Programozható chipkártyák – elmélet és gyakorlati tapasztalatok [1]

Írta:
Berta István Zsolt és Mann Zoltán Ádám
smartcard@ebizlab.hit.bme.hu

Konzulens:
Dr. Vajda István egyetemi tanár

Budapest Műszaki és Gazdaságtudományi Egyetem
Híradástechnikai Tanszék
Üzleti Adatbiztonság Laboratórium (E-BizLab)

A chipkártyák (*smart card*) mindennapjaink részévé váltak: bankkártyák, telefonkártyák, igazolványok formájában lépten-nyomon találkozunk velük. Azonban ezek az alkalmazások a modern chipkártyák technikai lehetőségeinek csupán egy töredékét hasznosítják. Jelen tanulmány célja annak bemutatása, milyen kriptográfiai lehetőségeket nyújtanak a programozható intelligens kártyák nagybiztonságú rendszerek kialakítására.

Chipkártyákról általában

A mai chipkártyák ősei az 1950-es évek óta (és mind a mai napig) széles körben alkalmazott mágneskártyák. Az ezeken elhelyezett mágnescsíkon néhány száz byte adatot lehet tárolni, mágneses úton olvasni, esetleg írni. Az 1970-es években merült fel mikroelektronikai áramkörök alkalmazásának lehetősége. A francia Bull cég 1979-ben készítette el első mikroprocesszorral is rendelkező kártyáját, melyen azonban a processzor és a memória még külön chipen helyezkedett el. Ez nem bizonyult kellően megbízható megoldásnak. A technikai fejlődés azonban csak az 1980-as években tette lehetővé az összes áramkör egyetlen chipre való integrálását. Franciaországban 1986 óta használnak chipkártyákat az utcai telefonokhoz.

A chipkártyák számos előnnyel rendelkeznek a mágneskártyákkal szemben. Egyrészt kevésbé érzékenyek mágneses zavarokra, másrészt pedig biztonságosabbak, hiszen a kártya processzora kontrollálja az olvasási és írási műveleteket. Ezáltal az adatokhoz közvetlenül nem, hanem csak közvetetten, előre definiált műveleteken keresztül lehet hozzáférni.

Természetesen a fejlődés azóta sem állt meg. Az intelligens kártyák egyre nagyobb memóriával és egyre nagyobb teljesítményű processzorral rendelkeznek. Ez lehetővé teszi, hogy a kártya ne csak végrehajtsa a kívülről érkező parancsokat, hanem önálló alkalmazások fussanak rajta (esetleg több is), a kártya operációs rendszere fölött. Az ezzel a tulajdonsággal rendelkező kártyák a programozható chipkártyák, melyek számos új előnyt hordoznak. Először is lényegesen nagyobb rugalmasságot: csupán egyféle kártyát kell legyártani, mely

[1] Ez a cikk a **Magyar Távközlés** 2000. áprilisi számában jelent meg.

ezután mindig újabb és újabb funkciókkal látható el újabb szoftver feltöltésével. Ebből adódóan a kártyákat elég a gyártási folyamat után specializálni, tehát kevesebb kártyatípusra van szükség, amelyek viszont így nagyobb példányszámban gyárthatók. Ez egyrészt a smart card technológia árának csökkenéséhez vezet, másrészt pedig ahhoz, hogy kisebb cégek is képesek kártyákat kiadni.

A programozható chipkártyák másik nagy előnye, hogy megfelelő operációs rendszer támogatás esetén, a kártyán egyszerre több alkalmazás is lehet. Ez megnyitja annak lehetőségét, hogy személyenként egyetlen kártya ellásson minden azonosítási, hitelesítési, adattárolási funkciót. Természetesen ez komoly problémákat vet fel, hiszen garantálni kell azt, hogy a rendszer minden szereplője pontosan annyi információt tudjon kinyerni a kártyából, amennyire jogosult. Végezetül pedig a programozható chipkártyák lehetőséget biztosítanak fejlett kriptográfiai algoritmusok és protokollok megvalósítására, amelyek nagymértékben növelhetik a rendszer biztonságát.

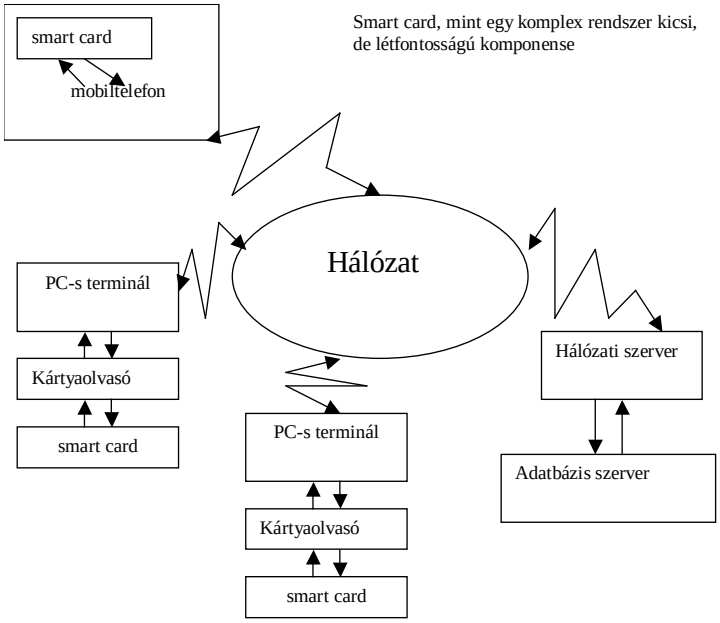
Fontos azonban említést tenni a programozható chipkártyák korlátairól is. A kártya fizikai biztonsága érdekében a teljes hardvert egyetlen chipben kell megvalósítani, hiszen különben a kártya buszrendszerére kapcsolódva értékes információkat lehetne nyerni. Így a processzor és a memória összteljesítményét korlátozza, hogy a termelődő hő elvezetése nincs megoldva. A másik kritikus faktor a kártya előállítási költsége. Ez már így is többszöröse az egyszerűbb kártyák költségének, bár nagyobb példányszám esetén nyilván drasztikusan csökkenthető. Minden esetre ez is gátat szabhat a teljesítmény növelésének.

Bár az elkövetkezőkben programozható chipkártyákról lesz szó, megemlítjük még az optikai kártyákat is, melyek főleg az egészségügyi szektorban népszerűek. Ezeknél mind az írás, mind az olvasás, és persze maga a pozícionálás is optikai úton történik, ami sokkal nagyobb pontosságot és ezáltal nagyságrendekkel nagyobb információsűrűséget tesz lehetővé. Emellett gyakran a kártya egész felülete információtárolásra szolgál. Az ilyen kártyák előállítása drága, viszont a tárolókapacitás a megabyte-os tartományban mozog. Ez előnyös, hiszen az orvosi szektorban a beteg kezeléseinek naplózásán kívül esetleg egész röntgenképeket is tárolni kell.

