STUDIA INFORMATICA 2001
Volume 22 Number 1 (43)

Jarosław FRANCIK, Krzysztof OGRODNIK
Politechnika Śląska, Instytut Informatyki

APLIKACJA KRYPTOGRAFICZNA ZINTEGROWANA Z SYSTEMEM WINDOWS

Streszczenie. Przedstawiona w artykule aplikacja kryptograficzna powstała w ramach prac badawczych dotyczących pełnej integracji oprogramowania ze środowiskiem systemu Windows. Celem integracji jest ułatwienie obsługi oprogramowania, a środkiem do realizacji tego celu – implementacja tzw. zerowego interfejsu użytkownika, który nie wykracza poza standardowe elementy dostarczane przez system. Program *SecureIt!*, posiadający taki właśnie interfejs, pozwala szyfrować i deszyfrować pliki i foldery, a na poziomie implementacji współpracuje z biblioteką *CryptoAPI*, dostarczając między innymi moduł dostawcy usług kryptograficznych (*CSP*) z implementacją algorytmu *Rijndael*.

CRYPTOGRAPHIC APPLICATION INTEGRATED WITH WINDOWS

Summary. A cryptographic application presented in the paper has been created as a part of the research on the software fully integrated with Windows environment. The goal of integration is to design optimally easy-to-use user interfaces, and the means to obtain it – implementation of so called zero user interface, which contains nothing but standard elements supported by the operating system. The *SecureIt!* application, made in conformity with this paradigm, supports encrypting and decrypting files and folders. It is compatible with *CryptoAPI* and provides a *CSP* module that implements the *Rijndael* algorithm.

1. Wstęp

Podstawowym celem przy projektowaniu graficznego interfejsu użytkownika jest łatwość jego obsługi – zgodnej z intuicją i nabytą przez użytkownika wiedzą. Dlatego wielkie znaczenie przywiązuje się do zgodności interfejsów z szeroko przyjętymi standardami. Standardy te w sposób najbardziej widoczny narzucają dwa programy, które można śmiało określić jako

najczęściej używane i najszerzej znane – chodzi o system operacyjny (w niniejszej publikacji ograniczamy się do systemów z rodziny *Windows*) i przeglądarkę WWW. Aplikacje, które są – w możliwie jak najpełniejszy sposób – z nimi zintegrowane zapewniają łatwość obsługi, oczywiście przy założeniu pewnej, choćby najskromniejszej umiejętności posługiwania się typowym oprogramowaniem [3].

Celem podjętych prac badawczych (finansowanych z funduszu Badań Własnych BW/2001) jest próba odpowiedzi na pytanie, jak daleko można się posunąć z integracją i czy możliwe jest tworzenie „interfejsów zerowych", to jest takich, które w zasadniczej swej części nie różniłyby się – z punktu widzenia użytkownika – od standardowych elementów systemu operacyjnego. W wyniku tych prac udało się usystematyzować zasady tworzenia aplikacji zintegrowanych z systemem operacyjnym *Windows*.

Wymiernym efektem prac stała się też aplikacja kryptograficzna, napisana zgodnie z koncepcją zerowego interfejsu użytkownika. Właśnie ten program, noszący nazwę *SecureIt!*, stanowi główny temat niniejszego artykułu.

2. Integracja i zerowy interfejs użytkownika

Założeniem do dalszej dyskusji jest posiadanie przez użytkownika podstawowych umiejętności obsługi systemu operacyjnego Windows w zakresie kopiowania i przenoszenia plików, uruchamiania aplikacji czy korzystania z menu kontekstowego.

Aplikacja zintegrowana z systemem (na poziomie interfejsu użytkownika) to taka, która w znacznym stopniu korzysta z elementów interfejsu użytkownika dostarczonych bezpośrednio przez system operacyjny. Elementy te realizują określone usługi na rzecz aplikacji, eliminując potrzebę realizacji części (lub całości) interfejsu przez samą aplikację. Przykładem może tu być wykorzystanie systemowych menu kontekstowych czy użycie standardowych okien dialogowych (np. tzw. *common dialogs*). Są to elementy dobrze znane użytkownikowi, przeto umiejętność posługiwania się częścią interfejsu zintegrowaną z systemem sprowadza się do podstawowej umiejętności obsługi systemu.

