Andrej DOBROVOLJC

# Microsoft CryptoAPI

Seminarska naloga pri predmetu Kriptografija in računalniška varnost

Prof. Aleksandar Jurišić

Novo mesto, junij 2000

# Kazalo

# 1. Uvod

Povečan obseg prometa preko nevarnega internetnega medija zahteva učinkovite mehanizme za zaščito sporočil, dokumentov, plačil in drugih transakcij. Ko govorimo o varnosti podatkov na internetu, moramo rešitve iskati na področju kriptografije. Med osnovne varnostne mehanizme sodijo predvsem šifriranje, dešifriranje in preverjanje pristnosti.

Proizvajalcem in ponudnikom operacijskih sistemov in programskih orodij pomeni internet po drugi strani odlično priložnost za razvoj povsem novih programskih rešitev, takšnih, ki doslej niso mogle obstajati. Mednje sodijo:

- Bančne aplikacije (Home Banking)
- Nakupovanje po internetu
- Varni e-mail
- Programi za skupinsko delo itd.

Področje kriptografije je zelo široko, zahtevno in večini programerjev neznano. Zahtevna je tudi implementacija teh algoritmov. Edino zagotovilo za pospeševanje uporabe varnostnih mehanizmov v programski opremi je pomoč programerjem v obliki knjižnic, programskih vmesnikov ali komponent, ki ponujajo ustrezno implementacijo kriptografskih standardov.

# 2. Osnovne značilnosti CryptoAPI

Microsoft CryptoAPI je programski vmesnik, ki omogoča programerjem hitro, enostavno in zanesljivo uporabo kriptografskih funkcij v svojih 32-bitnih Windows aplikacijah. Potrebno je zgolj osnovno poznavanje varnostnih funkcij in njihov namen, prihranjeno pa nam je poznavanje algoritmov in njihova implementacija. Gre torej za podobno rešitev, kot je uporabljena pri tiskalnikih in grafičnih karticah, kjer programerja ne zanima natančna konfiguracija strojne opreme ampak le njena uporaba.

Microsoft je prvo uradno različico CryptoAPI 1.0 poslal na trg v letu 1996. Podprte so bile funkcije za:

- generiranje, izmenjavo in nadzor ključev,
- šifriranje in dešifriranje,
- hash funkcije, digitalne podpise in njihovo verifikacijo.

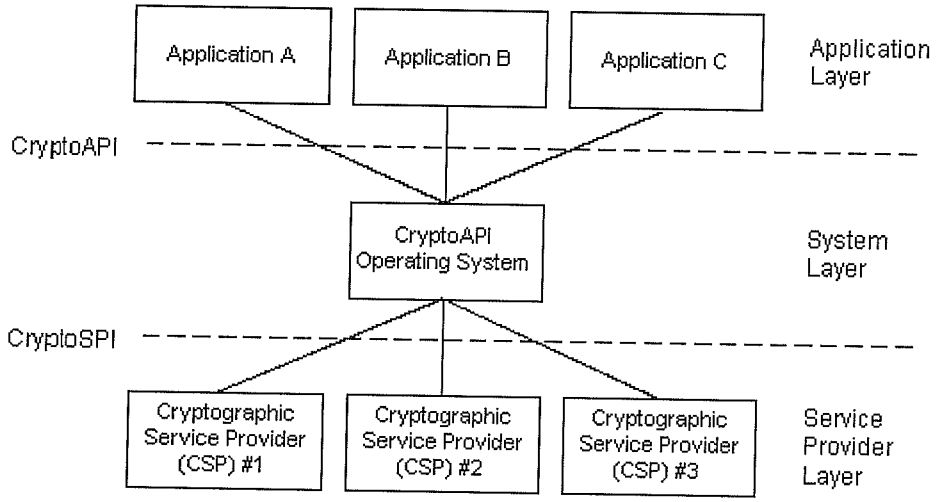
V letu 1998 so programski vmesnik nadgradili z verzijo 2.0, ki je aktivna še danes. Zaradi potrebe po varnejši komunikaciji in naraščajočega trenda uporabe certifikatov so vanj vgradili podporo za nadzor in uporabo digitalnih certifikatov.

Podpora za CryptoAPI je vgrajena v Windows 2000, Windows NT 4.0, Windows 98 in Windows 95 OSR2. Prinaša jo tudi Internet Explorer od vključno verzije 3.0 naprej. Za razvoj aplikacij z uporabo CryptoAPI lahko uporabimo C/C++ prevajalnike, Delphi, Visual Basic, VBScript, Javo ali ActiveX komponente.

# 3. Arhitektura CryptoAPI

CryptoAPI ima modularno zgradbo. Vse kriptografske operacije se izvajajo v posebnih sistemskih modulih za kriptografsko strežbo - *cryptographic service providers* (CSP). To so izmenljivi moduli, ki jih po potrebi dodamo ali odvzamemo sistemu. Vsak CSP vsebuje kriptografske algoritme, ki so popolnoma neodvisni od aplikacij, ki jih uporabljajo.

Programi ne komunicirajo direktno s CSP moduli. Uporabljajo CryptoAPI funkcije, ki so realizirane v sistemskih dinamičnih knjižnicah *Advapi32.dll* in *Crypt32.dll* . Operacijski sistem posreduje klice CryptoAPI funkcij ustreznim CSP modulom preko CryptoSPI (*system program interface*) programskega vmesnika. CryptoSPI vmesnik je enak pri vseh CSP modulih. V splošnem naj bi dobro napisani kriptografski programi delovali z različnimi CSP moduli. Uporabnik naj bi se sam odločil za ustrezen CSP modul s pričakovano stopnjo zaščite brez spreminjanja programa samega. Če to res želimo doseči, se moramo izogibati direktnih CSP klicev in uporabi posebnosti, ki so realizirane v posameznih CSP modulih. Omejiti se moramo zgolj na posredovanje podatkov in izbiro želene enkripcije.




Slika 1: Sistemska arhitektura CryptoAPI

# 4. Osnovne kriptografske funkcije

Osnovne kriptografske funkcije so tiste funkcije, ki so na voljo programerjem za uporabo pri razvoju njihovih kriptografskih aplikacij. Preko njih poteka vsa komunikacija s CSP moduli. Funkcije so napisane posplošeno z namenom univerzalne uporabe v povezavi s katerimkoli CSP modulom.