Az 1996-ban alakult PC/SC Workgroup a PC és smartcard piac legnagyobbjait tömöríti magába: Bull CP8, Gemplus, Hewlett-Packard, IBM, Microsoft, Schlumberger, Siemens Nixdorf, Sun Microsystems, Toshiba, Verifone. Az együttműködés eredményeként 1997 decemberében nyilvánosságra hozott specifikáció az intelligens kártyák, a kártyaolvasók és a PC-k együttműködését definiálja. A PC/SC természetesen nem az egyetlen ilyen jellegű tömörülés. A teljesség igénye nélkül megemlítünk még néhányat: OpenCard, MultOS, JavaCard, Muscle. Ezek közül két specifikáció egy-egy konkrét megvalósítását fogjuk konkrétan bemutatni: a PC/SC ajánlást megvalósító Microsoft Smart Card for Windows-t valamint egy JavaCard-megvalósítást, a Bull Odyssey 1.2 kártyáját.

Chipkártya alapú rendszerek

A chipkártyák rendszerint egy nagyobb rendszer egy komponensét alkotják. Ennek megfelelően a chipkártyák működését, lehetőségeit is ebben az összefüggésben érdemes vizsgálni. (1. ábra)



1. ábra – Smart Card, mint egy kompelx rendszer eleme

Egy chipkártya alapú rendszer tipikus szereplői a kártya gyártója (card manufacturer), kibocsátója (card issuer) és birtokosa (card holder). Ezek közül az első kettő egybeeshet. További szereplő lehet a terminálok tulajdonosa (terminal owner); ez például a telefonkártya szolgáltatás esetén rendszerint a telefontársaság (amely egyben a kártya kibocsátója is), de például bankkártyák esetén lehetőség van más bank termináljainak, vagy elektronikus kártyaelfogadóhelyek (Electronic Point of Sale) használatára. Tovább bonyolíthatja a helyzetet, hogy a kártya birtokosa nem feltétlenül egyezik meg a kártyán lévő adatok tulajdonosával (data owner); jó példa erre a hitelkártya, ahol a kártyán lévő adat tulajdonosa a bank, hiszen csak ő tudja ezen adatokat megváltoztatni. Programozható kártyák esetén nem egyezik meg a kártya gyártója a kártyán futó programok készítőivel.

Mindez azért rendkívül fontos, mert a szereplők számával arányosan nő a támadási lehetőségek száma is. (Támadás alatt itt természetesen adatok elleni támadást értünk, aminek célja lehet például egy szolgáltatás illetéktelen használata, közvetlen anyagi haszon, titkos információk megszerzése stb.) Ha csupán egyetlen szereplő volna, akkor kommunikáció híján a rendszer a lehető legbiztonságosabb lenne. Azonban, amint áttérünk egy kétszereplős modellre, azonnal megjelenik a lehetőség a két fél közti csatorna megtámadására, vagy arra, hogy az egyik fél támadást intézzen a másik ellen. Hasonlóan a funkciók minden további bontása újabb támadási lehetőségeket kínál.

A rendszer elleni támadások aszerint csoportosíthatók, hogy a támadó mely komponensen, illetve mely komponensek közötti csatornán intéz támadást a rendszer ellen. A támadó meg is egyezhet valamelyik szereplővel. Például lehet a támadó a kártya birtokosa (telefonkártya végtelenítése), vagy akár a szoftvergyártó is (kiskapuk elhelyezése), másrészt az is előfordulhat, hogy egy külső támadó a terminálokat veszi célba (hamis bankautomaták).

Jól látható tehát, hogy a programozható chipkártyák használatának ára van: új szereplők jelennek meg, ezzel új támadási lehetőségeket biztosítva. Ezt egyrészt figyelembe kell venni chipkártya alapú rendszerek tervezésénél, másrészt a chipkártyák által nyújtott kriptográfiai szolgáltatásokat is ennek fényében kell értékelni.

A fenti példák (telefonkártyák, bankkártyák stb.) mellett számos más rendszer is használ chipkártyákat, például a mobil távközlésben használatosak a SIM-kártyák. Ígéretes a chipkártyák internetes alkalmazásának lehetősége is, amelynél a következő funkciók körvonalazódnak:

- Login információk (felhasználói azonosító, jelszó) tárolása kártyán;
- Login dinamikus jelszó segítségével (pl challange and response elven);
- Erőforrás-hozzáférési jogosultságok kezelése;
- Elektronikus levelek titkosítása;
- Üzenetek hitelesítése és kriptográfiai ellenőrzőösszeggel való védelme;
- Digitális pénz az elektronikus kereskedelemben.

A csak Internet-hozzáférésre szolgáló, gyakran nyilvános számítógépek integritásának megóvása további kihívásokat jelent. Gondolunk itt első sorban az Internet kávézók és teleházak PC-ire és NC-ire (Network Computer). Ezen gépek állandó támadási lehetőségeknek vannak kitéve (amibe véletlen rongálás, például egy vírus letöltése is beletartozik), így semmiképp sem tekinthetők megbízhatónak. Ilyen esetekben olyan autentikációs protokoll alkalmazása látszik megfelelőnek, melynek során nemcsak a számítógép ellenőrzi a kártya tulajdonosának jogosultságait, hanem a gép is azonosítja magát a kártya felé. Sőt, elképzelhető az is, hogy a nyilvános gépek adminisztrátora bizonyos időközönként egy chipkártyával ellenőzi a gépek, illetve a rajtuk lévő szoftver integritását. Ebben az esetben a kártya tekinthető a megbízható félnek. Az integritás ellenőrzésére a számítógép memóriaképének egy egyirányú függvénnyel képzett tömörítvényét kell a kártyára tölteni, illetve ellenőrizni. (Egyirányú függvény alatt olyan függvényt értünk, melynek inverzét csak próbálgatással lehet meghatározni, ez pedig elég nagy adattér esetén reménytelenül hosszú időt vesz igénybe.)

Probléma marad azonban az, hogy a kártya – saját megjelenítő eszköz hiányában – csak a számítógépen keresztül tud jelzést adni a felhasználónak, illetve adminisztrátornak. Márpedig ha a számítógépet nem tekintjük megbízhatónak, a kapott eredményt sem tekinthetjük annak. Nagymértékben növelné a chipkártyák biztonságát, ha legalább egy LED segítségével jelezhetnék a felhasználónak, ha valami problémát észlelnek.

