DIPLOMAMUNKA

Fülöp Arnold

Debrecen
2007

Debreceni Egyetem
Informatikai Kar

SMART KÁRTYÁK

Témavezető:
Dr. Herendi Tamás
egyetemi adjunktus

Készítette:
Fülöp Arnold
programtervező matematikus

Debrecen
2007

Tartalomjegyzék:

1. Bevezetés

A mai világunkban fontos szerephez jut az, hogy bizonyos adataink, információink biztonságban legyenek, mások ne tudjanak hozzáférni, elrejthessük azokat illetéktelenek elől. Már az ókorban is volt szerepe a szövegek titkosításának, Julius Caesar például az ábécé betűinek ciklikus eltolásával állította elő rejtjelezett üzenetét. A középkorban az egyházak közötti levelezésnél használtak titkosítást, manapság pedig a társadalom szinte minden szférájában jelen van. Gondoljunk csak például a katonai, diplomáciai illetve államtitkokra, vagy éppen a privát jellegű fontos információkra, melyek jogtalan megszerzése súlyos anyagi és erkölcsi károkat okozhat.

Az adatok titkosításának a tudományát kriptológiának hívjuk, ami a görög *„krüptosz”* (rejtett) és *„logosz”* (szó) szavakból származik. Ennek a tudománynak két ága van: a *kriptográfia* és a *kriptoanalízis*. A kriptográfia olyan módszerekkel, eljárásokkal foglalkozik, amelyek biztosítják az üzenetek titkosságát, védettségét illetve hitelességét. A kriptoanalízis a kriptográfiai algoritmusok vizsgálatával foglalkozik, a titok megfejtésére, feltörésére tartalmaz eljárásokat.

A titkosításnak alapvetően két komponense van. Az egyik maga a titkosító algoritmus, amelyet bárki ismerhet, a másik pedig egy kulcs, ami az algoritmus egy paramétere. Így feltörés esetén egyszerűen csak lecseréljük a kulcsot. Maga a kulcs lehet például egy jelszó, amit meg kell jegyeznünk, lehet valamilyen személyi azonosításra szolgáló egyedi testi tulajdonság (például ujjlenyomat) és lehet egy tárgy is. Ilyen tárgy a smart kártya is, amely titkos adatok elektronikus tárolására képes.

A smart kártyák korunk biztonsági rendszereinek egyre gyakoribb elemei. A nanotechnológia dinamikus fejlődésével gyorsan terjedtek a programozható mikrocsipek, melyek mára az adatvédelem plasztikus komponensévé váltak. Piaci jelenlétüket leginkább a mobiltelefonok elterjedése fokozta, mivel a telefonszámokhoz tartozó adatok a legtöbb esetben egy-egy SIM kártyán vannak tárolva. Használatosak a hitelkártyák világában (Franciaország), a tömegközlekedésben (Japán, Nagy-Britannia), illetőleg nagyobb cégeknél személyigazolási célokra. Terjedőben van az orvosi adminisztrációba való integrációja (Németország) és mára már nem utópia egy elektronikus pénztárca lehetősége sem. A mobilkommunikáció jól mutatja a smart kártyák gyakorlati hasznát, hiszen a

hálózat működéséhez szükséges adatok bizalmas kezelése kritikus, mivel azok illetéktelen felhasználása komoly anyagi veszteséget okozhat az eredeti kártyabirtokosnak. A SIM kártyák lehetővé teszik, hogy az azonosításhoz kellő adatok fizikailag a kliensnél legyenek, a visszaélésektől védve.

A smart kártyák ereje abban rejlik, hogy nem csupán adathordozó eszközök, hanem a beléjük épített mikroprocesszor képes „védelmezni" a tárolt információkat. A számítógép és a memória egyetlen egységet alkot, ezért nem lehetséges a mikroprocesszor kikerülése, az információk direkt kinyerése. Elvileg tehát egy ilyen kártya eltulajdonítása haszontalan. Mégis vannak módszerek, melyek az intelligens kártyák feltörésére tesznek kísérletet, hogy azok ne jelentsenek plusz biztonságot a privát adatok számára. Természetes, hogy az ilyen „betörések" ellen a kártyaeszközök gyártói, forgalmazói, fejlesztői igyekeznek védekezni, azonban fontos kiemelnünk, hogy a lehetséges rések nagyon sokfélék lehetnek. A jelen értekezés célja, hogy osztályozza, és rendszerbe sorolja a smart kártyák elleni támadásokat, valamint több szempontból értékelje az ismert betörési módszereket.

A dolgozatot a smart kártya történetének áttekintésével kezdjük. Részletesebben elemezzük a mikroszámítógép szerepét, különös figyelmet szentelünk a kártyabirtokos vagy használati jogokkal felruházott személy azonosításának módszereire, mivel ez biztonságtechnikai szempontból kiemelten érdekes. Az általános áttekintés után érintőlegesen bemutatunk két élő; egy fizikai és egy alkalmazó rétegű szabványt. Ezek ismertetése a smart kártyák belső „lelkivilágának" bemutatására szolgál. A szabványok után érintőlegesen foglalkozunk a titkosítási algoritmusokkal, mivel a szoftveres töréseknek ezek „gyengeségei" az alapjuk.

2. Történeti áttekintés és osztályzás

Az adathordozó kártyák, a rájuk integrált technológia szerint, nagyon különbözőek lehetnek. Legegyszerűbb változatuk az ún. mágneses memóriakártyák, ezeket az 1920-as évektől kezdve használják. Érdemi részük mindössze egy mágnesezhető felületi rétegű szalag (ferromágneses részecskék találhatók a szalag anyagában, így ez a magnószalaghoz hasonló technológia), melyre a kibocsátó valamilyen kis méretű, statikus értéket helyez el (megfelelő szerkezettel újraírható a szalag, de a kémiai rendszer gyors öregedése miatt csak egyszer, gyártáskor élnek a lehetőséggel). Az adatábrázolás, a vonalkódokhoz hasonló módon történik; az északi és déli pólusú sávok szélessége, valamint váltakozása reprezentálja az információt. A leolvasó eszköznek mindössze annyi dolga van, hogy a szalagot letapogatva dekódolja és értelmezze a hordozott adatokat. Természetesen a tárolt információt lehet rejtjelezni, de mivel a ráírt anyag statikus, a használt titkosítási eljárás feltörése után az összes hasonló mágneses memóriakártya „meztelenné" válik, elveszti csekély biztonságát. Mivel előállításuk gyors és olcsó, valamint nem minden esetben életbevágóan fontos a titoktartás, manapság is sok helyütt alkalmazzák őket (bankkártyák, blokkoló kártyák, klubkártyák, beléptető kártyák). Főbb hibái közé tartozik, hogy kicsi a hibatűrő képességük, könnyen hamisíthatók és alacsony a tárolási kapacitásuk (1kB alatt). Elmondható, hogy a mágneses memóriakártyák a fizetési folyamatok elektronikus automatizálása végett jöttek létre, a kártyán rögzített információ biztonságára kisebb hangsúlyt helyezve.

A mágneses adattárolásnál jobb technológia az optikailag írható/olvasható felületek használata. A kártya egy részét a klasszikus CD-hez hasonló réteggel vonják be. Az optikai memóriakártyák tehát nem jelentenek elvi újítást a delejes társaikhoz képest, csupán jóval nagyobb a kapacitásuk (akár 4MB, ez jelentősen több a smart kártyák tárméreténél).

A következő generációt az IC-s (integrated circuit – integrált áramkör) memóriakártyák jelentik. Gyakorlatilag csak fizikai különbség van köztük és az egyéb memóriakártyák között. Az IC-s memóriakártya egy írható, olvasható emlékezőáramkörből áll, előnye a szalagokkal szemben, hogy az adattartalom dinamikus (könnyedén és elvben akárhányszor módosítható). Információvédelemről itt még ugyancsak nem beszélhetünk, mert a kártya olvasásának/írásának módja nyilvános, egy olcsó célkészség segítségével bárki hozzáférhet

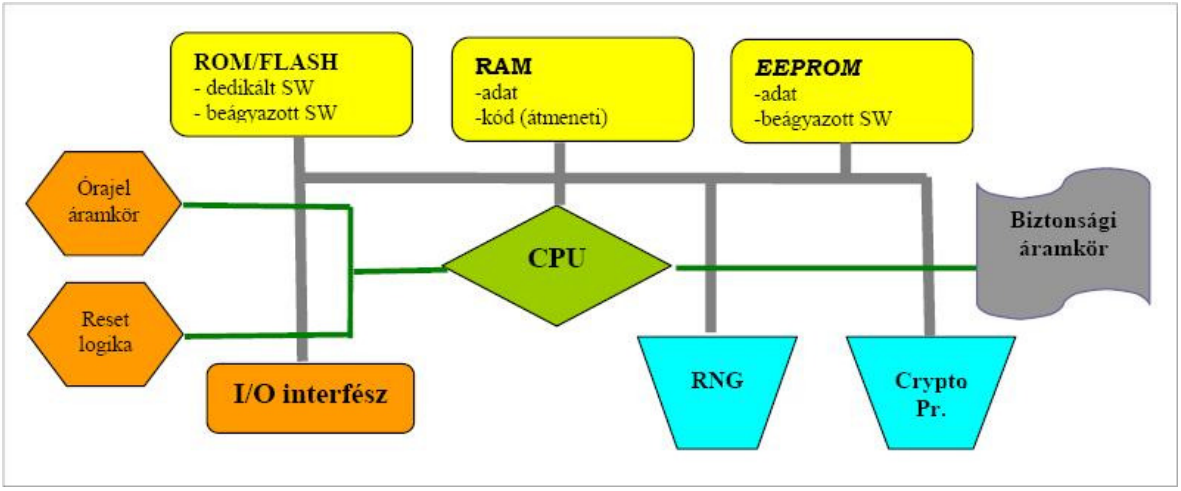
a tartalmához (ami lehet rejtjelezve). Olyan területeken, esetekben érdemes használni, amikor nem lényeges az adatok titokban maradása. Mellette szól, hogy a bonyolult smart kártyáknál számottevően kisebb áron előállítható, valamint, hogy az adatok változása esetén nem kell eldobnunk a legyártott lapkát.

Egy biztonságosabb technológia a csipkártya (más terminológiában: védett memóriájú kártya). Az elnevezést, a külső hasonlóságuk miatt, szokták használni – tévesen – a smart kártyákra is, de valójában egy sokkal egyszerűbb szolgáltatásról van szó. Igazi intelligencia nincs a IC-ben, csupán egy speciális áramkör van a memória köré szervezve. A csip logikája kártyánként különböző, arra szolgál, hogy valamilyen ügyes algoritmussal óvja a memória tartalmát, de ez nem programozható át. Felfogható a rendszer úgy is, mintha az adatírás, -olvasás egy speciális logika szerint zajlana, ezért szokták ezt a részosztályt logikai kártyáknak is nevezni. Egy adott titkosító módszer hardveres implementációjával elérhető, hogy az információ biztonságban tárolódjon a kártyán. Hibája, hogy nem képes haladni a korral, ha például egy kódoló algoritmust elméletileg feltörnek, vagy a számítási kapacitás „felnő" a kulcstérben való brute-force kereséshez, a védelem tovább nem garantált. Mivel egy fix lépésekből álló áramkör tervezése egyszerűbb és olcsóbb egy általános célú mikroprocesszornál (sőt, rengeteg optimalizálási lehetőség adódhat), ezért csak a rövidtávon érdekes adatok privát tárolására alkalmas. A Moore-féle empirikus törvény segítségével jó közelítéssel megbecsülhetjük, hogy egy adott méretű kulcs mikor válik gyengévé (belátható időn belül, próbálgatással megfejthetővé), így – az adat elöregedési idejét figyelembe véve – meg tudjuk választani az alkalmas kulcsmértet. Egy sifrírozási eljárás elvi feltörésének idejére természetesen nem tudunk jóslatot adni, ezért mindig fennáll a veszély, hogy csipkártyánk elavul. Ha egy közepesen biztonságos monofunkciós, viszonylagosan alacsony költségű kártyára van szükségünk, egy jól megtervezett csipkártya a megfelelő választás. Használatuk széles körben elterjedt, ilyenek a törzsvásárlói (pontgyűjtő) kártyák vagy a nyilvános telefonok kártyái. A kategóriába tartatozó csipkártyákat tovább tudjuk csoportosítani aszerint, hogy csak az írás joga van-e valamilyen módon (pl. titkos kóddal) védve – az olvasás publikus –, vagy bármilyen hozzáférés.