Vsak kriptografski program pri svojem delu uporablja vsaj en CSP modul, kompleksnejše aplikacije pa jih istočasno lahko uporabljajo tudi več. Zaradi možne uporabe več CSP modulov hkrati, se v vseh klicih kriptografskih funkcij pojavlja kot vhodni parameter sklic na ustrezen CSP modul. Zaradi zagotavljanja zaupnosti podatkov, so reference do podatkov v CSP modulih drugačne od tistih, ki jih vidimo iz aplikacije. Iz programa torej nikakor ne moremo priti do podatkovnih struktur znotraj samega CSP modula, s čimer je zagotovljena zaščita pred napadom.

Osnovne kriptografske funkcije so razdeljene v naslednje skupine:

- Service Provider funkcije,
- Generiranje in izmenjava ključev,
- Šifriranje in dešifriranje,
- Hash funkcije in digitalni podpisi.

Definicije vseh CryptoAPI funkcij se nahajajo v *WinCrypt.h* datoteki.

## 4.1. Service Provider funkcije

Service provider funkcije so namenjene vzpostavljanju in prekinjanju povezav programov s posameznimi CSP moduli, določanju privzetih CSP modulov posameznih uporabnikov in opravljanju drugih operacij v zvezi z vzdrževanjem CSP modulov.

Najpogosteje uporabljeni funkciji, ki se pojavljata prav v vsakem kriptografskem programu na osnovi CryptoAPI sta:

| | |
|---|---|
| *CryptAcquireContext* | V podanem CSP modulu poišče skladišče ključev določenega uporabnika in vrne sklic nanj (handle). |
| *CryptReleaseContext* | Sprosti sklic na skladišče ključev, ki je bil pridobljen pri klicu funkcije *CryptAcquireContext.* |

## 4.2. Generiranje in izmenjava ključev

Kriptografski ključi predstavljajo osrednji del vseh kriptografskih shem. Glede na namen se ločijo na simetrične ključe, ter na pare privatnih in javnih ključev. Ključe generiramo s funkcijama:

| | |
|---|---|
| *CryptGenKey* | Generira naključni simetrični ključ ali par javnih/privatnih ključev. Pri simetričnem ključu kot vhodni parameter nastopa specifikacija algoritma za katerega se ključ generira. Pri paru privatni/javni ključ je uporabljeni algoritem za generiranje ključa določen s podanim CSP modulom. |
| *CryptDeriveKey* | Generira simetrični ključ na osnovi vhodnih podatkov (npr. password). Ne generira privatnih/javnih ključev. Funkcija zagotavlja identičen ključ pri istem CSP modulu, istem algoritmu in istih vhodnih podatkih. |

V obeh primerih z vhodnmi parametri lahko vplivamo na dolžino ključa, inicializacijske vektorje, chaining način in druge lastnosti ključa. Funkciji kot rezultat vračata ročico (handle) ključa, kar nam omogoča njegovo uporabo pri naslednjih klicih. Skupaj v paru s ključem v rezultatu funkcije nastopa tudi informacija o algoritmu za katerega je bil ključ generiran.

Ključi se zaradi varnosti nahajajo interno v CSP modulih oziroma v njihovih skladiščih ključev. Vsakemu uporabniku določenega CSP modula je dodeljeno svoje interno skladišče. V primeru potrebe po izmenjavi ključev potrebujemo funkcije za izvoz in uvoz ključev (*CryptExportKey, CryptImportKey*). Ko ključ zapusti interno skladišče ključev, dobi posebno šifrirano obliko (BLOB), ki je primerna za prenos po nevarnem mediju.

## 4.3. Šifriranje in dešifriranje

Pri šifriranju obsežnejših količin podatkov se uporabljajo simetrični kriptografski algoritmi. Šifrirni ključ je enak dešifrirnemu. Simetrični ključ (session key) je lahko poljubne dolžine v velikosti od 40 do 2000 bitov (odvisno od uporabljenega algoritma). Osnovni funkciji te skupine sta:

| | |
|---|---|
| *CryptDecrypt* | dešifrira tajnopis z uporabo podanega simetričnega ključa |
| *CryptEncrypt* | šifrira čistopis z uporabo podanega simetričnega ključa |

Obe funkciji potrebujeta za izvedbo operacije simetrični ključ, ki poleg sebe nosi tudi informacijo o šifrirnem algoritmu. Kot vhodni parameter namesto konkretnega ključa nastopa sklic nanj (handle). Dobimo ga z uporabo ene izmed funkcij *CryptGenKey* ali *CryptImportKey*. Vhodne podatke posredujemo izbrani funkciji z uporabo kazalca, preko istega kazalca pa funkcija tudi vrne šifrirane ali dešifrirane podatke kot rezultat. V primeru večje količine podatkov funkcijo kličemo večkrat zapored. Ob tem moramo paziti na poseben *Boolean* parameter *Final* s katerim povemo, kdaj imamo v obdelavi zadnji blok podatkov. Glede na uporabljeni algoritem se zadnji korak namreč nekoliko razlikuje od predhodnih (bločni, tokovni, chaining).

Kot zanimivost naj omenim, da funkciji šifriranja in dešifriranja v Franciji zaradi omejevalnih uvoznih zakonov v Windows NT in 2000 okolju nista dovoljeni. CSP modul v tem primeru vrne napako. Pravi razlog te zakonske omejitve mi ni znan.

## 4.4. Zgoščevalne funkcije in digitalni podpisi

Funkcije te skupine se uporabljajo za digitalno podpisovanje podatkov. Vsi uporabniki javnega ključa, ki tvori par s pri podpisu uporabljenim privatnim ključem, lahko preverijo pristnost podpisa in celovitost prejetega dokumenta.

Pred uporabo zgoščevalne funkcije je potrebno zgraditi hash objekt (*CryptCreateHash*) z vsemi potrebnimi informacijami o hash algoritmu. Zgoščevanje nad konkretnimi podatki izvedemo s funkcijo *CryptHashData*. Rezultat zgoščevanja se zapisuje v ločeni hash objekt, zato pri večji količini vhodnih podatkov enostavno večkrat zaporedoma kličemo *CryptHashData* funkcijo.

*CryptSignHash* funkcija se uporablja za digitalno podpisovanje podatkov. Algoritmi za digitalno podpisovanje so počasni, zato se ne uporabljajo nad večjimi količinami podatkov. Nad podatki najprej izvedemo zgoščevalno funkcijo in rezultat zgoščevanja na koncu podpišemo. Pri *CryptSignHash* funkciji to izgleda tako, da ustrezen hash objekt nastopa kot vhodni podatek funkcije.

