
Chipkártyákról általában


A chipkártyák számos előnytel rendelkeznek a mágneskártyákkal szemben. Egyrészre kevésbé érzékenyk 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 műveletek 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 kivártól 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 elegendő a gyártási folyamat után specializálni, tehát kevesebben kártyatípusra van szükség, amelyek viszont igazán nagyobb példányszámban gyártathatók. Ez egyrészt a smart card technológia árának csökkentéséhez 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 lehetőségeit, hogy személyenként egyetlen kártya ellásson minden alapszintű, hitelesítési, adattárolási funkcióit. 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 lehetőségét biztosítanak fejlett kriptográfiai algoritmusok és protokollok megvalósítására, amelyek nagymértékben növelhetik a rendszer biztonságát.

Fontos azonban említeni a programozható chipkártyák korlátairól is. A kártya fizikai biztonsága érdekében a teljes hardvert egyetlen chipben kell megvalósítani, hiszen különben a kártya buszrendszerére kapcsolódva értekes információkat lehetne nyerni. Így a processzor és a memória összeltismeretével korlátolható, hogy a termelőhő lővelétele nincs megoldva. A másik kritikus faktor a kártya előállítási költsége. Ez már úgy 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 chipkártyáról lesz szó, megemlíthetjük még az optikai kártyákat is, melyek főleg az egészügyi 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 információsértéket tesz lehetővé. Emellett gyakran a kártya egész felülete információtaolásra 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.


**Chipkártya alapú rendszerek**

A chipkártyák rendszerint egy nagyobb rendszer egy komponensét alkotják. Ennek megfelelően a chipkártyák működését, lehetőségeit is ebben az összedefiniáltan érdemes vizsgálni. (1. ábra)
Egy chipkártya alapú rendszer típusos 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 kártyaelfoglóhelyek (Electronic Point of Sale) használatára. Tovább bonyolíthatja a helyzetet, hogy a kártya birtokosa nem feltétlenül egyezik meg a kártyán levő adatok tulajdonosával (data owner); jó példa erre a hitelkártya, ahol a kártyán levő 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 illetékében használata, közvetlen anyagi haszon, titkos információk megszerzése stb.) Ha csupán egyetlen szereplő volna, akkor kommunikáció hián a rendszer a lehető legbiztonságosabb lenne. Azonban, amint áttérünk egy kétszereplős modellre, azonnal megjelenik a lehetőség a két fél közti csatorna megújulására, 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 komponense, 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 (lamus 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 más számos más rendszer is használ chipkártyákat, például a mobil távközlésben használatos SIM-kártyák. Ígéretes a chipkártyák internetes alkalmazásának lehetősége is, amelyenlél a következőkön körvonalazódnak:

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

A csak Internet-hozzáférésre szolgáló, gyakran nyilvános számítógépek integritásának megóvása további kihívásokat jelent. Gondolkunk ilyen esetekben az Internet kávézók és teleházak PC-ire és NC-ire (Network Computer). Ezen gépek állandó támadási lehetőségeknek vanak kitűve (amibe véletlen rongálás, például egy vírus letöltése is bele tartozik), így semmiképp sem tekinthetők megbizhatóak. Ilyen esetekben olyan autentikációs protokoll alkalmazása létszik megfelelőnek, melynek során nemcsak a számítógép ellenőrzi a kártya tulajdonosának jogosultságait, hanem a gép is azonosítja magát a kártya felé. Sok elképzelhető az is, hogy a nyilvános gépek adminisztrátorai bizonyos idejökként egy chipkártyával ellenőrizik a gépeket, illetve a rajtuk levő softver integritását. Ebben az esetben a kártya tekinthető a megbizható félnek. Az integratív ellenőrzésére a számítógép memóriaképén egy egyirányú függvényel képzett tömörfényt kell a kártyára tölteni, illetve ellenőrizni. (Egyirányú függvény alatt olyan függvényt értünk, melynek inverzéért csak próbálhatóval lehet megdöntölni, ez pedig elég nagy adott tartományt reménytelenül hosszú idejű vesz igénybe.)