A ma elérhető legmodernebb kártyatechnológiát a többfunkciós, általános használatú mikroszámítógéppel ellátott lapocskák képviselik. Bombaszerű elterjedésüknek leginkább relatíve borsos áruk szab gátat. Egy BBC (British Broadcasting Corporation) felmérés szerint 2003-ban 2,8 milliárd smart kártya volt világszerte piaci forgalomban (ennek 40% európai tulajdonosnál). Elsőnek 1969-ben Jurgen Dethloff rukkolt elő a miniszámítógép

ötletével. A szakirodalom a smart kártyák születését 1974-re datálja, ebben az évben egy franci mérnök, Roland Moreno világszerte szabadalmaztatta J. D. vízióját. 4 évvel később, a szintén francia származású, Michel Ugon a szabadalmat adaptálta a bankkártyák világába. A smart kártyák ekkor nyerték el mostani, plasztiklapos formájukat (léteznek egyéb megjelenési formák is). Az ötlet bevált, M. Ugon megkapta a Francia Becsületrendet, a franciaországi bankok azóta is alapvetően intelligens csippel felszerelt bankkártyákat alkalmaznak. A 80-as években több országban is elkezdték a smart kártyák ipari előállítását. Elterjedését tovább segítette, hogy a városi nyilvános telefonok rongálása ellen a francia közigazgatás smart kártyákkal működő készülékek bevezetésével lépett fel. Az évszázad végére pedig, a GSM telefonos kommunikáció terjedésének hatására, a legtöbb európai háztartásban megjelent. A kompatibilitás és egységesítés érdekében a gyártók és a forgalmazók egy közös nemzetközi szabványt dolgoztak ki (ISO 7816), amellyel még foglalkozunk a 4. fejezetben.

A smart kártyák belsejében általában egy 8-bites RISC mikroprocesszor helyezkedik el (már létezik korszerű 16- ill. 32-bites mikroszámítógép is), melyhez társulhat egy kripto- vagy egy numerikus-processzor a specifikus titkosító számítások elvégzéséhez. A számítógép „egyik oldalára” egy 8 lábas bemeneti interfész van kötve (a csip külső, vezérlő felülete), a „másik oldalon” pedig a külvilágtól védett memória helyezkedik el. Ennek részei és azok mérettartományai: RAM (512-1024B), ROM (16-32kB), EEPROM (4-32kB). A belső processzor egyfeladatos, vagyis nem támogatja a többszálú programok futtatását.



1. ábra

Egy smart kártya belső felépítése

Érdekes megfigyelni, hogy míg a számítógépek teljesítménye rohamosan nő, addig a smart kártyák a 25 éves történelmük során alig fejlődtek. Ez persze csak látszólagos stagnálás, egy adott erejű smart kártya előállítási költségei idővel folyamatosan csökkentek. Nem is feltétlenül szükséges egyébként, hogy a lapocskák egy átlagos 80-as évekbeli számítógép szintjét meghaladják. A smart kártyák nem képesek önmagukban üzemelni (nincs energiaforrás), ezért a működtetésükhöz szükség van egy beágyazó környezet (CAD – Card Acceptance Device, magyarul kártyaolvasó vagy terminál) használatára. A hardverpiacon a forgalmazó cégek a legkülönbözőbb olvasószerkezeteket alkalmaznak: USB-s, PCMCIA, hálózati, floppymeghajtóba dugható, mobiltelefon stb. A lapkák egy újabb generációja már nélkülözni tudja ezt a fizikai érintkezést, ezekben egy apró rezgőkör kommunikál az éteren keresztül a terminállal (contactless). Amellett, hogy használatuk sokkal kényelmesebb, jóval drágábbak és komplexebbek, mert nem az olvasótól kapják a működéshez szükséges energiát, hanem saját apró akkumulátoruk van. Valójában tehát állandóan aktívak. Léteznek olyan lapkák is, melyek mindkét úton (érintkező és rádióhullám) tudnak kapcsolódni a kártyaolvasóhoz (kombi-kártya).

A számítástechnika történelmére jellemző, hogy az újdonsült technikákat általában több kereskedő, a riválisoktól eltérő formában valósítja meg. Így káosz alakul ki, mivel a rendelkezésre álló hardverek nem kompatibilisek egymással, ezért egy általános szoftver készítése lehetetlenné válik. Az egységesítés első lépése a már említett ISO szabvány kiadása volt. Nem elégséges azonban a fizikai paraméterek rögzítése, igénnyel találkozunk a platform független szoftverek támogatására is. Mivel a Java nyelv régóta élen kíván járni az architektúra független applikációk fejlesztésének támogatásában – ez az egyik fő célterülete –, ezért 1997 tavaszán kiadásra került a JavaCard API 1.0 elnevezésű specifikáció (ma használt verzió 2.2.2). Ennek hatására a szoftveresek egy olyan standardhoz jutottak, mellyel tetszőleges smart kártyán futtatható programokat készíthetnek. A rendszer kulcsa egy ágens komponens, az ún. Java Virtuális Gép (továbbiakban JVG), mely egy absztrakt számítógép szimulátora. A programokat a JVG-hez kell lefordítani, amit majd az adott kártyán lévő szimulátor értelmez és futtat. Persze fontos, hogy minden smart kártya típushoz létezzen egy JVG, amit a gyártók feladata – talán érdeke is – biztosítani. Az erőforrások szűkössége okán technikai problémát jelent mind a JVG, mind a használt applikáció együttes tárolása, pláne ha az egy titkosító eljárás egy nagy méretű kulccsal. A másik probléma a szimulációnak köszönhető sebességcsökkenés. Egy Java-s applikáció 30-szor lassabban hajtódik végre, mint gépi

kódban megírt társa. Napjainkban intenzív kutatás tárgya, hogy ezt a Java-s struktúrát hatékonyan realizálni tudjuk (első megvalósulások: Cyberflex, GemXPresso).

A Java kártyák (röviden JK) nagy előnye, hogy a homokozónak is gúnyolt JVG felett futó programok elől deprimálni tudunk bizonyos tevékenységeket, így növelve a memória tartalmának biztonságát. Másik pozitívum, hogy maga a működést biztosító Java kód le-föl tölthető a JK-ra, ezért nem szükséges azt folyton a memóriában tárolni, elég futtatás előtt feltölteni. A multifunkcionalitást az applikációk folyamatos cseréjével könnyen biztosítani tudjuk.

A Dallas Semiconductor az 1998-as JavaOne konferencián egy sajátságos smart kártya implementációval rukkolt elő, a mikroszámítógépet egy ujjra húzható gyűrűben helyezte el. Az ISO 7816-os szabványtól eltérően, a gyűrű egy „1-wire logic" (1 huzalos logika) típusú interfésszel van felszerelve (iButton), melynek lényege, hogy csak egy csatornán (szokványosan 6-8) keresztül kapja a vezérlőtől a bemenetet. A gyűrűhöz egy terminált is kifejlesztettek, amiben két független gyűrű fér el. Manapság már nem szokatlanok az ilyen ötletes mikroszámítógép beágyazások.

A becslések szerint a jövőben egyre több területen és helyen fognak smart kártyákat alkalmazni. Erre utal, hogy egyre több bank hitelkártya rendszere felel meg az EMV (Europay, MasterCard, Visa) intelligens bankkártyákra vonatkozó előírásoknak világszerte. Az EMV a nagy bankkártya társaságok közös szabványa, mely a smart kártyákon alapuló kártyák felé támasztott elvárásokat rögzíti. További bíztató előjel, hogy a Microsoft Windows 2000 és Windows XP operációs rendszerei már támogatják a smart kártyás ügyfél-azonosítást, ez mindenképp a kártyák elterjedését segíti. Egyéb piacuraló operációs rendszerek is igyekeznek támogatni a miniszámítógépeket; pl. a linuxos kártyakezelő könyvtárak kifejlesztésének szorgalmazására alakult a M.U.S.C.L.E. projekt.

A fejlődésre regresszív hatással lehet a MagnePrint technológia megjelenése, ami a jelenlegi bankkártyákon meglévő mágnescsík ujjlenyomatszerű, egyedi mintája alapján azonosítja az ügyfelet, nehézzé téve ezáltal a kártya másolását, hamisítását. Ez a jelenlegi kártyaállomány lecserélése nélkül teremt magasabb fokú biztonságot, míg a smart kártyára való áttérés az összes plasztiklap bevonását indukálja, ami nem kis költséggel jár. A fokozatos csere segít ezen, de akkor a két rendszert párhuzamosan kell üzemben tartani.

Történeti összefoglaló:

Év	Esemény
1969.	Jürgen Dethloff felveti a smart kártya gondolatát.
1970.	Kunitaga Arimura professzor szabadalmaztatja a smart kártya ötletét, de csak Japánban.
1974.	Roland Moreno világméretű szabadalmat nyújt be a smart kártya megvalósítására.
1979.	A Motorola megtervezi első smart kártyás mikroprocesszorát.
1982.	A Motorola Skóciában smart kártya gyárat létesít.
1985.	Franciaországban smart kártyát használnak telefonkártyaként.
1987.	Franciaországban kísérletileg banki használatban kipróbálják a mikroszámítógépes kártyát.
1987.	ISO/IEC 7816-os szabvány alapjai.
1996.	EMV szabvány.
1997.	A Motorola bejelenti a „Világméretű SmartCard Divízió” létrejöttét.
1997.	JavaCard API 1.0-s specifikáció.

3. A smart kártyák szerepe és jelentősége

Felmerül a kérdés, hogy egyáltalán milyen feladatok elvégzésére lehet használni egy ilyen apró számítógépet. Szimulációk elemzésére (meteorológiai modellezés, tudományos kutatás, mesterséges intelligencia) alkalmatlan, mert szűkös erőforrásaival nem versenyezhet a több száz számolási egységet magába foglaló szuperszámítógépekkel, gépfürtökkel. Vezérlésre (routing, realtime alkalmazások, célrobotok) szintén nem használható, mivel a külvilággal csak egy lassú és szűk keresztmetszetű csatornán képes kommunikálni. Egyedüli értelmes alkalmazása az adathordozás, adatmanipuláció, információkezelés lehet. Mint azt már korábban említettük, a számítógép szerepe, hogy izolálja a külvilágot a memóriában tárolt adatoktól, így védve azokat az illetéktelen felhasználástól.

3.1. Alkalmazások

Egy újszerű technika bemutatásakor fontos, hogy érveljünk hasznossága mellett. Ez a fejezet arra hivatott, hogy bemutasson néhány olyan alkalmazási területet, ahol a smart kártyák az eddig bevált eszközöknél valamilyen szempontból (akár több tekintetben is) jobb megoldást kínál. Sokszor találkozunk olyan szabadalmakkal, melyek áruk vagy nehézkes működtetésük miatt örökké csak papíron léteznek. A smart kártyák már több területen bizonyítottak (GSM telefonok) és számos olyan alkalmazási terület van, ahol egyre szélesebb körben kerülnek felhasználásra (e-mail titkosítás). Mennél inkább elterjednek, annál inkább érdekes az általuk szavatolt biztonság. A teljesség igénye nélkül, tekintsük tehát át a smart kártyák jelenlegi, valamint rövid távon lehetséges felhasználásait.

Egyszerű bankkártyák: legelső és legtesthezállóbb szerepköre a smart kártyának az elektronikus számlakezelés segítése. Rajtuk tárolódik az ügyfél egyedi azonosítója, amit az ATM (Automatic Teller Machine) vagy POS (Point Of Sale) terminál a kapcsolódó bankszámla kiválasztásához használ. Alapvető elvárás, hogy a birtokoson/birtokosokon kívül senki sem férjen hozzá a számlához, ne lehessen jogtalanul tranzakciókat végrehajtani rajta. Minden számlaműveletnél tehát szükséges a végrehajtó autentikációja,

ezt hivatott a mikroszámítógép ellátni, az alkalmazható módszerekről később lesz szó. Ismeretes, hogy anyagi értékkel bíró kreditünk nem csupán egy banknál lehet, ezért ebbe a kategóriába tartoznak a telefonkártyák, törzsvásárlói/törzsköri kártyák illetve az utazókártyák (travel card) is. A telefonkártyákat senkinek nem kell bemutatni. A törzsvásárlói kártyákat általában üzletláncok alkalmazzák; minden vásárláskor pontokat gyűjthetünk rájuk, ezeket később levásárolhatjuk. A törzsköri kártyákat egy célzott közönségnek szánják (hallgatók, nyugdíjasok, autósklub, tv csatorna előfizetők), ők a kártyával bizonyos anyagi kedvezményekhez juthatnak. Az utazókártya a busz-, villamosjegy modern változata, a csipen lévő kreditből a tömegközlekedést vehetjük igénybe. Hangsúlyozni kell, hogy ezen kártyatípusok mindegyike számlapénzt hordoz, ezért bizalmas kezelést kíván!