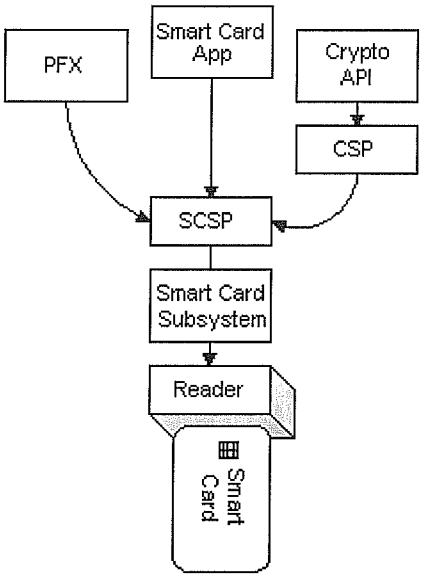
Preverjanje digitalnega podpisa izvedemo s *CryptVerifySignature* funkcijo. Tu kot vhodni parametri nastopajo digitalni podpis, javni ključ in hash objekt.

Potem, ko je bil izvršen klic funkcije *CryptSignHash* ali *CryptVerifySignature*, spreminjanje hash objekta ni več možno. *CryptHashData* funkcija ne vpliva več na podatke v hash objektu. Hash objekt je torej treba le še uničiti (*CryptDestroyHash*).

# 5. Cryptographic Service Provider (CSP)

Cryptographic Service Provider (CSP) moduli opravljajo kriptografsko strežbo. V njih so implementirani kriptografski standardi in algoritmi. Naloga CSP je tudi varovanje privatnih ključev. Vsak CSP ima namreč podatkovno bazo skladišč ključev. Vsako skladišče ima enolično ime, ki je ključ do uporabe CSP modula. Če podatkovna baza ne obstaja, CryptoAPI funkcije ne delujejo. Običajno obstaja privzeto skladišče ključev z logon imenom za vsakega uporabnika.

Dejansko je CSP dinamična knjižnica (DLL) skupaj s pripadajočim digitalnim podpisom, katerega pristnost CryptoAPI periodično preverja. Vsak CSP modul v dinamični knjižnici podpira predpisane funkcije CryptoSPI programskega vmesnika. Del funkcij je lahko realiziranih v hardveru (Smart Card, mikrokontrolerji, ipd.), s čimer lahko povečamo varnost in učinkovitost (shranjevanje privatnih ključev, izvajanje kriptografskih algoritmov).




Slika 2: Uporaba hardvera za realizacijo CSP modula

## 5.1. Tipi CSP modulov

Področje kriptografije je široko in se še razvija. Obstaja veliko algoritmov in standardov. Ti so razdeljeni po skupinah ali družinah glede na način izvedbe posameznih operacij. Tudi če dve podobni shemi uporabljata isti algoritem, se lahko pojavlja razlika med njima v dolžini uporabljenih ključev ali katerem drugem parametru.

Microsoft CryptoAPI je zasnovan tako, da različni tipi CSP modulov predstavljajo družine algoritmov. Vsaka družina določa nabor algoritmov in standardov za izvedbo ključnih kriptografskih funkcij. Ko se program poveže s CSP modulom določenega tipa, deluje vsaka CryptoAPI funkcija v skladu z algoritmi, ki sestavljajo to družino. Posamezno družino oziroma CSP tip sestavljajo algoritmi in postopki za naslednja področja:

- *Algoritem za izmenjavo ključev* — Vsak CSP tip določa in mora podpirati natanko en algoritem za izmenjavo ključev.
- *Algoritem za digitalni podpis* — Vsak CSP tip določa in mora podpirati natanko en algoritem za digitalno podpisovanje.
- *BLOB format za ključ* — Določa format BLOB strukture. Uporablja se za uvoz in izvoz simetričnih in privatnih ključev iz CSP.
- *Format digitalnega podpisa* — Omogoča verifikacijo podpisa s katerimkoli CSP modulom istega CSP tipa.
- *Metoda za generiranje simetričnih ključev* — Določa način generiranje simetričnega ključa.
- *Dolžina ključa* — Nekateri tipi določajo dolžino ključev.
- *Privzeti način delovanja* — Nekateri tipi določajo privzeti način delovanja za nekatere operacije: na primer privzeti bločni algoritem ali privzeto zgoščevalno funkcijo.

Obstaja nekaj predefiniranih osnovnih CSP tipov. Skupaj z osnovnimi podatki so zbrani v naslednji tabeli:

| **CSP tip** | **Izmenjava ključev** | **Digitalni podpis** | **Enkripcija** | **Zgoščevalne funkcije** |
|---|---|---|---|---|
| PROV_RSA_FULL | RSA | RSA | RC2, RC4 | MD5, SHA |
| PROV_RSA_SIG | - | RSA | - | MD5, SHA |
| PROV_RSA_SCHANNEL | RSA | RSA | CYLINK_MEK | MD5, SHA |
| PROV_DSS | - | DSS | - | MD5, SHA |
| PROV_DSS_DH | DH | DSS | CYLINK_MEK | MD5, SHA |
| PROV_DH_SCHANNEL | DH | DSS | RC2, RC4, CYLINK_MEK | MD5, SHA |
| PROV_FORTEZZA | KEA | DSS | SKIPJACK | SHA |
| PROV_MS_EXCHANGE | RSA | RSA | CAST | MD5 |
| PROV_SSL | RSA | RSA | različno | različno |

Tabela 1: Predefinirani CSP tipi

- PROV_RSA_FULL je najpogosteje uporabljeni in hkrati splošno uporaben CSP tip. Podpira namreč vse ključne kriptografske funkcije. Skupaj sta ga definirala Microsoft in RSA Data Security.

- PROV_FORTEZZA CSP tip vsebuje kriptografske protokole in algoritme v lasti National Institute of Standards and Technology (NIST).
- PROV_MS_EXCHANGE je bil razvit za potrebe Microsoft Exchange mail aplikacij in vseh drugih aplikacij združljivih z Microsoft Mail.

Čeprav je nekaj CSP tipov med seboj delno kompatibilnih, uporabljajo namreč nekatere iste algoritme, morajo soodvisni programi za pravilno delovanje vedno uporabljati isti CSP tip. Vsak CSP modul nekega tipa mora podpirati vse predpisane algoritme tega tipa, lahko pa podpira tudi dodatne. Kreiramo lahko tudi povsem svoje CSP tipe.