Probléma marad azonban az, hogy a kártya – saját megjelenítő eszköz hányában – csak a számítógépen keresztül tud jövőt adni a felhasználónak, illetve adminisztrátornak. Márhonnan pedig a számítógépet nem tekinthetik megbizhatónak, a kapott eredményt sem tekinthetjük annak. Nagyjából a chipkártyák biztonságát, ha legalább egy LED segítségével jelezhetnénk 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 programfájlok folyamatának körülményessége (az elkészült programot gyakran több lépéshoz 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 sajátos probléma, hogy a felhasználó bármikor megszakíthatja a folyamatban lévő műveletet a kártyának az olvasásból történő kihúzásával, s így annak adat menetének állapotban maradhatnak. Ez ellen olyan terminálokkal szokás védékezni, melyek a kártyát elnyelik, és csak a művelet végén adják vissza. Azonban ez a felhasználóknál gyakran bizalmatlanságot vált ki. Gondos programozással megvalósítható tranzakció kezelés a kártyán (de a hardware támogatás jelentősen megkönnyíti a mutat). Ez az adatbáziskezelők világából kölcsözőtt kifejezés azt jelenti, hogy a rendszer a kritikus műveletekről garantálja, hogy vagy teljesen befejezése, vagy pedig (ha valamiért ez nem lehetséges) a rendszer visszajut a tranzakció előtti állapotába.


![Header Body Diagram](image)

2. ábra – A parancs APDU struktúrát

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.

![Data SW1 SW2 Table](image)

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 hálózatok kiegészítő alkatrészeinek tekinti. A kártyák feladata tehát elsősorban a hálózati belépés (logon) illetve hálózati erőforrás-hozzáférés kezelésének támogatása, másrészre pedig – az Outlook részeként – elektronikus levelek titkosítása, hitelesítése.

A WinCardon egy 8 bites RISC AVR MCU processors 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 sб. számára, továbbá egy ATMEL kriptográfiai műveleteket támogató koproccesszorral, 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 messziről a valóságtól. A WinCard félrendszerre 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, Challenge and Response jellegű azonosítási protokoll.

A kártya képes alkalmazások egymás utáni futtatására, de a párhuzamos futtatást nem teszi lehetővé. A kártya félrendszerének 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, villamosbérlés, 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 sajjos nem tekintető két párhuzamosan futó processznek. Így a működés sokkal inkább függvényhívás-szerű (4. ábra).

![Diagram](image-url)

4. ábra – PC-kártya kommunikáció

A Microsoft ezzel a kártyával a Java Cardokat szeretnéné 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 történné, 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 amilyen különböző egy assemblyből, hogy nem látjuk a kártya regiszterei, s használhatjuk a basic vezérlési szerkezetet. Támaszkodhatunk viszont a kártya filerendszerére s kriptográfiai koporsózorára, 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**


<table>
<thead>
<tr>
<th>Sub WinCardTest()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dim l As Long</td>
</tr>
<tr>
<td>Dim i As Long</td>
</tr>
<tr>
<td>i = 0</td>
</tr>
<tr>
<td>l = 1</td>
</tr>
<tr>
<td>a hatványozást a kártya processzora nem támogatja</td>
</tr>
<tr>
<td>For i = 1 To hossz</td>
</tr>
<tr>
<td>l = l * 10</td>
</tr>
<tr>
<td>Call ScwGetCommByte ' be kell olvasni a teljes bemenetet, hogy</td>
</tr>
<tr>
<td>' ne legyen hiba</td>
</tr>
<tr>
<td>Next i</td>
</tr>
<tr>
<td>While i &lt; l</td>
</tr>
<tr>
<td>i = i + 1</td>
</tr>
<tr>
<td>' ez itt a kártya leterhelése</td>
</tr>
<tr>
<td>Wend</td>
</tr>
<tr>
<td>ScwSendCommByte NO_ERROR ' minden rendben van, kiléphetünk</td>
</tr>
<tr>
<td>End Sub</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>művelet végrehajtása</th>
<th>10 db ++</th>
<th>100 db ++</th>
<th>1000 db ++</th>
<th>10000 db ++</th>
</tr>
</thead>
<tbody>
<tr>
<td>művelet végrehajtása</td>
<td>3 másodperc</td>
<td>8 másodperc</td>
<td>80 másodperc</td>
<td>780 másodperc</td>
</tr>
</tbody>
</table>