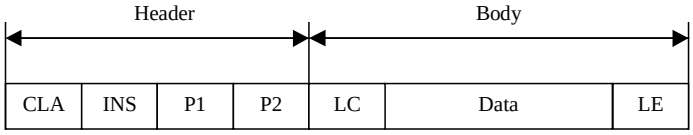
A chipkártyák programozása

A chipkártyák jellemző paraméterei (tipikusan 3-8 MHz órajelű 8 bites processzor, 8-32KB memória) az 1970-es évek számítógépeire emlékeztetnek. Ezt súlyosbítja az a körülmény, hogy a kártyák nem tudnak közvetlenül a külvilággal kapcsolatot tartani, csupán a kártyaolvasón keresztül. Ez nagyban nehezíti a hibakeresést. További probléma a rendszer lassúsága és a programírás folyamatának körülményessége (az elkészült programot gyakran több lépésben lehet csak a kártya által ismert formátumra hozni, ezután pedig fel kell tölteni a kártyára). A nehéz programozhatóságért cserébe egy kompakt, biztonságos kis számítógépet

kapunk.

További sajátságos probléma, hogy a felhasználó bármikor megszakíthatja a folyamatban lévő műveletet a kártyának az olvasóból történő kihúzásával, s így annak adatai inkonzisztens állapotban maradhatnak. Ez ellen olyan terminálokkal szokás védekezni, melyek a kártyát elnyelik, és csak a művelet végén adják vissza. Azonban ez a felhasználókból gyakran bizalmatlanságot vált ki. Gondos programozással megvalósítható tranzakció kezelés a kártyán (de a hardware támogatás jelentősen megkönnyíti a munkát). Ez az adatbáziskezelők világából kölcsönzött kifejezés azt jelenti, hogy a rendszer a kritikus műveletekről garantálja, hogy vagy teljesen befejeződnek, vagy pedig (ha valamiért ez nem lehetséges) a rendszer visszajut a tranzakció előtti állapotába.

A kártyával való kommunikáció alapegysége az APDU (Application Protocol Data Unit). A kártyával történő kommunikáció tipikusan félduplex (bár a szabvány lehetőséget biztosít duplex kommunikációra is): a terminál egy APDU-val elindít egy műveletet a kártyán, amely a művelet végeztével válaszol(hat). Az ISO 7816 szabvány leegyszerűsítve a 2. ábrán látható parancs-APDU struktúrát fekteti le.



2. ábra – A parancs APDU struktúra

Az egyes utasítások utasításosztályokba (CLA) sorolhatók. (pl: szabványos ISO utasítások, GSM utasítások stb), az utasítás (INS) kódja pedig a csoporton belül választja ki az utasítást, tehát kódjaik együtt azonosítják a végrehajtandó műveletet. P1 és P2 opcionális paraméter, ezeket követheti az adatok hosszát jelző byte, maga az adatmező, majd a várt válasz hossza. A kártya válasza (3. ábra) adatokat s két státusz szót tartalmaz.

3. ábra – A válasz APDU struktúrája

Microsoft Smart Card for Windows

A környezet bemutatása

A Microsoft meglehetősen későn lépett be a smart card piacra, de kártyájuk hardware tekintetében napjaink egyik legerősebbje. Ugyanakkor a Microsoft – kijelentéseik szerint – az általuk készített intelligens kártyákat is csak Windows NT-s hálozatok kiegészítő alkatrészeinek tekinti. A kártyák feladata tehát első sorban a hálózati belépés (logon) illetve hálózati erőforrás-hozzáférés kezelésének támogatása, másrészt pedig – az Outlook részeként – elektronikus levelek titkosítása, hitelesítése.

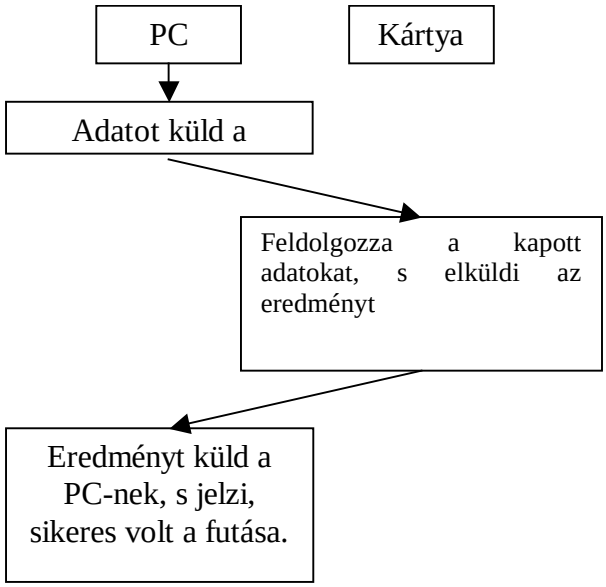
A WinCardon egy 8 bites RISC AVR MCU processzor van, s rendelkezik emellett 32 kilobyte Flash Program Memoryval az applikációk számára, 32 kilobyte EEPROMmal a tárolandó adatok és 1 kilobyte SRAM-

mal a változók, dinamikus adatok, stack stb. számára, továbbá egy ATMEL kriptográfiai műveleteket támogató koprocesszorral, mely megvalósítja a DES, triple-DES, RSA, SHA és CRC műveleteket.

A WinCardot a Microsoft úgy hirdeti, mint egy konzol nélküli Windows NT-t. Ez persze ebben a formában túlzás, de mégsem áll annyira messzire a valóságtól. A WinCard filerendszere erős védelmet biztosít, s rugalmasan beállítható rajta, mely felhasználó mely file-okon milyen műveleteket végezhet. Ezt hozzáférési listákkal (Access Control List) adhatjuk meg, amelyek a kártya /s/a/ alkönyvtárában helyezkednek el. Ebben a könyvtárban minden file egy-egy hozzáférési lista, mely megadja, milyen műveleteket milyen felhasználók jogosultak végrehajtani.

A kártya több felhasználó kezelésére képes. Maximum 127 különböző "ismert felet" (Known Principal) képes elkülöníteni. A felhasználókat jelentő file-ok tartalmazzák, hogy a felhasználó azonosítása milyen módon történhet. Ez lehet PIN-kód vagy DES alapú, Challange and Response jellegű azonosítási protokoll.

A kártya képes alkalmazások egymás utáni futtatására, de a párhuzamos futtatást nem teszi lehetővé. A kártya filerendszerének elkészítése előtt meg kell tervezni, milyen alkalmazások kerülnek majd rá, s létre kell hozni a hozzájuk szükséges felhasználókat, csoportokat, hozzáférési listákat. Egy kártya tehát több előre eltervezett funkciót tölthet be (lehet személyi igazolvány, villamosbérlet, telefonkártya és hitelkártya is egyben), de nem lehet egy villamosbérletnek azt mondani, hogy mostantól legyen bankkártya is. A kártyán futó és a kártyát kezelő alkalmazás sajnos nem tekinthető két párhuzamosan futó processznek. Így a működés sokkal inkább függvényhívás-szerű (4. ábra).