## 5.2. *Microsoft CSP moduli*

Microsoft je v sodelovanju z RSA Data Security razvil osnovni CSP modul imenovan **Microsoft Base Cryptographic Provider**. Njegovo interno ime je MS_DEF_PROV in se uporablja pri vzpostavljanju povezave kriptografskega programa z želenim CSP (*CryptAcquireContext*). Hkrati je to privzeti CSP modul na vseh Microsoft Windows operacijskih sistemih. Microsoft Base Provider je implementacija PROV_RSA_FULL CSP tipa in fizično obstaja v *RsaBase.DLL* datoteki. Podpira torej vse najpomembnejše kriptografske funkcije (izmenjava ključev, enkripcija/dekripcija, digitalni podpisi in zgoščevalne funkcije). Že na samem začetku (CryptoAPI 1.0) so ga zasnovali tako, da je bil primeren za izvoz iz ZDA.

V Microsoftovi ponudbi obstajata še dve izvedbi PROV_RSA_FULL CSP tipa, ki podpirata daljše ključe in dodatne algoritme. V naslednji tabeli so zbrane vse pomembnejše razlike med temi tremi Microsoft PROV_RSA_FULL tipi: **Base Provider**, **Strong Provider** in **Enhanced Provider**. Podane dolžine ključev so privzete vrednosti.

| **Algoritem** | **Base Provider** | **Strong Provider** | **Enhanced Provider** |
|---|---|---|---|
| RSA public-key signature algorithm | Ključ: 512 bit | Ključ: 512 bit | Ključ: 1,024 bit |
| RSA public-key exchange algorithm | Ključ: 512 bit | Ključ: 512 bit | Ključ: 1,024 bit |
| RC2 block encryption algorithm | Ključ: 40 bit | Ključ: 40 bit | Ključ: 128 bit |
| RC4 stream encryption algorithm | Ključ: 40 bit | Ključ: 40 bit | Ključ: 128 bit |
| DES | Ni podprt. | Ključ: 56 bit | Ključ: 56 bit |
| Triple DES (2 key) | Ni podprt. | Ključ: 112 bit | Ključ: 112 bit |
| Triple DES (3 key) | Ni podprt. | Ključ: 168 bit | Ključ: 168 bit |

Tabela 2: Pregled algoritmov in dolžin ključev posameznih MS PROV_RSA_FULL tipov.

Strong Provider (MS_ENHANCED_PROV) in Enhanced Provider sta navzdol kompatibilna z Base Providerjem. Strong Provider obstaja že od verzije CryptoAPI 1.0 in se je doslej uporabljal le v ZDA in Kanadi, Enhanced Provider pa je nov in prihaja skupaj z Windows 2000.

Ostali Microsoft CSP moduli so zbrani v tabeli 3.

| Provider | Tip | Interno ime |
|---|---|---|
| Microsoft DSS Cryptographic Provider | PROV_DSS | MS_DEF_DSS_PROV |
| Microsoft Base DSS and Diffie-Hellman Cryptographic Provider | PROV_DSS_DH | MS_DEF_DSS_DH_PROV |
| Microsoft DSS and Diffie-Hellman/Schannel Cryptographic Provider | PROV_DH_SCHANNEL | MS_DEF_DH_SCHANNEL_PROV |
| Microsoft RSA/Schannel Cryptographic Provider | PROV_RSA_SCHANNEL | MS_DEF_RSA_SCHANNEL_PROV |

Tabela 3: Microsoft CSP moduli

# 6. Pisanje lastnih CSP modulov

Pisanja lastnega CSP modula naj bi se lotili le v izjemnih situacijah. CSP moduli, ki so že razviti in predstavljajo del Microsoft Windows instalacij, naj bi zadostovali večini programskim varnostnim zahtevam. Dvomi o varnosti kljub vsemu obstajajo. Veliko programerjev in uporabnikov se namreč upravičeno sprašuje, če v Microsoftovih produktih morda ne obstajajo neka stranska vrata za NSA, skozi katera lahko zlorablja varnostne mehanizme. Namigovanj v tej smeri je na internetu vse polno in večina se sklicuje na Microsoftovo politiko ekskluzivne pravice podpisovanja CSP modulov.

Razlogi za pisanje svojih CSP modulov so naslednji:

- Implementacija novih ali nepodprtih algoritmov,
- Uporaba svojega hardvera za potrebe varnosti,
- Neodvisnost modula od Microsofta.

Preden se lotimo pisanja lastnega CSP modula naredimo naslednje:

- Izberemo kriptografske algoritme in podatkovne strukture.
- Poiščemo ali pripravimo implementacije izbranih algoritmov.

Razvoj lastnega kriptografskega CSP modula zajema naslednje korake:

1. Kreiranje CSP DLL knjižnice z izvozom vseh CryptoSPI funkcij.
2. Pisanje CSP setup programa, ki vnese ustrezne nastavitve v *Registry*.
3. Testno podpisovanja CSP modula.
4. Testiranje CSP modula.
5. Pridobivanje podpisa od Microsofta za razviti CSP modul.
6. Testiranje uradno podpisanega CSP modula.

Kreiranje CSP DLL knjižnice poteka na povsem običajen način, kot pri drugih DLL knjižnicah. V primeru hardverske rešitve moramo razviti še ustrezne gonilnike, ter morebitno programsko opremo na uporabljeni napravi. Vsak CSP mora imeti realizirane in izvožene vse funkcije CryptoSPI, imenovane tudi vstopne točke (Entry Points). Vseh skupaj je 23. Vsaka vstopna točka direktno ustreza neki kriptografski funkciji v CryptoAPI vmesniku. V primeru, da CSP neke funkcije ne podpira, mora ta vsaj vračati ustrezno kodo napake (E_NOTIMPL).

Naloge setup programa so naslednje:

- Kopiranje DLL datotek in potencialnih gonilnikov na ustrezna mesta v operacijskem sistemu.
- Vpis nastavitev v *Registry*:
  - registracija CSP modula (ime, tip, digitalni podpis),
  - nastavitev računalniku privzetega CSP (en privzeti CSP za vsak CSP tip),
  - nastavitev uporabnikom privzetega CSP (en privzeti CSP za vsak CSP tip).

Za testiranje novo razvitih CSP modulov Microsoft ponuja posebno orodje *Cryptographic Service Provider Developer's Kit (CSPDK)*, ki pa je kljub sprostitvi izvoznih omejitev za področje kriptografije še vedno na voljo le programerjem v ZDA in Kanadi. Brez tega orodja je razvoj svojega CSP modula praktično onemogočen. V paketu so namreč zbrani program za testno podpisovanje (**Sign.exe**), datoteki *csp.c in csp.def,* ter posebna različica *Advapi32.dll*, ki je pogoj, da CSP deluje s testnim podpisom.

