestawu korzystając z klucza **S0** i cały zestaw dostarcza administratorowi w celu wpisania danych do odpowiedniego klucza HAK2. Dane mogą zostać wpisane tylko na określoną stronę pamięci w określonej strefie i tylko gdy klucz **S0** w kluczu HAK2 jest taki sam jak klucz użyty przez programistę do wytworzenia podpisu.

Jeżeli podany (w podpisanym zestawie danych) numer fabryczny klucza HAK2 wynosi 0xFFFFFFFF to dane mogą zostać wpisane do klucza o dowolnym numerze fabrycznym, w przeciwnym wypadku dane mogą zostać wpisane tylko do jednego określonego klucza HAK2.

***Uwaga!***

***Uprawnienia dostępu do strefy D0 (tj. Z0) ustala programista: przy kasowaniu klucza przez administratora używane są domyślne uprawnienia DZ0 (z DZAR) zamiast podanych.***

Wartość 0 uprawnień (wszystkie bity wyzerowane) jest nieprawidłowa.

## 2.Używanie klucza HAK2

### 2.1 PRZYGOTOWANIE KLUCZA HAK2

#### 2.1.1 Etap 1 - przez programistę

Programista ustala wartości wszystkich wpisywanych przez siebie stałych. Następnie wpisuje je do klucza HAK2 i blokuje możliwość ich modyfikowania i kasowania.

Po wpisaniu stałych, a przed ich zablokowaniem programista w każdej chwili może skasować wszystkie wpisane przez siebie dane przywracając początkowy stan klucza HAK2.

Po zablokowaniu stałych programisty nadal istnieje możliwość ich skasowania, ale odpowiedni rozkaz wymaga wykazania się znajomością klucza S0, którego wartość zna tylko programista.

Przed zablokowaniem stałych programista może się dodatkowo upewnić, czy wpisane klucze (szczególnie **S0** - wymagany do kasowania) są na pewno zgodne z przyjętymi przez niego wartościami i czy stosowany przez niego algorytm szyfrowania jest na pewno taki sam jak w kluczu HAK2. Służą do tego wyliczane przez HAK2 sygnatury wpisanych do niego kluczy - wynik zaszyfrowania odpowiednim kluczem ciągu 8 bajtów zawierających same zera.

W momencie blokowania stałych programisty następuje przepisanie domyślnych wartości stałych administratora do obszaru stałych administratora i wpisanie jednego użytkownika o uprawnieniach administratora do pamięci haseł (z domyślnym identyfikatorem **DUID** i hasłem **DPASS** określonymi przez programistę).

#### 2.1.2 Etap 2 - przez administratora

W niektórych zastosowaniach klucza HAK2 etap ten (lub jego część) również odbywa się u programisty. Najczęściej jednak dostarcza on przygotowany(e) w etapie 1 klucz(e) HAK2 użytkownikowi programu i umożliwia mu wykonanie etapu 2 za pomocą przygotowanego przez siebie oprogramowania. Oprogramowanie to może dokonać części lub nawet całości etapu 2 automatycznie, bez interakcji z administratorem.

Administrator loguje się pod hasło otrzymane od programisty (bądź program rozpoznaje pusty klucz i loguje się automatycznie - patrz strona 11 rozdział 2.2.6 Użytkownik programisty (PU)) i:

- (opcjonalnie) resetuje klucz ustalając nowy podział pamięci **RNGS** i uprawnienia do stref danych **ZAR**,
- zmienia hasło na własne,
- ustala i wpisuje klucz **A**, jednakowy do wszystkich kluczy HAK2 tworzonej instalacji,
- wpisuje identyfikator **AID** i opis **AOP** egzemplarza klucza HAK2 (może to być identyfikator/opis instalacji, jednakowy we wszystkich egzemplarzach kluczy HAK2 danej instalacji), pamiętać należy, że dane te są jawne, dostępne do swobodnego odczytu,
- wpisuje stałe **PAD** (o ile przewiduje to aplikacja),
- dodaje kolejnych użytkowników i/lub administratorów,
- jednego z użytkowników lub administratorów ustala właścicielem egzemplarza klucza HAK2 - wpisuje jego identyfikator **UID** do stałej **OUID**,
- wpisuje dane do strefy (lub stref) pamięci.

Administrator może zawsze (po zalogowaniu się) zresetować klucz HAK2 do stanu w jakim został dostarczony przez programistę. Gdyby tak się zdarzyło, że zapomniane zostaną wszystkie hasła administratorów klucza HAK2, nadal istnieje możliwość jego zresetowania za pomocą kodu kasującego **RAC**.

## 2.2 KORZYSTANIE Z KLUCZY HAK2

### 2.2.1 Znajdowanie klucza HAK2

Znalezienie klucza HAK2 polega na uzyskaniu od drivera HID listy wszystkich urządzeń typu HID podłączonych do USB. Następnie na podstawie unikalnej kombinacji numerów **VID** i **PID** można wśród nich wyszukać klucze HAK2. Rozróżnienie poszczególnych kluczy HAK2 jest możliwe na podstawie ich numerów fabrycznych. Znalezienie przez aplikację "swoich" kluczy jest możliwe na podstawie **SID** i **SOP**.

### 2.2.2 Sprawdzanie obecności klucza HAK2

Znalezienie "swojego" klucza HAK2 jest w pewnym sensie sprawdzeniem jego obecności. Jednak wszelkie wykorzystywane w tym celu informacje są jawne. Istnieje więc możliwość zrobienia urządzenia USB (bądź programu udającego takie urządzenie), które zgłosi się dokładnie tak samo.

Wiarygodne sposoby identyfikacji klucza HAK2 to:

- wykorzystanie tajności klucza **S1** - jeżeli zaszyfrowanie losowych danych wewnątrz programu daje ten sam wynik co ich zaszyfrowanie przez klucz HAK2 to znaczy, że po obu stronach użyto tego samego klucza **S1** (zamiast umieszczać w programie klucz **S1** narażając go na ujawnienie można posłużyć się przygotowanym wcześniej odpowiednio dużym zestawem losowych danych i wyników ich zaszyfrowania),
- zaszyfrowanie fragmentu programu - klucz **S1** jest wykorzystywany do odszyfrowania fragmentu kodu lub danych (choćby pojawiających się na ekranie tekstów) programu przy każdym jego uruchomieniu,
- umieszczenie fragmentu programu (kodu lub danych) w pamięci klucza HAK2 i wczytywanie przy każdym uruchomieniu programu.

