Berta István Zsolt
IV. évfolyam, Műszaki Informatika szak

Mann Zoltán Ádám
IV. évfolyam, Műszaki Informatika szak

A hitelesség biztosításának lehetőségei intelligens smartcard segítségével[1]

Konzulens: Dr. Vajda István egyetemi tanár
Híradástechnikai Tanszék

1999. október 17.

[1] Ez a dolgozat a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Karán a 2000. évi Tudományos Diákköri Konferencián 1. helyezést ért el; a 2001. évi Országos Tudományos Diákköri Konferencián 2. helyezést ért el.

Tartalom

1. Munkánk körülhatárolása

Jelen tudományos diákköri dolgozat az 1999. évben, a Budapesti Műszaki Egyetem Híradástechnikai Tanszék Üzleti adatbiztonság laboratóriumában végzett munkánkat foglalja össze. Az általános kriptográfiai valamint smartcardokkal kapcsolatos vizsgálódásainkat a gyakorlatban is folytathattuk, amikor lehetőségünk nyílt a Microsoft Smart Card for Windows fejlesztőkörnyezetének tesztelésére, és smartcard alapú kriptográfiai alkalmazások készítésére, kipróbálására.

Az elkövetkezőkben részletesen ismertetjük a programozható smartcardok alkalmazási lehetőségeit a hitelességvizsgálat terén. Összegezzük tapasztalatainkat, méréseinket, bemutatjuk fejlesztéseink eredményét, valamint általában a smartcardok jövőbeli alkalmazhatóságával kapcsolatos elgondolásainkat.

2. A programozható smartcardokról

2.1. Bevezetés

Ebben a fejezetben áttekintjük a smartcardok történelmét, majd a második részben megvizsgáljuk, milyen tulajdonságokkal rendelkező kártyák vannak ma a piacon. Definiáljuk a programozható chipkártya fogalmát, s bemutatjuk, milyen cégek milyen termékekkel szálltak be a versenybe, s összefoglaljuk ezen termékek képességeit, s működésük filozófiáját. Végül szót ejtünk szabványosítási törekvésekről és különböző érdekcsoportok szerveződéséről is.

2.2. A múlt – A különböző kártyatípusok kialakulása

2.2.1. Az első kártyák

Az utóbbi években immár Magyarországon is megszokottá vált, hogy az emberek tárcájában különböző kártyák lapulnak. Legismertebb képviselőik a bank- és telefonkártyák, valamint az új diákigazolványok. Ezeken kívül azonban még számos más kártya van használatban, például különböző cégek törzsvásárlói kártyái, TV-dekóderek, GSM-kártyák stb. Minek köszönhető ez a hihetetlen fejlődés, mit nyújtanak a ma használatos kártyatípusok, és milyen további lehetőségek rejlenek a smartcardokban? E kérdésekre próbálunk a továbbiakban választ adni.

Az "intelligens kártyák" intelligencia dolgában igen különböző képességekkel rendelkezhetnek. Alapvetően két osztályt különböztethetünk meg: csak adattárolásra alkalmas, illetve önálló számítási és feldolgozási kapacitással is rendelkező kártyákat.

A kizárólag adatok tárolására alkalmas kártyák őse valószínűleg a névjegykártya. Az első műanyag alapú kártya 1950-ből, a Diners Clubtól származik. Az 50-es évek végére további két cég csatlakozott a kezdeményezéshez: az American Express és a Carte Blanche. Az első hitelkártya a Bank of Americatól származik, ebből lett később a VISA. Az Interbank egy más rendszert hozott létre, Mastercard néven. Ezek a kártyák azonban csak bevésett, illetve domborított azonosítók "tárolására" voltak alkalmasak.

2.2.2. Mágnes- és optikai kártyák

Az 1970-es években jelentek meg az első, mágnescsíkot tartalmazó kártyák, mégpedig az International Air Transportation Associationnél (IATA[2]). Ezen a kártyán a mágnescsík 210 bit/inch információt tárolt, ami 79 alfanumerikus (7 bites) karakternek felel meg. Ma kompatibilitási okokból a mágnescsíkot három sávra osztják; az első sáv felel meg a korábbi csíknak, és csak olvasható információkat tárol. A második sáv 75 bit/inches sűrűséggel

további 40 szám tárolására alkalmas, míg a harmadik sáv 107 számjegy írható és olvasható tárolását teszi lehetővé.

Ennél lényegesen nagyobb adatmennyiség tárolható az optikai kártyákon. 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 előállítási költsége rendszerint lényegesen nagyobb, mint a mágnescsíkkal ellátottaké, viszont a tárolókapacitás a megabyte-os tartományban mozog. Ezért ilyen kártyákkal főleg az orvosi szektorban találkozhatunk, ahol a beteg kezeléseinek naplózásán kívül esetleg egész röntgenképeket is tárolni kell.

2.2.3. Chipkártyák

A kártyák piacán egyre nagyobb számban vannak jelen a chipkártyák, melyek mikroelektronikai áramkörök felhasználásán alapulnak. Az első ilyen jellegű ipari próbálkozások az Innovatron cég 1974-es megalakulásához köthetők. A Bull 1979-ben készítette el első mikroprocesszorral is rendelkező kártyáját, melyen azonban a processzor még külön chipen helyezkedett el. Ez nem bizonyult kellően megbízható megoldásnak. A technikai fejlődés azonban csak a 80-as években tette lehetővé az összes áramkör egyetlen chipre való integrálását. Franciaországban 1986 óta használnak chipkártyákat az utcai telefonokhoz.

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 a kívülről érkező parancsokat hajtsa végre, hanem önálló alkalmazások futhatnak a kártyán (esetleg több is), a kártya operációs rendszere fölött. Ennek következtében a biztonságot már maga a kártya garantálhatja, nem kell azt a terminálra/kártyaolvasóra bízni.

Általában a biztonság kulcsfontosságú kérdés a smartcardokkal kapcsolatban. A memória rendszerint zónákra van osztva, amelyekhez különböző hozzáférési attribútumok és kulcsok (pl. PIN – Personal Identification Number) rendelhetők. Ezen kívül az újabb kártyák külön kriptográfiai processzorokkal is el vannak látva, így fejlett kriptográfiai módszereket alkalmazhatnak az adatbiztonság érdekében (pl. DES – Data Encryption Standard).

Mint látható, rengeteg különböző típusú kártya van. Ezek között valamelyest rendet a nemzetközi szabványok teremtenek, melyek azonban rendszerint valamely már de facto kialakult szabványt emelnek jogerőre. A témában az első szabványok (ISO 7810 és ISO 7811) a kártyák fizikai tulajdonságait rögzítették. További kapcsolódó szabványok az ISO 7812 és ISO 7813. Kizárólag a chipkártyákról szól az ISO 7816 szabvány. Ez azonban lehetővé tesz hibrid kártyákat is, ahol pl. mikrochip és mágnescsík egyaránt jelen van.

[2] A szövegben előforduló rövidítések magyarázataikkal együtt megtalálhatóak a függelékben

Az utóbbi évtized vitathatatlanul az Internet és a PC-k széleskörű elterjedésének évtizede. Nem véletlen tehát, hogy az utóbbi években nagyon erős aktivitás tapasztalható a smartcardok PC-s és internetes alkalmazásával kapcsolatban. Annál is inkább, mivel a smartcardok számos lehetőséget biztosíthatnak az Internet biztonságosabbá tételére, pl. üzenetek titkosítása, hitelesítése, elektronikus kereskedelem. Ezen kívül elterjedtek a nyilvánosan használható PC-k (Internet kávéház, teleház), amelyek sajátos integritási problémákat vetnek fel.
Ebből a szempontból kulcsfontosságú esemény a PC/SC Workgroup megalakulása (1996 május). Ez a csoport 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 hoztak egy specifikációt az intelligens kártyák, a kártyaolvasók és a PC-k együttműködéséről. Ez a specifikáció megfelel az ISO 7816 szabványnak, továbbá annak más bővítéseivel (EMV, GSM) is kompatíbilis.

2.3. A jelen - A mai piac vizsgálata

2.3.1. Mi az a programozható smartcard?

Egy smartcard programozható, ha rendelkezik processzorral és memóriaegységgel, továbbá képes felhasználói program futtatására. Így a kártya kibocsátója (nem a gyártó) képes applikációkat a kártyára letölteni, s azokat futtatni. Ezek a kártyák sokcélú eszközök, s az, hogy végül mire is használjuk őket, esetleg csak forgalomba hozásuk után derül ki.
Ezek a smartcardok rendelkeznek processzorral s (illékony illetve nem illékony) memóriaegységgel. Nemcsak egy „beégetett” mikroprogramot (operációs rendszert) képesek végrehajtani, hanem akármit, amit a kártya kibocsátója rájuk telepít. A program megírásához nincsen szükség a kártya belső architektúrájának részletes ismeretére, s nincsen szükség a kártya gyártójának méregdrága hardverére, szaktudására s ismereteire. Bármelyik kisebb-nagyobb cég, amelyik kártyás alkalmazás fejlesztésére adja a fejét, megteheti ezt, mindössze szoftverre s egy olvasóra van szüksége.
E kártyák tulajdonképpen egyenértékűek egy lassú számítógéppel, csupán az input/output perifériákban különböznek. Egy smartcardnak természetesen nincsen képernyője s billentyűzete, nincsenek soros, illetve párhuzamos portjai. Egyetlen úton tud kommunikálni a külvilággal: a kártyaolvasón keresztül.

2.3.2. Miért jó programozható smartcardot használni?

Egy hagyományos smartcard „egyszerű” adattároló egységként működik. A kártya kibocsátója definiálhat rajta különféle file-okat, s minden file-hoz megadhatja, milyen jogosultságok, jelszavak szükségesek a hozzáférésükhöz. Ezek után a kártya birtokosa, a felhasználó, magával hordozza a kártyát, s különféle kártyaolvasókba behelyezi azt. (Például

amennyiben a kártya egy metrójegy szerepét tölti be, a kártyatulajdonos behelyezi kártyáját a metróállomásokon lévő olvasókba, s ezzel „érvényesíti jegyét".) Az olvasó műveleteket végez a kártyán lévő adatokkal, olvassa őket, s – jogosultságainak függvényében – esetleg ír is.
Egy programozható kártya esetén más a helyzet. A kártyán lévő adatokon itt nem az olvasó - vagy az olvasóhoz kapcsolódó számítógép – végzi a műveleteket, hanem a kártyán futó szoftver maga. Az a program, amit a kártya kibocsátója készített. A program üzeneteket kap az olvasón keresztül, azokat dolgozza fel. A kártya kibocsátója a kártyán lévő alkalmazás megírásakor műveleteket definiál a kártyán, s az adatokat később csak ezeken a műveleteken (kapukon) keresztül lehet elérni.

2.3.3. Miért rossz programozható smartcardot használni?

Erre a kérdésre a válasz egyszerű. Nem rossz. A programozható smartcardok mindent tudnak, amit a hagyományos kártyák. Az általunk használt Microsoft kártya is megfelel az ISO 7816-4 szabványnak, amely definiálja az APDU-kat, (select file, read binary, stb.) amelyeket hagyományos smartcardok elfogadnak. Így funkcionálisan nem kevesebbek adattárolásra használatos társaiknál.
Egyetlen hátrányuk van: az áruk. Sokkal drágábbak ugyanis, mint a kevesebbet tudó hagyományos kártyák. Igaz, itt is, mint mindenütt, az árban az egyik meghatározó tényező az, hogy mekkora példányszámban kerül valami forgalomba. Programozható kártyákat még nem használnak olyan széles körben, hogy igazán megérje őket nagy példányszámban gyártani. Ugyanakkor, míg egy hagyományos kártyát csak egy előre meghatározott célra lehet alkalmazni, addig egy intelligens kártyát praktikusan bármilyen kártyás célra használhatunk. Így a példányszámot jóval magasabbra lehet majd emelni, mint a hagyományos kártyák esetében. A jövőben várható, hogy áruk jelentősen csökkenni fog.

2.3.4. Egy példa alkalmazás leírása

Vegyük egy telefonkártya példáját! Nézzük meg, mi történik, ha a telefonkártya szerepét nem programozható, tehát hagyományos kártya tölti be!
A kártya kibocsátója beállítja a kezdet kezdetén, hogy a kártyán mondjuk 50 egység van. Eladja a kártyát a végfelhasználónak, aki utána telefonál vele. Az beteszi a kártyát egy készülékbe. Ekkor a telefonkészülék ellenőrzi, tényleg telefonkártyát tettek-e bele. Ha ezt sikerül megállapítania, kiírja, hány egység van még a kártyán. Ezek után a felhasználó tárcsáz, s a készülék rendszeresen levon a kártyán lévő egységekből. Ezt hogyan teszi? Először is kiolvassa, hogy hány egység van a kártyán, utána csökkenti ezt a számot annak függvényében, hogy helyi beszélgetést folytat az illető, vagy pedig Amerikával cseveg, majd visszaírja az eredményt a kártyára.
Intelligens kártya esetében más a helyzet. A kártya kibocsátója itt is inicializálja a kártyát. Amikor a felhasználó beteszi a kártyát a készülékbe, hogy telefonáljon vele, nemcsak a

készülék állapítja meg, hogy tényleg telefonkártyát tettek-e bele, hanem a kártya is ellenőrzi, hogy tényleg telefonkészülékbe rakták-e be. Ezután a telefonkészülék küld egy üzenetet a kártyának, amelyben megkérdezi, hány egység van rajta. A kártya válaszol. A felhasználó tárcsáz, s a készülék kiszámítja, hány egységet kell levonni. Amikor éppen csökkenteni kell az egységek számát, a telefonkészülék üzenetet küld a kártyának, hogy csökkentse az egységek számát. A kártya ezt a műveletet saját maga végzi el.

A lényeges különbség az, hogy az utóbbi esetben a feldolgozandó adatok nem kerültek ki a kártya biztonságos környezetéből. Tehát ha a kibocsátó a következő műveleteket definiálja a kártyán: *lekérdezés()* és *csökkentés()*, és a kártya adatait csakis ezeken a műveleteken keresztül lehet elérni, annak a lehetősége, hogy valaki az egységeken más műveleteket végez, teljesen ki van zárva. Nem lehetséges például növelni a kártyán lévő egységek számát. Egyáltalán írni sem képes a kártyára senki más, mint a rajta futó program.

Meg kell jegyeznem, a fenti két művelethez még hozzátartozik egy, amivel a kártya és a telefonkészülék kölcsönösen megállapítják egymás kilétét. Létezhet továbbá még egy *inicializálás()*, művelet is, mellyel a kibocsátó beállítja a kártya egységeinek számát. Igaz, jobb, ha ezt nem vesszük be a műveletek közé, s a kibocsátó a filerendszer felírásával inicializálja a kártyát. Ebben az esetben viszont a kártyán lévő egységek számát növelni vagy alaphelyzetbe állítani senki sem tudja. Még a kibocsátó sem képes erre, s így a kimerült kártyát újra hasznosítani nem lehetséges.

2.3.5. Alkalmazások fejlesztésének lehetőségei

A smartcardok technikai okokból „kis” erőforráskészlettel rendelkeznek. Például 8 kilobyte EEPROM már igen soknak mondható. Processzoruk is viszonylag lassú, s egy bizonyos határnál gyorsabb nyilvánvalóan nem lehet, hiszen a műanyag tokban igen nehéz megoldani a kellő hűtést. Ezek után természetes lenne, hogy e kártyákra programot assembly nyelven lehet írni.

Csakhogy ez nem így van. A kártyakibocsátók rettegnek attól, hogy elkötelezik magukat egy kártyatípus mellett, s ezzel kiszolgáltatottá válnak az adott kártyagyártóval szemben. Az egymástól eltérő gyártmányú vagy esetleg csupán eltérő típusú kártyáknak gépi kódja s így assembly nyelve gyökeresen különbözhet egymástól. Ezért elképzelhető, hogy egy kártyakibocsátó kifejleszt egy alkalmazást az egyik cég kártyájára, majd át kell térnie más típusú kártyákra, s teljesen át kell írnia a programját, esetleg újra kell kezdenie az egész fejlesztést.

Így a fejlesztések bátorítása, ösztönzése végett szabványok alakultak ki, melyeket a kártyáknak ki kell elégíteni. Az egyik ilyen a Java Card.