Aplikacje mogą się charakteryzować różnym stopniem integracji. W najskromniejszym przypadku może się ona ograniczać do wykorzystania wspólnych dialogów plikowych (*common file dialogs*) i do skojarzenia rozszerzenia nazwy pliku z obsługującą ją aplikacją. Na przeciwnym biegunie znajdują się aplikacje, których cały interfejs użytkownika został zintegrowany z systemem operacyjnym. Nazywamy je aplikacjami z **zerowym interfejsem użytkownika**.

Jeżeli wszystkie elementy interfejsu użytkownika, które są charakterystyczne dla danej aplikacji, zostały przez nią wytworzone i stanowią niezależne elementy wizualne (okna, menu itp.), zdefiniujemy jako elementy **własne** (ang. *native*), w przeciwieństwie do elementów **zin-**

tegrowanych z systemem operacyjnym, stanowiących (pojęciowo) część elementów wizualnych generowanych przez system operacyjny, wówczas przez pojęcie zerowego interfejsu użytkownika (ang. *zero user interface*) rozumieć będziemy taki, w skład którego wchodzą wyłącznie elementy zintegrowane, brak natomiast zupełnie elementów własnych. Należy przy tym zaznaczyć, że rozróżnienie na elementy własne i zintegrowane nie jest ostre: opcja dodana do systemowego menu kontekstowego jest wytworem aplikacji, a zatem w pewnym sensie jej elementem własnym; z drugiej strony przez użytkownika jest postrzegana jako część obiektu wygenerowanego przez system. Wydaje się, że poszczególne elementy są pojęciowo odbierane przez użytkownika jako zintegrowane wówczas, gdy są one zgodne z ogólnymi założeniami mieszczących je większych elementów systemowych. W tym sensie będziemy skłonni uznać za zintegrowane nawet wysoce autonomiczne twory, jakimi są okna elementów Panelu sterowania. Podejście takie jest pragmatyczne – wtyczka Panelu sterowania jest praktycznie jedynym środkiem pozwalającym na szczegółowe dostrojenie aplikacji dysponującej poza tym zerowym interfejsem użytkownika.

Dotąd opisana została integracja na poziomie interfejsu użytkownika. Istnieje też inny jeszcze aspekt integracji – na poziomie wewnętrznych mechanizmów systemu operacyjnego. Integracja tego typu ma za cel zapewnienie zgodności z modułami systemu operacyjnego lub dostarczonymi przez niezależnych producentów; jest podstawą, na której można budować rozwiązania otwarte, gotowe do dalszej rozbudowy.

3. Techniczna realizacja integracji

Obecnie zostaną przedstawione dwie podstawowe techniki użyteczne przy tworzeniu aplikacji z zerowym interfejsem użytkownika. Pierwsza z nich, polegająca na wykorzystaniu skojarzeń plikowych, jest popularnie stosowana, bardzo prosta w implementacji, jednak charakteryzuje się poważnymi ograniczeniami. Stosowanie rozszerzeń jest natomiast uważane za znacznie trudniejsze; technika ta daje jednak praktycznie nieograniczone możliwości integracji z systemem.

Skojarzenia plikowe można założyć ręcznie korzystając z zakładki *Typy plików* w oknie *Opcje folderów* programu *Eksplorator Windows*, a programowo stosując proste wpisy w rejestrze systemowym [3]. Korzystając z tych elementarnych usług systemowych można łatwo uzyskać następujące cechy funkcjonalne, związane z poszczególnymi typami (rozszerzeniami) plików:

- nadanie ikony (skojarzenie typu pliku z wyświetlaną ikoną);
- zdefiniowanie domyślnej operacji wykonywanej po kliknięciu ikony pliku;
- rozszerzenie systemowego menu kontekstowych o dodatkowe, charakterystyczne opcje.

Wadą tego rozwiązania jest bezwarunkowe powiązanie z typem pliku – rozróżnianym według rozszerzenia nazwy. Nie można definiować odmiennego zachowania dla plików o tym samym rozszerzeniu. Trudno też definiować funkcjonalność dla plików o różnych rozszerzeniach. Niemożliwe jest operowanie na zbiorach plików definiowanych jako „wszystkie z wyjątkiem".