1. táblázat – WinCard számítási sebességének teszteredményei

Az első két eredményen még az látásk, 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 futatható 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.

DEs titkosító rutin természetesen futathatja a PC-n is. Sőt, akkor sokkal gyorsabb is lehetne. Miért jó, hogy kártyán valósítottuk meg? Ahogy végignézzük a fenti négy parancsot, rögtön feltehető, hogy "hiba-állomány" 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 tény leg így van, akkor - mivel nem definiálnunk 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 képítésen hozzáférni a kulcshoz, hanem a kártya gazdája is. Ha van a zsebében a kártya, nem képes kinézni a titkos kulcsot befől, csak használni tudja.
A kártya nem hajlandó kiadni magából a kulcsot, az egyetlen esély annak megszerzésére a DES felkérés. Ez a rendszerrel célszerű ilyen szintű biztonság létrehozása volt.

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 c rendszer – a DES ereje és feltételezéseink alapján – biztonságosnak nevezhető.

A Java Card specifikáció

**A Sun-féle általános Java Card környezet**

A Sun Microsystems által létrejött Java Card specifikáció fő célja az, hogy csökkentse a programozható kártyákba való befejezés kockázatát, s ezzel elősegítsze a technológia fejlődését.

Amíg a chipkártyákat pusztán assembly nyelven lehet programozni, a fejlesző cégek kiszolgáltatott válak 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 orvosolására született a Java Card programozási környezet (API).

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

Egy Java Card applet forráskódját nézvegetve feltűnik, hogy a kód alacsony szintű. Kevés az objektum-létrehozás, s rengeteg a bitművelet és shiftelés. Látott a kódon, hogy a programozónak minden órajel útemért 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 személygyűjtést (garbage collection), így ha létrehozunk egy objektumot, azt nem tudjuk elpusztítani. Az applet és az általa létrehozott objektumok életra addig tart, míg az alkalmazást le nem törljük a kártyáról. A fentiekből – ha a program robosztusságát mindene 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.

Mindezen 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 móda lényegében a következő: miután az applet feltöltésre került, 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 kivülről atomnak minősül. Tranzakció közben a kiirandó adatok egyufferben tárolódnak, s csak akkor kerülnek ténylegesen kiíráshoz, ha a tranzakció véget ér. Sajnos minden buffer mérete véges, s ha beteliik, akkor a tranzakciókezelés nem működik. Ennek elkerülésére gondos programozás szükséges.


**A konkrét Bull környezet**


A Bull fejeszkőkörnyezet, az OdysseyLab néhány utility mellett három programból áll. A konvertáló program platform-független byte kódok a kártya gépi kódjára fordít. A feltöltött program menedzseli a kártyán lévő appleteket. Ennek segítségével lehet letörlni a kártyát, illetve feltölteni egy új appletet. Végül a harmadik programnak már nem a fejlesztés, hanem a tesztelés fázisában van szerepe: egy egyszerű terminálfelület 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 kártyaadórasó detektálására. Nem tartalmaz viszont semmilyen programot vagy programozási könyvtárat, amely a PC oldali felhasználást lehetővé téne.

**Példa alkalmazás bemutatása: One Time Pad**

Léven, hogy a Java Card nem rendelkezik kriptográfiai koproceszorral, a kriptográfiai műveleteket tisztán szoftvers úton kell megvalósítani. A Bulltől nem kapunk semmilyen kriptográfiai könyvtárat vagy Java package-t, így arra kényesztünk, hogy saját magunk valósítsuk meg az adatbiztonsági műveleteket. Az ilyen irányú fejlesztés első lépései 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. Háttránya viszont, hogy n bit rejtjelezéséhez n bit hosszú kulcsra van szükség. Egy kulcsbitet nem lehet kétszer felhasználni, így kizárólag "nagy méretű" kulcsok esetében értelmes.