2.3.6. Java Card

A Sun Microsystems a Java nyelvet azért alkotta meg, hogy legyen egy nyelv, amely platformfüggetlen, s mellyel fejlesztett alkalmazások a legkülönbözőbb architektúrájú gépeken futnak. A java appletek – mivel Internetes terjesztésre találták ki őket – kellően kicsik. Mivel egy intelligens smartcard tekinthető számítógépnek is, természetesen vetődik fel a kérdés, hogy miért ne lehetne ez a platform is java kompatibilis.

A válasz is természetes: Egyrészt, mert a java byte-kódot generál, amit egy, az adott gépen futó virtuális gép (java VM), értelmez, s ez eredendően lassú. A smartcardok processzora így is jelentősen lassabb, mint a PC-ké, s ez mindig is így lesz. Másrészt a java nyelv túl komplex ahhoz, hogy programjait smartcardokon futtassuk. Maga a java virtuális gép is rengeteg helyet foglal el.

A Java Card API is az objektum orientált szemléletet követi. Minden egyes applet, amit a kártyára letöltük, egy-egy objektum, amely önálló életet él. Rendelkezik attribútumokkal és metódusokkal. A külvilág számára látható metódusok pedig argumentumként egy APDU-t (az az adategység, amit a PC küld el a kártyának egy csomagban) kap.

A Java Card appletek összefonódnak a webes appletekkel. Egy web böngészőben futó applet közvetlenül nem képes kommunikálni a kártyával, hanem szükséges egy áthidaló program, egy servlet a server oldalon. Ez a servlet tartja a kapcsolatot a szerver gépben lévő kártyán futó applettel (cardlet).

A Java Card appletek szervesen illeszkednek a java elképzelésekhez és az objektum orientált világhoz, s rendelkeznek a magasszintű java nyelv erejével s hordozhatóságával. Természetesen eredeti formájában a java nem alkalmas smartcardra. A Java Cardokon alkalmazott java az igazinak egy leegyszerűsített változata. A programozó a Java Card.framework, Java Card.security és Java Cardx.crypto csomagokat használhatja, melyek támogatják a kártyákba gyakran beépített kripto-koprocesszorok használatát. A másik változtatás pedig, hogy a kártyákon nem byte-kód fut, hanem gépi kód, amit egy kártyaspecifikus konvertáló programmal állíthatunk elő.

Tehát egy Java Card alkalmazás kártyára telepítésének menete a következő: A programozó megírja a programot, s lefordítja byte-kódra egy tetszőleges java fordítóval. Megteszi erre a célra a teljesen ingyenes JDK is. Ezek után a programot egy – most már kártyaspecifikus – konvertáló programmal lefordítja gépi kódra. S végül egy – szintén kártyafüggő – feltöltőprogrammal elhelyezi művét a smartcardon.

A program fejlesztése során tehát a kibocsátó nem kötelezte el magát egyetlen gyártó egyetlen terméke mellett, hanem a java appletet kártyafüggetlen (sőt, ingyenes) eszközökkel állítja elő. Ha később mégis kártyát akar váltani, akkor az új gyártó új kártyájához nem kell megvennie a fejlesztőrendszert, hanem elég neki a konvertáló program és a feltöltő eszköz. Sőt, az új kártyára az eredeti java applet változtatás nélkül átmehet.

Ez persze csupán elmélet, s könnyen lehet, hogy a gyakorlat ettől jelentősen különbözik. Mindenesetre ez komoly rugalmasságot jelentene a fejlesztők számára. A Java Card API

pedig igen elterjedt, a nagy kártyagyártó cégek közül sokan mögé álltak, s dobtak piacra Java Cardokat. Például a Bull kártyája az Odyssey I. és a DelaRue kártyája a GalactIC. Sőt, jelentek meg java gyűrűk is, melyek Java Card technológiát használnak ugyan, de alakjuk nem hitelkártyához, hanem inkább pecsétgyűrűhöz hasonlítható. De természetesen nem a Java Card az egyetlen irány.

Nézzünk, hogy épül fel egy egész egyszerű digitális pénztárca alkalmazás!

```
public class Wallet
{
       public int lekerdezes() { … }
       public void betet( int mennyit ) { … }
       public void kivet( int mennyit ) { … }
};
```

2.3.7. A MULTOS

A MULTOS lényege az, hogy megoldást ad több teljesen különálló alkalmazás futtatására s elkülönítésére egyetlen kártyán. Itt nem a nyelvet rögzítették le, hanem csupán az operációs rendszert. A MULTOS egy smartcardos operációs rendszer, amely egyfajta viselkedést rögzít le, amelyet a kártyán futó applikációk felé az operációs rendszer tanúsít, illetve egy viselkedést, amelyet az applikációknak a kártya operációs rendszere felé tanúsítani kell. Bárki fejleszthet MULTOS alá C, assembly vagy akár java nyelven – persze egy bizonyos kártyán futó MULTOS alá. A MULTOS pedig kártyagyártóknak eladja a MULTOS specifikációját, hogy azok megírhassák saját kártyájukra a MULTOS-t.

2.3.8. WinCard - A Microsoft kártya

A Microsoft hamar előállt a PCSC specifikáció egy referencia implementációval, majd 1998 októberében hivatalosan is bejelentette a Smartcards for Windows rendszert. A tervek szerint a smartcardok szerves részét fogják képezni a jövő Windows-os rendszereinek: szerepet fognak kapni a logon folyamatban, az Outlook részeként az üzenetek hitelesítésében, valamint az elektronikus kereskedelemben.

Ez a kártya is magas szintű nyelven, Visual Basicben programozható. Ebben is, s más dolgokban is a Microsoft-féle kártya természetesen a Microsoft-világhoz illeszkedik. Itt a kártyán nem objektum van, hanem applikáció. Nem metódusai vannak, amiket meghívhatunk, hanem egyetlen belépési pontja van, amit el lehet indítani. Nagyon hasonlít egy exe file-hoz, ami paramétereket kap, s ezek függvényében találja ki, mi a feladata.

Ugyanaz a digitális pénztárca alkalmazás, amit a Java Card esetében láttunk, most egy program lenne, melynek törzse így festene:

```
Sub Main
       Select Case scwGetCommByte() ‘beolvasunk egy byteot
              Case “b”: betet()
              Case “k”: kivet()
              Case “l”: lekerdezes()
       End select
       …
End Sub
```

Tehát, hogyha egy komplex programot hoz létre a felhasználó, amely egymástól jelentősen eltérő funkciókat tartalmaz, akkor vagy ő hoz létre „metódusokat” az applikációban, vagy több applikációt ír, amelyek egy-egy metódusnak felelnek meg.

2.3.9. OpenCard

Az OpenCard egy, a PC/SC-hez hasonló ipari tömörülés. A tagok között is sok közös van. Lényeges különbség azonban, hogy míg a PC/SC-ben a Microsoft jelentős szerepet játszik, sőt, a PC/SC specifikáció jelenleg egyetlen megvalósítása a Microsofttól származik, addig az OpenCardban a Microsoft nincs jelen. Az OpenCard ennek megfelelően nem is (csak) a javarészt Wintel platformra épülő PC-ket célozza meg, hanem egy annál általánosabb, platform-független szabványt kíván létrehozni.
A platform-függetlenség kulcsa a Java nyelv használata. Ettől még azonban az OpenCardnak nincs sok köze a Java Cardhoz, hiszen itt nem csupán a kártya programozási felületéről van szó. Ehelyett az OpenCard a teljes kártya/olvasó/számítógép/alkalmazás együttműködésre ad előírást. Ennek neve: Open Card Framework (OCF). Ez egyrészt egy magas szintű specifikáció az alkalmazások fejlesztői számára, mely elrejti előlük a kártya és az olvasó fizikai sajátosságait. Másrészt egy alacsonyabb szintű specifikáció a szolgáltatók számára, amely lehetővé teszi a szolgáltatások különböző cégektől származó építőelemekből való összeállítását.
Az OpenCard tagjai: 3-G International, American Express, Bull, Dallas Semiconductor, First Access, Gemplus, IBM, Intellect, Liberate Technologies, Newcom Technologies, Toshiba, TOWITOKO, SCM Microsystems, Schlumberger, Siemens, Sun Microsystems, UbiQ, Visa International, XAC Automation

2.3.10. MUSCLE

A M.U.S.C.L.E. egy új törekvés a smartcardos technológiák integrálására a linuxos világba. A Linux alapelveinek megfelelően ezen törekvés mögött nem egy profitorientált cég áll, hanem a világ linuxos közössége. Nem is új szabványok kidolgozása és elfogadtatása a cél, hanem a meglévő specifikációk (ISO 7816/4, PC/SC, OpenCard, PKCS11) integrálása a Linux platformra, továbbá smartcardok és smartcard olvasók linuxos meghajtóinak készítése.

3. Biztonsági megfontolások

3.1. Bevezetés

A smartcardok kezdettől fogva bizalmas információkat tároltak (legyen az akár számlaszám, jelszó, vagy orvosi feljegyzések), ráadásul elveszíteni is könnyebb egy kártyát, mint mondjuk egy PC-t, így a kártyák biztonságossága mindig is elsőrendű szempont volt. Persze mindig voltak olyanok, akik számára kihívást (vagy meggazdagodási lehetőséget) jelentett a kártyák feltörése, és erre egyre trükkösebb módszereket találtak ki. Az újabb kártya-generációk aztán felkészültek az ilyen támadásokra is, így a feltöréshez újabb trükkökre volt szükség. Ez egy soha véget nem érő játék, amiből az alábbiakban néhány tipikus támadást és az azok elleni védekezést mutatjuk be.

A lehetséges támadások alapvetően két csoportra oszthatók aszerint, hogy a támadás a kártya fizikai vagy logikai szintjét veszi célba. Ennek megfelelően beszélhetünk fizikai és logikai biztonságról.

3.2. Fizikai biztonság

Egy kártya fizikai manipulálásához rendszerint igen komoly és költséges felszerelésre van szükség (mikroszkóp, lézeres vágóberendezés, mikromanipulátor stb.), ami csak nagyon keveseknek áll rendelkezésre. Ennek ellenére fel kell arra készülni, hogy valakinek sikerül celláról cellára kiolvasni az EEPROM tartalmát. Ezért legalább a PIN-t és egyéb fontos adatokat mindenképp titkosítva kell tárolni.

Azonban ha a támadó tudja a titkosított PIN-t, és ismeri a titkosításhoz használt egyirányú függvényt, akkor, mivel a PIN tipikusan 10000 különböző értéket vehet fel, várhatóan néhány ezer próbálkozással meg tudja határozni.

Ezért lehetőleg meg kell akadályozni, hogy az EEPROM állapota kiolvasható legyen. Így bipoláris technológia nem alkalmazható, hiszen ezen látszik a cellák állapota, tehát MOS technológiára van szükség. [Simmons1992 *(12.4.1.)*]

Másrészt viszont a kártya készítése és tesztelése során szükség lehet arra, hogy az egyes cellák tartalma minden további nélkül kiolvasható legyen. Ezért a kártyáknak általában két állapota van: teszt és felhasználói mód, amelyek közt csak az egyik irányban van átjárás. Így a tesztelés végeztével a felhasználói módba való átkapcsolás rendszerint egy biztosíték kiégetésével történik.

Persze ha valakinek sikerülne eltávolítani a biztosíték fölül a passziváló réteget, és áthidalni a biztosítékot, a kártya újra visszakerülne tesztmódba.

Ennek megakadályozására egy további, logikai kapcsolót is be szoktak építeni az EEPROM-ba. A kártyát csak úgy lehetne visszakapcsolni tesztmódba, ha valaki először az EEPROM-

ban lévő kapcsolót átállítja, majd a biztosítékot áthidalja. Viszont amíg a biztosíték nincs áthidalva, nem lehet hozzáférni ahhoz az EEPROM-területhez.
Egy további lehetőség lenne a passziváló réteg eltávolítása után az EEPROM megfelelő kontroll és címző csatornáinak meghajtásával kiolvasni az adatbuszon keresztül az egyes memóriacellák tartalmát.
Pillanatnyilag ez nem lehetséges, pusztán a chip kicsiny méretei miatt. Lehet persze, hogy egy-két éven belül olyan fejlett mikromanipulációs módszerek lesznek ismertek, amik ezt lehetővé teszik. Azonban várhatóan addigra a chipek mérete is le fog csökkenni, sőt, arra lehet számítani, hogy a mikromechanikai módszerek még sokáig le lesznek maradva a chipek optikai gyártási technológiáihoz képest.
Eddig statikus támadásokat néztünk, vagyis olyanokat, melynek során a kártya nincs bekapcsolva. Most vizsgáljuk meg, milyen támadási lehetőségek kínálkoznak a kártya működése közben.
Elvileg egy támadó egy kellően gyors számítógéppel becsatlakozhat a kártya és az olvasó közé anélkül, hogy azok ebből valamit is észrevennének. Eltárolva a dialógust, legközelebb a támadó megtéveszthetné az olvasót, amennyiben egyszerűen úgy járna el, mint korábban a kártya, vagy a kártyát, ha az olvasót utánozza.
Ezellen általában dinamikus jelszavakkal lehet védekezni, vagyis minden dialógus során más jelszó használatával. Ez rendszerint azon alapul, hogy az egyik fél elküld egy véletlenszámot a másiknak, az elkódolja a véletlenszámot a titkos kulccsal, majd az eredményt visszaküldi. A másik fél ebből meg tudja állapítani, hogy birtokában van-e a titkos kulcs.
Egy lehetséges támadás a kártya véletlenszám-generátorának kicselezése. Ehhez annyi véletlenszámot kell generáltatni vele, hogy túlcsordulás miatt ugyanazok a számok ismétlődjenek.
A mai kártyák azonban ez ellen már úgy védekeznek, hogy túlcsordulás esetén blokkolódnak.

3.3. Logikai biztonság

A kártya elleni mindenfajta logikai támadás alapját a kártya/olvasó kommunikáció alapos ismerete jelenti. Manapság adottak a technikai feltételek arra, hogy akárki készítsen egy hamis kártyát, aminek vagy a kommunikáció kiismerésében, vagy a kommunikáció ismeretében a kártya utánzásában veheti hasznát. Mindenképpen szükség van azonban a kártya utasításkészletének, illetve a kártyának küldhető APDU-knak az ismeretére. Ez rendszerint nem (teljesen) publikált, de a legtöbb kártya esetén a következő módon könnyen kiismerhető. Először olyan APDU-kat küldünk a kártyának, amelyekben az utasításosztályt (CLA) változtatjuk 0-tól FF-ig. Tipikusan 2-3 esettől eltekintve „érvénytelen utasításosztály” üzenetet kapunk válaszként. Ezután a talált 2-3 osztályt pásztázzuk végig, az utasításkódot emelve 0-tól FF-ig.

A kártya számára tehát egyedül kriptográfiai módszerek jelenthetnek biztonságot. Megfelelő algoritmusok alkalmazása és átgondolt kulcsgondozás mellett az ilyenfajta támadásokból a támadónak nem származhat előnye.

A korai kriptográfiai algoritmus implementációk súlyos hibája volt, hogy a végrehajtási idő erősen függött a kulcstól és a kódolandó üzenettől. Így a támadónak csak a kódolás idejét kellett megmérni, és ez olyan mértékben lecsökkentette a lehetséges kulcsok halmazát, hogy azt a nyers erő módszerével végig lehetett nézni.

Ez ellen a mai kártyák két fajta védelmet tartalmaznak: egyrészt a megvalósításkor gondosan ügyelnek arra, hogy a futási idők ne függjenek a kulcstól. Másrészt hibaszámlálókkal teszik lehetetlenné a próbálgatásos módszereket. Ha a hibás próbálgatások száma elér egy előre meghatározott küszöböt, a kártya blokkolódik.

További probléma volt, hogy számos kártyatípusnál a hibaszámláló tényleges inkremántálása csak a hibakód kiadása után történt meg. Így ha a támadónak sikerült elég gyorsan megszakítani a kártya áramellátását, megakadályozhatta a hibaszámláló növelését, és így végigpróbálgathatta a lehetőségeket. Más típusoknál pedig a kártya áramfelvételéből lehetett következtetni arra, hogy az összehasonlítás eredménye pozitív vagy negatív volt-e, és annak alapján ugyancsak az áramellátás megszüntetésével lehetett megakadályozni a hibaszámláló inkrementálását.