Rozszerzenia powłoki i przestrzeni nazewniczej (ang. *shell & namespace extensions*) [2, 3, 5-7, 11] pozwalają nie tylko ominąć te problemy, ale dodatkowo wprowadzić nowe funkcje, niemożliwe do uzyskania przy wykorzystaniu standardowego mechanizmu skojarzeń plikowych. I tak możliwe jest między innymi:

- dynamiczne określenie ikony, zawartości menu kontekstowego oraz menu kopiowania i przenoszenia pliku – na podstawie dowolnych cech pliku, a nie tylko rozszerzenia jego nazwy,
- implementacja dodatkowego arkusza w oknie *Właściwości* pliku,
- specjalne zarządzanie operacją „przeciągnij i opuść",
- wprowadzenie dodatkowych obiektów o charakterze folderu, lecz odmiennym zachowaniu, widocznych w okienkach *Eksploratora Windows*.

Szczegóły implementacyjne rozszerzeń powłoki i przestrzeni nazewniczej wykraczają poza ramy tematyczne niniejszego opracowania. Ograniczymy się w związku z tym do przedstawienia jedynie najważniejszych informacji, odsyłając jednocześnie do publikacji [3, 6, 7, 11].

Rozszerzenie powłoki systemowej to moduł zwiększający możliwości tej powłoki poprzez dodanie nietypowych środków prezentacji i manipulacji elementami powłoki, szczególnie obiektami plikowymi (dokumentami). W tab. 1 zebrano dostępne w systemie *Windows* typy rozszerzeń powłoki.

Tabela 1

Typy rozszerzeń powłoki

Nazwa	Opis	Interfejsy
Icon	dynamiczne ikony	*IExtractIcon, IPersistFile*
Context menu	menu kontekstowe	*IContextMenu, IShellExtInit*
Property sheet	arkusze (zakładki) Właściwości	*IShellPropSheetExt, IShellExtInit*
Copy hook	kopiowanie obiektów	*ICopyHook*
Drop target	*drag-and*-drop: upuszczanie	*IDropTarget, IPersistFile*
Data object	*drag-and*-drop: kopiowanie	*IDataObject, IPersistFile*
Quick view	szybki podgląd	*IFileViewer, IPersistFile*
Briefcase reconcile	łączenie wersji dokumentu	*IReconcilableObject, IPersistFile*

Technicznie rzecz biorąc, moduł rozszerzenia powłoki to składnik *COM* (dawniej: serwer *OLE*) implementujący obiekty używane przez powłokę. Ma on postać pliku *DLL*. Dla każdego z ośmiu typów rozszerzeń określone są interfejsy (w terminologii *COM*), które składnik

będący rozszerzeniem musi implementować (tab. 1). Dla przykładu, interfejs *IExtractIcon*, niezbędny przy tworzeniu rozszerzenia typu *Icon*, zawiera dwie funkcje:

- *GetIconLocation* – zwraca nazwę pliku zawierającego ikonę oraz indeks ikony w pliku.
- *Extract* – zwraca uchwyt do ikony.

W zależności od tego, czy ikona jest dostępna w pliku, czy generowana na bieżąco i udostępniana dynamicznie, wystarczy zaimplementować tylko jedną z powyższych funkcji. Druga funkcja powinna wówczas zwracać standardową wartość *E_NOTIMPL*.

Jak wspomniano, rozszerzenia powłoki są składnikami *COM*. Nakłada to dodatkowe warunki na ich implementację. Chodzi między innymi o przydzielenie identyfikatora *GUID* i zaimplementowanie metod interfejsu *IUnknown*. Szczegółową dyskusję tych zagadnień można znaleźć w [3, 9], natomiast krótki przegląd, za to przeznaczony dla programistów zamierzających pisać moduły rozszerzeń powłoki – w [11].

Moduły rozszerzeń powłoki należy rejestrować w systemie za pomocą wpisu do rejestru o następującym przykładowym formacie:

```
[HKEY_CLASSES_ROOT\.XXX]="XXXFile"
[HKEY_CLASSES_ROOT\XXXFile]="Plik rozszerzenia XXX"
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{identyfikator CLSID}]="XXX Ext"
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{identyfikator CLSID}\
InProcServer]= "xxx.dll"
[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{identyfikator CLSID}\
ThreadingModel]= "Apartment"
[HKEY_CLASSES_ROOT\XXXFile\shellex\IconHandler]="{identyfikator CLSID}"
[HKEY_CLASSES_ROOT\XXXFile\DefaultIcon]=%1
```