Működése egyszerű. Vegyünk n bit rejtjelezző 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)

![Diagram](image)

5. ábra – A One Time Pad működése

Mivel a nyílt és a rejtett szöveg bitjei független valószínűségi változóknak tekinthetőek, 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 előre.

Chipkártya esetén nem tárolhatunk akárhány kulcsbitet. A kártyán 7040 byte hely van, s ebből az alkalmazásunk elfoglal 500 byte-ot. Ha 6000 byte hosszú kulcsot tárolunk, a kártyára már nem fér más. Mire használhatjuk ezt az algoritmust? Először is készíteni kell a "kulcs felvitelét" 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 méreténel. Amennyiben a One Time Padet titkos kulcsok (pl. 8 byte hosszú DES kulcsok) kicsereelésére használjuk, a 6000 byte 750 kulccsereére elegendő. Ha hetente cserélünk kulcsot, 14 évig használhatjuk a kártyákat.

A One Time Pad működéséből adódik, hogy a kódotlan és dekkódoló algoritmus megegyezik. Eredmény a kódoló és a kódolt adat hossza. Az eredmény kiolvasása a konkrét Bull kártya egy jellegzetességeből adódik. Az általunk megvalósított One Time Pad a 2. táblázatban bemutatott APDU-kkal működik. Példa: egy 0xE4080000003112233 APDU segítségével titkosítjuk 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 fogal helyet?

- Védeljük vele a kulcsot. Ha a kártyát fizikailag el nem lopják, a kulcs védve marad, mert a smart cardot nem lehet lemasolni;
- 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, lemasolhatatlanságát, s felhasználás utáni megsemmisülését.

**Sebességmérés**

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

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

2. programlista – Az Odyssey kártya sebességének mérésere 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).

<table>
<thead>
<tr>
<th>Műveletek száma</th>
<th>Szükséges idő</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.000</td>
<td>7 sec</td>
</tr>
<tr>
<td>1.000.000</td>
<td>69 sec</td>
</tr>
<tr>
<td>10.000.000</td>
<td>690 sec</td>
</tr>
</tbody>
</table>

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 mérésere más módszer alkalmaztunk: egy (a PC-n futó) ciklusban 1000-szer hívtuk meg programunkat N=0 bemenettel. Ez 58 másodperct 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 termékeknek, 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)

<table>
<thead>
<tr>
<th>Nézőpont</th>
<th>Microsoft</th>
<th>Bull – Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>A Microsoft Smart Card for Windows Professional kártyája</td>
<td>A Bull Odyssey I kártya 1.2-es változata</td>
</tr>
<tr>
<td>Fejlesztőeszköz</td>
<td>A Microsoft Visual Basic 6.0, s hozzá a Smart Card for Windows Plug-In (1999. májusi béta verzió)</td>
<td>Egy egyszerű test editor, a JDK 1.1.7 és a Bull OdysseyLab</td>
</tr>
<tr>
<td>PC oldali szoftverfejlesztés támogatása</td>
<td>A Visual basic 6.0 PC oldali fejlesztést támogató része, programkönyvtárak</td>
<td>Semmi</td>
</tr>
</tbody>
</table>

4. táblázat – Összehasonlítási szempontok


Azzal együtt, hogy a Microsoft kártyáját hardware paramétereit tekintve sokkal gyorsabbnak hihetnénk, méréseink tanulsága szerint messze nem ez a helyzet. A használat során a Microsoft-kártyát mindig is lomhának érezzük; különösen a nagy overhead volt zavaró. A Bull-kártyájánál 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 egyszerűségükönél fogva igen jellemző adatokat szolgáltatnak.