Természetesen mihelyst fény derült e hibákra, megtörténtek az ellenintézkedések. Tervezéskor gondosan ügyelnek arra, hogy a hibakódot csak a hibaszámláló inkrementálása után adják ki. További szempont, hogy a kártya áramfelvételéből semmilyen következtetést ne lehessen levonni az éppen végzett műveletre, illetve annak eredményére vonatkozóan. Ezen kívül az is elterjedt módszer, hogy még összehasonlítás előtt megnövelik a hibaszámlálót, majd pozitív eredmény esetén lecsökkentik. Így a támadó nem tud profitálni az áramellátás elvágásából.

További támadási lehetőség kínálkozik, ha hagyjuk, hogy a kártya és az olvasó kölcsönös autentikációja megtörténjen, majd a további kommunikációt egy közbeékelt, elég gyors számítógéppel kontrolláljuk, illetve módosítjuk.

Ezellen úgy lehet védekezni, hogy az autentikáció után sem nyílt üzenetváltás folyik a terminál és a kártya között, hanem egyrészt a lehallgatás ellen titkosítjuk az üzenetet, másrészt módosítás ellen (kriptográfiai) ellenőrző összeggel látjuk el.

A 3.2. és 3.3 fejezetek anyagához a [Rankl-Effing1997 *(8.6)*]-ot használtuk.

3.4. „Jó" kártya tervezésének elvei

A fentiek alapján mi a következő feltevéseket, elvárásokat fogalmaztuk meg a kártya biztonságosságával kapcsolatban.

A kártyán bizonyos kapukat, műveleteket definiáltunk, s a rajta lévő adatokat elérni csak ezeken keresztül lehet. Ezzel kívánjuk azt biztosítani, hogy a kártya adatai bizonyos feltételeknek megfeleljenek, s ne lehessen rajtuk illegális műveleteket végrehajtani. A

külvilág szeme elől elrejtjük az adatokat, s abból, ami a kártyán folyik, senki nem láthat semmit, csak azt, amit a rajta futó program kiküld.

Ez hasonló jelenség ahhoz, amit például az objektum-orientált programozás esetében tapasztalunk. Összefogjuk az adatokat s a rajtuk elvégezhető műveleteket egy egységgé (encapsulation), s az adatokat elrejtjük az árgus szemek elől (information hiding). Egy kártyával dolgozó program a kártyát fekete dobozként látja: nem tudja, mi van benne, csak azt tudja, mit küld be, s mit kap eredményül.

Ha a kártya interface-ét precízen definiáljuk, s megoldjuk, hogy a kártyán futó eljárások ne akadjanak össze, akkor egy támadó bármilyen sorrendben és számban indítja is el a kártya műveleteit, sem inkonzisztens állapotot nem alakíthat ki, sem pedig olyan irányba nem tudja módosítani az adatokat, ami a kibocsátónak nem állt szándékában.

A tervezés során a következő feltételezésekkel éltünk:

- *Jól* megtervezett kártya esetében a smartcardon lévő adatokhoz kizárólag a kártyán definiált műveleteken keresztül lehet hozzáférni. Ezeket a műveleteket a tervező definiálja s programozza le.
- Nincsenek a kártyán a tervező számára ismeretlen műveletek. Tehát a kártya gyártója vagy a kártya operációs rendszerének írója nem helyezett el a smartcardon sem szoftver sem hardver kiskapukat, melyekkel hozzáférhet a kártya adataihoz a fenti műveleteket megkerülve.

A második feltétellel kapcsolatban csak találgatni lehet. Ha viszont az első nem igaz, akkor a smartcard technológiának nincsen értelme. Ha akárki leemelhet adatokat a kártyáról, akkor annak biztonsága egy floppy diskével azonos. Habár ez a feltétel szó szerint biztosan nem érvényes, hiszen multik vagy titkosszolgálatok esetleg képesek lehetnek olyan erőforrások és célhardver felhalmozására, amely a kártya feltöréséhez szükséges, a "normális" üzleti világban a kártyák biztonságosnak mondhatók, s a hétköznapi emberekkel és kisvállalatokkal, kisebb országokkal szemben a kártya feltörhetetlen. Igaz, a smartcardok ilyen korlátairól igen kevés az ismeretünk, mivel mind a gyártók, mind az illetékes titkosszolgálatok mélyen hallgatnak a témában.

Szintén érdekes eset, hogy amíg a hagyományos kártyák csupán adathordozó funkciót látnak el, illetve csak arra alkalmasak, hogy az olvasó (mondjuk egy PC) ellenőrizze a kártya valódiságát (érvényességét), egy intelligens kártya alkalmazható egy PC azonosítására. Természetesen amennyiben a kártya felfedez valamilyen problémát, a PC-vel (vagy olvasóval) kapcsolatban, akkor felmerül az a gond, hogy a kártyabirtokosnak erről figyelmeztetést adni a kártya csupán a nem megbízható PC-n keresztül tud… Jó lenne ilyen szempontból, ha a kártya is el lenne látva kijelző eszközzel. Egyetlen LED, amely ég, hogy ha minden rendben van, s nem ég, ha baj van, sokat növelne a kártyák használhatóságán.

Hogy kell egy kártyát *jól* megtervezni?

- *APDU-kkal ne lehessen rajta műveleteket végezni!* Vagyis, ne lehessen a már kész kártyát a PC-ről háttértárként kezelni. Egyedül az alkalmazásfuttatás legyen megengedett, írni-

olvasni ne lehessen. Ennek implementálására egy egyszerű ötletet találtunk. Rendeljünk hozzá minden egyes file-hoz egy-egy olyan hozzáférési listát, amely nem tartalmazza az ismeretlen felhasználót. Minden file-hoz csak olyan felhasználók férhessenek hozzá, akiknek azonosításához valamilyen jelszó szükséges. Amennyiben azt akarjuk elérni, hogy egy file-t csak egy program legyen képes elérni, akkor a file használata előtt azonosítsa magát a program az egyetlen jogosult felhasználóként, majd a művelet végeztével jelentkezzen ki. Mivel ez a felhasználó nem felel meg valós embernek, a jelszó csak a programban van benne, s azt senki nem ismeri. Így csak e program képes e file-t elérni.

- *Ne lehessen a kártyát definiálatlan állapotba hozni semmilyen inputtal sem!* Ennek megoldása sem túl komplikált. Sőt, ez a feltétel alapkövetelmény PC-s programok esetében is. Köznapi szóhasználattal élve ez a "majombiztosság".
- *Ne lehessen a kártyát definiálatlan állapotba hozni semmilyen parancskombinációval* (vagy alkalmazásfuttatás-kombinációval) *sem!* Ez egy nehezebb kérdés. Sajnos a kártya csak olyan programok kezelésére alkalmas, amelyek először egy löket adatot olvasnak a PC-től, majd egy löket adatot elküldenek a PC-nek, majd terminálnak. Ha PC-kártya párbeszédet akarunk megvalósítani, akkor a kártyán futó alkalmazásnak le kell tárolni az állapotát, s amikor legközelebb indítjuk, vissza kell olvasnia. Ebben az esetben a párbeszéd megvalósítható, viszont ekkor a PC felelőssége a kártya alkalmazását újraindítani. Ilyen esetben erőteljesen fellép a több alkalmazást kezelő kártyák esetén az a probléma is, hogy a programok össze ne akadjanak a kártyán. Az integritás biztosítására mindenképp tranzakció menedzsmentre van szükség, ami azonban a programok komplexitását jelentősen növeli.
- Ne lehessen a kártyát definiálatlan állapotba hozni reseteléssel vagy a tápfeszültség megvonásával. Ez talán a legnehezebb kérdés, s nem is biztos, hogy megoldható. Amennyiben a felhasználónak lehetősége van a kártyát az olvasóból bármikor kivenni, képes tetszőleges pillanatban leállítani a kártyán futó program futását. A hardvergyártó felelőssége az olyan processzor előállítása, amely ilyen esetben nem kerül definiálatlan állapotba. A kártya operációs rendszerével szemben szintén jogosan támaszthatunk ilyen követelményeket. Amennyiben ezek nem teljesülnek, az azt jelenthetné, hogy esetleg véletlenül sérülhetne a kártyán lévő adatok integritása. A PC-s operációs rendszer gyártójának felelőssége az, hogy az operációs rendszer ne álljon le, ha a felhasználó működés közben kiveszi a kártyát. Sajnos a Windows NT nem felel meg ezen követelménynek. A kártyás alkalmazás tervezőjének felelőssége pedig az, hogy lehetőleg ne lehessen kárt okozni a kártyás program adataiban ilyen módszerrel. Ez rendkívül nehezen biztosítható, különösen a kártya assembly nyelvének részletes ismerete nélkül. Szerintünk legjobb megoldás a problémára az, hogy a smartcard rendszer megtervezésekor olyan olvasókról gondoskodunk, amelyekből a felhasználó nem veheti ki a kártyáját a rendszer engedélye nélkül. (Pl.: bankkártya automaták)

4. Alkalmazási környezet

4.1. Bevezetés

Ezen fejezetünkben először bemutatjuk a hitelesség biztosításának öt fő pillérét, majd később a Mézga család példájával illusztráljuk, hogy a mindennapi emberek életét hogyan változtathatná meg egy világméretű smartcard alapú rendszer, s megmutatjuk, hogy a hitelesség biztosítása hogyan, mi módon jelentkezik az egyszerű emberek számára.

4.2. A hitelesség biztosításának öt fő pillére

4.2.1. Hitelességvizsgálat

Hitelességvizsgálat alatt azt értjük, hogy két fél úgy kommunikálhat egymással, hogy bármikor ellenőrizheti a partnerétől kapott üzenet hitelességét, vagyis azt, hogy tényleg a partnere küldte-e az üzenetet, s hogy tényleg azt küldte-e amit ő megkapott. Ezt titkos, csak kettejük számára ismert, információdarabka (titkos kulcs) segítségével érik el.

A módszer lényege abból áll, hogy az üzenet mögé odaírják a titkos kulccsal készített kriptográfiai ellenőrző összeget, amit csak a titkos kulcs és az üzenet segítségével lehet előállítani. Az üzenet fogadója is legenerálja a kriptográfiai ellenőrzőösszeget ugyanazzal a titkos kulccsal, s ha ezek megegyeznek, akkor biztos lehet az üzenet hitelességében és integritásában.

4.2.2. Hozzáférésvédelem

Hozzáférésvédelem alatt azt értjük, hogy egy felhasználó csakis akkor nyerhet belépést egy rendszerbe, ha előtte azonosítja magát a rendszer felé.

Az azonosítás történhet egy titok alapján, amit csak a jogosult személy és a rendszer ismer (pl.: jelszó), történhet tárgy alapján, amit csak a jogosult személy birtokol (pl.: kártya), illetve történhet biometriai jellegzetességek alapján (pl.: ujjlenyomat). Ennek egyik funkciója az, hogy a rendszer megfelelő jogosultságokkal látja el az felhasználót, a másik pedig az, hogy azonosítja, s megállapítja, melyik felhasználó ő a sok közül. Ha valaki sikeresen azonosítja magát az adott módszer segítségével a rendszer felé, az ezután azonos lesz azzal, akinek a rendszer felismerte. S akár ő az, akár nem, a rendszer nem fogja tudni megkülönböztetni tőle. A felhasználó azonosítása emellett jelenti azt is, hogy az illető beleegyezik valamibe. Például egy bank automatája esetén a PIN kód beírása jelenthet beleegyezést egy tranzakcióba.

A titok esetében legegyszerűbb módszer a jelszó, vagy PIN (personal identification number) kód. Ennek fő előnye, hogy a felhasználó a fejében tárolja, így tőle azt eltulajdonítani nem lehet. Ezek hossza az ISO 9564-1 szabvány szerint négy és tizenkét számjegy között lehet. A hosszú kódok azért jobbak, mert egy támadó próbálgatással kevésbé képes feltörni őket,

viszont ezeket nehezebb megjegyezni is. Sok emberben már kialakult a képesség négyjegyű PIN kódok memorizálására, viszont ha ezt a hosszat megnöveljük, a szokatlanság miatt komoly ellenállásba ütközünk. [Rank-Effing1997 *(8.1.1.)*] A PIN alapú hozzáférésvédelemnek előnyei:

- A kód nem eltulajdonítható
- Könnyen megvalósítható

Hátránya viszont, hogy nincsen túl sok variáció, próbálgatással elég hamar ki lehetne találni egy PIN kódot. Ezért szükséges az, hogy ha a felhasználó „sokszor" hibázik, akkor a PIN kódot letiltsuk. Ez viszont a tisztességes felhasználót is komoly stresszhelyzet elé állítja. Egy PIN-t alapjában véve nem nehéz megjegyezni, de sajnos ma az embereknek sok-sok jelszót s kódot kell fejben tartaniuk, s az a célszerű, ha mindegyik más. Nem csoda, hogy nemcsak a kezdőkkel, laikusokkal fordul elő, hogy elfelejtik a jelszavukat. A PIN kódnak hibája továbbá, hogy az puszta adat, így könnyen lemásolható. S mindenki, aki birtokában van a PIN-nek, s azonosítja magát vele, a rendszer számára azonos a PIN tulajdonosával. Így a PIN-re vagy jelszóra kínosan vigyázni kell, titokban kell tartani, s biztonságos helyen kell tárolni.
Akkor is különösen óvatosnak kell lenni, amikor beírjuk jelszavunkat, hiszen ha valaki megfigyeli azt, könnyen megszerezheti jogainkat. Megoldás a problémára a dinamikus jelszavak használata.
Ha tárgy végzi el az azonosítást, akkor felmerül annak a gondja, hogy a tárgyat ellophatják, s ekkor nem jogosult embereket is beengedhet a rendszer. Az a helyzet is előállhat, hogy a jogosult felhasználó otthon hagyja a tárgyat, s ekkor őt sem engedi be.
Tárgy esetében viszont megvan annak a lehetősége, hogy nemcsak a rendszer azonosítja a tárgyat (például intelligens kártya), hanem a tárgy is a rendszert.
Történt például már olyan bűncselekmény Magyarországon, hogy néhány találékony ember felállított egy hamis bankjegykiadó automatát. A gyanútlan bankkártyatulajdonos odalépett a hamis automatához, behelyezte a kártyáját, s beütötte a PIN kódját. Ezután az álautomata kiírta, hogy a kártya lejárt, s bevonja. Így a felhasználó „odaadta" a kártyáját a PIN kódjával együtt a bűnözőknek, akiknek csupán el kellett vinni a kártyát egy valódi automatához, s be kellett írnia a tulajdonostól kapott PIN-t, s így hozzáférhettek a tulajdonos pénzéhez.
Ilyen esetben például segíthet, ha nemcsak a rendszerhez tartozó készülék azonosítja a kártyát, hanem ha először a kártya azonosítja a készüléket. Jó, ha ilyenkor a kártya visszajelzést tud küldeni a felhasználónak.
Nem intelligens kártyák esetére is létezik megoldása e problémának, s ez a következő:
Van a kártyában egy titkos kód, amit a felhasználó a saját PIN-jével képes módosítani. A bank pedig képes ezt a számot olvasni, ha azonosítja magát. A felhasználó beteszi a kártyát az automatába, az pedig azonosítja magát a kártya felé, s kiolvassa a felhasználó által beállított kódot, s kiírja a kijelzőjére. A felhasználó elolvassa a titkos kódot, s ha felismeri, akkor tudja,

az automata sikeresen azonosította magát a kártya felé, s az automata tényleg a banké. Ekkor nyugodtan beütheti a PIN-jét.

Intelligens kártya esetén jóval egyszerűbb megoldani, hogy a kártya azonosítsa az olvasót. Annak a problémának a megoldása viszont továbbra is fennáll, hogy hogyan jut el az esetleges vészjelzés a felhasználóhoz.

Egyik legjobb azonosítási módszer a biometriai alapon történő. Ekkor az adott személy egyedi biometriai jellegzetességeit próbáljuk megfigyelni. Egy jó biometriai módszer esetében pedig fontos, hogy

- ne legyen eltulajdonítható
- minél biztosabban felismerhető legyen
- hatékonyan mérhető legyen
- kevés adatot kelljen tárolni
- a személy öregedésével ne változzon jelentősen
- mind a jellegzetesség, mind mérési módszer elfogadható legyen a felhasználók számára

Mi emberek is biometriai jellegzetességek – például arc – alapján próbáljuk azonosítani egymást. Elterjedt biometriai módszerek még a következők:

- Ujjlenyomat
- Aláírás
- Hangminta
- Arcfelismerés
- Retinalenyomat

Az ujjlenyomatot a rendőrség is régóta használja. Így nagy biztonsággal felismerhető. Hamisítani szinte lehetetlen. Két hibája van: az egyik, hogy eltulajdonítható: valakinek a kezét erőszakkal is rányomhatják egy ujjlenyomat felismerőre. A másik pedig éppen megbízhatóságából ered. Az emberekben sok-sok év alatt kialakult az a kép, hogy ujjlenyomatot a bűnözőktől vesznek. Ha egy biometriai alapon működő felismerő rendszert keresekedelmi alkalmazásokban akarunk használni, lényeges, hogy a rendszer bizalmat keltsen az emberekben, s ne viszolyogjanak annak használatától.