Klucz **S1** (a dokładniej jego początkowy fragment **S1A**) można użyć do szyfrowania algorytmem DES (w odróżnieniu od DES3). Algorytm ten jest obecnie mało przydatny - jego bezpieczeństwo mocno ogranicza mała długość klucza. Zalecane jest zablokowanie funkcji szyfrowania kluczem **S1A** poprzez wyzerowanie bitu 0 w stałej **RSVD** podczas przygotowywania klucza HAK2 przez programistę. Powoduje to poprawę bezpieczeństwa klucza **S1**.

### 2.2.3 Szyfrowanie danych

Klucz **A** nie jest ustalany przez programistę. W każdej instalacji programu ma inną wartość ustaloną przez administratora programu. Dlatego może on być wykorzystany do szyfrowania danych, które mają być dostępne tylko dla użytkownika programu w danej instalacji. Dotyczy to zarówno danych przechowywanych na dysku jak i danych przesyłanych między poszczególnymi komputerami tej samej instalacji.

Bezpośrednie zastosowanie klucza HAK2 (z kluczem **A**) do szyfrowania danych jest nieefektywne czasowo (patrz niżej) i nie daje zmienności stosowanych do szyfrowania danych kluczy. Jedną z najprostszych możliwości jest szyfrowanie "pośrednie": dane są szyfrowane na komputerze za pomocą wygenerowanego losowo klucza **K**, który następnie zostaje zaszyfrowany kluczem HAK2 (z kluczem **A**). Klucz **K** w tej zaszyfrowanej postaci może być jawnie przesyłany między komunikującymi się ze sobą komputerami lub dołączony do szyfrowanego bloku danych na dysku - jest on bezużyteczny bez klucza HAK2 z właściwym kluczem **A**.

Klucz HAK2 umożliwia również zaszyfrowanie porcji danych (wiadomości SAM) z określeniem ich adresata (wewnątrz grupy o tych samych kluczach **A** lub **S1**) - odszyfrować potrafi je tylko HAK2 o numerze fabrycznym wskazanym przy szyfrowaniu (zwróci też numer fabryczny nadawcy). Dzięki temu możliwe jest przysyłanie poufnych danych do wybranych członków grupy.

Pojedyncza operacja zaszyfrowania/odszyfrowania ciągu 8 bajtów algorytmem DES za pomocą klucza HAK2 zajmuje około 22 ms, a algorytmem DES3 około 25ms. Mała różnica między tymi czasami wynika z dużego udziału w nich czasów transmisji i jej szyfrowania. W obu wypadkach między komputerem a kluczem HAK2 przesyłane są takie same ilości danych, których szyfrowanie wymaga dodatkowo 6-krotnego wykonania algorytmu DES.

Dla porównania pojedyncza operacja szyfrowania 8-bajtowego bloku algorytmem DES na komputerze PC z procesorem klasy Pentium 1.7GHz zajmuje około 450 ns (49 000 razy szybciej), a algorytmem DES3 około 1.35 μs (18 500 razy szybciej).

### 2.2.4 Podpisywanie danych

Dowolny ciąg danych można zabezpieczyć przed modyfikacją przez nieuprawnione osoby: wystarczy wytworzyć i dołączyć do nich podpis. Odpowiedni algorytm łączy ciąg danych i tajny klucz dając w wyniku podpis charakterystyczny

ny dla tego ciągu danych. Ciąg danych z dodanym podpisem może być następnie przesłany w sposób jawny: adresat znający tajny klucz może zweryfikować autentyczność ciągu danych przez wyliczenie podpisu i porównanie z otrzymanym. Bez znajomości tajnego klucza nie da się wytworzyć poprawnego podpisu, tak więc adresat ma pewność co do poprawności danych i tożsamości ich nadawcy.

Klucz HAK2 umożliwia weryfikację podpisu wytworzonego przy pomocy klucza **S0**. Programista może to na przykład wykorzystać w celu niezależnego udostępniania poszczególnych funkcji programu poszczególnym użytkownikom jego kopii. Informacja o udostępnionych funkcjach może się znajdować w pliku (może być jawnym tekstem) na którego końcu znajduje się podpis wszystkich zawartych w nim informacji. Plik może zostać związany z konkretnym użytkownikiem kopii programu poprzez wpisanie do pliku nazwy użytkownika. Aby całkowicie uniemożliwić użytkownikom wymienianie się między sobą plikami określającymi udostępnione funkcje można dostarczać każdemu z nich klucz HAK2 z innym kluczem **S0**.

Inną możliwością jest przechowywanie informacji o uprawnieniach w strefie pamięci danych programisty (patrz strona 6 rozdział 1.3.3 Strefy pamięci danych D0, D1 i D2) klucza HAK2. W tym wypadku nawet przy jednakowych kluczach **S0** można wiązać uprawnienia tylko z jednym użytkownikiem kopii programu poprzez umożliwienie ich zapisu tylko do konkretnego egzemplarza klucza HAK2.

## 2.2.5 Sesja

Wszystkie istotne funkcje klucza HAK2 stają się dostępne dopiero po zalogowaniu się użytkownika, czyli otwarciu jego sesji. Wygenerowany wówczas (losowo) klucz transmisyjny chroni wymianę danych pomiędzy programem a kluczem HAK2 - transmisja szyfrowana jest algorytmem DES. Trwanie sesji jest ograniczone w czasie (może trwać najwyżej **MST** \* 250ms, bądź nieskończoność dla **MST=0**) i w liczbie przesłanych danych (maksymalnie 256 rozkazów). Dzięki temu uniknięto użycia 3-krotnie wolniejszego algorytmu DES3 przy zachowaniu odporności klucza transmisyjnego na złamanie złożonymi metodami kryptoanalitycznymi (wymagają one z reguły dużej ilości danych zaszyfrowanych tym samym kluczem). Szyfrowanie transmisji przebiega w taki sposób, że dane zaszyfrowane wyglądają za każdym razem inaczej, nawet jeśli faktycznie przesyłane dane są identyczne. Dzięki temu transmisja jest odporna na powtarzanie danych przez atakującego. Sesja jest zrywana przy wystąpieniu błędów nietypowych w normalnym działaniu (np. zły format danych, dane nieprawidłowo zaszyfrowane itp.). Jest to dodatkowe zabezpieczenie przed próbami modyfikacji przesyłanych danych.

Część klucza transmisyjnego generuje program, a część HAK2. Program przesyła do klucza HAK2 identyfikator **UID** (bądź nazwę **NAME**) logującego się użytkownika i swoją część klucza transmisyjnego zaszyfrowaną hasłem podanym przez użytkownika. W odpowiedzi HAK2 przesyła swoją część klucza (również zaszyfrowaną) oraz uprawnienia użytkownika.