Testno podpisovanje CSP DLL datoteke moramo opraviti ob vsaki spremembi oziroma ponovnem prevajanju. Podpis se hrani v *Registry*-ju razen v Windows 2000, kjer je podpis integriran v DLL. Za testno podpisovanje uporabimo program **Sign.exe**, ki kot rezultat vrača podpis v datoteki *.sig* .

V skladu z ameriškimi izvoznimi zakoni glede kriptografije je dolžnost Microsofta, da vrši nadzor nad kriptografskimi rešitvami. Zaradi tega izvaja striktno politiko digitalnega podpisovanja vsakega CSP, ki bo deloval z njihovimi operacijskimi sistemi.

# 7. Primer uporabe CryptoAPI

Za lažji prehod s teorije v prakso je Microsoft poskrbel z nekaj primeri uporabe. Najdemo jih v sklopu MSDN knjižnice. Preden pa se česarkoli lotimo, se moramo nekoliko podrobneje spomniti ozadja celotne arhitekture CryptoAPI. Kriptografske funkcije se izvajajo le v CSP modulih in do njih moramo imeti dostop. Pogoj za to je obstoj skladišča ključev želenega CSP modula. Če ta še ne obstaja, ga moramo zgraditi. Pomagamo si lahko z enim od primerov ali pa poiščemo prav temu namenjen programček *InitUser.exe* na microsoftovi domači strani. Ko je to za nami, se lahko prosto lotimo raziskovanja CryptoAPI funkcij.

Podrobneje sem si ogledal pristop in razvoj programov za šifriranje in dešifriranje, ter pripravil ustrezna programa *Encrypt.exe* in *Decrypt.exe*. Njun izpis je dodan nalogi, kot priloga.

Osnovna naloga omenjenih programov je šifriranje in dešifriranje datotek. Klicni parametri pri obeh so identični.

- Vhodna datoteka (čistopis/tajnopis)
- Izhodna datoteka (tajnopis/čistopis)
- Geslo (neobvezno)

Programa lahko delujeta na dva različna načina. Razlika se pojavlja pri načinu generiranja ključev. V primeru, da je geslo podano, se le to uporablja za generiranje ključa. Kadar pa

gesla ni, se generira naključni simetrični ključ, šifrira z ustreznim javnim ključem, izvozi in nazadnje v tej šifrirani obliki zapiše v izhodno datoteko.

V prvem primeru lahko določeno datoteko dešifriramo z uporabo gesla. Program na osnovi gesla ključ lahko vedno zgradi. V drugem primeru moramo najprej iz datoteke prebrati šifrirani ključ, ga uvoziti v skladišče ključev uporabljenega CSP modula in po uspešnem uvozu dešifrirati datoteko.

V programih je uporabljen privzeti PROV_RSA_FULL CSP modul in njegovo privzeto skladišče ključev. Za šifriranje in dešifriranje sem uporabil RC2 algoritem (pri tem CSP modulu je sicer na voljo tudi RC4) in za zgoščevanje MD5 hash funkcijo.

# 8. Zaključek

Odločitev za ali proti uporabi CryptoAPI programskega vmesnika je odvisna od več različnih dejavnikov. Pomembni so predvsem varnost, enostavnost uporabe, strateška pomembnost in nenazadnje pogled na trg programske opreme, ki je ponujeno rešitev sprejela. Pogosto so dobre reference tiste, ki imajo največjo težo pri odločitvi za.

Programsko opremo na trgu, ki uporablja CryptoAPI lahko razdelimo na dva dela:

- programi razviti na osnovi Microsoft CSP modulov
- programi razviti na osnovi lastnih CSP modulov ali zgolj ponudba posebnih CSP modulov

Med ponudniki novo razvitih CSP modulov so najzanimivejši naslednji:

- **Atalla** - *TrustMaster CSP hardware security engine*: hardverska rešitev z uporabo PCI slot kartice (omogoča hitro podpisovanje in verifikacijo digitalnih podpisov, ter drugih zahtevnih računsko zahtevnih operacij)
- **BBN Corporation** – hardversko podprto generiranje in shranjevanje kriptografskih ključev
- **DataKey** – *SignaSURE CSP* : generiranje in nadzor ključev za podpisovanje, ter digitalno podpisovanje z uporabo pametne kartice ali posebnega pametnega ključa (*Smart Key hardware token*)
- **Hewlett Packard** – *International Cryptography Framework (ICF), ICF CSP*: programska rešitev z uporabo kriptografskih funkcij preko CryptoAPI do ICF CSP
- **Information Resource Engineering (IRE)** – *IRE CSP*: podpira RSA, DSA, Diffie-Hellman (512 do 2048 bitni ključi) DES, 3DES, MD5, SHA1, MAC, HMAC, IPv6 IPSEC generiranje ključev in digitalno podpisovanje z uporabo pametne kartice. IRE CSP bo v bodočnosti podpiral tudi *IRE/Analog Device* napravo z uporabo posebnega kriptografskega čipa za pospeševanje kriptografskih funkcij, ter nadzor ključev.
- **PC/SC Workgroup :** Bull, Schlumberger, Siemens Nixdorf in Spyrus so napovedali razvoj svojih CSP za uporabo s kriptografskimi pametnimi karticami.
- **Rainbow Technologies, Internet Security Group (ISG)** – *CryptoSwift PCI board:* razvili so nekaj svojih CSP za podporo svojemu izdelku CryptoSwift, ki temelji na hardverski rešitvi s PCI kartico. Napisali so svoj CSP, ki je kompatibilen z PROV_RSA_FULL CSP tipom. Ima podporo za RSA do 2048 bitov, Diffie-Hellman do 1024 in DSA.

- **Real Software:** razvili so svoj CSP za CryptoAPI 1.0 z uporabo eliptičnih krivulj, poleg tega pa podpirajo tudi DSA, DES, SHA in druge znane algoritme.
- **Spyrus** – *EES LYNKS Privacy Card:* podpirajo Fortezzo za vladne ustanove in ponujajo rešitve na osnovi smart kartice.
- **Trusted Information System, Inc. (TIS)** – *TIS RecoverKey CSP*

Prevladujejo rešitve z uporabo dodatne strojne opreme ali pametnih kartic. Po eni strani lahko sklepamo, da se povečini vsi zavedajo pomena večje varnosti z uporabo dodatnih strojnih komponent, po drugi strani pa vidimo, da vlada veliko nezaupanje do Microsofta oziroma do uporabe rešitev tretje osebe.