4. ábra – PC-kártya kommunikáció

A Microsoft ezzel a kártyával a Java Cardokat szeretné legyőzni. A Java Cardban nemcsak az a pozitívum, hogy a Java nyelv platformfüggetlen, hanem az is, hogy a kártyára való fejlesztéshez nem szükséges egy új programozási nyelvet megtanulni. Sőt, a fejlesztés magas szintű nyelven történhet.

A Microsoft ebben sem akart elmaradni. A terv az volt, hogy a fejlesztés Visual C-ben történne, de végül a C-t túl komplex nyelvnek találták és a Visual Basicet választották. (Ez abból látszik, hogy a

dokumentáció jelentős része C nyelvű példaprogramokat mutat be, s C programozásra ösztönzi az olvasót.) Így lett a Microsoft smartcardos fejlesztőrendszere a Visual Basic 6.0 egy PlugIn-je. Természetesen a nyelv PC-s változata túl komplex volt ahhoz, hogy minden jellegzetességét, erősségét meg lehessen tanítani a kártyának. Nemcsak arról van szó, hogy a grafikai utasításokat vették ki a kártyán futó basicből, hanem magát a nyelvet is átalakították.

Nem használhatunk például string változókat, lebegőpontos aritmetikát, változó méretű tömböket. Komoly hiányosság, hogy nem lehetséges a kártyán semmilyen hibakezelés. Ahhoz, hogy a WinCardra komoly, az üzleti életben is használható alkalmazást lehessen fejleszteni, ezt a Microsoftnak mindenképpen korrigálni kell.

Nem használhatjuk a Visual Basic könyvtári függvényeinek nagy részét sem. Így a nyelv, amin fejlesztünk, csupán annyiban különbözik egy assemblytől, hogy nem látjuk a kártya regisztereit, s használhatjuk a basic vezérlési szerkezeteit. Támaszkodhatunk viszont a kártya filerendszerére s kriptográfiai koporcesszorára, s ennek kezelésére külön könyvtári függvények vannak, melyek a következő csoportokra oszthatók: felhasználók azonosítása, kriptográfia, file kezelés, kommunikáció a PC-vel.

Jó tulajdonsága a rendszernek, hogy a kártyát használó alkalmazás nem a kártyaolvasóval kommunikál, hanem egy NT service-szel, a Smartcard base components-szel. Így a programozó úgy fejleszthet WinCardra programot, hogy nem kell tudnia, a felhasználó(k)nak majd milyen olvasója lesz. A SmartCard Base Components kapcsolódik majd az olvasó driveréhez, s kezeli azt.

Sebességmérés

Egyik alkalmazásunkkal a kártya processzorának számítási sebességét próbáltuk mérni. Erre egy külön applikációt készítettünk, mely az input hosszától exponenciálisan függő számú lépésre kényszeríti a processzort. N hosszú input esetében N db byte fogadását és 10^N db inkrementálás (++) műveletet kellett elvégeznie. A mérés végrehajtására szolgáló szubrutint az 1. programlista, a kapott eredményeket az 1. táblázat tartalmazza.

```
Sub WinCardTest()
    Dim l As Long
    Dim i As Long
    i = 0

    l = 1   ' a hatványozást a kártya processzora nem támogatja
    For i = 1 To bemenetHossz
        l = l * 10
        Call ScwGetCommByte ' be kell olvasni a teljes bemenetet, hogy
                            ' ne legyen hiba
    Next i

    While i < l
        i = i + 1       ' ez itt a kártya leterhelése
    Wend
    ScwSendCommByte NO_ERROR    ' minden rendben van, kiléphetünk
End Sub
```

1. programlista – A WinCard sebességének mérésére használt program

Feltételezésünk szerint az időadatok alapvetően két komponensből tevődnek össze: egyrészt magából a számításhoz szükséges időből, másrészt a kártya/gép kommunikáció, kártya resetelés stb.-re fordított időből

(továbbiakban: overhead). A táblázatban látható adatok alapján egy inkrementálás elvégzése mintegy 80 millisekundumot vesz igénybe, az overhead pedig kb. 2 másodperc.

10 db ++ művelet végrehajtása	3 másodperc
100 db ++ művelet végrehajtása	8 másodperc
1000 db ++ művelet végrehajtása	80 másodperc
10000 db ++ művelet végrehajtása	780 másodperc

1. táblázat – WinCard számítási sebességének teszteredményei

Az első két eredményen még az látszik, hogy nem a műveletek száma dominál, hanem a PC-kártya kommunikáció. A továbbiak viszont már nagyjából reális képet adnak a kártya processzorának sebességéről. Látható, hogy a kártya sebességben messze elmarad a PC mögött. Csakis azokat a műveleteket célszerű tehát rajta elvégezni, amelyek biztonsági szempontok miatt nem kerülhetnek ki a kártya védett környezetéből.

A DES csomag

Az általunk kártyába plántált DES csomag nem más, mint egy futtatható állomány, amely képes az inputját egy beépített titkos kulcs segítségével titkosítani, vagy a titkosított adatból az eredeti adatot visszaállítani. E program a következő utasításokat képes végrehajtani:

- Bemenet titkosítása: Az "e" parancs hatására a következő 8 byte-ot a kártya titkosítja, s kimenetként ezt küldi ki az outputra.
- Nyílt szöveg visszaállítása: az előző művelet inverze. A "d" parancs hatására a következő 8 byte bemenetből a kártya visszaállítja a nyílt szöveget.
- Kulcs betöltése a kártyába: ennek a műveletnek a segítségével lehet megváltoztatni a kártyában tárolt kulcsot egy, a felhasználó által meghatározott értékre. Input: a "l" parancs. Output: a NO_ERROR üzenet.
- Kulcs generálása: szintén a kulcs megváltoztatására szolgál, de itt véletlenszerűen generál egy DES kulcsot. Erre a kártya kripto-koprocesszorának véletlenszám generátor funkcióját használjuk. A DES kulcs ezután 56 db random bit lesz. Input: az "r" parancs. Output: a NO_ERROR üzenet.

DES titkosító rutin természetesen futhatna a PC-n is. Sőt, akkor sokkal gyorsabb is lehetne. Miért jó, hogy kártyán valósítottuk meg? Ahogy végignézzük a fenti négy parancsot, rögtön feltűnik, hogy "hiányzik" közülük egy: a kártyán lévő kulcs kiolvasása a kártyából. Így a kártya biztosítja azt, hogy a rajta lévő adatokhoz csupán az előre meghatározott műveleteken keresztül lehet hozzáférni. Amennyiben ez tényleg így van, akkor – mivel nem definiáltunk olyan műveletet, hogy a kulcs kiolvasása – senki nem fér hozzá a kulcshoz, akármilyen jelszavakat szerez is meg.