Aláírás felismerés esetén a probléma majdnem ugyanaz, mint a PIN kód esetében: a felhasználót megmérettetés elé állítjuk, s cselekednie kell ahhoz, hogy kiállja a próbát. Ez pedig ugyanúgy stressz alá helyezi.

Hangminta esetén is felmerül ez a gond. Másik problémája az, hogy nem eléggé biztonságos. Előnye viszont, hogy nagyon olcsó, és nem eltulajdonítható az adott személytől.

Az arcfelismerés nagyon drága, de látványos azonosítási módszer. Hibája, hogy viszonylag messziről is elvégezhető. Fontos egy azonosítási rendszerben, hogy az azonosítás megkezdése a felhasználó részéről tudatos félreérthetetlen lépés legyen, s így jelenthesse annak beleegyezését a rendszerbe való belépésre. Az például, hogy a felhasználó ráhelyezi a kezét az ujjlenyomat-felismerőre, kimondja a jelszavát, vagy aláír egy csekket, nemcsak azonosítási,

hanem beleegyezési funkciót is betölt. Az viszont, ha valakiről egy képet kapunk – esetleg messziről – távolról sem nevezhető beleegyezésének.
A retinalenyomat is a drágább módszerek közé tartozik. Ez viszont biztonságát tekintve az egyik legjobb. Infravörös sugarakat lőnek be a pupillán, s a visszavert fényből egy CCD kamera segítségével lehet következtetni az illető kilétére. Hibája, hogy az emberek általában féltik a szemüket, s nem szívesen teszik közel azt különféle berendezésekhez.

4.2.3. Rejtjelezés

Adat rejtjelezése alatt azt értjük, hogy egy transzformáció segítségével megváltoztatjuk annak alakját, természetét, méghozzá úgy, hogy az eredeti adatot csak az tudja visszaállítani, aki az inverz transzformáció birtokában van.
Maga a kódolás-dekódolás egy algoritmus (gép) segítségével történik. Ezt a gépet nagyon nehéz titokban tartani. S ha egyszer már kitudódott, milyen gépet használunk, a rendszer nem ér semmit, meg kell változtatni. Erre jó módszer az, hogy a gépünknek, algoritmusunknak van egy paramétere. Ezt nevezzük kulcsnak. A kulcs kicsi, rövid és könnyű változtatni. A jó rejtjelezőre az igaz, hogy akkor is megőrzi titkainkat, ha a támadó ismeri az algoritmust, s az egyetlen, amit nem ismer, az a kulcs, amit a rejtjelezéskor használtunk.
Ezt nevezzük Kerckhoff feltételnek. Tehát a támadóról feltételezzük, hogy mindent ismert a rejtjelezőnkről, kivéve az aktuális kulcsot.
A kulcsunkat egy kulcstérből választjuk. Ha ennek mérete kicsi, akkor a támadónak nincs nehéz dolga. Egyszerűen végig kell próbálgatnia az összes kulcsot az üzenetünk megfejtéséhez. Célszerű így, ha a kucsteret nagy méretűre választjuk, s ezzel lecsökkentjük a kimerítő kereséses (brute force) támadások esélyeit.
Két fő ágát ismerjük a rejtjelezésnek. Egyik a szimmetrikus (vagy titkos) kulcsú, másik az aszimmetrikus (vagyis nyilvános) kulcsú. Titkos kulcs esetén a kódolás és a dekódolás azonos, nyilvános kulcs esetén pedig különböző kulccsal történik.

4.2.4. Digitális aláírás

A digitális aláírás célját tekintve nagyon hasonlít a hitelességvizsgálathoz. Szintén arra szolgál, hogy egy kapott üzenetnek a hitelességét és integritását ellenőrizni lehessen. A különbség csupán az, hogy hitelességvizsgálat esetén két fél közös titkos kulcs segítségével kommunikálhat egymással hitelesen. Ekkor felmerül a kulcs biztonságos eljuttatásának problémája. Ha kettőnél több félből álló csoport akar egymással kommunikálni, akkor vagy minden két személynek van egy közös kulcsa, vagy egy közös kulcs van csupán, de ekkor nem megállapítható, hogy a csoportból ki küldte az üzenetet.
A digitális aláírás viszont nyilvános kulcsú rendszerben lehetséges. Ekkor minden résztvevő rendelkezik egy nyilvános és egy titkos kulccsal. Ha **A** hiteles üzenetet kíván küldeni **B**-nek, akkor annyit kell csak tennie, hogy az üzenetét (vagy egy tömörítvényét) elkódolja a saját titkos kulcsával, s odaírja az üzenet végére.

B úgy ellenőrizheti az aláírást, ha ő is képzi az üzenetnek ugyanazt a tömörítvényét, majd a kapott aláírást dekódolja **A** nyilvános kulcsával. Ha a két tömörítvény megegyezik, tudja, hogy az üzenet tényleg **A**-tól jött, s hogy **A** tényleg azt küldte.

4.2.5. Kulcsgondozás

Hogyha egy rendszerben kulcsok vannak, akkor őket védeni kell, hiszen ők jelentik annak fő gyenge pontját.

Titkos kulcsú rendszerben biztosítani kell, hogy a kulcshoz annak tulajdonosán kívül más ne férjen hozzá. Ha az egyik fél titkos üzenetet kíván küldeni a másiknak, el kell juttatnia hozzá a titkos kulcsot is. Figyelnie kell rá viszont, nehogy illetéktelen kezekbe kerüljön a kulcs, hiszen ekkor más is el tudná olvasni a titkos üzenetet.

Nyilvános kulcsú rendszer esetében a titkos kulccsal kapcsolatban szintén meg kell oldani a tárolás problémáját, de a titkos továbbítás itt nem merül fel. Felmerül viszont az a probléma a nyilvános kulccsal kapcsolatban, hogy el kell juttatni minden lehetséges partnerhez. Sőt, ekkor biztosítani kell e kulcstáblázat integritását is, hiszen ha valaki megváltoztathatja a kulcstáblát, akkor feltörhet üzeneteket. [Györfi-Vajda1991 *(Kript. 4.)*]

Mindkét rendszerben lényeges a kulcsok generálásának kérdése. Itt általában véletlen szám generátor használunk, de lényeges, hogy valódi véletlen számokat állítsunk elő. Az pedig igaz, hogy algoritmussal véletlent előállítani nem lehet. Szokás ezért billentyűleütések közti időt figyelni, vagy valamilyen módon egyéb emberi komponenseket is felhasználni valódi véletlen generálására.

Tehát a kulcsgondozás fogalma a következőket tartalmazza:

- Kulcsok generálása
- Kulcsok tárolása
- Kulcsok továbbítása

4.3. Mézga Géza egy napja

Reggel fél hétkor Mézga Géza az Aida nyitányára ébred, mert neki ez a kedvence, s az ébresztőóra tudta ezt, mert Géza előző este beletette a smartcardját. Miközben öltözködik, a smartcardot behelyezi a videofonba, mely előfizetésének ellenőrzése után ellátja őt a legfrissebb hírekkel, tőzsdei információkkal. A kártya természetesen ellenőrzi az információk *hitelességét* is. Mielőtt elhagyná a lakást, a kártya segítségével aktiválja a riasztót. Ezután a kártyával bezárja a lakást, és lemegy a liften a garázsba. A kártyával kinyitja és elindítja az autóját, és elmegy a munkahelyére.

Munkahelyére menet egy kicsit rálép a gázra, s egy rendőr le is inti. Géza átadja kártyáját a rendőrnek, aki olvasója és a rendőrség központi számítógépe segítségével megállapítja, hogy Gézának már nem ez az első gyorshajtása, sőt, a múlt héten tilosban is parkolt. Gézának nem kell attól tartania, hogy a rendőr több pénzt emel le a kártyáról, vagy esetleg lekéri az egyenlegét, megszerzi a jelszavát. A kártya *védi a kulcsokat* s az adatokat. Miután a

rendőrségi olvasó leemeli a kártyáról a bírság összegét, a rendőr visszaadja Gézának a kártyáját, aki boldogan teszi azt zsebre.

Ahogy lihegve beér a munkahelyére, beteszi kártyáját az ajtóban lévő olvasóba, s azonosítja magát a hivatal *hozzáférésvédelmi* alkalmazása felé. A retina-leolvasó ellenőrzi Géza személyazonosságát a kártyáján lévő retinaminta alapján, s megállapítja, hogy hősünk már a héten harmadszor késett el. Íróasztalához érve Géza a telefonba helyezi kártyáját, s az mostantól a neki szóló hívásokat erre a készülékre irányítja át. Géza rögtön meg is hallgatja az előző nap óta az ő számára érkezett üzeneteket. Ezután kártyája segítségével Géza bejelentkezik a számítógépébe, illetve a belső vállalati hálózatra. Az orvosi adatok ellenőrzése alapján Mézga Géza kap egy figyelmeztetést, hogy másnap tüdőszűrésre kell mennie.

Egy közeli étteremben ebédel kollégájával, aki a Máris és tsa cégnél dolgozik. Az ebédet kártyájukkal fizetik. Mivel emellett egy hosszútávú üzleti együttműködésben is megállapodnak, az étterem egyik számítógépén (persze először kártyáikkal ellenőrzik az éttermi szoftver integritását) megírják a szerződéstervezetet, melyet mindketten kártyájukkal *digitálisan aláírnak*, s elektronikus formában juttatják el cégeikhez.

Délután hazamegy, és kártyájával élelmiszereket rendel a szupermarketből. Egyben egy figyelmeztetést is kap, hogy az a fajta sajt, amit régebben gyakran rendelt, most újra kapható. Este az egész Mézga család tévét néz a kártyán lévő dekódoló alkalmazás segítségével. Mivel a képminőséggel nincsenek megelégedve, utasítják a kártyát, hogy a szolgáltatónak panaszt tegyen.

Késő este Paula felhívja régi ismerősét, Huffnagel Istvánt, és egy titkos beszélgetést folytat le vele, saját smart cardját használva a beszélgetés *rejtjelezésére*. Mivel a telefonok nyilvános kulcsú titkosítási algoritmust használnak, Paula úgy is beszélhet Huffnagel Pistivel titkosan telefonon, hogy nem osztottak meg titkos kulcsot előtte. Kártyája segítségével megállapítja, hogy másnap délután ráér, így akkorra beszélnek meg találkozót.

Az előzőkben megmutattuk, hogyan alakíthatná át a smartcard technológia egy mindennapi család életét. A későbbiekben majd megmutatjuk, ebből mi lehetséges, s mi nem. Ezen fejezetünk alapötletét [Zoreda-Oton1994 *(3.)*]-ből kölcsönöztük.

5. A mi fejlesztésünk

5.1. Bevezetés

Mi a Microsoft Smart Card for Windows nevű programozható smartcard bétaverziója segítségével hoztunk létre kártyán futó alkalmazásokat. E fejezet elején leírjuk a fejlesztőrendszert, amivel dolgoztunk, majd elkezdjük az általunk készített applikációk ismertetését. Részletes méréseket végeztünk a kártya különféle részeinek sebességéről, s ezek eredményeit is e fejezetben ismertetjük. Ezután bemutatjuk legkomolyabb alkalmazásunkat, a kártyán futó, kulcsot a kártyában biztonságosan tároló DES programcsomagunkat. Elmagyarázzuk működési elvét, s röviden ismertetjük PC oldali felhasználói felületét. Bemutatjuk, ezen csomag segítségével hogyan valósítottuk meg a hitelesség biztosításának öt fő pillérét.

5.2. A fejlesztőrendszer bemutatása

5.2.1. Magas szintű nyelv

A Microsoft ezzel a kártyával a Java Cardokat szeretné legyőzni. A Java Cardban nemcsak az a pozitívum, hogy a java 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 valahol félúton meggondolták magukat (talán a C-t túl komplex nyelvnek találták), s a Visual Basic mellett döntöttek. (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.

Ezek szerint a programozónak egyrészt nem kell új nyelvet kitanulnia, másrészt használhatja a Visual Basic fejlett fejlesztői környezetét. Természetesen a basic-kel kapcsolatban is felmerültek a java-hoz hasonló problémák. 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.

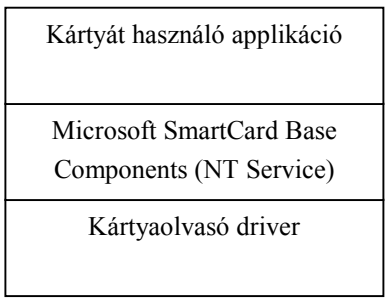
5.2.2. Korlátozások a Visual Basichez képest

Nem használhatunk például string változókat, lebegőpontos aritmetikát (ezt azért, mert a kártya processzora nem tudja), változó méretű tömböket. Komoly hiányosság (s ezt a Microsoftos dokumentáció is elismeri), hogy nem lehetséges a kártyán semmilyen hibakezelés. Amennyiben a kártyán futó program hibára jut, hibaüzenettel terminál. Ahhoz,

hogy a WinCardra komoly, az üzleti életben is használható alkalmazást lehessen fejleszteni, ezt a Microsoftnak mindenképpen korrigálni kell. Jelen helyzetben ugyanis, ha a kártya nem kap annyi byte üzenetet, amennyit vár, akkor a rajta futó program lefagy. Ez jelentősen csökkenti a WinCard biztonságát, s a programozónak figyelnie kell arra, hogy rosszindulatú programok ilyen támadásokkal ne tudják definiálatlan helyzetbe állítani a kártyát.
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 (for ciklus, while, if-then-else,case, stb). Támaszkodhatunk viszont a kártya filerendszerére s kriptográfiai koporcesszorára, s ennek kezelésére külön könyvtári függvények vannak a fejlesztőrendszerben. A kártyán futó könyvtári függvények a következő csoportokra oszthatók: felhasználók azonosítása, kriptográfia, file kezelés, kommunikáció a PC-vel.

5.2.3. Illeszkedés a Windowshoz

A Microsoft a WinCardot tulajdonképpen Windows 2000 alá fejlesztette ki. Csakhogy e szoftvernek van egy aprócska szépséghibája, mégpedig az, hogy még nincs. Így a WinCard toolkit hajlandó futni NT alatt is 4-es Service Packkel.
A WinCard igen szervesen illeszkedik az NT rendszeréhez, olyannyira, hogy adtak ki hozzá példaalkalmazásokat, melyek csak egy megfelelő WinCard segítségével teszik lehetővé az NT-be való bejelentkezést. Egy másik példa alkalmazás ugyanezt teszi, csak telefonon keresztül, s a RAS-ra kapcsolódik rá.



1. Ábra

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. Ez a SmartCard Base Components kapcsolódik majd az olvasó driveréhez, s ez kezeli azt.

5.2.4. Emuláció

Kártyán fejleszteni alkalmazást nem könnyű. A kártyaolvasó ugyanis soros portra csatlakozik, s a kommunikáció igen lassú. Hosszú, amíg az alkalmazásunkat letöltjük a kártyára, s lassú, amíg az lefut. Ráadásul egy EEPROMot nem lehet akárhányszor írni sem. Mindenképpen kellemes dolog, hogy a fejlesztőrendszerhez tartozik egy emulátor is, s így a programot először a PC-n próbálhatjuk ki, s elég a végén a már kész programot letölteni a kártyára.

2. Ábra – Választási lehetőség egy program indításakor

Sajnos az élet nem ilyen szép, hiszen sajnos jelentős különbség van a kártyán futó program és az emuláció között. Emuláció esetén például a timeout lehetősége nem foroghat fenn, kártya esetén viszont igen. Ha a program kártyán fut, akkor feltétlenül ügyelni kell rá, hogy a kártya pontosan annyi byte-ot fogadjon, amennyit a kártyát kezelő program elküld. Amennyiben nem ez történik, vagy a nem fogadott byte-ok pattannak vissza a feladóhoz, vagy pedig a kártyán futó program fagy le.