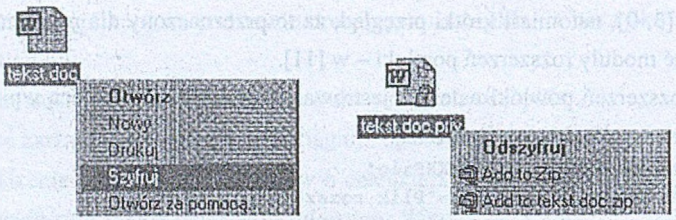
Moduły rozszerzeń przestrzeni nazewniczej pozwalają na dołączenie do przestrzeni nazewniczej Windows (czyli tego, co ogląda się w oknie Mój komputer) nowych obiektów o charakterze wolumenów dyskowych, folderów lub plików. Ich implementacja przypomina implementację modułów rozszerzeń powłoki [3, 6, 7, 11] i wymaga zaimplementowania w składniku *COM* następujących interfejsów: *IPersistFolder*, *IShellFolder*, *IEnumIDList* oraz *IShellView*.

4. Aplikacja kryptograficzna SecureIt! – opis funkcjonalny

Jako przykład integracji technologii kryptograficznych z systemem Windows przedstawiony zostanie program *SecureIt!*. Program ten umożliwia łatwą w obsłudze i efektywną realizację szyfrowania i deszyfrowania plików. Program odznacza się wysokim stopniem integracji z systemem Windows, praktycznie zerowym interfejsem użytkownika oraz otwartą i rozszerzalną architekturą zgodną ze standardem *CryptoAPI*.

Po zainstalowaniu programu *SecureIt!* polecenia szyfrowania i deszyfrowania plików są dostępne bezpośrednio z powłoki systemu Windows. Podstawowym sposobem uzyskania

dostępu do funkcji programu jest wykorzystanie systemowego menu kontekstowego, dostępnego dla ikon plików. W menu tym pojawiają się, w zależności od wybranego pliku, polecenia *Szyfruj* lub *Odszyfruj* (rys. 1). W celu dostosowania domyślnych parametrów szyfrowania dla wybranego pliku należy wyświetlić menu kontekstowe wraz z naciśniętym klawiszem *Shift* i skorzystać z polecenia *Szyfruj korzystając z...* Bardzo często podczas zarządzania plikami zachodzi potrzeba szyfrowania lub deszyfrowania plików wraz z ich przenoszeniem (kopiowaniem). Program *SecureIt* ułatwia tego typu operacje poprzez dodanie do menu kontekstowego pojawiającego się podczas kopiowania lub przenoszenia plików odpowiednich poleceń.

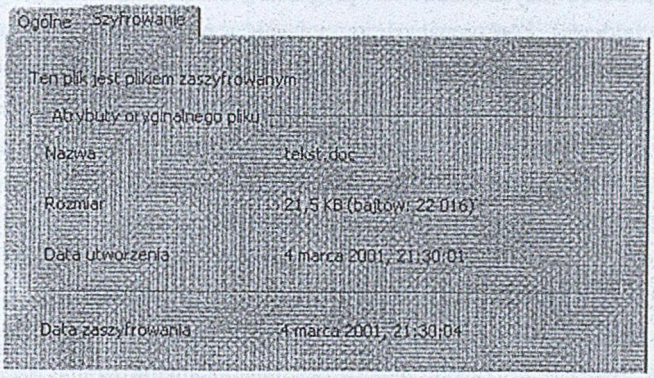


Rys. 1. Ikony i menu kontekstowe pliku dokumentu i jego zaszyfrowanej wersji w systemie z zainstalowaną aplikacją *SecureIt!*
Fig. 1. Icons and context menu for a document file and its encrypted version in a system with *SecureIt!*

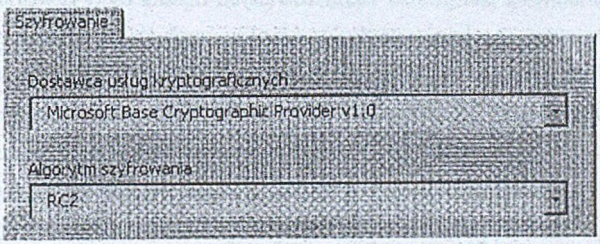
W celu zapewnienia efektywnej pracy w systemie Windows program *SecureIt!* ułatwia rozróżnianie pomiędzy plikami zaszyfrowanymi i niezaszyfrowanymi. Pliki zaszyfrowane, mimo że posiadają własne rozszerzenie (*.priv*), są reprezentowane za pomocą ikony podobnej to tej, która była używana z plikiem niezaszyfrowanym. Ikona taka jest jedynie wzbogacona o dodatkowy motyw graficzny – symbol kłódki (w podobny sposób system generuje ikony dla skrótów – *shortcuts*). Każdy plik zaszyfrowany posiada także dodatkowy arkusz właściwości dostarczający informacji o atrybutach pliku przed zaszyfrowaniem i o sposobie zaszyfrowania pliku (rys. 2).