Kot zanimivost velja omeniti, da tudi Nova ljubljanska banka ponuja uporabnikom rešitev za domače bančništvo, ki sloni na CryptoAPI. Podjetje Zaslon, avtor aplikacije, je uporabilo knjižnice Activ Card Gold, ki sloni na CryptoAPI 2.0, uporablja pa Microsoftov Base Cryptographic Provider (PROV_RSA_FULL).

# LITERATURA IN VIRI


[1] Supporting CryptoAPI in Real-World Applications, Microsoft MSDN Library, 1997

[2] Elizabeth Wiewall, Secure Your Applications with the Microsoft CryptoAPI, Microsoft MSDN Library, 1996

[3] Microsoft CryptoAPI Overview, Microsoft MSDN Library, 1998

[4] John Boyer, Digital Signatures with the Microsoft CryptoAPI, Dr.Dobb's Journal, June 1998

# Dodatek seminarski nalogi – Microsoft CryptoAPI

V Tipru smo razvili modularno tipkovnico MID. Osnovna ideja pri njenem snovanju je bila ponuditi uporabnikom možnost, da si tipkovnico prilagodijo po svojih lastnih željah. Njena najpomembnejša lastnost je programirljivost tipk. V ta namen smo razvili program MIDWIN, ki uporabniku na enostaven in prijazen način omogoča programiranje tipk.

Praksa je pokazala, da je programiranje z MIDWIN programom pogosto celo preveč enostavno opravilo in se ga lotevajo nepooblaščene osebe. Slednje povzroča težave vzdrževalcem strojne opreme, saj instalirana oprema naenkrat ne deluje več tako, kot je bilo sprva načrtovano. Nekateri kupci so nas zato začeli spraševati po funkciji zaklepanja vsebine tipkovnice. Problem ni enostaven, saj moramo v izogib možnim bodočim težavam ob funkciji zaklepanja poskrbeti tudi za funkcijo odklepanja. Slednja mora biti dovolj dobro zaščitena, da je ne bi mogla izkoristiti nepooblaščena oseba. Hkrati mora biti odklepanje za pooblaščene osebe dovolj enostavno, da se sploh odločijo za uporabo zaščitnega mehanizma.

Zaščito MID tipkovnice sem zasnoval na CryptoAPI programskem vmesniku. Uporabil sem privzeti CSP modul (PROV_RSA_FULL), saj ponuja dovolj dobro zaščito za moj primer. Hkrati je to edini CSP, ki se pojavlja na vseh Microsoft Windows operacijskih sistemih. Pri kreiranju hash objekta za kreiranje simetričnega ključa sem uporabil MD5 zgoščevalno funkcijo. Pri enkripcija in dekripcija gesla je uporabljen RC2 bločni algoritem.

Celoten zaščitni mehanizem sestavljata dve dinamični knjižnici (DLL) *MidEncrypt.DLL* in *MidDecrypt.DLL* . Uporabnik prejme skupaj z instalacijo MIDWIN programa vedno le *MidEncrypt.DLL* , medtem ko imamo *MidDecrypt.DLL* le v Tipru in ga uporabljamo samo za primere, ko uporabnik pozabi geslo s katerim je tipkovnico zaklenil.

Osnovna ideja je sila enostavna. Uporabnik zaklene tipkovnico z vpisom gesla (omejitev do 12 znakov). Le to se ob zaklepanju enkriptira in zapiše v rezervirano mesto v FLASH pomnilniku. Velikost pomnilnika je 8KBit (0000h - 1FFFh). Rezervirano je področje od 50h – 6Fh, česar uporabniki ne vedo in je tudi ob uspešnem branju celotnega pomnilnika nemogoče ugotoviti. Vpisovanju gesla je namenjena funkcija *StorePassword,* ki v sebi združuje enkripcijo gesla in uporablja Tiprovo knjižnico funkcij MidApi za komunikacijo s tipkovnico. Pred vpisom novega gesla funkcija *StorePassword* preveri pravilnost obstoječega gesla in šele nato izvede zahtevano operacijo. Funkcija se izvede tudi v primeru, da tipkovnica pred tem še ni bila zaščitena

Odstranjevanju zaščite tipkovnice je namenjena funkcija *RemovePassword.* Vhodni parameter te funkcije je obstoječe geslo, ki je vpisano v tipkovnici. Ob pravilnem vnosu funkcija pobriše tisti del flash pomnilnik v tipkovnici, ki je rezerviran za geslo.

Ob odklepanju tipkovnice mora uporabnik podati geslo, ki je bilo uporabljeno za zaklepanje. Po vpisu se to enkriptira in nato primerja z enkriptiranim geslom, ki je vpisano v tipkovnici. Vse skupaj je izvedeno v eni sami funkciji *CheckPassword.* V

sebi torej združuje enkripcijo in branje vsebine tipkovnice (enkriptirano geslo) z uporabo MidApi programskega vmesnika.

Na strani uporabnika je v celotni shemi uporabljena le funkcija enkriptiranja. Ne obstaja možnost, da bi uporabnik na enostaven način prebral enkriptirano geslo in tudi nima na voljo funkcije s katero bi najdeno geslo dekriptiral.

V primeru, da uporabnik pozabi geslo s katerim je tipkovnico zaklenil, brez posredovanja Tipra ne more narediti ničesar. V knjižnico sem dodal funkcijo *ExportFlashMemory* , ki prebere vsebino celotnega FLASH pomnilnika tipkovnice in jo zapiše v binarno datoteko. To datoteko mora poslati na Tipro, kjer z ustreznim programom v binarni datoteki poiščemo enkriptirano geslo in ga dekriptiramo. Originalno geslo lahko nato pošljemo nazaj osebi, ki je dekripcijo zahtevala.