Jeżeli:

- nie ma użytkownika o podanym identyfikatorze (bądź nazwie),
- jest użytkownik, ale podane hasło jest błędne (licznik błędnych prób logowania jest zwiększany - o ile nie jest przepełniony),
- licznik błędnych logowań jest przepełniony (3 błędne próby),
- token (pierwsze 4 bajty części klucza wygenerowanej przez program) został powtórzony,

to HAK2 zwraca błąd logowania.

Przepełnienie licznika błędnych prób logowania powoduje, że żadna próba logowania nie zostanie zaakceptowana. Licznik zmniejszany jest o 1 co minutę - daje to szansę na kolejną próbę. Po maksymalnie 3 minutach od przepełnienia licznik jest wyzerowany (o ile nie było kolejnych nieudanych prób) - dostępne są znowu wszystkie 3 próby. Kasowanie licznika następuje również po wyjęciu klucza HAK2 z portu USB.

Uprawnienia równe 1 oznaczają, że dany użytkownik jest administratorem. Po zalogowaniu się administratora udostępniane są wszystkie te operacje, które może w kluczu HAK2 wykonać jedynie administrator.

Uprawnienia z zakresu 2 do 255 oraz 0 oznaczają zwykłego użytkownika (bez uprawnień administratora). Wartości te mogą być dowolnie wykorzystane przez programistę.

Program jest z reguły tak skonstruowany, że przed wykonaniem każdej operacji (lub ich ciągu) w kluczu HAK2 dokonuje logowania użytkownika. Logowanie (tj. weryfikacja hasła użytkownika i wytworzenie klucza transmisyjnego) zajmuje stosunkowo dużo czasu, dlatego istnieje mechanizm podtrzymywania sesji dla tego samego użytkownika: pod pewnymi warunkami możliwe jest zachowanie poprzedniego klucza transmisyjnego.

## 2.2.6 Użytkownicy

Wszystkie istotne funkcje klucza HAK2 stają się dostępne dopiero po zalogowaniu się użytkownika (patrz strona 10 rozdział 2.2.5 Sesja). Programy korzystające z klucza HAK2 mogą potrzebować dostępu do pewnych funkcji również bez interakcji z użytkownikiem (np. sprawdzenie podpisu pod licencją przy uruchomieniu programu). Do tego celu mogą zostać wykorzystani użytkownicy: programisty i systemowy.

### Użytkownik programisty (PU)

- Identyfikator (UID): **PUID=0xFFFF**.
- Hasło (PASS): **DPASS**.
- Uprawnienia (UPR): 0.

Umożliwia:

- weryfikację autentyczności ciągu danych,
- sprawdzenie obecności i autentyczności klucza HAK2 - szyfrowanie kluczem **S1**,
- szyfrowanie wiadomości SAM kluczem **S1**,
- odczyt/zapis/kasowanie w udostępnionych strefach danych,
- odczyt niektórych stałych programisty i administratora.

Hasło tego użytkownika ustalone jest przez programistę w trakcie przygotowywania klucza HAK2. Jego **UID**, **PASS** i **UPR** nie są przechowywane w pamięci haseł i dlatego nie mogą być zmodyfikowane - pozostają stałe do momentu skasowania klucza przez programistę. Użytkownika **PU** nie można też usunąć - istnieje stale w czasie normalnego użytkowania klucza HAK2 (tj. od wpisania stałych programisty do ich skasowania). Użytkownik ten służy do ochrony programu: ma najbardziej ograniczone uprawnienia spośród wszystkich użytkowników. W jego sesji możliwe jest wiarygodne sprawdzenie podpisu ciągu danych (patrz strona 9 rozdział 2.2.4 Podpisywanie danych) - wynik sprawdzenia przesyłany jest w formie zaszyfrowanej. Szyfrowanie to oparte jest o hasło **DPASS**, z tego powodu musi być ono w programie chronione (ukryte). Wskazane jest, aby hasło to było inne w każdym kluczu HAK2, np. powiązane w jakiś tajny sposób z numerem fabrycznym klucza, a w programie wylizane.

### Użytkownik

- Identyfikator (UID): dowolny różny od **SUID** (patrz **Użytkownik systemowy (SU)**) i **PUID**, czyli z zakresu 0x0000..0xFFFFD.
- Hasło (PASS): dowolne.
- Uprawnienia (UPR): dowolne różne od 1.

Umożliwia wszystko to, co użytkownik programisty, i:

- szyfrowanie kluczem **A**,
- zmianę hasła.

Użytkowników do pamięci haseł wpisuje administrator, ich liczba ograniczona jest stałą **MH**.

### Użytkownik systemowy (SU)

- Identyfikator (UID): **SUID=0xFFFE**.
- Hasło (PASS): dowolne.
- Uprawnienia (UPR): dowolne.

Umożliwia wszystko to, co użytkownik.

Ostatnie 4 bajty hasła użytkownika systemowego (jego sygnatura **SUS**) są jawne i dostępne do swobodnego odczytu. Znajduje on zastosowanie w złożonych aplikacjach, które:

- wykorzystują większą liczbę egzemplarzy klucza HAK2, z których nie wszystkie są dostępne jednocześnie (np. osobiste klucze użytkowników aplikacji sieciowej),
- potrzebują automatycznego (bez interwencji człowieka) dostępu do kluczy HAK2 na prawach użytkownika, z tego powodu posiadają własne hasło,
- dokonują okresowej automatycznej zmiany swego hasła.

Dzięki sygnaturze **SUS** i przechowywanej historii haseł mają one możliwość identyfikacji nieaktualnego hasła w kluczu HAK2, który nie był dostępny w chwili okresowej zmiany hasła. Następnie dokonują automatycznej aktualizacji hasła. Niezależnie od wpisanych uprawnień użytkownik systemowy nigdy nie jest administratorem - jego hasło jest mniej bezpieczne (tj. efektywnie krótsze) od haseł pozostałych użytkowników.

### Właściciel

- Identyfikator (**UID**): **OID** - stała administratora.
- Hasło (**PASS**): dowolne.
- Uprawnienia (**UPR**): dowolne.

Umożliwia wszystko to, co użytkownik, i:

- szyfrowanie wiadomości SAM kluczem **A**.

Jest to wyróżniony użytkownik: może on posiadać strefę danych, do której w żaden sposób nie będzie miał dostępu nie tylko inny użytkownik, ale nawet administrator - z wyjątkiem sytuacji, w której to właśnie administrator jest właścicielem.