Parametry szyfrowania plików dostosowuje się za pomocą dodatkowej aplikacji (wtyczki) *Panelu Sterowania* (rys. 3). Do głównych opcji podlegających konfiguracji należą:

- nazwa modułu dostawcy usług kryptograficznych,
- algorytm używany podczas szyfrowania,
- tryb użycia hasła.



Rys. 2. Arkusz właściwości pliku zaszyfrowanego programem *SecureIt!*
Fig. 2. Property sheet of a file encrypted with *SecureIt!*



Rys. 3. Wtyczka Panelu sterowania przeznaczonego do konfigurowania programu *SecureIt!*
Fig. 3. Control Panel applet for *SecureIt!* configuration.

Program *SecureIt* współpracuje z dostawcami usług kryptograficznych (ang. *Cryptographic Service Provider, CSP*) – standardowymi modułami systemowymi, które implementują algorytmy i inne usługi. Standardowo dostępny jest moduł *Microsoft Base Cryptographic Provider*, oferujący – do szyfrowania plików – niezbyt mocny algorytm *RC2*. *SecureIt!* jest rozpowszechniany wraz z bardziej zaawansowanym modułem, implementującym algorytm *Rijndael* [8, 10], będący od niedawna standardowym algorytmem szyfrującym w USA. Ponieważ *CSP* jest standardem systemowym, toteż *SecureIt!* może korzystać zarówno z dowolnych modułów dostarczonych przez niezależnych producentów, jak również rozpowszechniany z programem moduł z algorytmem *Rijndael* może być stosowany przez inne funkcje i aplikacje kryptograficzne pracujące w systemie.

Pliki szyfrowane są za pomocą klucza generowanego z hasła pochodzącego od użytkownika. Użytkownik ma do wyboru kilka trybów użycia hasła:

- program pyta o hasło przy każdym szyfrowaniu pliku,
- program pyta o hasło raz – na początku pracy – i zapamiętuje je do końca sesji,
- program pyta o hasło raz i zapamiętuje je przez określony czas.

Jedną z zaawansowanych możliwości programu *SecureIt!* jest możliwość tworzenia tzw. bezpiecznych folderów. Funkcjonalność ta, polegająca na automatycznym szyfrowaniu plików kopiowanych lub przenoszonych do takiego folderu, może znacznie ułatwić zarządzanie dużą liczbą plików. Bezpieczne foldery w programie *SecureIt!* mogą obejmować pojedynczy folder lub drzewo folderów. W przypadku, gdy nie wszystkie pliki w bezpiecznym folderze mają być szyfrowane, program umożliwia specyfikację rozszerzeń plików do szyfrowania.

5. Uwagi implementacyjne

Opisane w poprzednim punkcie możliwości funkcjonalne programu *SecureIt!* wymusiły metodę integracji tego programu z systemem *Windows* za pomocą rozszerzeń powłoki i przestrzeni nazewniczej. Oto przyczyny podjęcia takiej decyzji projektowej:

- Menu kontekstowe dla plików zaszyfrowanych można było łatwo zrealizować korzystając z prostego mechanizmu skojarzeń plikowych (pliki te mają stałe rozszerzenie *.priv*); jednak pliki niezaszyfrowane można zdefiniować tylko jako kategorię *„wszystkie z wyjątkiem .priv"* – niemożliwą do zamodelowania za pomocą skojarzeń. Wobec tego konieczna była implementacja rozszerzenia powłoki typu *Context Menu* (tab. 1).
- Ikony plików zaszyfrowanych są tworzone dynamicznie, na podstawie oryginalnej ikony pliku sprzed zaszyfrowania i stałego elementu – wizerunku kłódki. Wymagało to zaimplementowania rozszerzenia powłoki typu *Icon* (tab. 1).
- Wprowadzenie dostosowanego arkusza właściwości wymaga zastosowania rozszerzenia powłoki typu *Property Sheet* (tab. 1).
- Implementacja funkcji *Bezpieczny folder* wymaga wprowadzenia rozszerzenia przestrzeni nazewniczej.