A kulcs mindenki elől biztonságban van. Nemcsak a támadó képtelen hozzáférni a kulcshoz, hanem a kártya gazdája is. Hiába van a zsebében a kártya, nem képes kinyerni a titkos kulcsot belőle, csak használni tudja

azt. Mivel a kártya nem hajlandó kiadni magából a kulcsot, az egyetlen esély annak megszerzésére a DES feltörése. Ezzel a rendszerrel célunk ilyen szintű biztonság létrehozása volt.
Akkor a legtitkosabb valami, ha senki nem ismeri. Az sem, aki beletöltötte a kártyába. Ennek módszere a véletlen kulcs generálása. Tehát arra használjuk fel a kártya processzorát, hogy saját maga állítson elő kulcsot, s azt ne adja ki senkinek. Így egyetlen entitásnak – még a kártya kibocsátójának – sem lehetnek ismeretei a kulcsot illetően. Így e rendszer – a DES ereje és feltételezéseink alapján – biztonságosnak nevezhető.

A Java Card specifikáció

A Sun-féle általános Java Card környezet

A Sun Microsystems által kifejlesztett Java Card specifikáció fő célja az, hogy csökkentse a programozható kártyákba való befektetés kockázatát, s ezzel elősegítse a technológia fejlődését.

Amíg a chipkártyákat pusztán assembly nyelven lehet programozni, a fejlesztő cégek kiszolgáltatottá válnak a kártyagyártóval szemben, mert minden chipkártya-típusnak különböző assembly nyelve lehet. Ha egy alkalmazást át kell vinni egyik kártyafajtáról egy másikra – vagy ugyanannak a típusnak egy másik altípusára – a teljes alkalmazást át kell írni. Ennek orvoslására született a Java Card programozási környezet (API).

A Java nyelv kifejlesztésének egyik célja beágyazott rendszerekben való alkalmazás, a másik pedig a weblapokon megjelenő programocskák (appletek) készítése volt, amelyek intelligens funkciókat vittek az addig javarészt statikus weblapokba. Ehhez képest gyökeresen más kihívást támasztott a Java Cardok esete. Míg egy Web-böngészőt futtató PC gyors processzorral rendelkezik, amely képes lehet appleteket hatékonyan futtatni, egy chipkártya néhány Mhz-en járó 8 bites processzora és néhány kilobyte memóriája merőben más sebességviszonyokat jelentenek. Így a Java Cardok nyelve, bár szintaktikailag Java, valójában sokkal közelebb áll a C-hez.

Egy Java Card applet forráskódját nézegetve feltűnik, hogy a kód alacsony szintű. Kevés az objektum-létrehozás, s rengeteg a bitművelet és shiftelés. Látszik a kódon, hogy a programozónak minden órajel ütemért meg kell küzdeni. A Java nyelven nem ilyen filozófiával szokás programozni.

A másik probléma az, hogy a kártyán lévő Java Virtuális Gép nem támogatja a szemétgyűjtést (garbage collection), így ha létrehozunk egy objektumot, azt nem tudjuk elpusztítani. Az applet és az általa létrehozott objektumok élete addig tart, míg az alkalmazást le nem töröljük a kártyáról. A fentiekből – ha a program robosztusságát mindenek felettinek tartjuk, ami például egy digitális pénztárca alkalmazásnál igen fontos szempont – logikusan adódó kötöttség, hogy a programozó köteles minden objektumot a konstruktorban létrehozni, s így oldja meg azt, hogy a memória el ne fogyjon menet közben.

Minden kártyás alkalmazás együttműködő objektumok egy nem üres halmaza, melyeknek egyike a javacard.framework.Applet leszármazottja. Ezen objektum egyes metódusai szolgálnak az alkalmazás belépési pontjaként. A működés módja lényegében a következő: miután az applet feltöltésre kerül, a rendszer meghívja annak *install()* metódusát. Az *install()* általában létrehozza az applet egy példányát, s meghívja a *register()* metódust. Futtatáshoz ki kell választani az adott appletet a megfelelő APDU-val, s a rendszer ekkor meghívja

annak *select()* metódusát. A Java Card appletnek nincs sokféle eseménykezelője, hiszen csak egyféle "közönséges" esemény történhet vele: a futtatás. Ilyenkor a *process(APDU)* metódus hívódik meg. Ez megkapja paraméterként a teljes APDU-t, s feldolgozhatja azt. Miután befejezzük a munkát az applettel, a rendszer meghívja annak *deselect()* metódusát.

A Java Card specifikáció támogatja a tranzakciókezelést. A programozó megadhat olyan blokkot, amely kívülről atominak minősül. Tranzakció közben a kiírandó adatok egy pufferben tárolódnak, s csak akkor kerülnek ténylegesen kiírásra, ha a tranzakció véget ér. Sajnos minden puffer mérete véges, s ha betelik, akkor a tranzakciókezelés nem működik. Ennek elkerülésére gondos programozás szükséges.

Hogy történik egy Java Card fejlesztés? Egy Java Card program byte kódja ingyenes eszközökkel (pl. JDK 1.1.7) elkészíthető. Így létrehozhatunk osztályokat, melyek egyike a javacard.framework.Applet leszármazottja. Az appletet konvertálni kell egy – a kártyagyártótól származó – konvertáló programmal. Ez kiszűri a nem Java Card konform elemeket, s az adott kártyatípus gépi kódjára konvertálja a programot. Ez egy az egyben feltölthető a Java Cardra, majd a megfelelő APDU segítségével meg kell hívni az *install()* metódusát. Ezután az applet működőképes.

A konkrét Bull környezet

A Bull Odyssey 1.2 a Java Card specifikáció egy megvalósítása. Sajnos a fejlesztőkörnyezet annyiban nem tesz eleget a specifikációnak, hogy kriptográfiai funkciókat nem tartalmaz. Az általunk vizsgált kártya 8 kilobyte EEPROM-mal rendelkezik. Használhatunk emellett a lokális változók átmeneti tárolására 512 byte RAM-ot is. Megfelel az ISO 7816 szabványnak, s így elfogadja a szabványos APDU-kat.

A Bull fejlesztőeszköze, az OdysseyLab néhány utility mellett három programból áll. A konvertáló program platform-független byte kódról a kártya gépi kódjára fordít. A feltöltő program menedzseli a kártyán lévő appleteket. Ennek segítségével lehet letörölni a kártyát, illetve feltölteni egy új appletet. Végül a harmadik programnak már nem a fejlesztés, hanem a tesztelés fázisában van szerepe: egy egyszerű terminálfelületet biztosít a kártyával való kommunikációra. Lehetőség van scriptek, függvények írására, s a nyelve támogat egyszerű elágazási- illetve ciklusműveleteket is.