Elektronikus pénztárca: eddig olyan elektronikus fizetőeszközök szerepeltek, melyek csak azonosításra szolgálnak, de nem tárolnak effektív pénzt. A smart kártyák elég biztonságosak ahhoz, hogy pénztárcaként működjenek. A felhasználó egy bankomatnál feltölti a lapkát, ekkor a számlája csökken a megadott mértékben, viszont a pénz megjelenik a kártyán. Fizetéskor csökken a bankkártyán lévő összeg és jóváíródik az eladó betétszámláján. Úgy kell elképzelni, mint egy valódi pénztárcát, azzal a plusszal, hogy ezt hiába lopják el, a megfelelő titkok vagy személyjegyek nélkül nem igazán tudnak vele mit kezdeni. Persze ilyen rendszer kiépítésekor garantálni kell, hogy minden résztvevő betartsa a szabályokat. Ezt a külső terminál és a mikroszámítógép kölcsönös hitelesség vizsgálata teszi lehetővé (kézfogásos protokoll). A korábban (sőt még manapság is) előszeretettel alkalmazott mágneskártyák nem alkalmasak egy elektronikus pénztárca megvalósítására. Előnye, hogy nincs szükség egy olyan központi szerverre (esetleg gépparkra), amin az összes tranzakció végbemegy, a fizetés helyben történik. Ennek adatvédelmi pozitívumai is vannak.

SIM kártya: hazánkban az ilyen típusú smart kártyából van a legtöbb. Leolvasó terminálja maga a mobiltelefon. Ez tárolja az előfizetésünkhöz tartozó telefonszámot, egyedi kulcsokat, a mobilhálózat paramétereit, egy telefonregisztert és a kapott/küldött SMS-eket. Használatukkal válik lehetővé, hogy függetleníteni tudjuk az előfizetést magától a készüléktől. A titkosítás itt olyan szempontból fontos, hogy ne lehessen hamisítani a SIM kártyákat, más ne kaphassa meg a nekünk címzett SMS-eket. Az sem mellékes szempont, hogy más ne tudjon telefonálni a mi kontónkra. A SIM kártyák védelmét kétszintű PIN-kódos konstrukcióval oldották meg.

„Kulcstartó" (cipher card): e megnevezést azért került idézőjelek közé, mert a smart kártyás terminológiában teljesen mást jelent, mint a közismert személyes dísztárgy. A kulcstartó kártyák valamilyen titkosításhoz használt kulcsokat tárolnak. Leginkább az aszimmetrikus kódolások titkos kulcsát/kulcsait hordozzák a memóriájukban. Jobb esetben arra is képesek, hogy ezt maguk állítsák elő, ami azért praktikus, mert sem a kulcs, sem az előállításához használt átmeneti számok (RSA esetében pl. a prímek) nem kerülhetnek napvilágra, örökké a kártya belső rejtélyét képezik (zero-knowledge azonosítási módszer). Amennyiben új kulcspárt kérünk a lapkától, az csak a nyilvános kulcsot közli velünk (a fogalmak tisztázása érdekében lásd a 6.3. alfejezetet). Az ilyen kártyák csökkentik a hanyag kulcskezelés (publikálás az Interneten, feljegyzés a személyes noteszbe, elfecsegés, nyilvános csatornán továbbítás stb.) veszélyeit.

Az aszimmetrikus titkosítások (részletesen 6.3. alfejezetben) esetében, egy nyilvános kulccsal rejtjelezett üzenetet csak az tud megfejteni, aki ismeri a hozzá tartozó titkos kulcsot. Mivel jelen esetben azt csak a smart kártya tudja, a csipben elhelyezett számítógép feladata a desifrírozás. Az eljárás a következő; valaki küld nekünk egy, a nyilvános kulcsunkkal titkosított üzenetet, ezt feltöltjük a kártyára, a csip az inputot a titkos kulcs ismeretében desifrírozza, majd a feladott üzenet eredeti szövegét visszaküldi a terminálnak. A módszer digitális aláírás készítéséhez is használható, ennek módjáról a 6.3. alfejezetben írunk. A jövőben sok ilyen smart kártya alkalmazás várható, az elterjedtebb e-mail kliensek (Microsoft Outlook Express, Netscape Messenger) már több éve támogatják ezt a fajta biztonsági megoldást. Az e-mailek területén mindkét funkcióra nagy igény van, hiszen a protokoll öregsége miatt (nem világméretű, nyitott hálózatra tervezték) sok a visszaélés. Az Interneten keresztül zajló e-vásárlás egyre inkább terjedőben van. A pénzintézetek különböző alternatív megoldásokat kínálnak az elektronikus árucsere korrekt lebonyolításához. Kézenfekvő megoldás a digitális aláírással hitelesített, titkosított vásárlási szerződés alkalmazása, amelyhez tartozó kriptográfiai paraméterek a „kulcstartóban" vannak. Egy ilyen aláírt szerződés jogilag bizonyítványa annak, hogy valóban az aláíró rendelte meg az adott terméket, szolgáltatást és nem valaki „szórakozott". Egy esetleges visszaélés azonban így sem zárható ki 100%-ig.

Világosan látszik, hogy ez egy forradalmian új technika, mely valóban a smart kártyák belső szuverén világát használják ki. Egész pontosan azt, hogy a mikroszámítógép képes olyan adatot generálni, amit „soha senki" – a macskakörmök a 100%-os biztonsággal szembeni szkepticizmust fejezik ki – nem fog megtudni.

Orvosi kórtörténet: már többször említettük, hogy a smart kártyák programozhatóságuk miatt összetettebb információkezelésre alkalmasak. Egy beteg (vagy kezelt) kórtörténetét a legtöbb civilizált országban már régóta számítógépeken tárolják. Ez általában egy komplex adatbázis, mely egy központi szerveren foglal helyet, a háziorvos, mentőorvos, műtős terminálja ehhez kapcsolódik és kéri le az adatokat. Világos, hogy ember fizikai paraméterei, gyógyszerérzékenysége, korábbi betegségei stb. egy hatalmas adathalmazt alkotnak, melynek részei más-más szakemberre tartoznak. Milyen jó lenne, ha ezeket az információkat a zsebünkben tudnánk hordani a hozzáférési szabályok megőrzésével. Erre az igényre megoldás az smart kártyák e területen való alkalmazása. Az orvosi anyagok a kártyán tárolódnak és különböző hozzáférési szinteket hozunk létre a módosításukhoz, ahol a kérelmező autorizációját a belső számítógép biztosítja. Például a vércsoportunk publikus, így baleset esetén a kiérkező mentős azonnal képes a segítségnyújtásra, de az allergiás érzékenységeink listájához csak a szakorvos férhet hozzá. Szembetegségünk paramétereit mondjuk egy optikus olvashatja, de írására csak a kedvenc szemészünk képes. Ilyen szofisztikált jogkörrendszer megvalósíthatón egy smart kártyán. Előnye, hogy a végső soron ránk vonatkozó adathalmaz a mi birtokunkban van, nem pedig egy központi adatbázisban. Ez a megoldás jobban támogatja a demokratikus államoknak a személyiségi jogokra vonatkozó követelményeit. Az illető maga mondhatja meg, melyik orvos meddig juthat el kórtörténete megismerésében, miközben rá van kényszerítve a társadalom egészségügyi normáinak betartására. Egy ilyen jelegű általános rendszer kiépítése és megszervezése kormányzati feladat, bármelyik résztvevő engedetlensége a szisztéma felborulásához vezethet. A smart kártyák ilyetén alkalmazásának progresszív terjedése várható a közeljövőben.

Multifunkciós: megoldható, hogy egy darab kártya egyszerre képes legyen az összes felsorolt funkció szolgáltatására. Képzeljünk el egy kártyalap méretű eszközt, mely a GSM kommunikációban is segítségünkre van, fizetni is tudunk vele, tárolja a digitális aláírásunkat, a benzinkútnál jutalompontot kapunk rá és a rendelőben is hasznát vesszük. Egy ilyen kártya realizálásakor hatalmas infrastrukturális beruházásokkal kell számolnunk, konkrétan, ki kell építenünk egy átfogó, biztonságos, kényelmes kártyakezelő hálózatot. Bár a mérnöki tudás már rendelkezésre áll egy ilyen univerzális kártya bevezetéséhez, az anyagi, politikai és szociális akadályok miatt az eddigi, ezt megcélzó kísérletek kudarcot vallottak. Példának okáért, a hazai diákigazolványok 1998-tól állnak az összes akkreditált felsőoktatási képzésben résztvevő hallgató rendelkezésére, mégis az utóbbi 6 évben kevés helyen használták ki a rajta helyet foglaló intelligens csip adta lehetőségeket.

3.2. A smart kártyákon alkalmazott azonosítás formái

Egy védett erőforráshoz való hozzáférés általában két lépésben történik. Az első fázist autentikációnak nevezzük és a felhasználó kilétének kétséget kizáró meghatározását jelenti. Ezt kövei az autorizáció, ami az alany jogokkal való felruházásának szakasza. Mondanunk sem kell, hogy a smart kártyára fejlesztett adatkezelő algoritmusok is ezt a logikát követik. Az azonosításkor nem elég elhinni a jelöltnek, hogy ő az, akinek mondja magát, ezt valamivel bizonyítania kell. Háromféle módszert szokás alkalmazni, az elsőt „birtoklok valamit", a másodikat „tudok valamit" vagy jelszavas, a harmadikat egyéni jegy szerinti autentikációnak nevezzük. Egy smart kártya használatához már eleve birtokolnunk kell magát a kártyát. Az azonosítás első fajtája tehát minden smart kártyás autentikációra jellemző. Lássuk most, milyen formában implementálják az azonosítás két utóbbi formáját a smart kártyák világában!

PIN-kód: a legelemibb (és egyben legkockázatosabb) védelem, ha egy titkos PIN-kóddal (Personal Identification Number) zároljuk az adatokat. A birtokos ismer egy jelszó szerepű számot (általában 4-8 számjegyű), ez maga a PIN-kód. Ennek párja a kártya memóriájában is tárolva van. A smart kártya, aktiválása (reset jel) utáni első lépése, hogy bekéri a felhasználótól a PIN-kódot. A megadott szám feltöltődik a kártyára, a belső számítógép összehasonlítja azt saját példányával, amennyiben nem azonosak, megtagadja a válaszadást. Az elv nagyon hasonlít az aktatáskák vagy a páncélszekrények zárjának működéséhez. Plusz defenzíva lehet, hogy 3 darab sikertelen próbálkozás után a smart kártya megsemmisíti önmagát (törli a memóriáját). Azért, hogy a feledékeny birtokosok ne szenvedjenek el nagy tragédiákat, lehetőség van egy középső szint beékelésére. Ilyenkor a „megsemmisült" kártyát egy jóval hosszabb, ún. PUK-kóddal még újra lehet éleszteni, de a PUK-kód háromszori elvétésével már tényleg örökre elérhetetlenné válnak az adatok. Ha a PUK-kódot megfelelően nagynak választjuk, ez a kevésbé merev szisztéma nem add több esélyt a véletlen kóddal próbálkozóknak. A védelmi rendszer komplexitását lehet inkrementálni több PIN-kód használatával. A csip tartalmának védelmét az biztosítja, hogy a felhasználó titokban tartja választott számkombinációját. Vigyázni kell a túl elemi számkódokkal (pl. 1234, vagy 1111), mert a birtokos ötlettelenségére alapozva, a támadók ezeket fogják először kipróbálni. Az ilyen jellegű kódok elutasítását beépíthetjük a PIN-kód felújító rutinba, így növelve a kártya robosztusságát.

Biometria: „minden ember különböző” – hangzik az ismert frázis, ha viszont így van, akkor kell, hogy legyenek sajátságos jegyek, melyek árulkodnak az egyed többitől való differenciáiról. A biometrikus azonosítás arról szól, hogy az azonosításra váró személytől valamilyen biológia mintát veszünk, amit összehasonlítunk egy referenciával. Amennyiben az eltérés nem szignifikáns, elfogadjuk a célszemély identifikációját. Az azonosításhoz egy könnyen, gyorsan és fájdalommentesen digitalizálható, egyedenként jócskán eltérő jegyre van szükségünk: ujjlenyomat, retina, írisz, arc, DNS, hang, aláírás, egérmozgatási stílus. Az ilyen jellegű kutatások és kísérletek az utóbbi időben komoly eredményeket hoztak. A hatékony összehasonlító algoritmus bonyolultsága mellett, a mintaanyag hatalmas mérete okoz gondot. Példának okáért az emberi DNS több mint 3 milliárd bázispárból áll, ami óriási, $2^{3\,000\,000\,000}$ bitnyi adatot jelent, ennek 99%-a minden emberben azonos, a maradék 1%-ot kell az azonosításkor megvizsgálni.