5.3. Néhány szó a kártyáról

5.3.1. Bevezetés

A kártya, amivel mi foglalkoztunk, a Microsoft Smartcard of Windows 1.0 nevet viseli, s 1999 májusában bocsátották ki béta verzióként a hozzá tartozó fejlesztőrendszerrel együtt. Néhány nappal e dolgozat beadása előtt kaptuk meg a következő, szeptemberi verziót.
A smartcard az adatbiztonság egyik kulcsa lehet a jövőben. Ez az egyik ok, amiért a Microsoft beleszállt ebbe az üzletbe is. A másik ok pedig az, hogy a Sun már benne van. Voltunk olyan szerencsések, hogy hozzájutottunk egy Microsoft-féle WinCardhoz, s erre alkalmazásokat is fejleszthettünk. Ez a kártya képességeit tekintve a jelen kor csúcsának felel meg.

5.3.2. A kártya képességei

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 számára s 1 kilobyte SRAM-mal a változók, dinamikus adatok, stack, stb számára. Rendelkezik továbbá egy ATMEL kriptográfiai műveleteket támogató koprocesszorral is, mely megvalósítja a DES, triple-DES, RSA, SHA és CRC műveleteket.
A WinCard egy intelligens kártya, képes több applikáció tárolására, elkülönítésére. Futtatni viszont egyszerre csak egyet képes. Ugyanakkor használható hagyományos ISO 7816-4 kártyaként a szabványos APDU-kkal. 127 felhasználót tud elkülöníteni egymástól, s rendelkezik filerendszerrel is, amelyben minden file-hoz hozzáférési listákkal adhatjuk meg, ki min milyen műveletet végezhet. A kártya erejét a már említett két bástya képezi: az applikációk és a filerendszer.

5.3.3. File rendszer

A WinCardot a Microsoft úgy hirdeti, hogy kártyája nem más, mint egy Windows NT, csak konzol nélkül. 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.
A jogosultságokat hozzáférési listával (access control list) adhatjuk meg, amelyek a kártya /s/a/ alkönyvtárában helyezkednek el. Ebben a könyvtárban minden file egy-egy hozzáférési lista, melyek megadják, hogy milyen műveleteket milyen felhasználók jogosultak végrehajtani. Ezt legfeljebb kétszintű boole formulákkal adhatjuk meg. Egy felhasználó „értéke" akkor igaz, ha az illető be van jelentkezve, egyébként hamis. Egy művelet végrehajtása csakis akkor lehetséges, ha egy hozzá tartozó boole formula igaz.

```
FILE /s/a/points ACL /s/a/administration
ACL_DATA RESOURCEOPERATION_FILE_READ DISJUNCTIVE {1 Customer 1 MERCHANT}
ACL_DATA RESOURCEOPERATION_FILE_INCREASE      DISJUNCTIVE {1 MERCHANT}
ACL_DATA RESOURCEOPERATION_FILE_DECREASE      CONJUNCTIVE {1 Customer 1 MERCHANT}
```

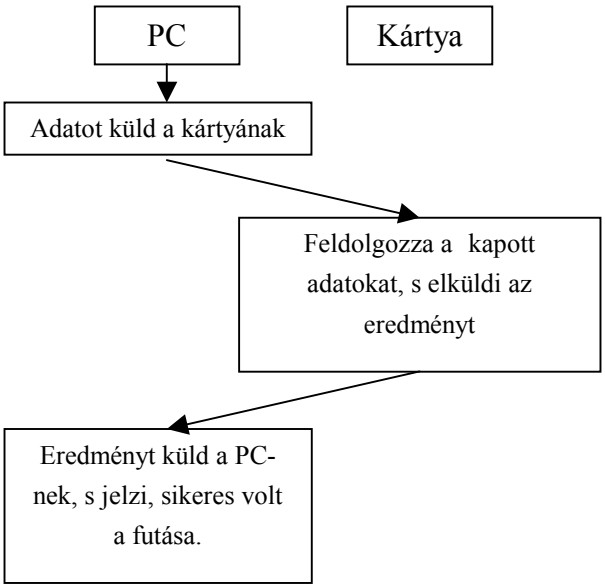
Ezek a sorok például egy loyalty rendszerből származnak, s azt jelentik, hogy a points nevű hozzáférési listához kapcsolt fileokon mind a kereskedő, mind az ügyfél végezhet olvasás műveletet, a fileokban lévő érték növeléséhez a kereskedő szükséges (az ügyfél vásárol a kereskedőnél), a csökkentéshez (a vásárló felhasználja a pontjait) viszont mindkettőjüknek jelen kell lenni. Minden file-hoz kapcsolódik egy ilyen hozzáférési lista. Sőt! Hozzáférési lista tartozik a hozzáférési listákhoz is. A fenti példa listájához egy másik (administration nevű) kapcsolódik. Egy jól megtervezett filerendszerben pedig egyik filehoz sem tartozik az alapértelmezés szerinti „mindenkinek mindent szabad" /s/a/default access control list file.

A kártya több felhasználó kezelésére képes. Maximum 127 különböző „ismert felet" (known principal) képes elkülöníteni. Ezek a felhasználók és a felhasználó csoportok. A felhasználókat jelentő file-ok tartalmazzák, hogy a felhasználó azonosítása milyen módon történhet. Ez lehet PIN kód vagy DES alapú challange and response.

```
FILE /s/k/Customer ACL /s/a/administration
KP_DATA {NO_GROUPS} PIN {1 2 3 4}
```

Ez a két sor itt az ügyfél nevű known principalt definiálja. Az ügyfél azonosítására itt PIN kód szolgál. A felhasználók nyilvántartását a /s/k/ könyvtár végzi. Itt minden egyes file megfeleltethető egy-egy felhasználónak, vagy felhasználócsoportnak. Amikor egy felhasználó (ember vagy gép) azonosítja magát a kártya felé, akkor a kártya ezt megjegyzi, s a kártyán futó programok képesek végrehajtani mindazon műveleteket, amelyekhez az adott felhasználónak joga van. Az illető addig bejelentkezve marad, amíg ki nem jelentkezik, vagy amíg a kártya nem resetelődik.

5.3.4. Applikációk



3. Ábra – PC-kártya kommunikáció

A WinCard intelligens kártya, s képes applikációkat futtatni. Minden applikáció rendelkezik egy-egy számmal (a /c00.rte a 0-val, a /c01.rte az 1-gyel, stb), s őket egy APDU-val lehet elindítani, mely paraméterként ezt a számot kapja meg.

A kártya nem képes applikációk párhuzamos futtatására, képes viszont több alkalmazás elkülönítésére, s egymás után való futtatására. Igaz, a kártya filerendszerének elkészítése előtt meg kell tervezni, milyen alkalmazások kerülnek majd rá, s létre kell hozni a hozzájuk szükséges felhasználókat, csoportokat, hozzáférési listákat. Tölthet be tehát egy kártya több funkciót (lehet személyi igazolvány, villamosbérlet, telefonkártya és hitelkártya is egyben), de ezeket előre el kell tervezni (nem lehet tehát egy villamosbérletnek azt mondani, hogy mostantól legyen bankkártya is).

A kártyán futó és a kártyát kezelő alkalmazás sajnos nem tekinthető két párhuzamosan futó processznek. WinCardra ugyanis (e fejlesztőrendszer segítségével) csak olyan programot lehet írni, amely a 2. ábrán látható séma szerint működik:

Így a működés sokkal inkább függvényhívás szerű. Nem válaszolhat tehát a PC a kártya által küldött csomagokra. Így nem lehet WinCardra írni olyan alkalmazást, amely esetén a kártya kérdéseket tesz fel a PC-nek, s feljegyzi a válaszokat. Ilyet végrehajtani csak olyan módon lehet, hogyha a kártya programja kilépés előtt eltárolja saját állapotát, s a PC-n futó program ismét elindítja őt. Ez persze ekkor már a PC felelőssége...

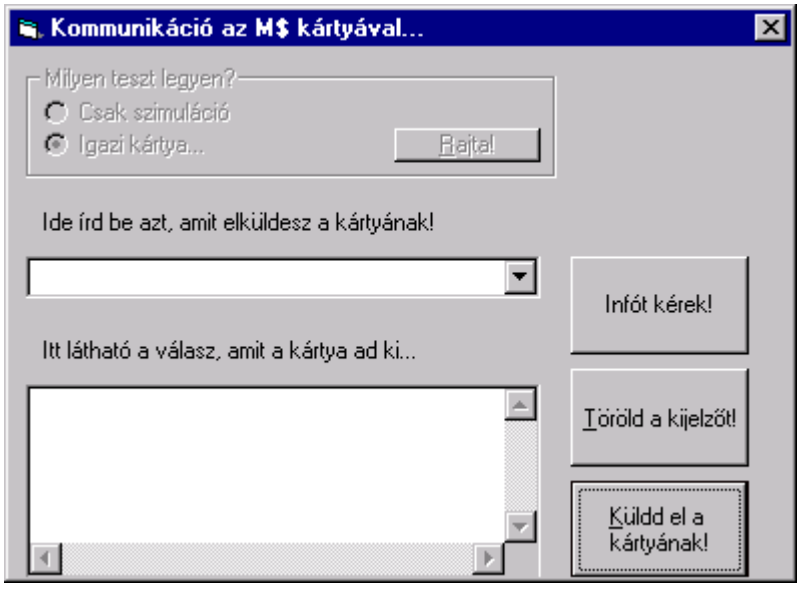
5.4. Windows host – Smartcard

Amikor azt írtuk, hogy kártyára fejleszteni alkalmazást, az úgy hangzott, mintha egyetlen egy alkalmazásról lenne szó. Pedig ez nem így van. Egy intelligens smartcardra írt program legalábbis két részből áll. Az egyik része az, ami a kártyán fut, s a műveleteket végzi a kártyán lévő adatokkal. A másik része pedig PC-n fut, s a felhasználó kéréseit továbbítja a kártya felé, illetve szép, grafikus formában fogalmazza meg a kártya válaszait.

5.5. Első programjaink

5.5.1. Terminál

Egyik programunk Windows alatt fut, s a kártyát kezeli. Nem más, mint egy egyszerű terminál, mely az inputként kapott stringet egyszerűen, egy az egyben elküldi a kártyának, majd a kapott választ kiírja. Az elején választhat a felhasználó, hogy emulált vagy valódi kártyával akar kommunikálni, majd megkezdődik a dolog lényegi része. A beírt stringet a terminál elküldi a kártyának (a protokoll szerinti formában), hogy az értelmezhesse azt. Az „infót kérek” gomb lehetőséget ad a „?” üzenet gyors kiadására. A kártya válaszát a terminál szöveges módban jeleníti meg, s csak a hibaüzeneteket fejti ki. Terminál programunk általános célú alkalmazás, a fejlesztés megkönnyítése végett fejlesztettük ki. Célja, hogy bármilyen, a kártyán futó programot könnyen tesztelhessünk a segítségével, s nem az, hogy egy konkrét programhoz felhasználóbarát interface-t adjon.



4. Ábra – a terminálablak

5.5.2. Számláló

Egyik programunk egy számlálót valósít meg a kártyán. Nagyon hasonló ahhoz, mint amit a telefonkártya programomnál leírtunk. Annyi a különbség, hogy ez a számláló 0-ról indul s felfelé számol. Ha elér egy értéket (99), akkor nem számol tovább. A következő műveletek értelmezhetők rajta: *inicializálás, növelés, érték lekérdezése*. Ilyen módon illegális érték nem írható bele. Jelen beállítások szerint mindenkinek joga van minden művelet végrehajtásához, de könnyen fejleszthető olyan irányba, hogy mondjuk inicializálni csak bizonyos PIN kód segítségével lehessen.

5.5.3. Zseb

A „Zseb” nevű programunk adattárolást valósít meg a kártyán. A program 10 db file-t (zsebet) kezel a kártyán. Név szerint /zseb0.dat - /zseb9.dat. Ezeket tudja a felhasználó írni, olvasni és törölni. A kártya, amin a Zseb program fut, funkcionálisan nem tud többet egy hagyományos kártyánál, viszont lényeges a különbség a belső működésben. Ugyanis itt nem a

PC írja bele az adatokat a kártyán lévő fileba, hanem csupán egy bytesorozatot küld el a WinCardnak (a megadott protokoll szerint), s a kártya ezt értelmezi, ennek alapján cselekszik, s dönti el, hogy írni vagy olvasni kell-e most.
parancsok:
c[x]: törli a(z) x. zseb tartalmát
r[x]: kiolvassa a(z) x. zseb tartalmát, s elküldi a felhasználónak
w[x]<10 byte>: beleírja a 10 byte tartalmát a(z) x. zsebbe

5.6. Az általunk kidolgozott protokoll

Mivel az emuláció eléggé finnyás, kidolgoztunk egy protokollt a kártya és a PC között, melynek segítségével a PC közölheti a kártyával, hány byte-ot fog küldeni, s a kártya visszajelezhet. Az első byte, amit a PC elküld a kártyának, azt jelzi, hány byte következik még utána. Az ez után következő byte a parancs. A protokollban csak azt rögzítettük le, hogy ha itt kérdőjel következik, akkor a kártya válaszképpen 3-4 byte-ban elküld egy azonosítót, melyből kiderül, melyik programunk fut a kártyán éppen.
A kártya válaszát is rögzítettük. Ha az első byte 0, akkor nem volt hiba. Az 1-es szintaktikai hibát jelent, tehát hogy a kártya nem értette meg a parancsot. A 2-es pedig azt jelzi, hogy a hiba filekezelés közben történt, stb…

5.7. Mérési eredményeink

5.7.1. A kártya számítási sebességének mérése

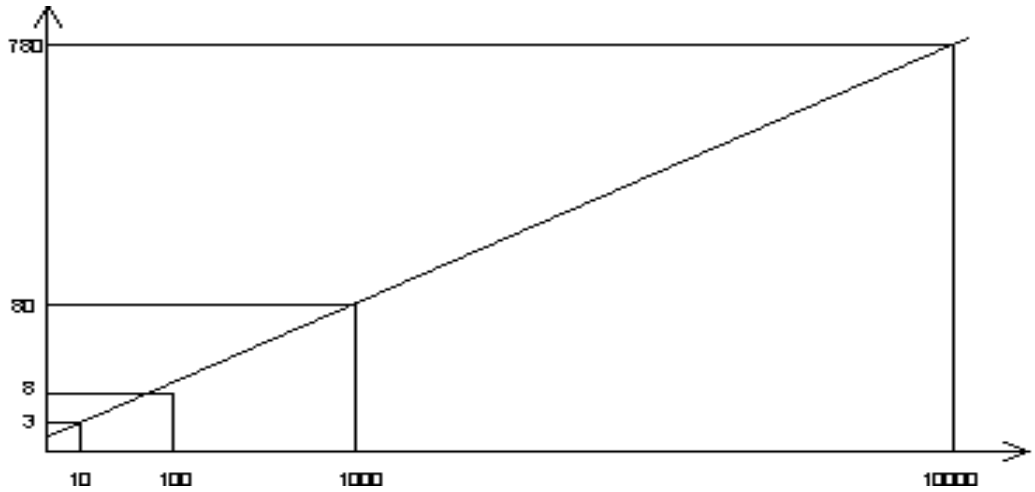
A kártya processzorának számítási sebességét próbáltuk mérni. Erre egy külön applikációt készítettünk, mely az input hosszától exponenciálisan függő számú lépésre kényszeríti a processzort. N hosszú input esetében n db byte fogadását és 10^n db inkrementálás műveletet kellett elvégeznie. Az ez alapján kapott eredmények:

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 – számítási sebesség teszteredményei

Feltételezésünk szerint a fenti 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).
Az overhead feltehetőleg nem függ attól, hogy a kártyának hány műveletet kell végeznie, ezért összességében is egy lineáris időfüggésre számítunk. A fenti adatokra tehát y=ax+b

alakú egyenest szeretnénk illeszteni, ahol „a” az egy művelet elvégzéséhez szükséges idő, „b” pedig az overhead, „x” a műveletek száma, „y” pedig az idő másodpercben.
Mivel a mérési pontatlanság nagyobb x-ek esetén kisebb relatív hibát okoz, ezért célszerű az x=1000 és x=10000-hez tartozó pontokra illeszteni az egyenest. Így a=7/90≈0.08 és b=20/9≈2.2 adódik. Ilyen „a” és „b” értékek mellett y(10)=3 és y(100)=10 adódik, ami azt jelenti, hogy ez az egyenes jól illeszkedik mérési eredményeinkre. Eszerint egy művelet



5. Ábra – a processzor számítási sebességének mérése

elvégzése 80 millisekundumot vesz igénybe, az overhead pedig kb. 2 másodperc.
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 tesztet egyébként a következő basic szubrutin végezte:

```
Sub matek()
    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
```

E mérési módszerünk elég látványosan bemutatja az inkrementálás műveletek sebességét. Valószínű viszont, hogy a különböző műveleteket különböző idők alatt hajtja végre a kártya.