### Administrator

- Identyfikator (**UID**): dowolny różny od **PUID** i **SUID**.
- Hasło (**PASS**): dowolne.
- Uprawnienia (**UPR**): 1.

Umożliwia wszystko to, co użytkownik, i:

- zapis stałych administratora,
- dodawanie/nadpisywanie/usuwanie użytkowników (w tym administratorów) i ich nazw,
- zmianę uprawnień **UPR** dowolnemu użytkownikowi z wyjątkiem programisty,
- ustanowienie właściciela,
- reset klucza HAK2 - skasowanie wszystkich danych, przywrócenie wartości domyślnych stałym administratora (stałym **RNGS** i **ZAR** nadanie wartości podanych przez siebie),
- odczyt stałych **PAD**,
- odczyt informacji o użytkownikach: ich identyfikatorów, uprawnień i nazw.

Po resecie klucza HAK2 w pamięci haseł istnieje tylko administrator domyślny z identyfikatorem **UID = DUID**, **PASS = DPASS**, **UPR = 1** (oczywiście stale dostępny jest też użytkownik programisty).

Administrator może ustanowić właścicielem egzemplarza klucza HAK2 jednego z istniejących w nim użytkowników - identyfikator użytkownika zostaje wpisany do stałej **OID**. Operacja ta jest jednorazowa - nie można zmienić ani nadpisać właściciela bez konieczności resetu klucza HAK2. Reset kasuje wszystkie dane (w szczególności stałą **OID**) - w ten sposób zabezpieczone są one również przed administratorem.

Liczba administratorów w kluczu HAK2 ograniczona jest tylko stałą **MH**, ale w większości zastosowań występuje tylko jeden administrator w każdym egzemplarzu klucza.

## 3.Oprogramowanie

Dla programistów przygotowana została prosta biblioteka do obsługi klucza HAK2 w systemie operacyjnym Windows'98 i następnych. Udostępniona jest ona w wersji źródłowej w języku C, dzięki czemu możliwe jest dostosowanie jej do indywidualnych potrzeb (np. do innego systemu operacyjnego) i włączenie do aplikacji.

### Uwaga!

*Zwracana przez większość funkcji biblioteki wartość typu 'int' oznacza status wykonania funkcji: 0-błąd, 1-sukces.*

### 3.1 FUNKCJE SZYFRUJĄCE

Częścią biblioteki są funkcje szyfrujące algorytmem DES (i jego odmianami) zadeklarowane w pliku "DES.h". Algorytm DES przy pomocy klucza K (56 bitów) przekształca jawny blok danych P (64 bity) w blok zaszyfrowany C (również 64 bity):  $C = EK(P)$ . Analogicznie przebiega odszyfrowanie:  $P = DK(C)$ . Podane przekształcenia wykonuje funkcja DES\_crypt(): zaszyfrowuje ('cr\_dir' = 0) lub odszyfrowuje ('cr\_dir' != 0) blok 'block' przetworzonym kluczem 'key'. Klucz 'key' generuje funkcja DES\_prepare\_key() z właściwego klucza DES 'raw\_key' - wytworzony w ten sposób klucz można używać do szyfrowania wielu bloków, jest to metoda efektywniejsza czasowo niż posługiwanie się kluczem DES bezpośrednio. Klucz 'raw\_key' podaje się w formie 8 bajtów, w których najmłodsze bity są ignorowane.

### Uwaga!

*Algorytm DES jest algorytmem symetrycznym - nadawca i odbiorca zaszyfrowanych danych muszą znać klucz szyfrujący, jest on ich wspólnym sekretem. Dane są tak bezpieczne, jak chroniący je klucz: musi być unikalny, pozostać niedostępny dla postronnych i trudny do zgadnięcia. Warunki te można spełnić poprzez wylosowanie wszystkich 56 bitów klucza - niewskazane jest, aby był on hasłem ani w ogóle ciągiem tekstu. Każde zaszyfrowane dane można odszyfrować wypróbując po kolei wszystkie możliwe klucze (tzw. atak metodą "brute force"): jedyną obroną jest olbrzymia liczba możliwych kluczy, co przekłada się na czas wykonania ataku. Zawężenie klucza do znaków tekstu istotnie zmniejsza liczbę możliwych kluczy powodując przyspieszenie tego rodzaju ataku.*

Odmianą algorytmu DES jest algorytm DES3 (3DES, Triple-DES, TDEA). Szyfrowanie nim składa się z 3 operacji DES z 3 kluczami K1, K2 i K3. Istnieją 2 wersje algorytmu:

- EDE: zaszyfrowanie  $C = EK3(DK2(EK1(P)))$ , odszyfrowanie  $P = DK1(EK2(DK3(C)))$ ,
- EEE: zaszyfrowanie  $C = EK3(EK2(EK1(P)))$ , odszyfrowanie  $P = DK1(DK2(DK3(C)))$ .

Pierwszą wersję realizuje funkcja DES3\_crypt(), drugą DES3\_EEE\_crypt(). Klucz HAK2 realizuje wersję EEE.

W pliku "Crypt\_utils.h" zadeklarowane są funkcje użytkowe wykorzystujące szyfrowanie algorytmem DES/DES3 (m.in. szyfrujące ciągi danych w trybach CBC i CFB). Funkcja CrUtil\_Sign\_data\_MM() umożliwia wytworzenie podpisu programisty pod dowolnym ciągiem bajtów.

### Uwaga!

*Wymienione powyżej funkcje dokonują szyfrowania programowo - nie korzystają z klucza HAK2. Są przydatne np. przy realizacji szyfrowania "pośredniego" (patrz strona 9 rozdział 2.2.3 Szyfrowanie danych).*

### 3.2 INICJALIZACJA BIBLIOTEKI

Bibliotekę należy zainicjalizować używając funkcji HAK2\_Init\_library() w celu nadania wartości początkowych niektórym zmiennym wewnętrznym. Numer wersji biblioteki w kodzie BCD zwraca funkcja HAK2\_Get\_lib\_version().

Po zakończeniu pracy z biblioteką (np. przed zamknięciem aplikacji) należy wywołać funkcję HAK2\_Close\_library() w celu zwolnienia zasobów systemowych zajętych przez bibliotekę. Sekwencję inicjalizacja/zakończenie można wykonywać wielokrotnie.