E mellett tartalmaz még a Bull CD-je egy JDK 1.1.7-et, dokumentációt, s néhány alkalmazást, például a kártyaolvasó detektálására. Nem tartalmaz viszont semmilyen programot vagy programozási könyvtárat, amely a PC oldali feljesztést lehetővé tenné.

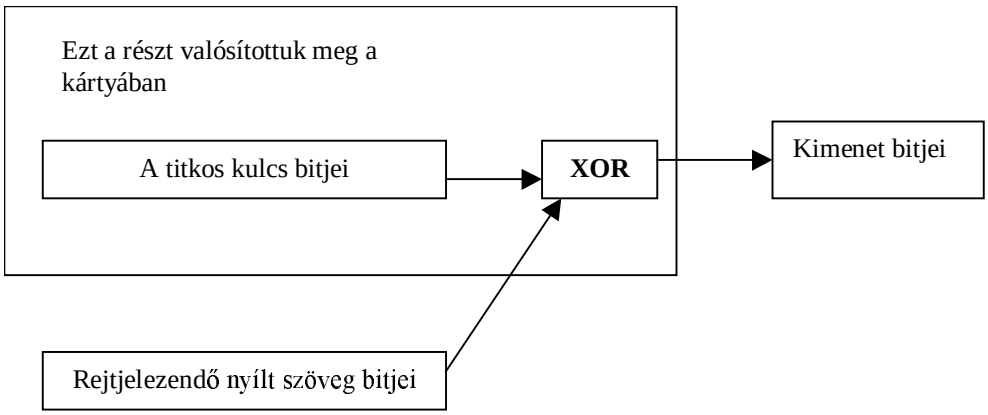
Példa alkalmazás bemutatása: One Time Pad

Lévén, hogy a Java Card nem rendelkezik kriptográfiai koprocesszorral, a kriptográfiai műveleteket tisztán szoftveres úton kell megvalósítani. A Bulltől nem kaptunk semmilyen kriptográfiai könyvtárat vagy Java package-t, így arra kényszerültünk, hogy saját magunk valósítsuk meg az adatbiztonsági műveleteket. Az ilyen irányú fejlesztés első lépcsője a One Time Pad.

A One Time Pad nem tartozik sem a legújabb, sem a legbonyolultabb titkosítási megoldások közé. Vernam francia katonatiszt találta fel az első világháború során, s később Shannon bizonyította be jelentőségét. Rendkívül egyszerű szerkezete, s könnyű megvalósíthatósága ellenére, ez a – jelenleg ismert – egyetlen feltétel

nélkül biztonságos titkosító. Kizárólag hallgatózó (tehát nem aktív) támadóval szemben tökéletes védelmet nyújt. Hátránya viszont, hogy n bit rejtjelezéséhez n bit hosszú kulcsra van szükség. Egy kulcsbitet nem lehet kétszer felhasználni, így kizárólag "nagy méretű" kulcsok esetében értelmes.

Mûködése egyszerû. Vegyünk n bit rejtjelezendõ nyílt szöveget, s vegyünk n véletlen bitet. Ez utóbbi lesz a kulcs. A titkosítás úgy történik, hogy képezzük a nyílt szövegnek a kulccsal való modulo 2-es összegét (5. ábra)



5. ábra – A One Time Pad működése

Mivel a nyílt és a rejtett szöveg bitjei független valószínûségi változóknak tekinthetõek, bizonyítható, hogy kölcsönös információjuk 0, s így a rendszer statisztikai módszerekkel feltörhetetlen. Hasonlóan eredménytelen a brute force támadás is. Amennyiben az összes lehetséges kulccsal elvégezzük az inverz transzformációt, az összes lehetséges üzenetet kapjuk meg. Így egyetlen lépést sem tettünk elõre.

Chipkártya esetén nem tárolhatunk akárhány kulcsbitet. A kártyán 7040 byte hely van, s ebbõl az alkalmazásunk elfoglal 500 byte-ot. Ha 6000 byte hosszú kulcsot tárolunk, a kártyára már nem fér más. Mire használhatjuk ezt az appletet? Először is készíteni kell a “kulcs felvitele” APDU-val két ugyanolyan kulccsal rendelkező kártyát. Ezután a két kártya birtokosa egymással titkosan kommunikálhat. Természetesen az üzenetek összhossza nem lehet nagyobb a kártyán lévő kulcs méreténél. Amennyiben a One Time Padet titkos kulcsok (pl. 8 byte hosszú DES kulcsok) kicserélésére használjuk, a 6000 byte 750 kulcscserére elegendő. Ha hetente cserélünk kulcsot, 14 évig használhatjuk a kártyákat.

A One Time Pad működéséből adódik, hogy a kódoló és dekódoló algoritmus megegyezik. Szintén megegyezik a kódolandó és a kódolt adat hossza. Az eredmény kiolvasása a konkrét Bull kártya egy jellegzetességéből adódik. Az általunk megvalósított One Time Pad a 2. táblázatban bemutatott APDU-kkal mûködik. Példa: egy 0xE408000003112233 APDU segítségével titkosíthatjuk a 0x11, 0x22 és 0x33 byte-okat. Az eredményt pedig a 0xE4C0000003 APDU-val olvashatjuk ki a kártyából.

Miért jó, hogy a One Time Pad chipkártyán foglal helyet?

- Védhetjük vele a kulcsot. Ha a kártyát fizikailag el nem lopják, a kulcs védve marad, mert a smart cardot nem lehet lemásolni;
- A kulcs nem kerülhet ki a kártyából. Ki lehet ugyan olvasni úgy, hogy kódoljuk a 0x00 byte-ot,

de akkor is csak olyan kulcsbiteket ismerhetünk meg, amelyeket már felhasználtunk;

- A felhasznált kulcsbitek megsemmisítését a kártya garantálja;
- Így a már továbbított üzeneteket nem lehet megfejteni a kulcs későbbi megismerésével.

Tehát a chipkártya segítségével egy biztonságos környezetet adtunk a One Time Padnek, amellyel garantáljuk a kulcs megismételhetetlenségét, lemásolhatatlanságát, s felhasználás utáni megsemmisülését.

Sebességmérés

A WinCardhoz hasonló sebességmérésre itt is sor került. A kártyán lévő appletet hívó APDU parancskódja szolgált az N bemenetként; a program ennek megfelelően 10^N dekrementálás (--) műveletet végzett (2. programlista).

```
short N = buffer[ISO.OFFSET_INS];
int szam = 1;
for( ; N>0 ; N-- )
       szam *= 10;           //hatványozás helyett
for( ; szam>0; szam-- );             //a tulajdonképpeni terhelés
```

2. programlista – Az Odyssey kártya sebességének mérésére használt program

Azonban, mint az már korábban is feltűnt, a Bull kártyája sokkal gyorsabb, mint a Microsoft-é, így a mérés is teljesen más tartományban zajlott (3. táblázat).