Azért az inkrementálást választottuk mérésünk eszközéül, mert az megítélésünk szerint eléggé tipikus és eléggé alapvető művelet egy programban.

5.7.2. A kártyában lévő kriptográfiai koprocesszor sebességének mérése

Az inkrementáláshoz képest egyáltalán nem nevezhető elemi műveletnek a DES számítása. Viszont célalkalmazásunk, az egyik későbbi fejezetben szereplő DES csomag pont ezt használja, s a kriptográfiai műveletek a smartcard technológia egyik értelmét adják. Természetesen a különböző kriptográfiai műveleteket különféle idők alatt végzi el a kártya. A kártyán létező DES, TRIPLE DES és RSA titkosító műveletek közül viszont egyértelműen a DES a legegyszerűbb, s legelemibb. Ezért ezt választottuk tesztelésünkhöz.
A tesztelést az általunk készített, és a következőkben bemutatott DES csomag ECB kódoló funkciójával végeztük. Tehát az inputot a PC nyolc byte-os blokkokra vágta fel, s egyenként küldte el a kártyának. A kártya a rejtjelezett nyolc byte-os csomagokat visszaküldte a PC-nek, s a PC ezt kijelezte. A következő eredményekre jutottunk:

1 DES	6 másodperc
2 DES	11 másodperc
3 DES	15 másodperc
5 DES	25 másodperc
10 DES	50 másodperc

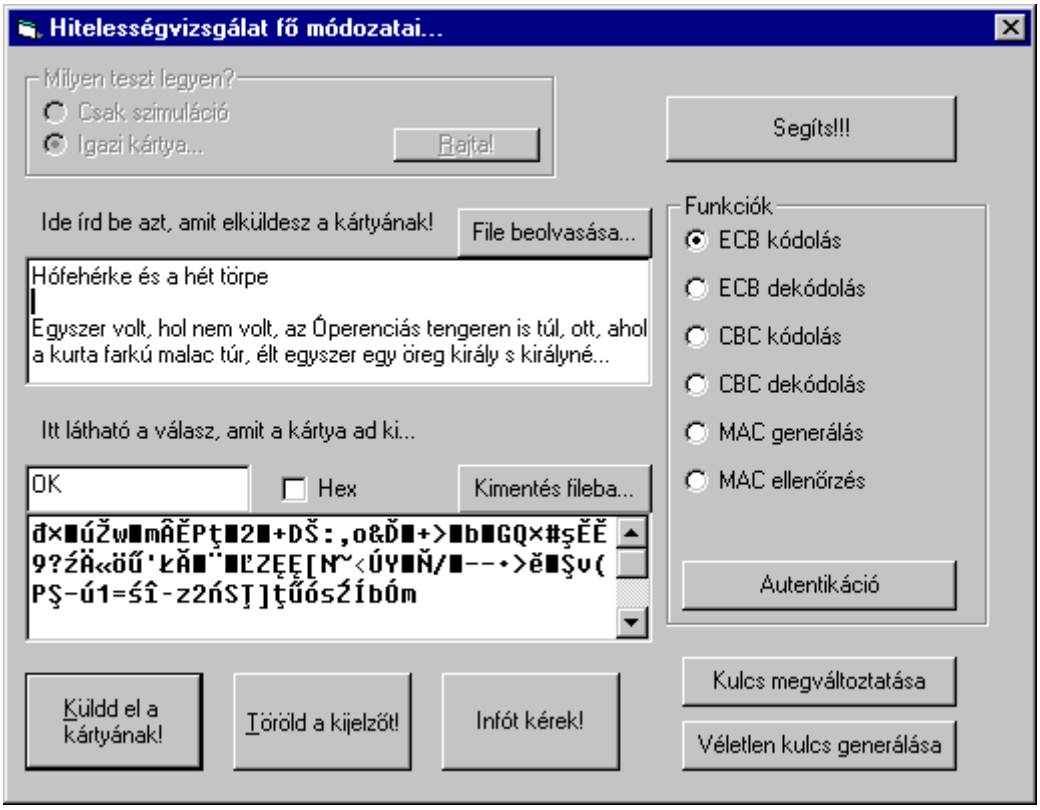
2. Táblázat – A kriptokoprocesszor sebességtesztje

A kártya által végrehajtott művelet ugyanaz volt, ha 1 db DES-t számítottunk, mintha 10-et, egyszerűen csak többször hajtottuk végre. A blokkokra tördelést, s a blokkok egymás után való elküldését a PC végezte el, de úgy ítéljük, ennek ideje kevéssé számított. Jelentős időt vett viszont igénybe a Visual Basic ablakok megjelenítése s az inicializálás. Szintén befolyásolja a sebességet az input beolvasása és az output kiírása. Az I/O műveletek ugyanis nagyon lassúak. E táblázat alapján egy DES művelet körülbelül öt másodperc hosszú, míg az overhead 1 másodperc.

Egyik korábbi programunkban MAC-t számítottunk a kártya segítségével, s akkor drasztikusan rosszabb eredményeket értünk el. A különbség az volt az akkori MAC számítás és a mostani között, hogy akkor a visszacsatolást a kártyán belül valósítottuk meg. Tovább lassította a dolgokat, hogy a kártyán hibásan működnek azon bitműveletek, melyeknek mindkét operandusa változó. Így a modulo 2 összeget képző visszacsatolást nekünk kellett kézzel megvalósítani.
Régebbi programunkban 25 másodpercbe telt MAC-t számítani minden nyolc byte után. Ez itt 7 másodpercre csökkent. A nagy időkülönbség oka kétségkívül a processzor alacsony számítási sebessége.

5.8. DES csomag

5.8.1. A csomag bemutatása



5. Ábra – A DES csomagunk PC-s felhasználói felülete

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 véletlen számgenerátor

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.

Ez természetesen nem a véletlen műve. A PC-n bárhol helyeznénk is el a kulcsot, egy támadónak lenne esélye arra, hogy megtalálja azt. Feltételezésünk az, hogy 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.

Tehát a kulcs a kártyán biztonságban van - legalábbis a kártya előállítójának állítása szerint.

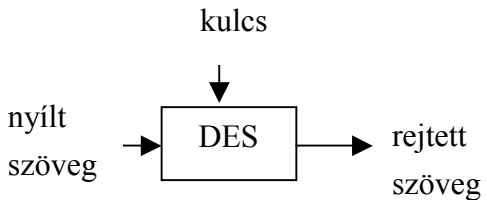
Ki elől van biztonságban? A válasz: mindenki elől. Nemcsak a támadó képtelen hozzáférni a kulcshoz, hanem a kártya gazdája is az. Hiába van a zsebében a kártya, nem képes kinyerni a titkos kulcsot belőle, csak használni tudja azt. Mivel a kártya nem hajlandó kiadni magából a kulcsot, az egyetlen esély 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. Ha a kártya felhasználója ezt a parancsot adja ki smartcardjának, akkor ezután a következő ismeretekkel rendelkezik:

- kulcs van a kártyán
- a kulcs, ami a kártyán van, nem azonos azzal a kulccsal, ami eddig volt a kártyán
- a kulcs bitjeinek eloszlása egyenletes

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 kibocsájtójának - sem lehetnek ismeretei a kulcsot illetően. Így e rendszer - a DES ereje és feltételezéseink alapján - biztonságosnak nevezhető.

A hitelesség megállapításának főbb módszerei közül számos megvalósítható DES csomagunk segítségével. Úgymint: hitelességvizsgálat, rejtjelezés, dinamikus jelszavak. S mivel DES esetén titkos kulcsokról van szó, felmerül a kulcsgondozás problémája is, amiben a kártya igencsak hathatós segítségünkre lehet.



6. Ábra – DES doboz

A kártyán egy közönséges DES kódolót valósítottunk meg. A bemenete a 8 byte-os nyílt szöveg, kimenete pedig a szintén 8 byte-os rejtett szöveg. Paramétere a titkos kulcs.

Ezt a rendszert egy PC-s program használja. Ez a program készít a DES elemből egy a gyakorlatban is használható eszközt. Ez a program építkezik a DES dobozból, s egy komplexebb rendszert alakít ki. A PC felelőssége az esetleg hosszú inputot 8 byte hosszú

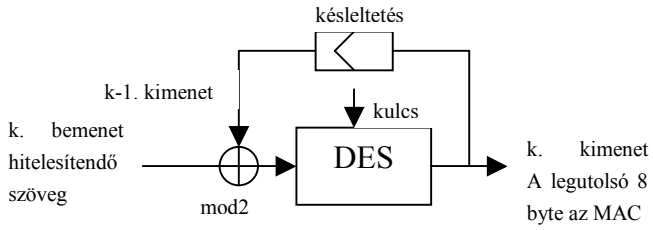
blokkokra tördelni, s őket a fekete doboznak tekintett kártyának elküldeni, majd a rejtett szöveget feldolgozni, s esetleg valamilyen formában a bemenetre visszacsatolni. (pl.: CBC, MAC)
Így egy egyszerű DES elemből egy sok célra felhasználható eszközt készítettünk, amire támaszkodva a hitelességvizsgálat főbb módozatait tekinthetjük át.

5.8.2. Hitelességvizsgálat

A hitelességvizsgálat megvalósítható úgy, hogy ez az információdarabka egy 56 bit hosszú DES kulcs. Amennyiben **A** fél ellenőrizhető hitelességű üzenetet szeretne küldeni **B** félnek, akkor megteheti a következő lépéseket:

1. A titkos kulcs segítségével legenerálja az üzenet nyolc byte hosszú MAC sűrítményét.
2. Az MAC-t odaírja az üzenet végére.
3. Elküldi az így keletkezett új üzenetet.

Ha B fél ellenőrizni szeretné az üzenet hitelességét, nem kell mást tennie, mint leválasztania az üzenet végéről a kapott MAC-t, s a titkos kulcs segítségével újra legenerálnia azt. Mivel ugyanaz a titkos kulcsa, mint A-nak, s az üzenet is ugyanaz, ugyanarra az eredményre kell jutnia, mint A-nak.
Amennyiben az üzenetet menet közben valaki módosította, úgy B nem azt az MAC-t kapja, mint ami az üzenet végén volt. Pontosabban kaphatja ugyanazt is, de ez csak a véletlen műve lehet, s ennek valószínűsége igen kicsi ($1:2^{64}$, ami igen kis szám, s egy ekkora valószínűséggel bekövetkező eseményre nem igen kell számítani)
Az MAC számításának elmélete a jobb oldali ábrán található. Visszacsatolt rendszerben működik a DES, s a következő kimenetet az aktuális bemenet és az előző kimenet modulo 2 összege adja. Ha a teljes kimenetet vesszük, akkor a kapott eredmény a CBC rejtjelezés, de ha csak az utolsó nyolc byte-ot, akkor azt nevezzük MAC-nek. Az MAC egyfajta kriptográfiai ellenőrző összeg, amit a titkos DES kulcs segítségével állítottunk elő. Az MAC függ a hitelesítendő szöveg minden bitjétől. [Davis-Price1992 *(5.5.)*]



7. Ábra – MAC számítás

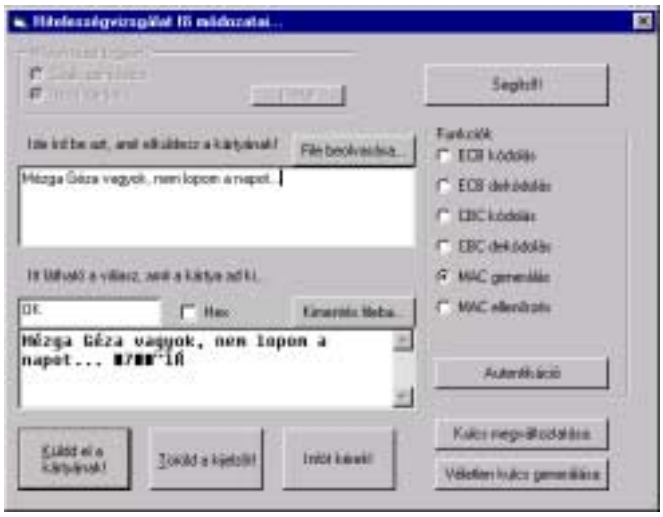
Jelen pillanatban a kártyán a rendszerből egyedül a DES doboz és a kulcs van. Voltak próbálkozásaink az egész rendszer kártyára helyezésére is, de jelentős sebességcsökkenés volt akkor tapasztalható. Erről bővebben a mérések fejezetben írunk.
A másik ok, amiért a visszacsatolás a PC-ben került megoldásra az az, hogy az egy APDU-ban küldhető adat hossza 40 byte-ra korlátozott. Ez kikerülhető lenne ha a kártyában állapotot

tárolnánk, de ez jelentősen megnövelné a rendszer komplexitását, s így csökkentené annak megbízhatóságát és sebességét. A visszacsatolást és az XOR műveletet a kártyán kívül, a PC-ben valósítottuk meg.

A kártyát kezelő PC-s program felelősségéhez tartozik az input blokkokra tördelése, a kártyás alkalmazás kódoló eszközként való használata, s a az output feldolgozása. Továbbra is kihasználjuk viszont a kártya előnyét: nem ismeretes a PC-s program számára a kártyában tárolt titkos kulcs. Így a kártya szükséges az MAC generálásához vagy ellenőrzéséhez.

A felhasználói felület is támogatja mind az MAC generálását, mind annak ellenőrzését. Ha az input ablakba beírunk egy szöveget, s MAC-t generáltatunk, az output ablakban feltűnik a szöveg, s mögötte a nyolc byte MAC. Ezt az üzenetet küldhetjük el társunknak, aki rendelkezik egy pont olyan kártyával, mint mi, s a kártya tartalmazza ugyanazt a titkos kulcsot, mint a miénk.

Társunk dolga ekkor annyi, hogy beviszi a kapott szöveget az input ablakba, s MAC-t ellenőriztet a programmal. A PC ekkor leválasztja a 8 byte MAC-t a szövegről, kiszámítja a maradéknak az MAC-jét, s összeveti a kettőt, s jelzi az eredményt. Ha nem egyeznek, társunk arra következtethet belőle, hogy az MAC-t más kulccsal generálták, nem azzal, amit mi megosztottunk vele, vagy pedig az üzenet tartalmát változtatták meg. Tehát az üzenet nem tekinthető hitelesnek.



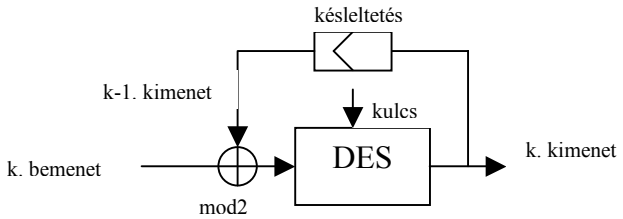
8. Ábra – MAC számítás

5.8.3. Rejtjelezés

DES esetében a rejtjelezést a DES függvény végzi el, melynek paramétere a titkos kulcs. Az inverz transzformáció is elvégezhető a titkos kulcs ismeretében. A DES alapú *rejtjelezésnek* két struktúrája lehetséges: az ECB és a CBC. Mindkettő olyan DES dobozkára épít, amilyet mi a kártyán kialakítottunk.

Az ECB (electronic codebook) egyszerű blokkrejtjelező. A bemenetet nyolc byte hosszúságú blokkokra tördeljük, s ezeket egyenként rejtjelezzük a DES segítségével.