Główna część programu *SecureIt!* została zaimplementowana jako biblioteka *DLL*, zawierająca moduły rozszerzeń powłoki i przestrzeni nazewniczej systemu *Windows*. Moduł rozszerzeń powłoki stanowi obiekt *COM* implementujący wymagane interfejsy i zarejestrowany w odpowiedni sposób w rejestrze systemowym.

Jako narzędzie służące do implementacji programu zostało wybrane środowisko *Microsoft Visual C++* i biblioteka *ATL (Active Template Library)*. Do wyboru tych narzędzi przyczyniło się kilka istotnych cech, między innymi ich popularność, efektywność – *C++* w połączeniu z *ATL* umożliwia tworzenie obiektów COM niemal optymalnych pod względem wielkości i szybkości działania.

W skład programu *SecureIt!* wchodzi wiele różnych rozszerzeń powłoki, między innymi dynamiczne ikony, arkusze właściwości, menu kontekstowe, a także rozszerzenie przestrzeni

nazewniczej. Specjalnie dla potrzeb tego projektu stworzone zostało narzędzie o nazwie *Active Shell Framework* (*ASF*). Jest to zbiór wtyczek (generatorów) i klas ułatwiających tworzenie modułów rozszerzeń powłoki w środowisku *Visual C++* w oparciu o bibliotekę *ATL*. Narzędzie to ułatwia tworzenie dowolnych rozszerzeń powłoki i przestrzeni nazewniczej, przejmując na siebie wszelkie szczegóły związane z wymogami systemowymi tego typu projektów. *ASF* integruje się z interfejsem użytkownika systemu *Visual Studio/C++*.

Do implementacji warstwy kryptograficznej została wykorzystana biblioteka *CryptoAPI*, dostępna w systemach *Windows 95*, *Windows NT 3.51* i nowszych (w systemach starszych niż *Windows 98/NT 4.0* warunkiem dostępności *CryptoAPI* było zainstalowanie programu *Internet Explorer* w wersji co najmniej 4.0). Biblioteka *CryptoAPI* stanowi pomost (interfejs) pomiędzy rzeczywistymi modułami zawierającymi implementację usług kryptograficznych (tzw, *CSP – Cryptographic Service Providers*) a aplikacjami użytkowymi. Faktyczna funkcjonalność, np. implementacja algorytmów kryptograficznych, generacja kluczy itp. jest zlokalizowana w modułach *CSP*. Dzięki takiej otwartej i modułowej architekturze możliwa jest rozszerzalność aplikacji – wykorzystanie nowych dostawców usług kryptograficznych implementujących silniejsze algorytmy.

W celu zwiększenia elastyczności i rozszerzalności, w skład programu *SecureIt!* wchodzi obiekt automatyzacji *COM*, umożliwiający między innymi szyfrowanie i deszyfrowanie plików. Obiekt ten może zostać wykorzystany w środowiskach skryptowych (np. w makrach), a także w dowolnych innych programach, umożliwiając zautomatyzowanie operacji szyfrowania i deszyfrowania danych.

6. Podsumowanie

Aplikacja *SecureIt!* powstała w zasadzie jako efekt uboczny prac badawczych nad interfejsem użytkownika, rozwinęła się jednak jako w pełni funkcjonalny kryptograficzny program użytkowy. Pozwala na szyfrowanie i deszyfrowanie plików. W odróżnieniu od standardowej implementacji szyfrowania istniejącej w systemie *Windows 2000* pliki zaszyfrowane przez *SecureIt!* są niezależne od stosowanego systemu plikowego i mogą być swobodnie przenoszone pomiędzy różnymi komputerami.

Porównując *SecureIt!* z innymi programami kryptograficznymi [1, 4] trzeba wskazać na istotną przewagę prezentowanego rozwiązania w dziedzinie interfejsu użytkownika. Stopień integracji programu z systemem *Windows* wyróżnia go zdecydowanie spośród konkurencyjnych rozwiązań. Jeśli chodzi o liczbę i wydajność zaimplementowanych algorytmów, to w tej konkurencji *SecureIt!* nawet nie stara się podejmować wyzwania: rozpowszechniany jest z

implementacją jednego tylko algorytmu *Rijndael*, którego kod źródłowy został pobrany z Internetu [8]. Jednak potencjalnie i tu może *SecureIt!* zdystansować konkurencję, gdyż jego otwarta architektura pozwala na wykorzystanie dowolnego algorytmu zaimplementowanego w postaci modułu *CryptoAPI/CSP* – a jest to postać standardowa, systemowa i dokładnie wyspecyfikowana [np. w 6, 7], potencjalnych więc implementacji może być mnóstwo.