Rendkívül hasznos lenne, ha ezeket a biometrikus azonosítókat egy smart kártyán tudnánk tárolni. A mintát fel tudjuk használni kilétünk igazolására, vagy lehet a kártya kulcsa (egy bonyolult PIN-kóddal analóg módon). A két felhasználás egyfajta ambivalenciát mutatat, az előbbi esetben az a célunk, hogy minél szélesebb körben ismert legyen a minta („ez tényleg az ő ujjlenyomata”), ezzel szemben az utóbbinál pont, hogy félteni kell azt a hamisítóktól. Az azonosítás folyamata igen egyszerű, egy terminál beolvassa például az ujjlenyomatunkat, átküldi a smart kártyának, az összehasonlítja a benne lévő etalonnal, és pozitív eredmény esetén szolgálatkész állapotba kapcsol. Sajnos a smart kártyák jelenlegi, fizikai paraméterei (számítási sebesség, tárméret) nem teszi képessé a biometrikus azonosításra, de sok gazdasági, tudományos projekt foglalkozik ilyen célú fejlesztéssel. A referenciamintát tárolhatnánk egy egyszerű adathordozón is akár, de úgy a támadó kezébe kerülhet és segítségével megalkotható egy hamis személyjegy. Persze ez nem egyszerű feladat, mert a leolvasók élő mintákat várnak, és vajon ki tud manapság az enyémhez hasonló lenyomatú retinát növeszteni? Mindenesetre jobb, ha a mintát egy olyan „dobozban” tartjuk, ahonnan nem lehet vagy nehéz kiszedni; az intelligens csip egy ilyen doboz.

A biometrikus azonosítás nagy problémája, hogy a fogyatékosok számára használhatatlan. Egy kéz nélküli személytől nem tudunk ujjlenyomatot venni, egy vaknak meg nincs élő retinája. Minden külső jegynek megvan a maga fogyatékossága, ezek egyikét sem használhatjuk általánosan. Áthidaló megoldás a DNS vizsgálata, de az mag még lassú és költséges eljárás, egy mikroszámítógép számára pláne elvégezhetetlen feladat. Egyébként még ez sem tökéletes technika, gondoljunk csak az egypetéjű ikerpárokra. Valamilyen

tudás, készség (írás, egérmozgatás, fogalmazási stílus) elemzése talán jobb lenne, de itt ismét belefutunk a fogyatékossággal kapcsolatos problémába.

4. Az ISO/IEC 7816-os szabvány

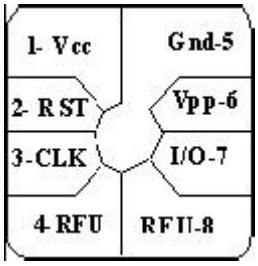
Az International Organization for Standardization (ISO) egy világméretű, elsősorban technológiai szabványok kibocsátásával és gondozásával fogalakozó szervezet. Az általuk kibocsátott szabványok igyekeznek garantálni a termékek közötti kompatibilitást, az elvárható minőséget, az egészség- és környezettudatosságot. Az ISO hálózatának 148 ország a tagja, központja Svédország. 1947 február 23-án rakták le az alapjait, az utóbbi 60 év alatt a legszélesebb körben elfogadott szabványkibocsátóvá vált. Általában igaz, hogy minden új keletű kémiai, gépészeti, elektronikai, informatikai technológiát igyekeznek ISO szabvánnyá tenni. Minden, a neve alatt kiadott szabványt egy szám azonosít, ezek mindegyike nyilvános, bárki számára hozzáférhető. Az ISO 7816 a smart kártyák nemzetközileg elfogadott standardja. A szabványt támogatja egy másik szervezet, az IEC (International Electrotechnical Commission), ezért szokták használni az ISO/IEC 7816 megnevezést is. Léteznek ugyan egyéb szabványok (pl. ISO 14443 – contactless smart kártyák, CEN 1546, ETSI – telekommunikáció, EMV – bankkártya-forgalmazók szabványa), de az elterjedtsége miatt ezt a szabványt vesszük most górcső alá.

Az ISO/IEC 7816 részei:

Fejezet	**Leírás**	**Kiadás**
ISO 7816-1	Fizikai paraméterek.	1987
ISO 7816-2	A kapcsolódási pontok leírása (kártyaolvasók részére).	1988
ISO 7816-3	Az elektronikus jelek és a fizikai szintű átviteli protokoll leírása.	1989
ISO 7816-4	Belső utasítások.	1995,1998
ISO 7816-5	Alkalmazások kezelésének leírása.	1994,1996
ISO 7816-6	Adatelemek, formátumok leírása.	1995
ISO 7816-7	SCQL (Structured Card Query Language) parancsok leírása.	1998
ISO 7816-8-9	Az egész piacot átfogó biztonsági követelmények.	1999
ISO 7816-10	Szinkron kommunikációs smart kártyák.	1999

A szabványt részekre (fejezetekre) szokás bontani, minden rész egy külön tulajdonsággal, elvárással foglalkozik. Az első rész a fizikai karakterisztikáját adja meg az integrált

áramkörnek. Definiálja a külső természeti hatások (mágneses mező, UVfény, statikus elektromos mező, hőmérséklet, mechanikai deformáció) elleni állóképesség küszöbértékeit illetve ezek beállításához ajánl néhány tesztelési technikát (ISO 10373). Hivatkozik továbbá a 7810-es ISO szabványra, ami a intelligens csipet bennfoglaló plasztiklap dimenzióit (szélesség: 85,72 mm/3,375 inch, magasság: 54,03 mm/2,125 inch, vastagság: 0,76 mm [±0,08]/0,03 inch) és a csip pontos helyét írja le. A csip külső felületének és a leolvasó szerkezet lábainak stabilan kell érintkeznie a számítások alatt, ezért nem hagyhatók figyelmen kívül a szabvány mechanikai tulajdonságokra vonatkozó előírásai.
A 2. rész a csip kommunikációs interfészének topológiájáról szól. Meghatározza, hogy a 8 külső láb (C1-C8) milyen sorrendben és milyen geometriai elrendezésben fedje le a csip külső felületét. Sajnos ezt a fejezetet sok gyártó nem veszi elég komolyan és saját lábelrendezéssel áll elő.



2. ábra

A csip szabványos felülete

A lábak funkciói:

Láb	**Kód**	**Magyarázat**
C1	VCC	A kártya tápellátása.
C2	RST	Ezen a lábon keresztül lehet az mikroszámítógépet alapállapotba hozni (reset jel).
C3	CLK	Órajel, mellyel szabályozni tudjuk a belső CPU sebességét, elsősorban az adatcserét bonyolító általános protokoll használja.
C4	RFU	Későbbi felhasználásra fenntartva.
C5	GND	A földelés.
C6	VPP	A mikroszámítógép programozásához használt tápellátás (opcionális).
C7	I/O	Half-duplex kommunikációs csatorna a mikroszámítógép és a terminál között.
C8	RFU	Későbbi felhasználásra fenntartva.

A következő, 3. szakasz az elektronikus szignálokat és az adatátviteli protokollok felépítését írja le. Elsősorban azoknak a villamosmérnököknek szól, akik a kártya és a leolvasó közötti legalacsonyabb szintű bitáramlást hivatottak megvalósítani. Az ISO szabvány megszabja az órajelek frekvenciatartományát, a feszültségértékeket (Vih, Vil, Vcc, Vpp, Voh, Vol stb.) és szervezésük karakterisztikáját. A specifikált protokollok részletes bemutatása nem kapcsolódik szorosan a dolgozat tárgyához, ezért nem kerül ismertetésre. Dióhéjban annyit érdemes tudnunk, hogy a kommunikáció egy inicializáló reset jellel indul, melyet a terminál ad ki. Az adatok oda-vissza, sorosan közlekednek egy 1 bites csatornán. Az adatcsere half-duplex módon (kétirányú, adott időben csak egyik üzenhet), hibavizsgálattal zajlik. A tápellátást a terminál szolgáltatja (VCC), a nem felejtő belső memória kaphat külön feszültségértéket (VPP).

A 4-es sorszámot viselő rész a smart kártya és az olvasó közötti üzenetek, utasítások, válaszok formáját deklarálja, biztonsági megfontolásokra is kitérve. A dokumentum támogatja, hogy a kártya memóriájában strukturáltan tudjunk információt elhelyezni. Igyekszik laza kereteket szabni, hogy a kártyafejlesztő cégeknek lehetőségük legyen kedvük szerint kiegészíteni a standard sémát.

Az adatszervezési logika teljesen analóg az elterjedt fájlos absztrakcióval. A fájlrendszer gyökerében egy kötelező előfordulású MF (master file) helyezkedik el. Ez alá szerveződnek fás adatszerkezetben az állományok. A fa csúcsaiban, a szokványos könyvtárakat kiváltó, DF-ek (dedicated file) helyezkednek el, ezek írják le az alájuk besorolt fájlok listáját. Minden DF-re külön zárat (védelmet) rakhatunk, így egy nagyon kifinomult hozzáférési hálózatot alakíthatunk ki. A fa levelei az ún. EF-k (elementary file), melyeknek két fajtáját különböztetjük meg. Míg a belső EF-khez (internal EF) csak a mikroszámítógép férhet hozzá, addig a működő EF-khez (working EF) bárki. Minden fájlnak – típusától függetlenül – egy 2 bájtos azonosítója van, kitüntetett a $3F00_{16}$ szám, ez az MF rögzített kódja. Egy konkrét állományt az elérési útjával (DF-ek sorozta az MF-től a levélig, végül a fájl saját kódja) címezhetünk meg. A protokoll négyféle fájlstruktúrát körvonalaz. A legelemibb a szekvenciális adatábrázolás, mely simán a bájtok egy sorozatát jelenti (transparent EF). Az ISO támogatja továbbá a blokkos fájlok kezelését, ami azt takarja, hogy a fájlban az egyes blokkok külön-külön azonosíthatók. A blokkméret lehet fix vagy változó. A 4. formátum a ciklikus blokkfájl. Egy adott állomány struktúráját attribútumai mutatják meg. A felhasználói hozzáférést minden egyes fájlnál egyedileg meghatározhatjuk. A szabvány elterjedtségére utal, hogy már létezik olyan szoftvercsomag

(SCFS – Smartcard Filesystem), mellyel UNIX-os operációs rendszer alá tudunk felmountolni egy ISO 7816-4-es fájlrendszert.

Fájl elnevezési konvenciók:

Azonosító	**Magyarázat**
0000_{16}	PIN1-et tartalmazó fájl (CHV1).
0001_{16}	Belső kulcsfájl autentikációhoz.
0002_{16}	Egyedi sorszámfájl.
0011_{16}	Külső kulcsfájl autentikációhoz.
0012_{16}	Privát RSA kulcsfájl.
0100_{16}	PIN2-őt tartalmazó fájl (CHV2).
1012_{16}	Publikus RSA kulcsfájl.
$2F01_{16}$	ATR (answer to reset) fájl.

A 4-es fejezet tárgyal még egy fizikai réteg feletti kommunikációs protokollt. Alapvetően egy mester-szolga (master-slave) típusú mechanizmusról van szó, ahol a mester a CAD, a feladatvégrehajtó pedig a smart kártya. Az üzenetváltás alapegységét az APDU-k (Application Processing Data Unit – szabad fordításban: alkalmazásprotokoll-adategységekben) képezik. Ennek két típusa létezik, melyek logikailag szorosan kapcsolódnak egymáshoz; utasítás, illetőleg válasz APDU. Az utasítás-válasz párok négy logikai csatornán közlekedhetnek, és a smart kártya teljes körű vezérlését teszik lehetővé. Az utasítás APDU egy 4 bájtos fejlécből és egy változó méretű törzsből tevődik össze. A fejléc első két bájtja (CLA, INS) lényegében az elvégzendő elemi feladatot határozza meg, a harmadik (P1) és negyedik (P2) bájt az utasítás paraméterei. A törzs az utasításhoz tartozó különleges adatokat tartalmazza (például, hogy mit írjunk egy fájlba). Első bájtja (Lc) – ha az nulla, akkor 2. és 3. bájtja – a törzs hosszát, a valódi adathalmazt követő Ls bájt (esetleg 3 bájt) pedig a várt válasz maximális méretét határozzák meg. Ezek megléte az adott utasítás jellegétől függ, akár az egész törzsrész is hiányozhat. A válasz APDU szintén egy változó kiterjedésű törzsrészből ($Lr \leq Lc$ hosszú), valamint két kötelező záró bájtból áll. A válasz APDU méretét az indukáló utasítás tartalmából konzekvensen ki lehet számolni. Az végső két bájt (SW1, SW2), az utasítás végrajtásának sikeréről ad tájékoztatást, hibafelderítésre használhatók. A kommunikációt a terminál kezdi a reset jel kiadásával, erre a kártya az ATR (answer to reset) fájl tartalmával válaszol. Az ATR a kártya típusát,

gyártóját és alapvető paramétereit írja le. A további működés a belső számítógép és a CAD interakciójának függvénye.