Andrej Dobrovoljc

Novo mesto, 1.8.2000

```
//-------------------------------------------------------------------
// Program for file encryption using optional user password.
// Supports two types of session key generation.
// USAGE: encrypt  <source file> <dest file> [ <password> ]
//-------------------------------------------------------------------

#include <stdio.h>
#include <windows.h>
#include <wincrypt.h>

#define ENCRYPT_ALGORITHM    CALG_RC2
#define ENCRYPT_BLOCK_SIZE   8

void HandleError(char *s);

BOOL EncryptFile(
     PCHAR szSource,         // in
     PCHAR szDestination,    // out
     PCHAR szPassword);      // in

void main(int argc, char *argv[])
{
    PCHAR szSource;
    PCHAR szDestination;
    PCHAR szPassword;

    // Validate argument count.
    if(argc != 3 && argc != 4) {
        printf("USAGE: encrypt <source file> <dest file> [ <password> ]\n");
        exit(1);
    }

    // Parse arguments.
    szSource       = argv[1];
    szDestination  = argv[2];

    if(argc == 4) szPassword = argv[3];

    if(EncryptFile(szSource, szDestination, szPassword))
    {
       printf("Encryption of the file %s was a success. \n", szSource);
       printf("The encrypted data is in file %s.\n",szDestination);
    }
    else
    {
        HandleError("Error encrypting file!");
    }

    exit(0);
}

static BOOL EncryptFile(PCHAR szSource, PCHAR szDestination, PCHAR szPassword)
//     szSource          - input plaintext file name
//     szDestination     - output ciphertext file name
//     szPassword        - password string or NULL if a password is not used
{
    FILE *hSource;
    FILE *hDestination;

    HCRYPTPROV hCryptProv;
    HCRYPTKEY hKey;
    HCRYPTKEY hXchgKey;
    HCRYPTHASH hHash;

    PBYTE pbKeyBlob;
    DWORD dwKeyBlobLen;

    PBYTE pbBuffer;
    DWORD dwBlockLen;
    DWORD dwBufferLen;
    DWORD dwCount;

    // Open source file.
    if(!(hSource = fopen(szSource,"rb")))
        HandleError("Error opening source plaintext file!");

    // Open destination file.
    if(!(hDestination = fopen(szDestination,"wb")))
        HandleError("Error opening destination ciphertext file!");
```

```
    // Get handle to the default provider.
    if(!CryptAcquireContext(&hCryptProv, NULL, NULL, PROV_RSA_FULL, 0))
       HandleError("Error during CryptAcquireContext!");

    // Create the session key.
    if(!szPassword )
    {
        //--------------------------------------------------------------
        // No password was passed.
        // Encrypt the file with a random session key and write the key
        // to the destination (ciphertext) file.
        //--------------------------------------------------------------

        // Create a random session key.
        if(!CryptGenKey(hCryptProv, ENCRYPT_ALGORITHM, CRYPT_EXPORTABLE, &hKey))
            HandleError("Error during CryptGenKey. \n");

        // Get handle to the encrypter's exchange public key.
        if(!CryptGetUserKey(hCryptProv, AT_KEYEXCHANGE, &hXchgKey))
            HandleError("User public key is not available and may not exist.");

        // Determine size of the key blob, and allocate memory.
        if(!CryptExportKey(hKey, hXchgKey, SIMPLEBLOB, 0, NULL, &dwKeyBlobLen))
            HandleError("Error computing blob length! \n");

        if(!(pbKeyBlob =(BYTE *)malloc(dwKeyBlobLen)))
            HandleError("Out of memory. \n");

        // Encrypt and export session key into a simple key blob.
        if(!CryptExportKey(hKey, hXchgKey, SIMPLEBLOB, 0, pbKeyBlob, &dwKeyBlobLen))
            HandleError("Error during CryptExportKey!\n");

        // Release key exchange key handle.
        CryptDestroyKey(hXchgKey);
        hXchgKey = 0;

        // Write size of key blob to destination file.
        fwrite(&dwKeyBlobLen, sizeof(DWORD), 1, hDestination);
        if(ferror(hDestination))
            HandleError("Error writing header.");

        // Write key blob to destination file.
        fwrite(pbKeyBlob, 1, dwKeyBlobLen, hDestination);
        if(ferror(hDestination))
            HandleError("Error writing header");
    }
    else
    {
        //--------------------------------------------------------------
        // Encryption with a session key derived from a password.
        // The session key will be recreated when the file is decrypted.
        // Password is required to create the session key for decription.

        // Create a hash object.
        if(!CryptCreateHash(hCryptProv, CALG_MD5, 0, 0, &hHash))
            HandleError("Error during CryptCreateHash!\n");

        // Hash the password.
        if(!CryptHashData(hHash, (BYTE *)szPassword, strlen(szPassword), 0))
            HandleError("Error during CryptHashData. \n");

        // Derive a session key from the hash object.
        if(!CryptDeriveKey(hCryptProv, ENCRYPT_ALGORITHM, hHash, 0, &hKey))
            HandleError("Error during CryptDeriveKey!\n");

        // Destroy the hash object.
        CryptDestroyHash(hHash);
        hHash = 0;
    }

    //--------------------------------------------------------------------
    //  The session key is now ready. If it is not a key derived from a
    //  password, the session key encrypted with the encrypter's private
    //  key has been written to the destination file.

    //--------------------------------------------------------------------
    // Determine number of bytes to encrypt at a time.
    // This must be a multiple of ENCRYPT_BLOCK_SIZE.
```

```
    dwBlockLen = 1000 - 1000 % ENCRYPT_BLOCK_SIZE;

    //--------------------------------------------------------------------
    // Determine the block size. If a block cipher is used,
    // it must have room for an extra block.
    if(ENCRYPT_BLOCK_SIZE > 1)
        dwBufferLen = dwBlockLen + ENCRYPT_BLOCK_SIZE;
    else
        dwBufferLen = dwBlockLen;

    // Allocate memory.
    if(!(pbBuffer = (BYTE *)malloc(dwBufferLen)))
        HandleError("Out of memory. \n");

    // In a do loop, encrypt the source file and write to the destination file.
    do
    {
        // Read up to dwBlockLen bytes from the source file.
        dwCount = fread(pbBuffer, 1, dwBlockLen, hSource);
        if(ferror(hSource))
            HandleError("Error reading plaintext!\n");

        // Encrypt data.
        if(!CryptEncrypt(hKey, 0, feof(hSource), 0, pbBuffer, &dwCount, dwBufferLen))
           HandleError("Error during CryptEncrypt. \n");

        // Write data to the destination file.
        fwrite(pbBuffer, 1, dwCount, hDestination);
        if(ferror(hDestination))
            HandleError("Error writing ciphertext.");
    }
    while(!feof(hSource));