Műveletek száma	Szükséges idő
100.000	7 sec
1.000.000	69 sec
10.000.000	690 sec

3.táblázat – Az Odyssey kártya tesztjének sebességeredményei

Jól látható, hogy egy művelet elvégzéséhez körülbelül 0.07 msec-ra volt szükség. Az overhead alig érzékelhető, így ennek mérésére más módszert alkalmaztunk: egy (a PC-n futó) ciklusban 1000-szer hívtuk meg programunkat N=0 bemenettel. Ez 58 másodpercet vett igénybe. Látható, hogy ez nagyságrendekkel nagyobb, mint ugyanennyi inkrementálás elvégzése, vagyis ezzel a módszerrel lényegében tényleg az overheadet mértük. Ezek szerint az overhead kb. 60 msec.

A két technológia összehasonlítása

Az összehasonlítás tárgyát nem pusztán két termék képezte. A Microsoft kártyáját, s a hozzá tartozó fejlesztőeszközt nevezhetjük terméknek, bár még nincsen kint a piacon. A Bull kártya esetében viszont külön kell választanunk a specifikációt a megvalósítástól. A WinCard oldalán egy homogén rendszer áll, melynek belsejébe nem láthatunk bele. Ez nemcsak a fejlesztés áttekintését teszi nehézzé, de a fejlesztőeszköz esetleges hibáinak felderítését is nehezíti. A Bull kártya esetében viszont jól látható a rendszer rétegszerkezete. (4. táblázat)

Nézőpont	Microsoft	Bull – Sun
Hardware	A Microsoft Smart Card for Windows Professional kártyája	A Bull Odyssey I kártya 1.2 –es változata
API	A Microsoft Smart Card for Windows Professional (1999. májusi béta verzió), mint programozási környezet	A Java Card API 2.1-es specifikációja
Fejlesztőeszköz	A Microsoft Visual Basic 6.0, s hozzá a Smart Card for Windows Plug-In (1999. májusi béta verzió)	Egy egyszerű text editor, a JDK 1.1.7 és a Bull OdysseyLab
PC oldali szoftverfejlesztés támogatása	A Visual basic 6.0 PC oldali fejlesztést támogató része, programkönyvtárak	Semmi

4. táblázat – Összehasonlítási szempontok

A Microsoft kártya határozottan fejlettebb, erősebb, mint a Bull Odyssey I 1.2. Míg az utóbbi 8 kilobyte EEPROM-mal rendelkezik, s 512 byte RAM-mal, a Microsoft kártyája 32 kilobyte EEPROM-ot, 32 kilobyte FLASH-ROM-t, s 1 kilobyte RAM-ot tartalmaz. Rendelkezik továbbá egy kriptográfiai koprocesszorral, s így hardware támogatással számolhat DES-t és RSA-t (Az RSA funkciót nem sikerült működésre bírnunk). A Bull-kártya nemcsak kripto-koprocesszort nem tartalmaz, hanem még a Java Card specifikációban leírt kriptográfiai package-eket sem, s így a titkosító műveleteket tisztán szoftveres úton kell implementálni. Szoftveres algoritmus egy célhardware-rel szemben feltétlenül alul marad a versenyben. Ugyanakkor a Microsoft kártya – számunkra érthetetlen okból – rendelkezik egy jókora késleltetéssel, amely minden egyes applikáció-futtatáskor jelentkezik. A Bull kártya válasza rövid kérések esetén azonnali.

Azzal együtt, hogy a Microsoft kártyáját hardware paramétereit tekintve sokkal gyorsabbnak hihetnénk, méréseink tanúsága szerint messze nem ez a helyzet. A használat során a Microsoft-kártyát mindig is lomhának éreztük; különösen a nagy overhead volt zavaró. A Bull-kártyánál ilyen problémát nem érzékeltünk. A mérések ezt be is igazolták: a Bull kártyája azonos idő alatt kb. 1000-szer annyi műveletet tud végezni, mint a

Microsoft-é, aminek ráadásul az overhead-je is körülbelül 30-szoros. Természetesen ezek az eredmények csak az általunk végzett, igen egyszerű mérésekre vonatkoznak, de úgy érezzük, e mérések éppen egyszerűségüknél fogva igen jellemző adatokat szolgáltatnak.

A hatalmas különbség okát nehéz megállapítani, hiszen nem tudjuk, pontosan mi történik a kártyákon és az egyes fejlesztőkörnyezetek belsejében. Lehetséges magyarázatként szóba jöhet a processzorok eltérő aritmetikai funkcionalitása, a Visual Basic fordító szuboptimalitása stb. Az overheadet indokolhatja az a körülmény, hogy a Microsoft-kártyát Windows NT alatt használtuk, ahol a soros porthoz való hozzáférés egy service-en keresztül történik, ami esetleg igen lassú lehet.

A Microsoft komplett fejlesztőeszközt kínál termékéhez. A smart card toolkitje a Visual Basic 6.0 egy PlugInje, amely szervesen kapcsolódik a Visual Basichez. A fejlesztésnek nemcsak az a része történik Visual Basic alatt, amely a program megírását tartalmazza, de programunkat Visual Basicből lehet a kártyára feltölteni s futtatni is. Így a kártyán futó s a kártyát kezelő alkalmazás egyazon környezetben készül el. Ugyanezen környezetben, egy emulátor segítségével ki is próbálhatjuk a kártyás alkalmazást, anélkül, hogy a kártyára le kellene tölteni. Sajnos az emuláció nem biztosít egy a valóshoz hasonló környezetet, így a fejlesztésnek feltétlenül komoly részét képezi a kártyán való tesztelés is. Szintén ugyanebből a fejlesztőkörnyezetből feltölthetjük az applikációt a kártyára, s futtathatjuk is. A Visual Basic – WinCard rendszer a fejlesztés közel minden fázisát lefedi. Ez a nagyfokú integráltság persze bizonyos esetekben hátrányos is lehet.

A Bull a kártyájához nem ad teljeskörű fejlesztőeszközt. A forráskód tetszőleges editorral elkészíthető. A fordításhoz a Bull CD-je pusztán JDK 1.1.7-t tartalmaz (a dokumentáció szerint Visual Caféval is működik), s e mellett foglal helyet az OdysseyLab nevezetű utilitycsomag. Ez tartalmaz egy konvertáló s egy feltöltő programot, valamint egy SmartLab nevezetű eszközt, amely a kártyával való kommunikációhoz biztosít egy környezetet. Ezt főként tesztelésre használhatjuk. Tehát a Bull egy kisebb programokból álló eszközcsomagot ad OdysseyLab néven, nem pedig egy fejlesztőeszközt. Szintén fájó, hogy nem támogatja az OdysseyLab a PC-oldali fejlesztést. Nekünk egy másik Bull smart cardos fejlesztőeszköz segítségével (amely a TB családba tartozó, régebbi típusú, ún. generikus kártyákhoz készült) sikerült a kártyaolvasót meghajtani. Emulátort a Bull egyáltalán nem biztosít kártyájához, az emulációt még a byte kódon lehet elvégezni egy Sun-féle (ingyenesen letölthető) emulátorral. A specifikációnak ez is csak hellyel-közzel felel meg, s ez sem támogatja a specifikáció kriptográfiai függvényeit.