### 3.3 MECHANIZM PNP (PLUG&PLAY)

Biblioteka przechowuje listę dołączonych kluczy HAK2, maksymalnie 127 (stała HAK2\_MAX\_NOF\_DEVS w pliku "HAK2.h"). Do odświeżenia tej listy służy funkcja HAK2\_OnDeviceChange(). Odświeżenie listy jest konieczne po zainicjalizowaniu biblioteki ("stworzenie" listy) i po każdym włożeniu lub wyjęciu klucza HAK2.

Klucz HAK2 tak jak wszystkie urządzenia USB posiada właściwości PNP i biblioteka umożliwia ich wykorzystanie. System Windows przy każdym zdarzeniu związanym z urządzeniem PNP (np. włożeniu lub wyjęciu) wysyła komunikat WM\_DEVICECHANGE do wszystkich aplikacji (ich okien nadrzędnych: "top-level windows", zobacz w dokumentacji Microsoft Platform SDK lub na stronie [www.msdn.com](http://www.msdn.com)). W zależności od typu urządzenia może to być nawet kilka komunikatów dla jednego zdarzenia. Istnieje możliwość otrzymywania komunikatów specyficznych dla urządzeń typu HID. W tym celu należy przekazać bibliotece uchwyt okna (adresata komunikatów) przy pomocy funkcji HAK2\_SetMsgsDestHandle(), a następnie wywołać funkcję HAK2\_Reg\_for\_msgs(). Można wtedy odświeżać listę urządzeń (funkcją HAK2\_OnDeviceChange()) tylko wtedy, gdy jest to konieczne, tj. po otrzymaniu komunikatu WM\_DEVICECHANGE dotyczącego urządzenia HID. Komunikat ten należy rozpoznać wykorzystując identyfikator GUID drivera HID (dostępny przez funkcję HAK2\_Get\_GUID()).

Każda zmiana w liście urządzeń (spowodowana włożeniem/wyjęciem klucza HAK2 lub zmianą w deskrytorze któregoś z urządzeń np. po wykonaniu rozkazu A\_WR) powoduje wysłanie przez bibliotekę komunikatu powiadamiającego do okna wskazanego funkcją HAK2\_SetMsgsDestHandle(). Funkcją tą definiuje się też identyfikator komunikatu powiadamiającego. W pierwszym parametrze (wParam) tego komunikatu biblioteka wysyła liczbę urządzeń przed zmianą, w drugim (lParam) liczbę urządzeń po zmianie.

Zastosowanie opisanego tu mechanizmu zademonstrowane jest w przykładowej aplikacji do konfiguracji klucza HAK2.

### 3.4 ZNAJDOWANIE KLUCZY HAK2

Do wyszukiwania kluczy HAK2 z listy biblioteki służy funkcja HAK2\_Get\_devs(). Zwraca ona liczbę urządzeń spełniających podane kryteria wyszukiwania i ich deskrytory. W deskrytorze znajdują się wszystkie dane identyfikujące klucz HAK2 zwracane przez rozkaz ID\_RD (w tym numer fabryczny służący do identyfikacji urządzenia w bibliotece). Kryteria wyszukiwania to dwa deskrytory: wzorzec i maska. We wzorcu ustawia się wartości szukane (np. identyfikator aplikacji **SID**), a w masce wyzerowuje się wszystkie pola nieznaczące. Tworzenie wzorców/masek ułatwiają stałe 'HAK2\_exact'/HAK2\_any' (całe deskrytory) oraz HAK2\_EXACT/HAK2\_ANY (do pól całkowitych).

- Przykład 1 - wyszukiwanie maksymalnie 5 kluczy HAK2 współpracujących z aplikacją o identyfikatorze 0x1234ABCD.

```
THAK2_Dev_dscr tmpl;
THAK2_Dev_dscr mask = HAK2_any;
THAK2_Dev_dscr dscr_tbl[5];
UINT nof_devs;

mask.SID = HAK2_EXACT;
tmpl.SID = 0x1234ABCD;

nof_devs = HAK2_Get_devs(&tmpl, &mask, dscr_tbl, 5);
if (nof_devs == 0)
    // nie znaleziono żadnego urządzenia
else
    // znaleziono 'nof_devs' urządzeń (być może więcej niż 5), deskrytory
    // pierwszych pięciu z nich zapisane w dscr_tbl[]
```

- Przykład 2 - wyszukiwanie jednego klucza HAK2 o numerze fabrycznym pomiędzy 0x12345600 a 0x123456FF.

```
THAK2_Dev_dscr tmpl;
THAK2_Dev_dscr mask = HAK2_any;
THAK2_Dev_dscr dscr;

mask.SN = 0xFFFFFFFF00;
tmpl.SN = 0x12345600;

if (HAK2_Get_devs(&tmpl, &mask, dscr, 1) == 0)
    // nie znaleziono żadnego urządzenia
else
```

```
// urządzenie znalezione
```

## 3.5 FUNKCJE WYKONUJĄCE ROZKAZY KLUCZA HAK2

Biblioteka umożliwia wykonanie każdego rozkazu klucza HAK2 (patrz strona 22 rozdział 4.3 Opis rozkazów klucza HAK2) przez wywołanie funkcji o tej samej nazwie poprzedzonej przedrostkiem "HAK2\_Cmd\_" (np. HAK2\_Cmd\_ID\_Rd()). Istnieją też funkcje integrujące działanie kilku różnych rozkazów i/lub wykonujące ich serię, np. HAK2\_Check\_sign(), HAK2\_DX\_rd\_n().

Pierwszym parametrem każdej z tych funkcji jest numer fabryczny klucza HAK2, w którym ma zostać wykonana żądana operacja. Status (kod błędu) wykonania ostatniej takiej operacji można odczytać funkcją HAK2\_Get\_status(). Zwracane kody zdefiniowane są w pliku "HAK2err.h".

### 3.5.1 Konfiguracja klucza HAK2

W nowym (lub skasowanym) kluczu HAK2 dostępny jest jedynie rozkaz identyfikujący (ID\_RD) i rozkazy konfigurujące (ERASE, PSK\_WR, PSD\_WR, PS\_LOCK). Klucze takie wyróżnia ustawiony bit 4 (HAK2\_FLAG\_LI) pola F w deskrypcorze.

Do konfiguracji klucza HAK2 (tj. wpisania stałych programisty) służy funkcja HAK2\_Setup(). Stałe programisty przekazywane są w strukturze 'setup\_data'. Funkcja wpisuje klucze **S0** i **S1** (rozkaz PSK\_WR), sprawdza poprawność otrzymanych sygnatur, wpisuje resztę stałych konfiguracyjnych (rozkaz PSD\_WR, 4x) wraz z wyliczoną sumą kontrolną i blokuje zapis (rozkaz PS\_LOCK). W przypadku niepowodzenia (np. na skutek podania nieprawidłowych stałych) należy skasować klucz (rozkaz ERASE) przed kolejną próbą konfiguracji.