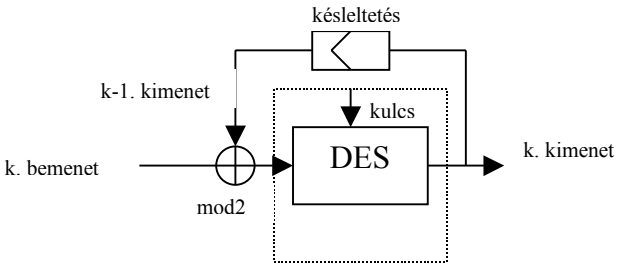
A CBC (cipher block chaining) [Davis-Price1992 *(4.2)*] egy jóval ravaszabb szerkezetű visszacsatolt rendszer. A következő bemenet mindig a következő nyílt szöveg blokk és az



8. Ábra - CBC kódolás

előző rejtett szöveg modulo 2 összegeként áll elő. Így azt érhetjük el, hogy a k. kimenet nemcsak a k. bemenettől függ, hanem függ az összes előző bemenettől is. Tehát ugyanannak a blokknak mindig más és más lesz a képe. A CBC rejtjelezőnek egyik paramétere a DES kulcs, másik pedig az IFB (initial feedback buffer), tehát az a visszacsatolás, amit a legelső bemenethez adunk hozzá modulo 2. Az IFB tartalma általában csupa 0. Mi is így dolgozunk.

Igaz, a kártyán csupán a DES dobozt valósítottuk meg, de a kártya segítségével a PC-s programunk mindkét rejtjelezést képes kódolni és dekódolni is. S mindezt a PC úgy végzi el, hogy a PC-s program nem ismeri a DES kulcsot, mivel azt a kártya tartalmazza. Így a rendszer biztonsága nem jelentősen kisebb, mintha a visszacsatolást is a kártyán valósítottuk volna meg. Viszont jelentős sebességnövekedést értünk el ennek segítségével.



9. Ábra -Mi van a kártyán?

DES csomagunk grafikus felhasználói felülete hatékonyan támogatja mindkét kódolási mechanizmust. Sőt, dekódolni is lehet velük elkódolt rejtett szöveget. Természetesen ez csak akkor igaz, ha ugyanazzal a kulccsal próbálunk dekódolni, mint amivel kódoltunk.

Ha a kártya által generált kulccsal kódoltunk, akkor abban bízhatunk, hogy azt a kártyából senki ki nem nyerheti. Még mi sem. Így, ha a kártya kulcsát megváltoztatjuk, akkor a generált kulcs megsemmisül. S abban is biztosak lehetünk, hogy senkinél sincs meg még egy példányban.

5.8.4. Hozzáférésvédelem

Hozzáférési jogosultság ellenőrzése és azonosítás történhet

- titok segítségével (PIN vagy challange and response)
- tárgy segítségével
- biometriai módszer segítségével

Mi, mivel kártyával foglalkozunk, a tárgy segítségével való hozzáférésvédelmet valósítottuk meg. Ezzel „áttoltuk a lovat a szomszéd utcába”: hogyan azonosítja magát a kártya? Biometriai jellemzői a smartcardoknak nincsenek, s tárgyakat sem birtokolnak, így marad a titok.

Lehetőség lenne, hogy a kártya egy jelszót küldjön el az olvasónak, s az olvasó ellenőrizze ezt. Ez egy nagyon rossz megoldás lenne. Ekkor ugyanis ha valaki ellopná a kártyát, behelyezné a saját olvasójába, a kártya rögtön elárulná az olvasónak a jelszót. Attól is félhetnénk minden egyes használatkor, hogy valaki lehallgatja a titkos jelszót, amint az a cél felé tart, s ezzel birtokába jut.

A megoldás egyszerű: a rendszer végül is ismeri a jelszót, akárcsak mi. Nincs szüksége rá, hogy megtudja azt. Arra van csupán szüksége, hogy megtudja, ismerjük-e a jelszót. Ezt hogyan érhetjük el? Az általunk megvalósított megoldás a „kihívás és válasz" módszer segítségével a következőképpen működik:

1. A rendszer küld a kártya számára egy kihívást. Ez nem más, mint egy *r* véletlen szám.
2. A kártya megkapja a véletlen számot. Nem csinál vele mást, mint kódolja a *k1* titkos kulcsa segítségével, s visszaküldi $E_{k1}(r)$-et a rendszernek.
3. A rendszer is kódolja *r*-et a saját *k2* titkos kulcsa segítségével, tehát kiszámítja $E_{k2}(r)$-et. Ha a két titkos kulcs megegyezik, a kártyától kapott eredmény is megegyezik majd a saját számításával. Így ha a rendszer azt kapta, amit várt, akkor nyugodtan felismerheti a kártyát.

Mi a DES csomagra támaszkodva valósítottuk meg ezt a dinamikus jelszó kezelést. Így a jelszó (a *k1* kulcs) nem más, mint egy 56 bit hosszú DES kulcs.

A PC-s program a következőket végzi el, ha az autentikáció gombra kattintunk:

1. generál egy véletlen számot
2. kódolja a számot a kártya segítségével
3. kódolja a számot a saját titkos kulcsa segítségével
4. összehasonlítja a két eredményt

A két eredmény két 64 bit hosszú szám. Annak valószínűsége, hogy két 64 bites számt véletlenül megegyezik, nagyon kicsi. Így ez az azonosítási módszer elég biztonságosnak (a DESsel azonos biztonságúnak) tekinthető.

5.8.5. Digitális aláírás

A digitális aláírás nyilvános kulcsú élő rendszerekben létező módszer. Eredeti elképzelésünk az volt, hogy kidolgozunk a kártyán egy nyilvános s egy titkos kulcsú kriptográfiát alkalmazó csomagot. Míg az utóbbi próbálkozásunkat hosszas küzdelem után végül siker koronázta, az előbbi sajnos kudarcba fúlt. Ennek fő oka az volt, hogy mind a kártyához tartozó fejlesztőeszköz, mind pedig annak dokumentációja még béta verzió, s a végleges változat nem készült el. A kártya már eljutott a végleges változatig, de ahhoz, hogy kódot írjuk rá, szükségünk lett volna a fejlesztőkörnyezetre, amivel kapcsolatban komoly dokumentáltsági hibákba s hiányosságokba ütköztünk. Így a kártya RSA funkcióját nem voltunk képesek működésre bírni.

Találtunk viszont egy titkos kulcsú protokollt, amely lehetővé teszi a digitális aláírást. Ezt meg lehet valósítani DES segítségével. [Schneier1996 *(2.6.)*]

A módszer lényege a következő:

A, **B**, **C**, **D**, és a többiek személyek, akik hitelesen akarnak kommunikálni egymással. Mindannyian rendelkeznek saját titkos kulccsal. **T** egy kitüntetett személy, akiben a többiek mind megbíznak, s megosztották vele titkos kulcsukat.

A hiteles üzenetet kíván küldeni **B**-nek. A következő lépések történnek ekkor:

1. **A** titkosítja az üzenetét a kulcsával, s elküldi **T**-nek. Beleírja az üzenetébe azt is, hogy ő **B**-nek kíván üzenni.
2. **T** ismeri **A** kulcsát, kibontja az üzenetet. Elolvassa azt, majd hozzáteszi azt, hogy tanúsítja, hogy az üzenet **A**-tól jött. (ezt onnan tudja, hogy **A** kulcsával kódolták)
3. **T** titkosítja az üzenetet **B** kulcsával, s elküldi **B**-nek.
4. **B** tudja, az üzenet **T**-től jött, mert **B** kulcsát csak **T** ismeri rajta kívül. Az üzenetben pedig benne van, hogy **A** küldte, s ezt **T** írta, **T** pedig soha sem hazudik.

Ez a módszer megvalósítható PC-s program csomagunkkal, de tulajdonképpen az MAC generálás funkciót használja (vagy a CBC/ECB rejtjelezések egyikét), tehát nem különbözik a többitől.

5.8.6. Kulcsgondozás

A kulcsgondozás jelenti kulcsok

- generálását
- tárolását
- továbbítását

RSA kulcs előállítása a kártyán a sebességviszonyok miatt reménytelen vállalkozás lenne. DES kulcsot viszont minden gond nélkül generálhatunk, itt nincsen szükség különösebb biztonsági megfontolásokra a kulcs természetét illetően. Generálunk 8 db random byte-ot, s ez lesz a mi DES kulcsunk. (A gép egyébként nem használtja a DES kulcs paritásbitjeit.) Azzal, hogy véletlen biteket választunk, nem követünk el nagy hibát, hiszen a DES 2^{56} db kulcsa között összesen 16 a gyenge kulcs. [Schneier1996 *(8.1)*]

A fő gond itt nem a véletlen nyolc byte-tal van, hanem azzal, hogy valóban véletlennek tekinthetjük-e azt, amit a kártya kiad. Erről nincsenek információink. Azt tudjuk csak kijelenteni, hogy a kártya dokumentációja nem mond lehetőséget arra, hogy a kártyából a véletlen szám generátor aktuális állását kinyerjük. Sőt, jól megtervezett kártya esetén a támadónak véletlen szám generálásához (s ennek megismeréséhez) sincsen joga.

A kulcsok tárolására adhatunk egy rövid választ: Ez a smartcard technológia lényege. A smartcardokat tulajdonképpen kulcsok tárolására és hordozására találták ki. Ők valójában nem mások, mint ezt a célt szolgáló biztonságos eszközök. A dolog lényege annak biztosítása, amit a biztonságról szóló fejezetben leírtunk.

A következő pontoknak kell teljesülnie ahhoz, hogy a kulcs ne kerülhessen illetéktelen kezekbe:

- Ne ismerje senki illetéktelen a kulcsot, már korábbról, mielőtt az a kártyába bekerült. Ez alapvető feltétel, s szervezési kérdés, nem szoftver probléma. Programunk lehetőséget biztosít kulcs véletlen generálására, amit biztosan nem ismer senki. Bizonyos esetekben persze ez nem lehetséges, például amikor két kártyát kívánunk előállítani azonos kulccsal. Ekkor van értelme a kulcs kártyára való feltöltésének.

- Ne lehessen a kulcsot fizikai eszközökkel megszerezni! – Ez a kártyagyártó felelőssége.
- Ne lehessen a kulcsot APDU-k segítségével kiolvasni! – Ennek megoldására a fent leírt módszert alkalmazhatjuk: definiálunk egy system nevű known principalt, s minden adatfilehoz csak ő férhet hozzá. Minden program jelentkezzen be a kártyába először system-ként, ha file-okat kíván kezelni, majd jelentkezzen ki, ha befejezte. Az pedig, hogy ő milyen PIN-nel vagy challange and response-zal azonosítja magát, legyen hétpecsétes titok. Igaz, még így is fennáll annak a veszélye, hogy egy támadó rájön erre, például azzal, hogy feltöri a challange and response DES-ét, de ekkor már oly mértékű számítási kapacitás birtokában kell lennie, amivel a kártyában tárolt titkos kulcsot is feltörheti.
- Ne lehessen a kulcsot az alkalmazástól megkapni! – Ez az alkalmazás tervezőjének a feladata. Ha nem írjuk ki a kulcsot soha a kimenetre, az nem fog magától kikerülni oda.

A kulcs továbbítása történhet:

- Smartcard segítségével, tehát a kulcsot hordozó személy a zsebében szállítja a kártyát. Ez nem kevésbé biztonságos, mint a tárolás esete. Ezen programunk erre készült. Itt csak így lehet kulcsot hordozni. A kulcs kártyán történő hordozásának pedig egyetlen értelme az, hogy azt a kártyával felhasználjuk, hiszen azt a kártyából kinyerni nem lehet.
- A PC irányába a kártyából: ilyen a mi programunkban nem történhet. A kulcs csak a PC-ből megy a kártya felé, de ez az előző pont anyaga.
- Hálózaton keresztül: a mi programunk ilyet sem tesz. Ezen egyébként segíteni lehetne megfelelő kódolás használatával.

5.9. Összefoglalás, munkánk értékelése

5.9.1. A mi eredményeink

Célunk a WinCard s a hozzá tartozó fejlesztőrendszer lehetőségeinek áttekintése, s a hitelesség biztosítás megvalósításának smartcardos lehetőségeinek felmérése volt. Ezek megvalósításához kifejlesztettünk egy DES kódolást-dekódolást megvalósító, s kulcsokat gondozó kártya-PC programcsomagot. Ennek segítségével próbáltuk megvalósítani a hitelesség biztosításának öt fő pillérét, melyek:

- hitelességvizsgálat
- rejtjelezés
- hozzáférésvédelem - dinamikus jelszavak
- digitális aláírás
- kulcsgondozás

Ezek közül hármat közvetlenül megvalósítottunk. Kártyánk képes rejtjelezni, s a programcsomag képes hitelességet ellenőrizni, illetve a PC dinamikus jelszó segítségével azonosítani a kártyát.

A kulcsgondozás viszont nem egy aktív funkció, amit egy rendszer képes „megtenni”. Ez egy problémakör, egy gondolkodásmód, melynek figyelembe vétele létfontosságú. Kártyánk védi, s ki nem adja a kulcsot senkinek. A kulcsot csak használni lehet, megismerni nem. Lehetőség van kulcsnak magán a kártyán történő generálására is, s ennek használata tovább növeli a rendszer flexibilitását s biztonságát. A kulcs továbbításának pedig egyetlen megengedett módja a smartcard. Így kijelenthetjük, hogy az öt pillér közül négyet sikeresen teljesítettünk.
Probléma egyedül a digitális aláírás megvalósítása körül történt, ugyanis a fejlesztőrendszer dokumentáltsági hibái folytán nem tudtuk beüzemelni a kártya RSA funkcióit. Klasszikus digitális aláírás megvalósításához viszont nyilvános kulcsú titkosírásra van szükség.
Leírtunk viszont egy három résztvevős protokollt, mely digitális aláírást valósít meg titkos kulcsú rendszerben, s az általunk készített programcsomagot ennek bármely szereplője használhatja.

5.9.2. Az öt pillér elvi megvalósíthatósága egy mai smartcardon

A smartcardok biztonságrechnikai alkalmazások terén magasan felülmúlják a PC-ket. Fő hártányuk a sebességben rejlik. Elég hamar nyilvánvalóvá vált számunkra, hogy kriptográfiai műveletek terén csakis a kártya processzorában implementált műveletekre támaszkodhatunk. Elvileg lehetséges lenne őket például függvényként megvalósítani, de ezek sebessége kritikán aluli lenne. A mérésekről szóló részben (5.6.2., 5.6.3.) ismertettük azon tapasztalatainkat, melyeket a teljes egészében a kártyán megvalósított MAC számító program implementálása közben szereztünk. A jövőben mindenképpen várható sebességnövekedés, de a hardver vagy mikroprogram, esetleg operációs rendszer szinten megvalósított funkciókkal a felhasználói programok nem versenyezhetnek.
A másik szűk keresztmetszet a PC-kártya kommunikáció. Ez ugyanis lassú (5.6.1.). Ez a továbbiakban is soros lesz, ugyanis a kártyán lévő kontaktusok működését leíró szabványok csak ilyet tesznek lehetővé. Így komoly sebességnövekedés itt nem jöhet szóba. Szintén korlátozva van az egy APDU-ban eljuttatható adatok mennyisége. Ez áthágható korlát, bár ez sebességcsökkenést jelent, hiszen a kártya processzorát kell ekkor igénybe vennünk.
A rejtjelező funkciót egy kártya képes ellátni, természetesen a fenti korlátozásokkal. Sajnos az egész nyílt szöveget el kell juttatnunk a kártyára, s ez így lassú. Mi 100 byte-os nagyságrendben mozgó inputot még ésszerű idő alatt tudtunk titkosítani. Egy sok kilobyte-os anyag viszont órákig megy keresztül a kártyán. Kis és rövid titkokat tehát hatékonyan kezelhetünk smartcarddal. A megabyte-os tartományt viszont jobb, ha elfelejtjük.
Hitelességvizsgálat és digitális aláírás esetén jobb a helyzet. Igaz, ha itt is leküldjük a kártyára az egész nyílt szöveget, s mindezt a kártya processzorán dolgozzuk fel, az előző bekezdésben leírt problémába ütközünk. Megtehetjük viszont, hogy először a PC-n tömörítvényt képzünk a nyílt szövegből, s utána azt írjuk alá a kártyával. Így jelentős sebességnövekedést érhetünk el, míg a kriptográfiai műveleteket továbbra is a biztonságos kártyán végezhetjük el. Ezzel az előfeldolgozással, amit például az MD5 vagy az SHA eljárás is elvégezhet, a biztonság

erejéből nem veszítünk, hiszen ezen eljárások nem titkosak, s semmilyen titkos paramétert nem tartalmaznak.