    //--------------------------------------------------------------------
    //  End the do loop when the last block of the source file has been
    //  read, encrypted, and written to the destination file.

    // Close files.
    if(hSource) fclose(hSource);
    if(hDestination) fclose(hDestination);

    // Free memory.
    if(pbBuffer) free(pbBuffer);

    // Destroy session key.
    if(hKey) CryptDestroyKey(hKey);

    // Release key exchange key handle.
    if(hXchgKey) CryptDestroyKey(hXchgKey);

    // Destroy hash object.
    if(hHash) CryptDestroyHash(hHash);

    // Release provider handle.
    if(hCryptProv) CryptReleaseContext(hCryptProv, 0);

    return(TRUE);
} // End of Encryptfile

void HandleError(char *s)
{
    fprintf(stderr,"An error occurred in running the program. \n");
    fprintf(stderr,"%s\n",s);
    fprintf(stderr, "Error number %x.\n", GetLastError());
    fprintf(stderr, "Program terminating. \n");
    exit(1);
}
```

```
//--------------------------------------------------------------
// Program for file decryption using optional user password.
// Supports two types of session key generation.
// USAGE: decrypt  <source file> <dest file> [ <password> ]
//--------------------------------------------------------------

#include <stdio.h>
#include <windows.h>
#include <wincrypt.h>

#define ENCRYPT_ALGORITHM   CALG_RC2
#define ENCRYPT_BLOCK_SIZE  8

void HandleError(char *s);

BOOL DecryptFile(
     PCHAR szSource,         // in
     PCHAR szDestination,    // out
     PCHAR szPassword);      // in

void main(int argc, char *argv[])
{
    PCHAR szSource;
    PCHAR szDestination;
    PCHAR szPassword;

    // Validate argument count.
    if(argc != 3 && argc != 4) {
        printf("USAGE: decrypt <source file> <dest file> [ <password> ]\n");
        exit(1);
    }

    // Parse arguments.
    szSource       = argv[1];
    szDestination  = argv[2];

    if(argc == 4) szPassword = argv[3];

    if(DecryptFile(szSource, szDestination, szPassword))
    {
        printf("\nDecryption of file %s succeeded. \n", szSource);
        printf("The decrypted file is %s .\n",szDestination);
    }
    else
    {
        HandleError ("\nError decrypting file. \n");
    }

    exit(0);
}

static BOOL DecryptFile(PCHAR szSource, PCHAR szDestination, PCHAR szPassword)
//     szSource         - input ciphertext file name
//     szDestination    - output plaintext file name
//     szPassword       - password string or NULL if a password is not used
{
    FILE *hSource;
    FILE *hDestination;

    HCRYPTPROV hCryptProv;
    HCRYPTKEY hKey;
    HCRYPTHASH hHash;

    PBYTE pbKeyBlob = NULL;
    DWORD dwKeyBlobLen;

    PBYTE pbBuffer;
    DWORD dwBlockLen;
    DWORD dwBufferLen;
    DWORD dwCount;

    BOOL status = FALSE;

    // Open source file (cipher text).
    if(!(hSource = fopen(szSource,"rb")))
       HandleError("Error opening ciphertext file!");

    // Open destination file (plain text).
    if(!(hDestination = fopen(szDestination,"wb")))
```

```
    HandleError("Error opening plaintext file!");

// Get a handle to the default provider.
if(!CryptAcquireContext(&hCryptProv, NULL, NULL, PROV_RSA_FULL, 0))
   HandleError("Error during CryptAcquireContext!");

//  Check for existence of a password.
if(!szPassword)
{
    // No password given. Try to decrypt file with the saved session key.

    // Read key blob length from source file, and allocate memory.
    fread(&dwKeyBlobLen, sizeof(DWORD), 1, hSource);
    if(ferror(hSource) || feof(hSource))
        HandleError("Error reading file header!");

    if(!(pbKeyBlob = (BYTE *)malloc(dwKeyBlobLen)))
        HandleError("Memory allocation error.");

    // Read key blob from source file.
    fread(pbKeyBlob, 1, dwKeyBlobLen, hSource);
    if(ferror(hSource) || feof(hSource))
        HandleError("Error reading file header!\n");

    // Import key blob into CSP.
    if(!CryptImportKey(hCryptProv, pbKeyBlob, dwKeyBlobLen, 0, 0, &hKey))
        HandleError("Error during CryptImportKey!");
}
else
{
    // Decrypt the file with a session key derived from a password.

    // Create a hash object.
    if(!CryptCreateHash(hCryptProv, CALG_MD5, 0, 0, &hHash))
        HandleError("Error during CryptCreateHash!");

    // Hash in the password data.
    if(!CryptHashData(hHash, (BYTE *)szPassword, strlen(szPassword), 0))
        HandleError("Error during CryptHashData!");

    // Derive a session key from the hash object.
    if(!CryptDeriveKey(hCryptProv, ENCRYPT_ALGORITHM, hHash, 0, &hKey))
        HandleError("Error during CryptDeriveKey!");

    // Destroy the hash object.
    CryptDestroyHash(hHash);
    hHash = 0;
}

//-------------------------------------------------------------------
//   The decryption key is now available, either imported from a blob
//   read from the source file or created using the password.
//   This point in the program is not reached if the decryption key is
//   not available.

//-------------------------------------------------------------------
// Determine the number of bytes to decrypt at a time.
// This must be a multiple of ENCRYPT_BLOCK_SIZE.
dwBlockLen = 1000 - 1000 % ENCRYPT_BLOCK_SIZE;
dwBufferLen = dwBlockLen;

// Allocate memory.
if(!(pbBuffer = (BYTE *)malloc(dwBufferLen)))
    HandleError("Out of memory!\n");

// Decrypt source file, and write to destination file.
do {
    // Read up to dwBlockLen bytes from source file.
    dwCount = fread(pbBuffer, 1, dwBlockLen, hSource);
    if(ferror(hSource))
        HandleError("Error reading ciphertext!");

    // Decrypt data.
    if(!CryptDecrypt(hKey, 0, feof(hSource), 0, pbBuffer, &dwCount))
        HandleError("Error during CryptDecrypt!");

    // Write data to destination file.
    fwrite(pbBuffer, 1, dwCount, hDestination);
    if(ferror(hDestination))
```

```
            HandleError("Error writing plaintext!");
        }
        while(!feof(hSource));

        status = TRUE;

        // Close files.
        if(hSource) fclose(hSource);
        if(hDestination) fclose(hDestination);

        // Free memory.
        if(pbKeyBlob) free(pbKeyBlob);

        if(pbBuffer) free(pbBuffer);

        // Destroy session key.
        if(hKey) CryptDestroyKey(hKey);

        // Destroy hash object.
        if(hHash) CryptDestroyHash(hHash);

        // Release provider handle.
        if(hCryptProv) CryptReleaseContext(hCryptProv, 0);

        return status;
} // End of Decryptfile

void HandleError(char *s)
{
    fprintf(stderr,"An error occurred in running the program. \n");
    fprintf(stderr,"%s\n",s);
    fprintf(stderr, "Error number %x.\n", GetLastError());
    fprintf(stderr, "Program terminating. \n");
    exit(1);
}
```