Az APDU-k szerkezete:

Utasítás:	CLA	INS	P1	P2	Lc	$D_1,D_2,\ldots,D_{Lc}$	Ls
Válasz:	$D_1,D_2,\ldots,D_{Lr}$			SW1		SW2	

A szabványos APDU utasítások:

CLA	**Elnevezés**	**Magyarázat**
$B0_{16}$	**READ BINARY**	Olvasás egy adott szekvenciális EF-ből.
$D0_{16}$	**WRITE BINARY**	Új adat írása egy szekvenciális EF-be.
$D6_{16}$	**UPDATE BINARY**	Egy konkrét szekvenciális EF már meglévő részét tudjuk vele felülírni.
$0E_{16}$	**ERASE BINARY**	Töröli a megadott szekvenciális EF egy részét.
$B2_{16}$	**READ RECORD(S)**	Egy blokkos EF adott blokkjának/blokkjainak olvasása.
$D2_{16}$	**WRITE RECORD**	Egy blokkos EF egyik blokkjának írása (felülírás, bináris VAGY-olás, bináris ÉS-elés).
$E2_{16}$	**APPEND RECORD**	Egy folytonos EF esetén a sorozat végéhez konkatenálja, egy ciklikus EF esetén viszont az első pozícióra írja az új rekordot.
DC_{16}	**UPDATE RECORD**	A parancs kezdeményezi egy blokk felülírását.
CA_{16}	**GET DATA**	Segítségével a futó alkalmazás egy változójának (rövid élettartamú adat) értékét olvashatjuk ki.
DA_{16}	**PUT DATA**	Segítségével a futó alkalmazás egy változójának (rövid élettartamú adat) értékét írhatjuk felül.
$A4_{16}$	**SELECT FILE**	Segítségével kiválaszthatjuk a későbbi utasítások célfájlját.
20_{16}	**VERIFY**	Egy kártyán tárolt adatot hasonlít össze az olvasóról érkezővel (pl. jelszó, PIN-kód).

CLA	Elnevezés	Magyarázat
88_{16}	**INTERNAL AUTHENTICATE**	Hatására a kártya egy belső titokkal sifrírozza a törzsben kapott kihívást. Kihíváson alapuló azonosításhoz használjuk.
84_{16}	**GET CHALLENGE**	Hatására a kártya egy tesztmintát (általában egy véletlen számot) küld az olvasónak, melynek aláírott változata alapján fogja azonosítani a terminált. Kihíváson alapuló azonosításhoz használjuk.
82_{16}	**EXTERNAL AUTHENTICATE**	Autentikálja – a legutolsó **GET CHALLENGE** utasításra kiadott tesztminta aláírt változata segítségével – a felhasználót/terminált. Kihíváson alapuló azonosításhoz használjuk.
70_{16}	**MANAGE CHANNEL**	A 4 logikai csatorna kezelésére használatos.
$C0_{16}$	**GET RESPONSE**	Olyan APDU-k fogadására használhatjuk (a terminál szemszögéből), melyeket a szabványos protokoll másképp nem tesz lehetővé.
$C2_{16}$	**ENVELOPE**	Olyan APDU-k küldésére használhatjuk (a terminál szemszögéből), melyeket a szabványos protokoll másképp nem tesz lehetővé.

Nézzük meg, milyen biztonsági mechanizmusokat támogatnak a szabvány ezen fejezetében definiált parancsok! A legalapvetőbb egy tárolt és egy megadott szöveg (általában PIN-kód) összehasonlítása, ezt a **VERIFY** utasítás végzi.

Egy másik lehetőség, hogy a kérelmező (CAD) bizonyítja, hogy egy olyan kulcs birtokában van, amit csak ő ismerhet. Ennek alapját a aszimmetrikus sifrírozások adják. A kártya a **GET CHALLENGE** utasításra egy véletlen tartalmú, „kihívó” szöveget küld válaszul, ezt kell digitálisan aláírnunk. Az aláírt szöveget visszatölthetjük a mikroszámítógépbe az **EXTERNAL AUTHENTICATE** utasítással. Amennyiben a várt titkos kulccsal rejtjeleztünk, a kártya a felhasználó nyilvános kulcsával vissza kell tudja állítani az eredeti szöveget és biztos lehet a kérelmező kilétében. Ez az ún. kihívásos azonosítás.

Az eljárás kicsit eltérő változata szimmetrikus titkosítással is működik. A smart kártya és birtokosa ismer egy közös kulcsot. Mind a kártya, mind a kérelmező titkosítja a **GET CHALLENGE** utasítás adta referenciaszöveget. Ha azonos eredményre jutnak, akkor nagy valószínűséggel egyező kulcsot használtak (ezt a smart kártya ellenőrzi az **EXTERNAL**

AUTHENTICATE hatására). A módszer gyengébb a digitális aláíráson alapuló változatánál, mivel ebben az esetben a titkot mindkét fél ismeri (smart kártya és birtokos), míg ott az kizárólag csak a birtokos előtt ismert. Egy konkrét példát találunk erre a 6.4.1. alfejezetben.
Az **INTERNAL AUTHENTICATE** szintén egy kihíváson alapuló azonosítás támogatását szolgáló parancs, azzal a különbséggel, hogy ezzel az utasítással a CAD hívja ki a smart kártyát és nem fordítva. Általában minkét (külső és belső) kihívásra sor kerül, ez a már megemlített, kézfogásos módszer.
A szabvány tehát alapvetően ezeket a módszereket támogatja, de a használt kriptográfiai algoritmus megválasztásában szabad utat enged. Mint az már a fájlrendszer bemutatásánál láttuk, az autentikációs módszer minden DF csomópontban külön-külön megadható, felülbírálható.
A szabvány ötödik fejezete a csipen futó applikációk azonosításának általános módszeréről szól. Az alkalmazásokat egyedi AID (Application Identifiers) kód azonosítja, ez két részből áll. Az első rész az ún. RID (Registered Application Provider Identifier), ez az 5 bájt hosszú szám a fejlesztő céget, az azt követő maximálisan 11 bájt hosszú szakasz pedig magát az algoritmust indexeli.
A hatodik fejezet a adatformátumokat hivatott egységesíteni (pl. képformátum, betűkészlet). Az adatbázis kezelők világában a lekérdező és definíciós nyelvek (SQL – Structured Query Language) már elnyerték létjogosultságukat. Egy az SQL-lel analóg nyelvet, az SCQL-t (Structured Card Query Language) formalizálja a ISO 7816 szabvány 7. fejezete. Ez a standard még csak kísérleti stádiumban van. A 8. és 9. fejezet a kártyakibocsátó iparág számára fogalmaz meg közös használatú, az adatbiztonságot elősegítő utasításokat. A általunk vizsgált utolsó fejezet (10.) a modern, szinkron kommunikációs smart kártyáknak szól.

5. Java Card-ok

A Java programozási nyelv története valamikor 1990 táján kezdődött. A Sun Microsystems-nél egy intelligens elektronikus eszközökhöz (pl. mobiltelefon, tejrendelő frizsider) használható programnyelv kifejlesztésébe fogtak (Green projekt). Az első sikeres változatot Oaknak keresztelte el egyik alkotója (James Gosling), a legenda szerint az irodája ablakából látható tölgy után. Ezt a megszólítást akkor már birtokolta egy másik programozási nyelv, ezért a Sun-nak új név után kellet néznie. Végül a projekt fejlesztői által elfogyasztott rengeteg kávé származási helyének (Jáva sziget – Indonézia) nevét adományozták neki (mi az angol „Java" írásmódot használjuk). Már-már úgy nézett ki, hogy a Green projekt kudarcba fullad, mivel a programozható elektronikus eszközök nem terjedtek a várt ütemben. A nyelvet az enyészettől a World Wide Web mentette meg. 1993 környékén a web már elég széles népszerűségnek örvendett és igény mutatkozott az addig többnyire statikus dokumentumok életre keltésére. A dinamikus tartalomhoz saját programozási nyelvre volt szükség.

Az elektronikus eszközökben megbúvó számítógép teljesítménye, architektúrája nagyon különböző lehet. A Green projekt dolgozóinak az a gondolata támadt, hogy az elkészült program mindig egy fizikailag nem létező, látszólagos processzor tárgykódjára forduljon. Ezt a processzort Java Virtuális Gépnek hívjuk, paraméterei, utasításkészlete kötött. A hardver kibocsátójának feladata, hogy a JVG-t implementálja az adott masinára, és onnantól kezdve az összes Java applikáció futtatható az eszközön. A nyelv portabilitása kapóra jött a webes fejlesztőknek. Ott is alapvetően az a probléma, hogy az összekötött sok ezer számítógéptípus színes palettáját kell a publikált programnak támogatnia. A befuccsolt Green projekt új erőre kapott, a piacvezető böngészők egyenként implementálták a JVG-t, ami a Java-t az „Internet programozási nyelvévé" tette. Sajnos az üzleti érdek keresztbe tud tenni egy egységesítési törekvésnek, nem úszta meg ezt a Java sem, ennek részletei azonban túlmutatnak e dolgozat témáján.

Általánosságban elmondható, hogy a Java egy modern, objektum orientált nyelv. Szintaxisa nagyban hasonlít a C, C++-hoz. Nem csupán támogatja az objektum, osztályok kezelését, valamint az azokkal kapcsolatos technikákat (öröklődés, virtuális függvények, konstruktor/destruktor stb.), hanem teljes mértékben objektum orientált. Az adatabsztrakció

magas szintjét teszi lehetővé. Támogatja a napjainkban divatos kivételkezelést. Sok korábbi feladatot igyekszik levenni a programozóról, ilyen például a szemétgyűjtés. A JVG ágens közbeiktatása miatt a Java-ban készített programok memóriaigénye jóval nagyobb, sebessége pedig harmada a velük ekvivalens C++-os, Delphis stb. programoknak. Mivel a kód sohasem közvetlenül a processzort irányítja, ezért egy Java-s program tevékenységét a JVG hangolásával tetszőlegesen korlátozhatjuk. Talán ez a tulajdonsága tette népszerűvé a smart kártyák területén. A mikroszámítógépre megírt interpreter figyelheti az illegális kódsorokat (pl. bizonyos fájlok olvasása) és elutasíthatja az ilyet tartalmazó programok végrehajtását. Előnyös az is, hogy a kártyán permanensen csak a JVG-t (smart kártyás terminológiában inkább a JKVG elnevezést szokásos használni) kell tárolni, miden érdemi tevékenységet végző alkalmazást elég közvetlenül a végrehajtás előtt letölteni az eszközre. Ez mindenképp elősegíti a multifunkciós smart kártyák tervezését, hisz minden célfeladat elvégzése előtt szimplán kicseréljük a használatos programot a korábbival és nem kell, hogy egyszerre jelen legyenek.

A Java nem szakadt el szülőterületétől, így a 90-es évek második felében megszülethetett a nyelv smart kártyás változata. A Java Kártyák (röviden JK) történetéről és szerepéről a 2. fejezetben már esett szó. Most a szabvány 2.2.2-es verzióját fogjuk áttekinteni.

A JavaCard API 2.2.2 (továbbiakban JK 2.2.2-es specifikáció) tulajdonképpen a Java nyelv speciálisan smart kártyákra tervezett könyvtárakkal való kiegészítése – értelemben leszűkítése. Ráépül az iparágban klasszikusnak számító szabványokra: ISO 7816, EMV. A fejlesztői környezetben lefordított programokat angolszász szóval cardleteknek hívjuk. Amikor a programozói felületről, a keretrendszerről, natív metódusokról és a JKVG-ről együtt beszélünk, a futtatási környezet (JCRE – Java Card Runtime Environment) elnevezéssel élünk. A valóságban a JKVG két részből áll: a Java Kártya Átalakító (JavaCard Converter) a terminálon helyezkedik el, ez végzi a bonyolult objektumkezelő feladatokat (pl. virtuális függvény tábla kezelése, hivatkozások feloldása), míg a puszta bájtkód értelmező a belső számítógépen helyezkedik el. A kettő közötti interfész maga a cardlet. Ráadásul a JVG-nek több olyan funkciója is van, amit a JKVG-be nem raktak bele. A szálak kezelését biztosító osztályok a kártyaprocesszor egyfolyamatos volta miatt kimaradtak. A túl nagy járulékos időköltség miatt a cardleteknél szemétgyűjtés sincs, amit egyszer lefoglaltunk – értsd egy végrehajtás alatt –, az többé nem szabadul fel. Az elemi típusoknak csak egy része használható: egészek, 1 dimenziós tömbök. A lebegőpontos számokat vagy a karakter típust nem értelmezi a JKVG.