Po skonfigurowaniu dostępne stają się wszystkie rozkazy jawne, wszystkie bity pola F w deskrypcorze zostają wyzerowane.

Programista może skasować klucz HAK2 znając wpisany do niego klucz **S0**. W tym celu musi pobrać z urządzenia liczbę losową (rozkaz RND\_RD), zaszyfrować ją kluczem **S0** (DES3, patrz strona 13 rozdział 3.1 Funkcje szyfrujące, funkcje DES\_prepare\_key() i DES3\_EEE\_crypt()) i przekazać jako parametr rozkazu ERASES0.

### 3.5.2 Utworzenie sesji użytkownika (patrz 1.3.2 Sesja)

Do utworzenia sesji (zalogowania użytkownika) służy funkcja HAK2\_Login(). W zależności od podanych parametrów użytkownik identyfikowany jest przez **UID** (rozkaz LOGIN) lub nazwę **NAME** (rozkaz LOGIN\_N). Parametr 'token' jest użyty jako część liczby losowej generowanej przez bibliotekę (pierwsze cztery bajty), jego zastosowanie opisane jest w **3.5.3 Sprawdzenie podpisu programisty**. Powtórzenie tej samej wartości w kolejnym logowaniu jest odrzucone przez klucz HAK2 (błąd logowania).

W zależności od rodzaju zalogowanego użytkownika dostępne stają się rozkazy (patrz strona 21 rozdział 4.2 Lista rozkazów klucza HAK2):

- administrator: wszystkie rozkazy szyfrowane,
- użytkownik programisty: rozkazy szyfrowane "P",
- pozostali użytkownicy: rozkazy nie będące rozkazami tylko dla administratora.

Szyfrowania transmisji (rozkażów szyfrowanych) po zalogowaniu dokonuje biblioteka - jest ono przezroczyste z punktu widzenia programisty. Sesję można zakończyć rozkazem LOGOUT.

### 3.5.3 Sprawdzenie podpisu programisty (patrz 2.2.4 Podpisywanie danych)

Zastosowaniem sprawdzania podpisu jest przede wszystkim weryfikacja autentyczności pliku licencji określającego dostępne opcje programu takie jak wybrane funkcje, liczba stanowisk, użytkowników, okres ważności itp. Do dokonania weryfikacji podpisu konieczna jest znajomość klucza **S0** służącego do jego wytworzenia. Z tego względu nie może ona być przeprowadzona w programie - umieszczenie w jego kodzie klucza naraża klucz na ujawnienie (umożliwiłoby to generację własnych plików licencji). Dlatego weryfikacji dokonuje klucz HAK2.

Istnieje jeszcze jedno zagrożenie dla programu: modyfikacja jego kodu w celu ominięcia weryfikacji. Aby maksymalnie utrudnić taką modyfikację należy dokonać sprawdzenia licencji w możliwie największej liczbie miejsc w programie (w jego kodzie) - wyszukanie i zmodyfikowanie ich wszystkich będzie czasochłonne. Nie może to być jednak proste wywo-

łanie funkcji i sprawdzenie zwróconego rezultatu - wystarczy wtedy zmodyfikować tę funkcję. Biblioteka wraz z kluczem HAK2 umożliwiają realizację bardziej skomplikowanego schematu.

W dalszej części punktu **3.5.3** pod pojęciem logowania należy rozumieć nawiązanie sesji, w której nastąpi (lub nastąpiło) sprawdzenie podpisu w kluczu HAK2 (tj. wykonanie rozkazu ST\_CHK, patrz poniżej).

Jako token przy logowaniu powinien zostać użyty czas systemowy (z rozdzielczością najlepiej ok. 40ms). Program sprawdza licencję funkcją Check\_sign(). Dzieli ona ciąg na bloki i wysyła do klucza HAK2 (rozkaz ST\_DTA), a następnie wysyła podpis (rozkaz ST\_CHK) i otrzymuje wynik sprawdzenia w postaci bloku ST\_RSP zaszyfrowanego bieżącym kluczem sesyjnym. Zestaw składający się z bloku i klucza zwraca programowi, który nie testuje zestawu natychmiast, ale zapamiętuje go. Najlepiej jest gromadzić większą liczbę zestawów (każdy pobierany osobno, w losowym czasie), a sprawdzenia losowo wybranego zestawu (lub kilku z nich) dokonywać w innych wątkach po upływie losowego czasu. Do testowania zestawu służy przykładowe makro HAK2\_CHECK\_SIGN(). W bloku ST\_RSP zawarte są:

- token użyty przy otwieraniu bieżącej sesji,
- 2 młodsze bajty numeru fabrycznego klucza,
- pierwszy bajt sprawdzanego ciągu danych,
- wynik sprawdzenia: 0 - podpis poprawny, inna wartość - podpis błędny.

Makro odszyfrowuje blok ST\_RSP w podanym zestawie i porównuje swoje parametry z zawartością bloku. Jako token powinien tu być przekazany aktualny czas systemowy. Może on się różnić od czasu logowania, dlatego jednym z parametrów makra jest maksymalna dopuszczalna różnica (tj. najdłuższy możliwy czas pomiędzy logowaniem a sprawdzeniem zestawu). Jeżeli sprawdzenie wypadnie pomyślnie, blok ST\_RSP pozostaje zaszyfrowany - można go testować ponownie w ten sam sposób.

Makro HAK2\_CHECK\_SIGN() podane jest jako przykładowe - programista powinien je zmodyfikować, aby nie ułatwiać wyszukania jego wystąpień w skompilowanym kodzie swego programu. Najlepiej gdy każde wywołanie makra będzie generowało inny kod - można do tego użyć zmylającej operacji wykorzystującej np. numer linii w pliku źródłowym i/lub część nazwy tego pliku.

### 3.5.4 Szyfrowanie

Do szyfrowania bloku 8 bajtów służą rozkazy DES\_S1A, DES3\_S1 i DES3\_A. Rozkaz DES3\_A\_8 szyfruje jednocześnie 8 bloków w trybie ECB - ten sam efekt można uzyskać wykonując rozkaz DES3\_A osobno dla każdego bloku, ale trwałoby to znacznie dłużej ze względu na narzut protokołu komunikacyjnego. Parametr 'cr\_dir' określa kierunek szyfrowania: 0 - zaszyfrowanie, 1 - odszyfrowanie.