W realizacji programu wykorzystano składniki COM stanowiące rozszerzenia powłoki (*shell extensions*). Aplikacje oparte na tej technologii są w powszechnym odczuciu uważane za trudne do napisania. Jednak usystematyzowanie zasad tworzenia tego typu oprogramowania zdecydowanie ułatwiło zadanie. Czynnikiem krytycznym, który nie tylko uprościł realizację programu *SecureIt!*, ale również i w przyszłości zdecydowanie uprzystępni realizację rozszerzeń powłoki i przestrzeni nazewniczej, jest zestaw wtyczek ułatwiających ich przygotowanie. Zestaw ten został pod nazwą *ASF* (*Active Shell Framework*) stworzony specjalnie dla potrzeb przedstawionych w artykule prac przez jednego z autorów (K. Ogrodnik).

Program *SecureIt!* jest dobrym przykładem realizacji przydatnego oprogramowania spełniającego wymogi definicji zerowego interfejsu użytkownika.

LITERATURA

1. Białas A.: Programy kryptograficzne. W: Grzywak A. (red.), Bezpieczeństwo systemów komputerowych. Wydawnictwo Pracowni Jacka Skalmierskiego, Gliwice 2000, str. 115-130.
2. Chappell D.: Understanding ActiveX and OLE, Microsoft Press, Redmond, WA, USA, 1996.
3. Cluts N. W.: Programming the Windows 95 User Interface. Microsoft Press, Redmond, WA, USA 1995.
4. Duchnicz A.: Bezpieczne bity. Test programów do szyfrowania danych. CHIP 10/99, str. 166-181.
5. Francik J.: Integracja rozwiązań kryptograficznych z systemami operacyjnymi Windows. W: Grzywak A. (red.): Bezpieczeństwo systemów komputerowych. Wydawnictwo Pracowni Jacka Skalmierskiego, Gliwice 2000, str. 175-197.
6. Microsoft Developers Network. CD-ROM. Microsoft Corp. 1999.
7. Microsoft Developers Network On Line. www.microsoft.com/msdn.
8. Rijmen V.: The block cipher Rijndael, http://www.esat.kuleuven.ac.be/~rijmen/rijndael
9. Rogerson D.: Inside COM, Microsoft's Component Object Model. Microsoft Press, Redmond, WA, USA, 1997.
10. Schneier B.: Kryptografia dla praktyków. WNT, Warszawa 1995.

11. Williams A.: Czarna księga MFC. Wydawnictwo Helion, Gliwice 1999.

Recenzent: Dr inż. Krystian Żymełka

Wpłynęło do Redakcji 26 marca 2001 r.

Abstract

The critical factor of success in the user interface design is being easy-to-use. This may be obtained by deep integration of an application with the operating system. Once the user knows the system, he or she practically knows also the interface of the application integrated with it. The goal of the R&D activities presented in this paper is to find out how deep such an integration may be. The answer may be the concept of the *zero user interface*, that contains no controls but those integrated with the operating system.

The possibilities of integrating applications with Windows system are discussed in deep. They involve file associations and shell/namespace extensions, the latter being more difficult to implement but much more powerful.

The main part of the article presents *SecureIt!* – an application that was started as a sample "zero-interface" applet and arose to be a fully functional, powerful cryptographic application. It encrypts and decrypts file, widely using shell & namespace extension COM based techniques to implement smart context menus, dynamical icons (that look just like icons of the original file with addition of a small image of a padlock, fig. 1), property sheets (fig. 2) and smart folders with automatic encryption/decryption facility. The program makes use of *CryptoAPI CSP* modules (*Cryptographic Service Provider*) to make its lower-end cryptographic interface compatible with widely-known system standard. The whole program can be configured with a Control Panel plug-in (fig. 3). *SecureIt!* has been implemented in *Visual C++* using *ATL* and *ASF* (*Active Shell Framework*) – a general tool created specially for *SecureIt!* to facilitate creating shell & namespace extensions.

Compared with other file encrypting applications, *SecureIt!* distinguishes with its highly-integrated, easy-to-use interface as well as open architecture, that may be expanded using any *CSP* module. It is also a very good example of a fully functional zero-interface application.