A JK-kra való programfejlesztés 5 lépésből tevődik össze. Először elkészítjük a cardlet forráskódját. Ezt aztán Java bájtkódra fordítjuk. A harmadik lépés, hogy egy konverter segítségével átalakítjuk a tárgykódot úgy, hogy a JKVG fent említett szétbontott működéséhez adekvát formátumú legyen. Az így kapott eredményt szimulátoron tesztelhetjük, validálhatjuk. Végül feltölthetjük a kártyára a végeredményt. Egy másik lehetőség, hogy a cardletet az Internet egy konkrét pontján helyezzük el, ahonnan a hálózati hozzáféréssel rendelkező kártyaolvasó igény esetén letöltheti és installálhatja a JK-ra. Az ilyen cardleteknek természetesen valahogy jeleznünk kell hitelességét, hogy a kártya is meggyőződhessen jogosságáról. Ezt kézenfekvően a program digitális aláírásával tehetjük meg.

Ez a bekezdés némiképp bepillant a JKVG programozásába, ezért alapszintű Java ismereteket feltételez. Egy cardlet mindig a **javacard.framework.Applet** leszármazottja. A megoldandó feladathoz tartozó specifikus algoritmusokat ennek egyes tagfüggvényeinek felülírásával implementálhatjuk. A cardlet betöltése után **install()** metódusa hívódik meg, itt végezhetjük el az egyszeri, statikus inicializációs feladatokat (pl. fájlok létrehozása). A telepítésnek a része kell legyen a **register()** metódus meghívása, különben a JCRE nem szerez tudomást a cardletről. Az ISO 7816-ból ismert APDU csomagokat a **process()** függvény kezeli. Ebben elágazások felhasználásával elérhető, hogy a különböző vezérlő utasításokra a kártya más-más módon reagáljon. Ha nagyon didaktikusak akarunk lenni, azt mondhatjuk, hogy tulajdonképpen a **process()** függvény maga a program. Megjegyezzük, hogy vigyázni kell a függvény tervezésekor, mert egy végtelen ciklus az egész kártya blokkolódásához vezet! A **select()** és **deselect()** parancsok a már a JCRE-ben regisztrált programok aktiválásának és deaktiválásának a függvényei. Mikor a Java környezet kap egy parancsot, hogy a tárolt néhány eljárás közül most az **App**-ot futtassa, akkor betöltés után rögtön az **App.select()** utasítás hívódik meg, melyben különféle előfeladatokat követhetünk el (pl. véletlenszám-generátor gyökerének beállítása vagy egy másik cardlet objektumának elérése). Mikor egy új cardletet választunk, az **App** program működése az **App.deselect()** végrehajtását követően felfüggesztődik. A eljárás párral tehát a cardletek közötti kapcsolgatást kezelhetjük. A program tervezésekor nem szabad figyelmen kívül hagyni, hogy a felhasználó bármikor kihúzhatja a kártyát a terminálból, leállítva ezzel a program futását. Ha ez valami összetett feladat közben történik meg, az nagyon súlyos következményekkel járhat. A probléma kikerülésére használhatjuk a **System.beginTransaction()** és a **System.commitTransaction()** eljárásokat. A **beginTransaction()** kiadását követően, a változtatások csak egy ideiglenes pufferre fejtik ki hatásukat. A JK memóriájának tartalma

csak a **commitTransaction()** parancs után, egyhuzamban módosul. Ezzel a trükkel lehet védekezni egy nem várt külső megszakítás negatív hatásai ellen.

A Java szabványos könyvtárak közül az JK-k világában csak a **java.lang** és a **java.io** lebutított változatait használhatjuk. A **javacard.framework** és a **javacardx.framework** könyvtárak tartalmazzák az alapvető ISO 7816-os szabványhoz kapcsolódó Java-s osztályokat, konstansokat, objektumokat. A kettő között az a különbség, hogy a **javacard.framework** tagjainak minden JK-n működniük kell, a **javacardx.framework** elemei közül viszont szabadon eldöntheti a gyártó, hogy melyeket támogatja kibocsátott kártyája. Mivel az utóbbi lényegében a 4. fejezetben bemutatott fájlrendszert kezelő függvényeket tartalmazza, a gyakorlatban lehet számolni azzal, hogy egy valamirevaló JK támogatni fogja. Az elterjedt biztonsági technikákat a **javacard.security** csomag használatával implementálhatjuk. Fontosak még a kriptográfiai társprocesszor használatát lehetővé tevő függvények, ezek a **javacardx.crypto** és **javacardx.cryptoEnc** könyvtárakban találhatóak meg.

A **javacard.framework** fontosabb osztályai:

Osztály	Rövid leírás
Object	Minden osztály őse, JK-s környezetben csak az **equals** metódust lehet használni.
Applet	Ennek leszármazottja az összes cardlet.
AID	A cardletek egyedi azonosítójának kezelésére szolgál.
APDU	Az ISO 7816-as szabvány kommunikációs protokolljának absztrakt osztálya. Megjegyzendő, hogy biztonsági okokból tagjai csak egy APDU puffert kezelnek, a tényleges adatküldés/fogadás a JKVG feladata. A puffer mérete 37 bájt, vigyázzunk a túlcsordulásra!
ISO	Az ISO 7816-os szabvány konstansainak beszédes elnevezéseit tartalmazza.
ISOException	A Java kivételkezelő szintaxisát követő, a válasz APDUk SW részének kezelésére szolgáló osztály. A kivételkezeléssel egyébként csínján kell bánnunk, mert a szemétgyűjtés hiánya miatt könnyen „felzabálhatjuk” a memóriát.
OwnerPIN	PIN-kód kezelését segítő osztály.
ProxyPIN	Egy cardlet ruházhat át biztonságos módon PIN-kódot valamelyik másik cardletnek ezen ágens osztály használatával.
Util	Pufferkezelő parancsok gyűjteménye.
System	Objektumok tranzienssé (leállítás után is elérhető) tételét, megosztását és a – már bemutatott – tranzakciókezelést segítő osztály.

Osztály Rövid leírás

FileSystem Az ISO 7816-os szabványban rögzített fájlkezelés Java-s reprezentációja.

A **javacard.security** fontosabb osztályai:

Checksum CRC (Cyclic Redundancy Check) ellenőrző összegek generálását támogató osztály.

KeyBuilder Kulcsgyártásra használható osztály (viszonykulcsokhoz).

KeyPair Kulcspárok tárolására szolgáló osztály.

MessageDigest Ezzel az osztállyal szövegek rövid kivonatát készíthetjük el – ami digitális aláíráshoz használatos –, a műveletet hash-elésnek is hívjuk.

RandomData Segítségével véletlen adatokat generálhatunk.

Signature Digitális aláírások kezelésének osztálya.

A **javacardx.crypto** fontosabb osztályai:

Cipher A sifrírozó eljárások absztrakt osztálya.

6. A titkosító algoritmusok

Ebben a fejezetben áttekintjük a különböző titkosítási és adatvédelemi módszereket. Minden típushoz igyekszünk néhány konkrét algoritmust is vázolni. Mindig kitérünk az adott algoritmus smart kártyákon való alkalmazhatóságára. A fejezet végén bemutatunk két gyakorlatban használt biztonsági protokollt. Az egyik a lehallgathatatlan kommunikációt, a másik a bizalmas, többrétegű adatkezelést megvalósító szisztémák iskolapéldája.

6.1. Hashfüggvények

Mikor egy szöveget visszafejthetően akarunk rejtjelezni, valamilyen bijektív leképezést kell használnunk, hogy az eredeti üzenet előállítható legyen a kriptogramból. Néha azonban elég, ha csak egyirányú a transzformáció. Ezt úgy képzelhetjük el, mint a húsdarálót, ahol a kijövő cafatokból soha többé nem állítható elő a nyers hús. Ki lehet találni a darálóval analóg viselkedésű matematikai függvényeket, ezeket összességében hashfüggvényeknek (vagy egyirányú függvénynek) nevezzük. A lehetséges hashfüggvények közül csak az olyanokat szeretjük, amelyek rendelkeznek néhány jó tulajdonsággal. Az egyik ilyen az ún. lavinahatás. Ha a bemenetet egy kicsit megváltoztatjuk, akár csak 1 bittel is, az eredmény lényegesen más lesz, a függvény nem mutat semmiféle folytonosságot. A hópehely méretű módosítás lavinává duzzad a végeredményben. Egy másik kritérium, hogy a hash-elés minden inputra fix méretű kimenetet produkáljon, és ez lehetőleg jóval kisebb legyen a bemenő paraméter átlagos méreténél. Ennek okát a felhasználási területek bemutatásakor érthetjük meg. Az előző elvárásból triviálisan következik, hogy biztos vannak olyan különböző bemenetek, melyeket a hashfüggvény ugyanoda képez. Mivel a függvény értelmezési tartományának számossága nagyobb a képtérénél, ezért elkerülhetetlen, hogy néhány függvényeredmény több lehetséges bemenetnek egyaránt hozzárendeltje legyen. Követelmény még az is, hogy a hash értékből ne legyen visszafejthető az eredeti adat, illetve annak egyenértékű – leképezés szempontjából ekvivalens – társa.
Az egyirányú függvényeknek két komolyabb alkalmazási területe van. Egyrészt szokták a jelszavakat hash értékük alapján tárolni. Mikor bejelentkezés történik, a kapott jelszót

szintén hash-eljük, majd a „darált” értékek egyenlőségét vizsgáljuk (a helyes működést a függvény tulajdonság garantálja). Amennyiben csak simán elmentjük a jelszómintát, és azonosításkor a kapott jelszót ezzel hasonlítjuk össze, fennáll a veszélye annak, hogy rossz kezekben kerül a jelszavakat tároló fájl, és onnantól kezdve nem beszélhetünk biztonságról. Ha viszont csak a hash értékeket tároljuk, az előző bekezdésben kimondott visszafejthetetlenségi tulajdonságnak köszönhetően – még ha a támadó ismeri is hashfüggvényünket – a megszerzett jelszófájl számára hasztalan. Másrétű felhasználási terület a lenyomatkészítés. Egy szöveges dokumentum digitális aláírásakor alapesetben az egész dokumentumot felhasználjuk az aláírás generálásakor. Egy 100 oldalas szerződés esetében ez nagyon sok számítási időt venne igényben. Egy gyorsan processzáló hashfüggvény alkalmazásával egy sokkal rövidebb és ráadásul fix méretű lenyomatát kapjuk a szövegnek. Így ezt már gyorsabban alá tudjuk írni, anélkül hogy a védjegy függetlenné válna a dokumentum tartalmától. A statikus méretet ezért kötöttük ki az előző bekezdésben. Az is látszik, hogy a lavina tulajdonság miatt, hiába írom át csupán a dokumentum 1%-át, a hiteles aláírás azonnal gyökeresen megváltozik. Nem lehet tehát az „eléírok egy 1-est” jellegű csalásokat véghezvinni. Ugyanezért nem alkalmazható a hash-elés biometrikus azonosításra, hisz ott csak statisztikusan egyezik a tárolt és a mért minta, sok a kettő közötti különbség, ami drasztikus hatással van a hash értékre. Azonos személy ujjlenyomata a legkülönfélébb hash értékeket produkálhatja, azok összehasonlítása értelmetlen. Vessünk most egy-egy pillantást a két legelterjedtebb hashfüggvényre!

6.1.1. MD5 (Message Digest)

A módszert 1992-ben publikálták. Tetszőleges hosszúságú szövegből 128 bites sorozatot generál. A pecsét kiszámítását egy kitöltési szakasz előzi meg. A szöveg végéhez konkatenáljuk az <10000…> végtelen bitsorozatot úgy, hogy az eredmény mérete 512-vel osztva pontosan 448-at adjon (ha már az eredeti üzenet is ennyit adott, akkor is végzünk kiegészítést). Ezzel még nem fejeztük be a szöveg kiegészítését, az egész végére kerül a szöveg 64 bites mérete (low-order bájtsorrendben), így a bemenet mérete az 512-nek többszöröse lesz. A lenyomat elkészítéséhez egy 128 bites puffert használunk, melyet induláskor a $0123456789ABCDEF_{16}$ értékkel töltünk fel. Ezek után végigiterálunk a szöveg 512 bites blokkjain, és egy keverőfüggvénnyel előállítjuk az aktuális blokk és a

puffer egy 128 bites „ötvözetét”, ami a puffer új értéke lesz. A keverőfüggvény a mindenki számára ismerhető szinusz függvény adott pontjaiban felvett értékeit használja fel a keveréshez. Az eljárás 32 bites processzorokhoz lett tervezve, de mivel csupán primitív számításokat igényel (kivéve talán a szinusz értékek kiszámítását, de ez táblázattal helyettesíthető), könnyedén implementálható smart kártyákra is. Hash-elhetjük pl. a PIN-kódokat az MD5 eljárással, bár ez a 16 bájtos méret miatt pazarlásnak tűnik, viszont a digitális aláíráshoz veterán pecsétfüggvénynek számít.