Funkcja HAK2\_CryptECB\_S1A\_S1\_A() integruje działanie wymienionych powyżej rozkazów szyfrujących. Służy do zaszyfrowania w trybie ECB wybranym algorytmem/kluczem ciągu danych o długości wskazanej przez 'data\_size'. Dane przed wysłaniem do klucza HAK2 dopełniane są zerami do długości będącej wielokrotnością 8 bajtów, co musi być uwzględnione przy określaniu rozmiaru przekazanego bufora. Długość danych po dopełnieniu zwracana jest za pośrednictwem 'data\_size'.

### 3.5.5 Funkcje administratora

Najważniejszą funkcją administratora jest przygotowanie klucza (lub kluczy) HAK2 do pracy w określonej instalacji. W tym celu:

- loguje się używając domyślnego identyfikatora **DUID** i hasła **DPASS** (program powinien to robić automatycznie: **DPASS** powinno pozostać tajne - patrz strona 11 rozdział 2.2.6 Użytkownik programisty (PU))
- opcjonalnie (jeżeli chce ustalić własny podział pamięci danych **RNGS** i/lub uprawnienia dostępu do stref danych **ZAR**) resetuje klucz HAK2 (rozkaz RESET) i loguje się ponownie,
- zmienia swoje hasło (rozkaz PASS\_CHG), wylogowuje się (ważne: w celu uniknięcia podtrzymania sesji) i loguje ponownie pod nowym (nie ma możliwości zmiany identyfikatora użytkownika - jeżeli nie odpowiada mu domyślny identyfikator **DUID**, musi dodać nowego administratora, po czym usunąć domyślnego),
- wpisuje klucz **A** wraz z identyfikatorem/opisem instalacji **AID/AOP** (rozkaz A\_WR),
- wpisuje stałe **PAD** - całe lub ich część (rozkaz PAD\_WR),
- dodaje nowych użytkowników/administratorów (rozkaz PASS\_ADD), może im również nadać nazwy (rozkaz PASS\_ADD\_N),

- ustala właściciela klucza HAK2 (rozkaz PASS\_WR\_OUID),
- wpisuje dane (licencyjne - ADP, własne lub dla użytkowników) do stref danych (rozказы DX\_WR\_S, DX\_WR/DX\_WR4).

Administrator może zmienić klucz A w trakcie późniejszego użytkowania klucza(y) HAK2 bez wpływu na pozostałe dane (rozkaz A\_WR). Dokonuje też pozostałych czynności administracyjnych:

- dodaje/nadpisuje nowych użytkowników i przydziela im nazwy,
- usuwa użytkowników (rozkaz PASS\_DEL),
- usuwa użytkownikom nazwy (rozkaz PASS\_DEL\_N),
- zmienia użytkownikom uprawnienia (rozkaz PASS\_WR\_UPR),
- usuwa dane ze stref danych (rozkaz DX\_DELN), może je też odczytywać (rozказы DX\_RD/DX\_RD4),
- wpisuje niewykorzystane stałe PAD (rozkaz PAD\_WR).

Do odczytu informacji o użytkowniku (identyfikatora, uprawnień i nazwy) służy rozkaz PASS\_RD\_UID.

Stałe w pamięci nieulotnej chronione są sumami kontrolnymi - ich przekłamanie sygnalizowane jest w polu F deskryptora klucza HAK2. W przypadku błędu w stałych administratora (flagi HAK2\_FLAG\_ACRC, HAK2\_FLAG\_RNGS, HAK2\_FLAG\_ZAR i HAK2\_FLAG\_OUID) może on klucz zresetować (rozказы RESET lub RESETC) i ponownie przygotować do pracy. Pozostałe błędy wymagają skasowania klucza HAK2 przez programistę i ponownego skonfigurowania.

### 3.5.6 Szyfrowane wiadomości adresowalne (SAM)

Do wygenerowania wiadomości SAM służy rozkaz SAM\_ENCR. Podany ciąg danych (40 bajtów) i numer fabryczny adresata (DEST\_SN) klucz HAK2 łączy ze swoim numerem fabrycznym (SRC\_SN) i zaszyfrowuje kluczem S1 (lub A) tworząc SAM (64 bajty). Odszyfrować może ją tylko klucz HAK2 o numerze fabrycznym DEST\_SN z identycznym kluczem A (lub S1). Do odszyfrowania SAM służy rozkaz SAM\_DECR, w odpowiedzi zwraca ciąg danych oraz numery fabryczne adresata (DEST\_SN) i nadawcy (SRC\_SN).

#### *Uwaga!*

*Wiadomość SAM dobrze nadaje się do przesyłania kluczy szyfrujących: szyfrowana jest z siłą 168 bitów (DES3) i jest zabezpieczona przed modyfikacją z siłą 64 bitów. Zasyfrowanie nawet identycznych danych daje inny wynik. Nie można jej odszyfrować nawet znając klucz A (lub S1): jest zabezpieczona przed dostępem administratora znającego klucz A (lub programisty znającego klucz S1) z siłą 128 bitów. Szyfrowanie kluczem A dostępne jest tylko dla właściciela klucza - dobrze chroni jego prywatne dane.*

## 3.6 KOMPILACJA BIBLIOTEKI

#### *Uwaga!*

*W poniższym paragrafie biblioteka nazywana będzie biblioteką HAK2.*

Bibliotekę HAK2 tworzą następujące pliki źródłowe:

- "DES.c", "DES.h",
- "Crypt\_utils.c", "Crypt\_utils.h",
- "dev\_find.c", "dev\_find.h",
- "usbio.c", "usbio.h",
- "HAK2.c", "HAK2.h", "HAK2\_dbg.h", "HAK2err.h".

Spośród plików nagłówkowych do aplikacji wystarczy włączyć plik "HAK2.h".