A hozzáférésvédelem és a kulcsgondozás alig ütköznek problémába. Mindkettő olyan eljárás, amelyre már régen alkalmaznak kártyákat, igaz, programozható képességüket még nem használják ki. Úgy is lehet mondani, hogy a kártyákat eredetileg erre a két módszerre találták ki. Mindkét esetben kicsi az adatforgalom, s kevés számításra van szükség, így az új ötletek is viszonylag könnyen implementálhatók.

6. Kutatásaink jövője

A programozható smartcardok előtt óriási jövő áll. Mi most csak a hitelesség biztosításának módszereit tekintettük át, s ez csak egy kis témakör, amiben ezen chipkártyákat használni lehet.

Használhatóak lehetnének például hálózatos világban. Elképzelhető lenne, hogy valaki a kártyája segítségével azonosítja magát egy világ méretű hálózat felé, s ennek segítségével hozzáfér a rendszerben akármilyen erőforráshoz, akár banki tranzakciókat is végezhet.

Lehetséges lenne kártyával betölteni az összes lehetséges igazolvány és hitelkártya szerepét. Szintén óriási lehetőség lenne kártyák alakalmazására az elektronikus kereskedelemi, s a digitális pénz alkalmazások terén. Kártyák használhatóak mind a világhálón lévő információk elérésére, s a számos szolgáltatóhoz való bejelentkezésre, de ugyanakkor használhatóak személyiségi jogok védelmére is.

Ezer és ezer további alkalmazást említhetnénk még, de ennek csak egy apró részébe vagyunk képesek jobban belemélyedni. A programozható kártyák előtt komoly jövő áll, s mi tovább szeretnénk foglalkozni a bennük rejlő lehetőségek kitapasztalásával, illetve a reálisnak ítélt alkalmazások megvalósításával.

Mi most a Smart Card for Windows 1.0-s béta (1999 május) verzióját vizsgáltuk, e dolgozat beadása előtt néhány nappal kaptuk meg a fejlesztőrendszer következő változatát (1999 szeptember). Az új eszközben érezhető egy kiforrottabb technika, s bizonyos hibákat (például amelyekbe az RSA alkalmazásakor ütköztünk), kijavítottak. Szintén megjelentek e kártyán bizonyos GSM kapcsolatok is.

További előrelépés kutatásaink szempontjából, hogy sikerült szert tennünk három Bull Odyssey I. Java Cardra, s a hozzájuk tartozó fejlesztőeszköz demo változatára. Így a közeljövőben lehetőségünk lesz a két konkurrens gyártó hardver és szoftver termékeinek alapos összevetésére.

7. Smartcardok: jelen és jövő

Igaz, hogy a smartcardok gyenge pontjai, a kommunikációs sebesség, a tárkapacitás és a számítási sebesség sokat fejlődtek, de valószínűleg mindig is alul fogják múlni a PC-k teljesítményét. Bizonyos dolgok elvi korlátokba is ütköznek.

A kommunikáció a kártya és a PC között sorosan valósul meg (ez a kontaktusok specifikációjából következik). Így ez eleve nem lehet különösen gyors. Az adat tárolása jelenleg nem illékony memóriával történik, (hogy ne legyen benne mechanika), ez pedig igen drága. A processzor esetében pedig komoly hőelvezetési problémák merülnek fel az órajelfrekvencia növelése esetén. Ráadásul minden alkatrészt egyetlen mikrochipen kell megvalósítani, ami növeli a nehézségeket. Nem tudjuk, meddig fognak nőni e paraméterek, de a PC-ket nem érhetik utol.

Milyen lesz a jövő smartcardja? Használjuk-e majd úgy mi is, mint Mézga Géza a negyedik fejezetben? Nem tudjuk. Az egyik elképzelés az, hogy a smartcardon tároljuk majd összes személyes adatunkat, s minden, amit szeretnénk magunkkal vinni. Tehát akár otthon vagyok, akár külföldön, egy szállodában, a saját smartcardommal ugyanolyan szolgáltatásokat kapok. Ugyanannyit pirul a pirítós, ugyanúgy be tudom zárni a kocsimat, ugyanúgy jelzi, ha a kedvenc együttesemnek koncertje lesz a közelben, s ugyanúgy olvashatom a postámat, megkaphatom az újságomat, fogadhatom a telefonhívásokat, akárhol is vagyok a világon.

Persze ehhez az kell, hogy a smartcard tartalmazza az összes adatot: legyen benne, hogyan szeretem a pirítóst, milyen zenére szeretnék ébredni, tartalmazza az ujjlenyomatomat, a retinamintámat, a jelszavaimat, PIN kódjaimat az összes lehetséges rendszerbe, s szükség esetén ő használja őket. Ehhez nagyon „erős” smartcardra va szükség.

A másik lehetőség közelebb áll a mai realitásokhoz. A kártya ne tartalmazza ezeket, hanem forduljon a hálózathoz. Tehát ha beteszem a kártyámat a pirítóssütőbe, akkor az azonosítson engem, s kérje le a saját adataim közül, hogy hogyan szeretem a pirítóst. Tehát a kártya csupán egy kulcs legyen a világ szolgáltatásaihoz, mellyel azonosíthatom magamat, s elérhetem az én egyedi beállításaimat.

Jelentős előrelépés lenne a mai helyzethez képest, ha az emberek nem sok-sok kártyát hordoznának magukkal, hanem csak egyetlen egyet. S nem kellene sok-sok jelszót és PIN-t fejben tartaniuk, hanem elég lenne egy, az, amit a kártya tud, s a kártya mendzselhetné az összes többi jelszót, kódot, amit a felhasználó nem „látna”.

Igaz, várhatóan nem fog a közeljövőben senki telefonbeszélgetést valós időben kártyán titkosítani nyilvános kulcsú titkosírással, mint ahogy azt Paula és Huffnagel Pisti tette, de számos momentum a Mézga család egy napjából, mint például az előző bekezdésben leírt is, műszakilag lehetséges. Szervezési, illetve jogi kérdés ezek megvalósítása. Elvileg a személyi igazolvány, jogosítvány, bankkártya, diákigazolvány, adókártya, TB kártya, villamosbérlet, HIR kártya, telefonkártya, parkolókártya, stb. egyetlen kártyára integrálása megvalósítható lenne. Sőt, mindemellett még tartlamazhatná az ujjlenyomatunkat, retinalenyomatunkat,

nyilvános és titkos kulcsunkat is, s a jelszavainkat a különféle számítógépes rendszerekhez. Ez ma műszakilag már nem probléma.

A gond azzal van, hogy a kártyaolvasók manapság még nem elég elterjedtek (ezért történik a diákigazolvány-chipkártya érvényesítése matrica segítségével), és jelentős probléma az is, hogy ne sértse mindez a személyiségi jogokat.

Smartcard rendszerrel ugyanis könnyű lenne egy orwelli rendőrállamot adminisztrálni. Amikor Mézga Géza beteszi a kártyáját a munkahelyi leolvasóba, fontos, hogy a főnöke ne érhesse el az ő orvosi adatait, vagy a bankszámláját, de még azt sincsen joga megtudni, hogy Géza milyen pirítóst kedvel, vagy hogy pontosan hány fokos vízben szeret fürdeni.

Mindezek a problémák viszont – már ma, a mostani kártyákkal is, például az általunk vizsgált WinCarddal – technikailag megoldhatóak, csak komoly szervezést igényelnek. Igaz, mindehhez meg kellene győzni az embereket, hogy a jól megtervezett kártyáról ezen információkat nem lehet kinyerni. Nagy előnye ugyanis a kártyás világnak a hálózatos világgal szemben, hogy míg utóbbi esetében az ember sohasem lehet biztos benne, hogy ki mikor hova tud betörni, s milyen információt lophat el tőlem, a kártya áttekinthető. Azt könnyű elmagyarázni valakinek, hogy „Ha nem teszem be a kártyát az olvasóba, akkor nem vehetnek el tőlem pénzt.”

Az intelligens smartcardok igen széles lehetőségeket nyitnak meg elektronikus biztonságtechnikai alkalmazások előtt, s ráadásul ezeket a lehetőségeket viszonylag alacsony áron jelentik. Nem szükséges ugyanis külön hardvert, külön mikrochipet gyártani minden egyes alkalmazáshoz, hanem általános célú, programozható eszközökön lehetséges a fejlesztés, amelyeket nagy példányszámban alacsony áron lehet előállítani. Egy kártya felprogramozásához pedig egy olvasón, egy PC-n s szoftveren kívül semmi sem szükséges. Így nem csak nagy multik, hanem kisebb cégek is képesek lehetnek saját smartcard alapú szoftverek kidolgozására, fejlesztésére.

A programozható kártyák nemcsak gazdasági teret nyitnak meg. Lehetséges, hogy egy kártya megvizsgálja egy számítógépen futó szoftver integritását. Elképzelhető az is, hogy egy kártya saját maga generál egy titkos kulcsot, s azt belőle kinyerni semmilyen módon nem lehet. Hajlandó viszont a beérkező adatokat titkosítva visszaküldeni a PC-nek. A lehetőségek száma jócskán megnőtt, s habár a smartcardok mind sebesség, mind kapacitás terén jócskán elmaradnak a PC mögött, mind hordozhatóság, mind pedig biztonság szempontjából könnyedén lekörözik azt.

Függelék

Rövidítések jegyzéke

ACL: Access Control List, hozzáférési lista. A kártya file-rendszerében minden állományhoz egy ACL tárolja, hogy milyen felhasználók milyen műveleteket végezhetnek az adott állományon.

APDU: Application Protocol Data Unit, a kártya és a kártyaolvasó közti kommunikációs protokoll egysége. Az olvasó minden parancsát egy APDU-ban küldi el a kártyának, a kártya pedig szintén egy APDU-ban válaszol. Az APDU felépítését, és így az olvasó/kártya interfészt az ISO 7816 szabvány részletesen leírja.

API: Application Programming Interface, alkalmazásfejlesztői interfész. Az API szubrutinok egy gyűjteménye, amivel egy adott eszköz programozható, pl. WinCard API.

ATM: Automatic Teller Machine, készpénzkiadó automata.

ATR: Answer To Reset, válasz a Reset jelre. Minden kártya/olvasó párbeszéd úgy kezdődik, hogy az olvasó küld egy Reset jelet a kártyának, amire a kártya az ATR-rel válaszol. Ebből kell minden olyan információnak kiderülni, ami a további kommunikációhoz kell (pl. használt protokoll, frekvencia, maximális áram és feszültség stb.) Ezen k0vül a kártya „bemutatkozik”: megad(hat)ja a típusát, gyártóját stb.

CBC: Cipher Block Chaining, láncolt blokk-kódolás. DES-en alapuló titkosítási eljárás, melynek során a nyílt üzenet 8 byte-os szegmenseit egy visszacsatolt rendszer segítségével kódolják. Minden bemenet az előző bemenetből és a rejtett szöveg előző szegmenséből jön létre egy visszacsatolás során.

CSP: Cryptographic Service Provider, kriptográfiai szolgáltató. A Microsoft terminológiájában a kártya egy CSP, amely a számítógép számára implementálja a CryptoAPI-t.

DES: Data Encryption Standard, adatkódolási szabvány. A 70-es évek óta szabványos és széles körben elterjedt szimmetrikus titkosítási eljárás, mely egy 56 bites kulcs segítségével 64 bites üzenetekből 64 bites kódot állít elő.

DF: Dedicated File. A kártyák file-rendszerében ez felel meg a könyvtáraknak.

ECB: Electronic Code Book. DES-en alapuló titkosítási eljárás, melynek során a nyílt üzenet 8 byte-os szegmenseit DES-sel, ugyanazon kulcs segítségével elkódolják, és a kapott 8 byte-os kódokat egymás mellé illesztik.

EEPROM: Electronic Erasable Programmable ROM, elektronikus úton törölhető programozható ROM. A kártya nem-illékony memóriája rendszerint ilyen.

EF: Elementary File. A kártyák file-rendszerében ez felel meg a hagyományos adarfile-oknak.

EMV: Europay-Mastercard-Visa. Üzleti tömörülés, mely az ISO 7816-ot kibővítő specifikációt dolgozott ki.

EPOS: Electronic Point Of Sale, elektronikus árusító és bankkártya elfogadó hely.
GSM: Global System for Mobile Communications. A mobiltelefonokban lévő SIM-kártyák is lényegében smartcardok, csupán méretük kisebb.
IATA: International Air Transportation Association, nemzetközi légiforgalmi szervezet. Az első, mágnescsíkot tartalmazó kártyák bevezetője.
IFB: Initial Feedback Buffer, kezdeti visszacsatolási puffer. Visszacsatolásos kódoláskor (pl. CBC) ez pótolja a 0. szövegszegmenst. Rendszerint a titkos kulccsal együtt kell megadni.
ISO: International Standard Organisation, nemzetközi szabványügyi szervezet.
JDK: Java Development Kit, a Sun ingyenes java fejlesztő eszköze
KP: Known Principal, a Microsoft kártya által ismert fél.
KPT: Known Principal Table. A KP-ket tartalmazó táblázat.
MAC: Message Authentication Code, üzenethitelesítő kód. Az egész üzenettől függő, fix hosszúságú (8 byte) kód. Előállítása: a CBC utolsó kimenete.
MD5: Message Digest 5. Ron Rivesttől származó üzenettömörítő és –hitelesítő egyirányú függvény, mely tetszőleges nyílt szövegből annak 128 bites lenyomatát készíti el.
MF: Master File. A kártyák file-rendszerében ez felel meg a gyökérkönyvtárnak.
MOS: Metal-Oxid Semiconductor. A kártyákban használatos EEPROMokban MOS technológiát kell alkalmazni annak érdekében, hogy a cellák állapotát ne lehessen mikroszkóppal megállapítani.
MS: Microsoft
MSDN: Microsoft Developers' Network, Microsoft fejlesztői hálózat. A Microsoft fejlesztői eszközökhöz járó MSDN CD illetve az MSDN online internetes oldala sok segítséget nyújt a fejlesztésben.
MUSCLE: Movement for the Use of Smart Cards in a Linux Environment, mozgalom a smartcardok linuxos környezetben való alkalmazására.
NVM: Non Volatile Memory, nem-illékony memória.
OCF: OpenCard Framework. Az OpenCard Consortium specifikációja.
PIN: Personal Identification Number, személyi azonosító szám. Rendszerint 4 számjegyű titkos azonosító, melynek begépelésével a kártya felhasználója jogosultságát igazolja.
PKCS: Public-Key Cryptography Standard, nyilvános kulcsú kriptográfiai szabvány az RSA Laboratories-tól.
RAM: Random Access Memory, véletlen hozzáférésű memória. Írható-olvasható memória, melynek statikus és dinamikus változata közül a smartcardok rendszerint a statikusat használják (SRAM).
RFU: Reserved for Future Use.
RISC: Reduced Instruction Set Computer, egyszerűsített utasításkészletű processzor.
ROM: Read Only Memory, csak olvasható memória. Ez tárol(hat)ja a kártya operációs rendszerét, ill. olyan adatokat, melyek nem változnak a kártya életciklusa során.
RAS: Remote Access Service, telefonos csatlakozási lehetőség a Windows NT-hez.

RSA: Rivest-Shamir-Adleman. Népszerű aszimmetrikus (nyilvános kulcsú) titkosítási eljárás, mely nevét három alkotójáról kapta.
SC: Smart Card.
SCOS: Smart Card Operating System., smartcard operációs rendszer.
SHA: Secure Hash Algorithm, üzenettömörítő és –hitelesítő egyirányú függvény.
SIM: Subscriber Information Module, a GSM-telefonokban az előfizetést igazoló smartcard.
SRAM: static RAM.
UAM: User Application Memory, a felhasználói programok által szabadon használható írható-olvasható memória.
VB: Visual Basic.
VM: Virtual Machine, a java úgy éri el a platformfüggetlenséget, hogy a java programokat az adott platformon futó java VM-ek értelmezik.
Wintel: Windows - Intel
WORM: Write Once Read Multiple, egyszer írható, sokszor olvasható memória.
WRU: Writing Reading Unit, kártyaolvasó.

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:
W. Rankl – W. Effing: Smart Card Handbook. John Wiley & Sons, 1997.
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

Egyéb, smartcardokkal kapcsolatos társaságok:
Bull: http://www.cp8.bull.net
Java Card: http://java.sun.com/products/javacard/htmldoc
M.U.S.C.L.E.: http://www.linuxnet.com
PC/SC Workgroup: http://www.smartcardsys.com
OpenCard: http://www.opencard.org