6.1.2. SHA-1 (Secure Hash Algorithm)

Keletkezése 1995 áprilisára tehető. Az MD5-höz hasonlóan 512 bites blokkokban dolgozza fel a szöveget, de 160 bites hash értéket generál. Az üzenet mérete nem lehet nagyobb 2^{64} bitnél. Ezt az algoritmust is az MD5-tel megegyező kiegészítési lépéssel kezdjük. Egy blokk keverése 4 szakaszból ál, minden szakasz 20 kört foglal magába. Minden kör egy elemi keverési lépést jelent, melynek programkódját itt nem közöljük. Elég annyit tudnunk, hogy szakaszonként változik a keverőfüggvény, illetve egy eltolásra használt konstans. Egy 512 bit hosszú blokk 16 darab 32 bites szóból áll, az SHA-1 ezeket cserélgeti, transzformálja körönként 5 db 32 bites munkaváltozó felhasználásával. A változók induláskor rögzített értékeket kapnak, később mindig az előző blokk feldolgozása utáni értékekkel folytatják munkájukat. A végeredményt, a szöveg bejárása után, a munkaváltozók összefűzésével kapott szám adja (5×32=160). Az algoritmus erősebb az MD5-nél, viszont sokkal nagyobb a számítási igénye, ráadásul nem követi az Intel számábrázolási konvencióját, ami a legtöbb architektúra esetében külön problémákat okoz. smart kártyán való alkalmazása, bár nem lehetetlen, de semmiképp sem egyszerű feladat.

6.2. Szimmetrikus titkosítás

Szimmetrikus titkosításnak nevezzük azt, amikor egy kulccsal sifrírozunk valamilyen nyílt szöveget, úgy, hogy ugyanazzal a kulccsal – esetleg egy másik eljárás használatával – vissza is lehet azt fejteni. Már nagyon régóta alkalmaznak szimmetrikus titkosításokat, sőt, egészen a XX. századig csak ilyen kriptográfiai módszerek voltak használatban.

Tökéletesen biztonságosnak nevezünk egy szimmetrikus rejtjelező algoritmust, ha végtelen számítási kapacitással sem lehet a nyílt szöveget visszafejteni. A tökéletes biztonságnak egyébként szükséges feltétele, hogy a kulcs mérete megegyezzen a nyílt szöveg mértével, valamint a kulcstér entrópiája nagyobb vagy egyenlő legyen, mint az üzenetek terének entrópiája. Ezt a tételt alapul véve alkalmaznak is OTP (One Time Pad – egyszer használatos bitminta) alapú titkosításokat. Ennek lényege és egyben gyengesége is, hogy a kulcs olyan hosszú, hogy le lehet vele takarni az teljes üzenetet. Általában nem várjuk el, hogy végtelen idő kelljen rejtjelezésünk töréséhez, a gyakorlatban megelégszünk azzal, ha „csak" tízszer annyi időbe kerül, mint a Föld kora. Cserében használhatunk lényegesen rövidebb állandó kulcsot. Most megnézzük a számítógépek világában legelterjedtebb szimmetrikus algoritmus prosperálásának főbb mérföldköveit, valamit egy továbbfejlesztett utódját.

6.2.1. A DES (Data Encryption Standard)

A DES elnevezésű algoritmust 1975-ben tették közkinccsé, bizonyos (nem elvi) részeit azonban ekkor még titokban tartották. 1977-ben vált nemzetközi szabvánnyá, azóta nagy karriert futott be a számítástechnika minden szegmensében. A DES a nyílt szöveget 64 bites blokkonként, 64 bites kulccsal titkosítja. A kulcs 64 bitjéből minden nyolcadikat ellenőrzési célokra használunk, így igazából a kulcs csupán 56 bites. Maga az algoritmus szerkezete a Feistel-struktúrát követi. Minden blokkot külön-külön dolgozunk fel. Először kötött módon összekeverjük a bemenet bitjeit, ennek inverzét a kódolás végén szintén elvégezzük. A 64 bites adatot két (bal, jobb) 32 bites részre vágjuk. A jobb oldali részt átadjuk egy speciális transzformátor függvénynek, a kimenet és a blokk bal oldalát, mint bitvektorokat, összeadjuk (másképp mondva XOR művelet végzünk). Majd az így kapott 32 bites számot a blokk jobb oldalára írjuk, a régi jobb oldal kerül a bal helyére. Ezt a lépéssort 16-szor hajtjuk végre, befejezésképp csinálunk még egy sima bal-jobb cserét. A transzformátor függvényt minden iterációban egy saját körkulcs vezérel, feladata, hogy megszüntessen minden linearitást. A 48 bites körkulcsokat, az 56 bit hosszú globális kulcs, két 28 bit nagyságú összefüggő darabjának (bal, jobb) iterációnkénti balra forgatásával (adott ciklusban 1 vagy 2 bitnyivel forgatjuk tovább), valamint a bitek keverésével kapjuk.

A megfejtő algoritmus a föntinek a fordított irányú megfelelője (a körkulcsok élőállítása és a cserék visszafelé zajlanak).
A DES alkalmazása a smart kártyák területén is nagyon elterjedt. Erre utal, hogy a JK 2.2.1-es specifikációban külön interfésze van: **DESKey**. Egyetlen gond vele, hogy megnyugtatóan biztonságos használatához a fent említett 64 bites, általánosan elfogadott méretnél nagyobb kulccsal kell implementálni. A nagy kulcsok kezelése a zsenge mikroszámítógépeken nehézkes.

6.2.2. A TripleDES és a 3DES

A TripleDES és a 3DES sifrírozások a fent bemutatott DES algoritmus egyfajta kiterjesztései. A trükk az, hogy háromszor akkora kulcsot használunk (192 bit), mint azt az ős esetében tettük. A kulcsot 3 db 64 bites darabra vágjuk, majd az elsővel DES sifrírozást hajtunk végre, a másodikkal desifrírozzuk az első menet eredményét, végül a harmadik kulcsdarabbal újabb DES-elés alá vetjük a szöveget (az angol elnevezések nyomán, szokás a CDC rövidítést használni). Észrevehető, hogy amennyiben az első két részkulcs azonos, valójában csak a harmadik kulccsal rejtjelezünk, az első két ellentétes kör kioltja egymást. Ez pedig úgy tekinthető, mintha a sima DES algoritmust alkalmaztunk volna, ezért mondhatjuk, hogy a címbeli kriptográfiai eljárások, annak egyfajta kiterjesztései. Ha mind a három kulcs különböző, akkor TripleDES-ről, amennyiben a két szélső kulcs azonos, akkor pedig 3DES-ről beszélünk. A visszafejtés két desifrírozásból és középen egy sifrírozásból áll (DCD), a kulcsokat fordított sorrendben kell felhasználni. Mindkét algoritmus kihasználja az eredeti DES azon tulajdonságát, hogy a be- és kikódolási procedúra megválasztása önkényes. Mindegy tehát, hogy először desifrírozunk, aztán sifrírozunk, vagy fordítva, végeredményben egyaránt a nyílt szöveget kapjuk vissza. A 3DES-t széles körben alkalmazzák, mert bonyolultsága megegyezik a szimpla DES-évél, viszont sokkal nagyobb biztonságot nyújt, ráadásul kompatibilis is vele. A TripleDES elterjedtsége kisebb, mert az emberiség jelenleg nem igényel ilyen erős titkosítást, sőt az USA-ban a törvény egyenesen tiltja az ilyen rejtjelezések polgári igénybevételét.

6.2.3. AES (Rijndael – Rijmen & Daemen)

1996-ban az NIST (National Institute for Standards and Technology) tendert írt ki egy DES-nél jobb algoritmus kiötlésére, a pályázatot AES-re (Advanced Encryption Standard) keresztelték el. 2000-ben hirdettek eredményt, a sok beérkezett algoritmus közül a Rijndael került ki győztesen, ami később a DES hivatalos utódja lett. A szabvány 3 darab kulcsméretet enged meg, ezek rendre 128, 192, 256 bit nagyságúak.

A DES-hez hasonlóan, ennek az algoritmusnak is egy ciklus képezi a vázát. A ciklus lépésszáma a kulcs méretétől függ. A ciklus magjában 4 különböző transzformáció foglal helyet, ezeket a Rijndael terminológiában rétegeknek nevezzük (**SubBytes**, **ShiftRows**, **MixColumns**, **AddRoundKey**). A algoritmus működésének megértéséhez vezessük be a state struktúra fogalmát, ami nem más, mint a nyílt szöveg bájtjainak mátrixa. A mátrix szélessége a kulcs mérete bájtokban, magassága fixen 4, és sorfolytonosan tartalmazza a titkosítandó bájtokat (a nyílt szöveg egy blokkját): B_{00}, B_{10}, B_{20}, … A **SubBytes** réteg egy egyszerű jelcserés kódolást hajt végre. A **ShiftRows** transzformáció adott mértékben elforgatja a state struktúra sorait. A következő réteg (**MixColumns**) a state struktúra elemeit 2-es maradékosztály fölötti 8-as rendű polinomokként kezeli, oszloponként kiszámolja ezek 4 darab lineáris kombinációját, az eredmények adják az új oszlopot (pl. $B'_{00}:=2\times B_{00}+3\times B_{10}+B_{20}+B_{30}$). A kombinációk úgy vannak kitalálva, hogy egy egyszerű 8-bites processzoron is nehézség nélkül implementálhatóak legyenek. A negyedik réteg (**AddRoundKey**) XOR-olja a körkulcs mátrixos alakját a state struktúrával. A körkulcsokat az eredeti kulcs „felduzzasztásával” kapjuk meg. Az így kiterjesztett kulcs ciklusszámlálóval indexelt darabja lesz az aktuális körkulcs. A desifrírozáshoz a fenti rétegek megfelelő inverzét alkalmazzuk.

A state struktúra:

kulcsméret: 128 bit	192 bit	256 bit
B_{00} B_{01} B_{02} B_{03}	B_{04} B_{05}	B_{06} B_{07}
B_{10} B_{11} B_{12} B_{13}	B_{14} B_{15}	B_{16} B_{17}
B_{20} B_{21} B_{22} B_{23}	B_{24} B_{25}	B_{26} B_{27}
B_{30} B_{31} B_{32} B_{33}	B_{34} B_{35}	B_{36} B_{37}

A Rijndael nem csupán rejtjelezési erősségben jobb a DES-nél, hanem a kis erőforrású architektúrákon való implementálhatóságban is. Nem kell mondanunk, hogy ez különösen

kedveltté teszi a smart kártya alkalmazások területén. A már sokat emlegetett Java Kártya szabvány az AES algoritmus fogadására is felkészült (**AESKey** interfész).

6.3. Aszimmetrikus titkosítás

Egészen a XX. századig a kriptográfia egyik alapelve volt, hogy a védett üzenetváltásban résztvevő felek rendelkeznek egy közös titokkal. Ezt a principiát cáfolják meg a aszimmetrikus rejtjelező algoritmusok. Itt ugyanis minden szereplőnek két kulcsa van; egy titkos és egy nyilvános. A titkos kulcsot csak a tulajdonosa ismeri, még a kommunikációs partner sincs tisztában vele. A nyilvános kulcs viszont mindenki számára elérhető, beleértve a potenciális támadókat, adattolvajokat is. A kulcspár két tagja a sifrírozás szempontjából egyfajta erős korrelációt mutat, de ennek ellenére nehézkes az egyik ismeretében a másikat kiszámolni. A *nehézkest* itt nyomatékosan értjük, mert a kulcsok között meglévő kapcsolat miatt ez sohasem lehetetlen. Az összes aszimmetrikusan titkosító algoritmus azt használja ki, hogy a nyilvános kulcsból csak exponenciális időben tudjuk megfejteni a titkos kulcsot. Azt pedig, hogy elvileg elképzelhetetlen egy gyorsabb algoritmus, a legtöbb esetben nem tudhatjuk bizonyossággal, ezért gyakran az emberiség „tudatlanságára" hagyatkozunk. A dolgot úgy lehet elképzelni, hogy nem azonos nehézségű Böhönyéről Budapestre (kulcspár előállítása), mint Budapestről Böhönyére (kulcspár törése) eltalálni. Egyszerűen jelenleg Böhönye nincs a főutakon kitáblázva, míg Budapest szinte mindenhol. Egy konkrét aszimmetrikus sifrírozás feltörése tehát egy hatékony nyilvános kulcs elemző számítás feltalálásán múlik, az analógiával élve: „Böhönye országos kitáblázásán".