#### *Uwaga!*

*Szyfrowania transmisji dokonuje biblioteka HAK2 - dane przekazywane do funkcji (parametry) i zwracane przez te funkcje mają postać jawną. Dlatego biblioteka HAK2 powinna zostać włączona do modułu aplikacji (statycznie): dzięki temu tajne dane (hasła, klucze itp.) wymieniane z kluczem HAK2 szyfrowane są w module aplikacji i nie wydostają się z niego w postaci jawnej. Dołączenie biblioteki HAK2 w formie modułu DLL powoduje między*

*innymi to, że tajne dane mogą zostać podsłuchane/zmodyfikowane przez moduł "udający" moduł biblioteki HAK2.*

Biblioteka HAK2 korzysta też z funkcji systemu Windows (bibliotek DLL) i wymaga właściwych plików nagłówkowych oraz bibliotek importowych. Pliki nagłówkowe będące częścią pakietu Microsoft Platform SDK (np. "windows.h", "setupapi.h") są dostępne w środowisku programistycznym bez dodatkowych zabiegów, podobnie biblioteki importowe (np. "kernel32.lib", "setupapi.lib") są dołączane domyślnie. Biblioteka HAK2 korzysta z drivera HID poprzez bibliotekę systemową "hid.dll" - do jej wykorzystania przy kompilacji potrzebne są pliki "hidsdi.h", "hidpi.h", "hidusage.h" i "hid.lib" z pakietu Windows DDK (Driver Development Kit).

#### **Uwaga!**

*Format bibliotek importowych (w tym "hid.lib") jest specyficzny dla środowiska programistycznego (kompilatora/linkera). W przypadku używania środowiska innego producenta niż Microsoft potrzebny jest specyficzny plik "hid.lib". Można go wygenerować z pliku "hid.dll" używając programu narzędziowego dostarczonego wraz ze środowiskiem. W przypadku pakietu Borland C++Builder jest to program "implib.exe". Ważne jest, aby kod w pliku "hid.lib" wyszukiwał funkcje w bibliotece "hid.dll" według nazw, a nie według indeksów - indeksy funkcji różnią się pomiędzy wersjami systemu Windows. W przypadku programu "implib.exe" z pakietu Borland C++Builder należy użyć przełącznika "-f".*

## 3.7 WIELOZADANIOWOŚĆ

Pojedynczy egzemplarz klucza HAK2 przeznaczony jest do pracy z jedną instancją aplikacji - w jednej chwili może istnieć tylko jedna sesja użytkownika w kluczu HAK2. Aplikacja wyszukuje "swój" klucz HAK2 poprzez identyfikator SID i nie próbuje komunikować się z pozostałymi kluczami. Dzięki temu nie ma konfliktów pomiędzy instancjami różnych aplikacji. Do takiego modelu bibliotekę można wykorzystać bez dodatkowych zabiegów. Praca wielu aplikacji z jednym kluczem HAK2 jest również możliwa, ale wymaga bardziej rozbudowanego oprogramowania - np. nawiązanie nowej sesji dla każdej operacji w kluczu HAK2 lub rodzaj programowego "serwera" kluczy HAK2.

Używanie kluczy HAK2 w systemie wielozadaniowym wymaga zapewnienia synchronizacji w dostępie do urządzeń przez poszczególne zadania (procesy). Wprawdzie system Windows zgodnie ze specyfikacją USB zapewnia integralność pojedynczego transferu USB (tj. przesłania ciągu bajtów żądanej długości w jednym kierunku), ale każdy rozkaz klucza HAK2 wymaga sekwencji 2 transferów: wysłanie rozkazu/parametrów i odebranie odpowiedzi. Istnieje duże prawdopodobieństwo wzajemnego zakłócania sekwencji rozkazu ID\_RD przez aplikacje w czasie jednoczesnego odświeżania listy urządzeń (np. po włożeniu lub wyjęciu klucza HAK2). Biblioteka rozwiązuje ten problem przy pomocy tzw. muteksów. Tworzy jeden taki obiekt dla każdego wykrytego klucza HAK2. Przed wykonaniem sekwencji rozkazu w kluczu HAK2 biblioteka (a dokładniej wątek, w którym się ona wykonuje) przejmuje muteks związany z kluczem, a po zakończeniu zwalnia. Jeżeli muteks jest zajęty przez bibliotekę z jednego wątku, biblioteki z pozostałych wątków (np. z innych aplikacji) czekają na swoją kolej próbując przejąć muteks.

Muteksy są obiektami globalnymi w systemie Windows identyfikowanymi poprzez nazwę - ciąg tekstowy. Biblioteka tworzy nazwy muteksów poprzez połączenie zapisanego tekstowo (heksadecymalnie) numeru fabrycznego klucza HAK2 z unikalnym identyfikatorem (szczegóły w kodzie źródłowym biblioteki). Dzięki temu nie ma konfliktów z wszelkimi innymi muteksami wykorzystywanymi w systemie.

Opisany mechanizm działa przy założeniu, że wszystkie aplikacje korzystające z kluczy HAK2 używają tego mechanizmu w ten sam sposób, a pozostałe aplikacje nie próbują komunikacji z kluczami odrzucając je na podstawie identyfikatorów VID/PID. "Odporność" na aplikacje nie stosujące się do powyższych zasad można uzyskać wykorzystując otwieranie urządzeń na wyłączność - parametr 'dwShareMode' funkcji CreateFile() (plik "dev\_find.c") równy 0. Rozwiązanie to nie jest jednak polecane w obecnej formie biblioteki - powoduje m.in. zajęcie przez aplikację wszystkich wolnych urządzeń HID. Ponadto nie działa ono w pierwszej wersji systemu Windows 98 (tzw. Gold).

## 3.8 WIELOWĄTKOWOŚĆ

Używając bibliotekę w aplikacjach wielowątkowych należy pamiętać, że wewnętrzna lista urządzeń jest zmienną globalną, a przechowywane w niej są także dane sesji (w tym klucz sesji). Wątki korzystające z osobnych kluczy HAK2 nie przeszkadzają sobie, natomiast konieczna jest synchronizacja wątków korzystających z tego samego klucza HAK2. Synchronizacja taka nie jest wbudowana w bibliotekę, można ją zapewnić używając sekcji krytycznych (obiektów typu CRITICAL\_SECTION) dla każdego urządzenia. Dostęp do urządzeń musi być również zsynchronizowany z odświeżaniem listy urządzeń - wewnętrznie w bibliotece urządzenia identyfikowane są na podstawie pozycji na liście. Można to

osiągnąć poprzez wejście w sekcje krytyczne wszystkich urządzeń z listy przed wywołaniem funkcji HAK2\_OnDeviceChange() odświeżającej listę.

***Uwaga!***

*Nie wolno wywołać funkcji HAK2\_OnDeviceChange() w sytuacji, gdy któryś z wątków procesu czeka na przejęcie muteksu. Funkcja ta zamyka uchwyty muteksów, co w tym momencie jest niedopuszczalne - zobacz w dokumentacji funkcji systemowej WaitForSingleObject(). Zastosowanie sekcji krytycznych opisane powyżej rozwiązuje ten problem.*