A hatalmas különbség okát nehéz megállapítani, hiszen nem tudjuk, pontosan mi történik a kártyákon és az egyes fejlesztőkörnyzetek belsőjében. Lehetetlenséges magyarázatként szóba jöhet a processzorok eltérő aritmetikai funkcionálisítása, 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 kompletten fejlesztőcsőkzt néhány termékekhez. A smart card tool kit a Visual Basic 6.0 vagy PlugInje, amely szervesen kapcsolódik a Visual Basichez. A fejlesztések 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 függetlenül az a kártyát kezelő alkalmazás egyazon környezetben üzemel el. Ugyanezen környezetben, egy emulátor segítségével ki is próbálhatjuk a kártyás alkalmazást, anélkül, hogy a kártyára le kellene töteni. Sajnos az emuláció nem biztosít egy a valóshoz hasonló környezetet, így a fejlesztésnek feltétlenül komoly részét képezi a kártyán való tesztelés is. Szintén ugyanazból a fejlesztőkörnyezetből feltöltethetjük az applikációt a kártyára, s futathatjuk 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 fejlesztőcsőkzt. A forráskód tetszőleges editorral elkészíthető. A forrásfájlakhoz a Bull CD-je pusztán JDK 1.1.7-t tartalmaz (a dokumentáció szerint Visual Caféval is működik), s e mellett foglalt 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 eszköcsomagot ad OdysseyLab néven, nem pedig egy fejlesztőcsőkzt. Szintén fájó, hogy nem támogatja az OdysseyLab a PC-oldali fejlesztést. Nekünk egy másik Bull smart cardos fejlesztőcsőkzt segítségével (amely a TB családába tartozó, régebbi típusú, ún. generikus kártyához készült) sikerült a kártyaoldalasztót meghajtani. Emulátort a Bull egyúttal a kártyánál vagy valós, meg a byte kódon lehet elvégezni egy Sun-féle (ingyenesen letölthető) emulátorral. A specifikációk ez is csak helyel-közössel felel meg, s ez sem támogatja a specifikáció kriptográfiai függyéveit.


A Java nyelv ezel szemben egy jól kidolgozott, áttekinthető, átfogó koncepciós rendelkező eszköz, s a Java Cardok nyelvén megkonstruálásakor a tervezők nem módosították semmit a Java-ban, csupán elvették belőle. Így a Java Cardok nyelve a Java egy részhalmaz. Egy Java Card applet szintaktikailag helyes Java programot jelent, igaz, logikailag nagyon szintén eltér a Java-tól. 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 lefrázását. 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 végrehajtási módja determinisztikusnak bizonyult. A WinCard esetében viszont rengeteg megmagyarázhatatlan eseménytal találtuk szembe magunkat. Amellett, hogy a dokumentáció meglehetősen szegényes, s ám az, hogy a működési cilindérték, a kártyán futó programmal kapcsolatban úgy éreztük, nem tartja be pontosan a Neumann-elvet, s az utasítások nem egymással követik egymást.

Összegzés

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


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 nevezettő modellje is), így aki rájuk fejleszt, nem válik kiszolgáltatott 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 technológia még gyermecképően jár, s még a fejlesztők sincsenek teljesen tiszta, hogy a technológia megfelelő. Ennek ellenére hisszük, hogy a közeli jövőben a programozható chipkártyák jelentősége megnő, és a technológia letisztulásával és az árak a nagy példányszám következtetében történő lecsökkenésével kiszorítják a hagyományos kártyákat.
Irodalomjegyzék

Általános kriptográfiai munkák:


Smartcardokkal kapcsolatos munkák:

Berta István Zsolt – Mann Zoltán Ádám: A hitelesség biztosításának lehetőségei intelligens smart card segítségével (TDK dolgozat)
Bruce Schneier – Adam Shostack: Breaking Up Is Hard To Do: MOdeling Security Threats for Smart Cards

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 nyelv specifikációja http://java.sun.com/docs/books/jls/html
PC/SC Workgroup: http://www.smartcardsys.com