A Microsoft kártyája a Visual Basic nyelvet használja, amely egyszerű, de ugyanakkor hiányzik belőle mindennemű egységes koncepció. A nyelv fő erősségét, az objektum-orientáltságot teljesen eltávolították, továbbá a nyelv egyes elemeit megcsonkították, átalakították. Nem tiszta, hol húzódik meg az a határ, amely elválasztja a kártyán is alkalmazható hívásokat, nyelvi elemeket a csak PC-n alkalmazhatóaktól. Egyes változtatások a nyelv lényegét is érintik. A WinCard basic nyelve mindössze annyiban hasonlít Visual Basicre, hogy hasonlóképpen strukturálatlan.

A Java nyelv ezzel szemben egy jól kidolgozott, áttekinthető, átfogó koncepcióval rendelkező eszköz, s a Java Cardok nyelvének megkonstruálásakor a tervezők nem módosítottak semmit a Javában, csupán elvettek belőle. Így a Java Cardok nyelve a Java egy részhalmaza. Egy Java Card applet szintaktikailag helyes Java programot jelent, igaz, logikailag nagyban eltér a Javától. Egy – a C nyelvet, s filozófiát ismerő – átlagos Java

programozó számára nem jelenthet gondot a Java Cardra való fejlesztés.

A Microsoft dokumentáltsága meglehetősen szegényes. Bizonyos függvényekről nincsen dokumentáció, s ha van, akkor a dokumentációban nem annyi paraméterük van, mint a valóságban. Látszik, hogy eredetileg nem Visual Basicben történt volna a fejlesztés, hanem Visual C-ben. A dokumentáció jelentős részét nem adaptálták basic-hez.

A Java Card programozási környezet megfelelő mennyiségű és minőségű dokumentációval rendelkezik, de a Bull kártya, tehát a konkrét implementáció, dokumentáltsága igen sok kívánnivalót hagy maga után. A gyártó itt sem közölte a kártya konkrét technikai paramétereit, a szükséges APDUkat, a file rendszer pontos leírását. A minőség – apróbb hibáktól eltekintve – megfelelő, gondok inkább a mennyiséggel vannak, illetve azzal, hogy nem térnek ki minden területre.

A Bull kártyája megfelel a Java Card specifikációnak, s az általunk próbált műveleteket ismerte, a végrehajtás módja determinisztikusnak bizonyult. A WinCard esetében viszont rengeteg megmagyarázhatatlan eseménnyel találtuk szembe magunkat. Amellett, hogy a dokumentáció meglehetősen szegényes, s számos lényegi hibát tartalmaz, a kártyán futó programmal kapcsolatban úgy éreztük, nem tartja be pontosan a Neumann-elvet, s az utasítások nem egymás után követik egymást.

Összegzés

Két, egymástól nagyon távol álló programozható chipkártyát vetettünk össze. Mások az erőforrásaik, más az architektúrájuk, más a filozófiájuk, más nyelven kell őket programozni, s az ISO 7816 szabvány által előírt jellegzetességeken túl egyáltalán nem kompatibilisek egymással.

A Microsoft-kártya paraméterei erősebbek. Több memóriával rendelkezik, van kripto-koprocesszora, s a hozzá tartozó fejlesztőeszköz a fejlesztés teljes spektrumát lefedi. Az erősebb hardware ellenére sebességben meg sem közelíti vetélytársát. A fejlesztőeszköz pedig még nem tisztult le kellőképpen, s a következő verzióig teljes átdolgozásra került, s az új béta verzió nem kompatibilis az eredetivel. A Microsoft Smart Card for Windows Professional 1999. májusi béta verziója nem alkalmas komoly fejlesztésre.

A Bull Odyssey ezzel ellentétben egy letisztult piaci termék, a nyilvános Java Card 2.1 specifikáció egy példánya. Java Cardokat nemcsak a Bull készít (ilyen például a De La Rue cég GalactIC nevezetű modellje is), így aki rájuk fejleszt, nem válik kiszolgáltatottá a kártyagyártó céggel szemben.

Az Odyssey mögött ott áll a Java Card specifikáció, egy jól átgondolt, logikailag tiszta, robosztus fogalomrendszer, amely igaz, hogy változik, de lényeget érintő változások nem következnek be rajta. Annak, aki ma programozható smart cardokkal foglalkozik, tudomásul kell vennie, hogy a techológia még gyermekcipőben jár, s még a fejlesztők sincsenek teljesen tisztában a lehetőségekkel. Ennek ellenére hisszük, hogy a közeli jövőben a programozható chipkártyák jelentősége megnő, és a technológia letisztulásával és az árak a nagy példányszám következtében történő lecsökkenésével kiszorítják a hagyományos kártyákat.

Irodalomjegyzék

Általános kriptográfiai munkák:
D.W. Davies – W. L. Price: Security for Computer Networks. John Wiley & Sons, 1992.
Bruce Schneier: Applied Cryptography. John Wiley & Sons, 1996.
Gustavus J. Simmons (Szerk.): Contemporary Cryptology. IEEE Press, 1992.
Györfi-Vajda: A hibajavító kódolás és a nyilványos kulcsú titkosítás elemei. Budapest, 1991.

Smartcardokkal kapcsolatos munkák:
Berta István Zsolt – Mann Zoltán Ádám: A hitelesség biztosításának lehetőségei intelligens smart card segítségével (TDK dolgozat)
W. Rankl – W. Effing: Smart Card Handbook. John Wiley & Sons, 1997.
Bruce Schneier – Adam Shostack: Breaking Up Is Hard To Do: MOdeling Security Threats for Smart Cards
J. L. Zoreda – J. M. Oton: Smart Cards. Artech House, 1994.

A Microsoft fejlesztőkörnyezettel kapcsolatos információk forrása:
Windows Smart Card Development Kit Help
http://www.microsoft.com/security/tech/smartcards

Webes hivatkozások
Bull: http://www.cp8.bull.net
Java Card: http://java.sun.com/products/javacard/htmldoc
Java nyelv specifikációja http://java.sun.com/docs/books/jls/html
PC/SC Workgroup: http://www.smartcardsys.com