Az aszimmetrikus titkosításnak nemcsak elméleti jelentősége van, több, gyakorlatias szemmel igencsak jelentős, jó tulajdonsággal rendelkezik. A klasszikus rejtjelezések nehézsége volt, hogy a feleknek egy biztonságos csatornán meg kellett egyezniük valamilyen titkos kulcsban. Ez a felgyorsult, elektronikus világban szinte elképzelhetetlen. Gondoljunk bele mennyi plusz fáradtsággal járna minden banki tranzakció előtt befáradni a bankba kulcsegyeztetés végett. Ráadásul egy n tagú közösségben $n \times (n-1)/2$ darab kulcsra van szükség ahhoz, hogy mindenki mindenkivel tudjon bizalmas párbeszédet folytatni. Ha nyilvános-titkos kulcspárt használunk, egyrészt nem kell titkos csatornán egyeztetnünk, másrészt csupán $2 \times n$ db kulcs kívánatos.

Az aszimmetrikus rejtjelezés prosperálását úgy tudjuk elképzelni, mintha először a nyílt üzenetet egy páncélládikába zárnánk, lelakatolnánk és elküldenénk a címzettnek. Ő nem tudja leszedni a lakatunkat, de rárakhatja a sajátját, és a duplán lezárt ládát visszaküldheti. Ekkor leszedhetjük saját lakatunkat, majd ismételten elpostázhatjuk, az immáron csak a címzett lakatja által óvott ládikót. A célszemély képes eltávolítani saját lakatját, és a sok munka után végre hozzájut az üzenethez. Sőt, még abban is biztos lehet, hogy tényleg tőlünk jött az üzenet (a lakat 100%-ig hiteles eredetét feltételezve), különben sosem került volna le a ládáról az eredeti lakat. A valóságba ez úgy működik, hogy először a saját titkos kulcsunkkal, majd a címzett nyilvános kulcsával sifrírozzuk az üzenetet. Fogadáskor a szöveg a címzett titkos, majd a saját nyilvános kulcsunkkal desifríroztatik.

A rendszer működése a nyilvános kulcs hitelén áll vagy bukik. Ha ugyanis mondjuk Bob akar Alice-nak üzenni (lásd a 13.2. függeléket), de Trudy képes saját nyilvános kulcsát, mint Alice-ét eladni, Bob jóhiszeműen Trudy kezére játssza bizalmas üzeneteit. Ezért fontos a nyilvános kulcsok hitelesítése, melyet egy harmadik fél (hitelesítő központ) tehet meg. Jelenleg, a szubszidiaritás elvét követe, hierarchikusan csatolt kulcshitelesítő szerverek látják el ezt a feladatot. Az általuk szolgáltatott kulcsok valódiságát saját önigazoló kulccsal garantálják. A rendszer nagyban hasonlít a piramisjátékhoz. Ha megbízunk egy ismerősünkben, akinek van olyan őszinte kapcsolata, aki igazolni tudja a nekünk üzenő személy nyilvános kulcsának valódiságát, akkor az ismeretségek bonyolult, skálafüggetlen hálózatának egy útját jártuk be. A kulcsszerverek hálózata, az ilyen ismerősi láncolatok informatikai realizációja.

Több ízben esett már szó a digitális aláírásról, melyet az aszimmetrikus titkosítás „tudom, hogy ki küldte” tulajdonsága teszi lehetővé. Ha az aláírás mondjuk Bob nyilvános kulcsával desifrírozva az eredeti szöveget adja, akkor biztosak lehetünk abban, hogy Bob az aláíró, hiszen csak ő ismeri a megfelelő titkos kulcsot. Valójában a gyorsabb számítás érdekében, nem az egész dokumentumot, hanem annak egy hash eredményét és a keletkezés dátumát szoktuk digitálisan aláírni.

Másik módja a digitális aláírás azonosításra való felhasználásának a kihíváson alapuló autentikáció. Alice küld Bobnak egy nagy véletlen számot, a kihívást. Ezt Bob titkosítja a titkos kulcsával és visszaküldi Alice-nak. Ha Alice Bob nyilvános kulcsával desifrírozva a visszakapott üzenetet az eredeti számot kapja, akkor biztos lehet benne, hogy Bob van a vonal másik végén (vagy legalábbis valaki, aki ismeri az ő titkos kulcsát). Most nézzünk meg két létező aszimmetrikus kriptográfiai módszert.

6.3.1. RSA (Rivest, Shamir, Adleman)

Az algoritmust Ronald Rivest, Adi Shamir és Leonard Adleman publikálták 1997-ben. Alapját a moduláris algebrák képezik. A titkosítás megértéséhez induljunk ki a matematikában jól ismert kis Fermat-tételből

$$a^{p-1} - 1 \equiv 0 \pmod{p}$$

Itt p egy tetszőleges prím, a pedig egy [0, p-*1*] intervallumba eső egész. A kongruencia mindkét oldalához adjunk hozzá egyet, majd szorozzuk meg az egészet a-val

$$a^p \equiv a \pmod{p}$$

A kis Fermat-tételnek létezik egy általánosabb, Euler φ függvényes alakja is (minden természetes számhoz a nála kisebb, vele relatív prímek számát rendeli)

$$\text{lnko}(a,n) = 1 \rightarrow a^{\varphi(n)} \equiv 1 \pmod{n}$$

Könnyedén bizonyítható, hogy az egyenlet mindkét oldalát pozitív egyész k-adik hatványra emelve, valamint a tétel második alakjára áttérve, a következő összefüggést kapjuk

$$a^{k\varphi(n)+1} \equiv a \pmod{p}$$

Keressünk két nagy prímet (p, q), ami valószínűségi prímtesztekkel „gyorsan” megtehető. Válasszuk a fönti n-t p és q szorzatának. Ekkor elemi módon bizonyítható, hogy $\varphi(n)=(p-1)(q-1)$. Ezt beírva az Euler függvény helyére, és a bal oldali kitevő egy szorzatalakját keresve ($ed=k\varphi(n)+1$) az alábbi feladathoz jutunk

$$ed \equiv 1 \pmod{(p-1)(q-1)}$$

Egy megfelelő (e, d) páros gyorsan meghatározható. A nyílt szöveget ezek után a min(p, q) méreténél kisebb méretű blokkokra bontjuk (általában a blokkméret az, ami adott, és ahhoz keresünk kellően nagy prímeket). A blokkokat sorban az e-edik hatványra emeljük, majd vesszük az n szerinti maradékát, így kapjuk meg a titkosított szöveget. Ha a kriptogram blokkjait most az d-dik hatványra emeljük, akkor előáll az eredeti betűsorozat

$$(a^e)^d \equiv a^{ed} \equiv a^{k\varphi(n)+1} \equiv a$$

Az e-t és az n-et nyilvános, a d-t titkos kulcsnak választva kész is van az aszimmetrikus sifrírozásunk. Rendkívül fontos tulajdonság, hogy a kódoló algoritmus igen egyszerű műveletekből áll, ráadásul lehetőség van némi optimalizálásra is. Ahhoz, hogy valaki kiszámolja d-t, az n-et kellene faktorizálnia, ami nagy számok esetén mai tudásunk alapján kivárhatatlanul lassú folyamat. Vannak azonban bizonyos speciális prímek, melyekből

generált (gyenge) kulcsok esetlenek abban az értelemben, hogy a velük titkosított üzenetek könnyebben visszafejthetők az általánosnál. Napjainkban használatos kulcsméret: 1028 bit. Az RSA implementációk elterjedésének sokáig a jogdíjfizetési kötelezettség szabott gátat, ami azonban 2000-ben feloldódott. Sok digitális aláírás RSA alapon működik, ezért a modern smart kártyából sem hiányozhat. Egyetlen baj vele a viszonylag méretes kulcs szükségessége, ami kis kapacitású tárolókat számottevően igénybe vesz. A JK 2.2.1-es specifikációban a kapcsolódó interfészek: **RSAPrivateCrtKey**, **RSAPrivateKey**, **RSAPublicKey**.

6.3.2. ECC (elliptikus görbék)

Mint arra már utaltunk, az RSA legfőbb problémája, hogy relatíve tetemes méretű kulcsok esetén nyújt csak megnyugtató biztonságot. Az elliptikus görbék (EG) matematikai vizsgálata egy sokkal kisebb kulcsigényű aszimmetrikus rejtjelezést adott az emberiségnek. Először tisztázni kell, mit nevezünk elliptikus görbének. Az EG olyan pontok halmaza a síkban, melyek kielégíti a következő egyenletet

$$y^2 = x^3 + ax + b$$

Az a-t és a b-t a görbe paramétereinek tekintjük. Az egyenlettel leírt halmazhoz definíció szerint még egy extra pontot hozzáveszünk; a végtelen pontot (jelölés: O). Az olyan paraméterezésű görbéket, melyekre igaz, hogy $D:=4a^3+27b^2=0$, nem tekintjük elliptikus görbéknek. Egyszerű igazolni, hogy minden elliptikus görbe szimmetrikus az x-tengelyre. Vegyünk most egy tetszőleges EG-t. Egy ezen lévő P pont ellentettje (-P) legyen x-tengelyre vett tükörképe (amiről már tudjuk, hogy eleme az EG-nek). A végtelen pont ellentettje önmaga (O=-O). Vezessünk be a pontokra egy összeadás műveletet!

Kezdetben tegyük fel, hogy $P\neq\pm Q$ és egyik sem végtelen. Ilyenkor az R=P+Q pont kiszámítása úgy történik, hogy a Q és a P pontokat összekötő egyenes EG-vel való 3. metszéspontját (-R) tükrözzük az x-tengelyre. Előfordulhat, hogy nincs külön 3. metszéspont. Ez esetben az egyenes a P és Q pontok egyikében az EG érintője, ott „kétszeres” metszéspontot feltételezünk, tehát P+Q=-P.

Amennyiben P=Q, de egyik sem a végtelén pont, akkor az R=P+P=:2P az EG P pontjához húzott érintő és a görbe metszéspontjának ellentettje. Ilyen metszéspont az E-n ($x^3+ax+b=0$) kívül mindig létezik, legyen 2E:=O.

A P=-Q esetben R=P+Q=P+(-P)=:O módon definiáljuk a művelet eredményét.

Maradt még az az eset, amikor Q végtelen és P tetszőleges. Ilyenkor a P+Q összeg nem más, mint P. Az EG pontjai a most meghatározott összeadás műveletére nézve csoportot alkotnak, melynek egységeleme az extremális végtelen pont (P+(-P)=O).

Térjünk most át a folytonos síkról a természetes számok halmazára. Ha az EG egyenletét moduláris aritmetikával írjuk fel, máris egy diszkrét rendszerhez jutunk

$$y^2 \equiv x^3 + ax + b \pmod{p}, \text{ ahol } p \text{ prím és } x,y,a,b \in [0, p-1] \text{ egészek}$$

Erre a furcsa görbére ([0, p-1] speciális részhalmazára) is definiálható az imént bemutatott összeadás, ez is hordozza a folytonos görbe kriptográfiai szempontból hasznos tulajdonságait, vagyis ez az összeadás is csoportművelet. A továbbiakban ezt értjük EG alatt, a folytonos definícióra csak a geometriai reprezentáció miatt volt szükség.

Nézzük most meg, hogyan használható fel az EG-k pontjainak matematikája aszimmetrikus titkosításra! Legelőször választunk egy tetszőleges EG-t és egy rajta lévő G bázispontot. Ezek legyenek nyilvánosak. Tegyük fel, hogy Alice akar Bobbal privát kommunikációt folytatni. Mindketten kiötölnek egy tikos egész számot, jelöljük ezeket a (Alice) illetőleg b-vel (Bob). Fejenként összeszorozzák a titkos kulcsukat – az imént választott számok – a G referenciaponttal

$$\overbrace{aG = G + G + \ldots + G}^{a}$$

Az összeg eredménye a nyilvános kulcs. Mikor mondjuk Alice üzen Bobnak, választ egy k véletlen egész számot, ami a kommunikáció során viszonykulcs szerepet fog betölteni. A közölni kívánt üzenetet bijektív módon leképezi az EG egy M pontjára (az ECC egyik sarkalatosan nehéz lépése). Ezek után elküldi a (kG, M+kbG) kódolt szöveget. Bob a kapott páros első tagját megszorozza a csak általa ismert b számmal, majd az eredmény ellentettjét hozzáadja a második taghoz

$$M + kbG + (-b(kG)) = M$$

Eve akkor tudná elolvasni az üzenetet, ha ismerné a b számot, amit a nyilvános bG-ből kéne tudnia kiszámolni. A visszafejtés módszerét a „diszkrét logaritmus elliptikus görbék felett” megnevezéssel illetik, csak exponenciális idejű algoritmust ismerünk rá, így Evenek, nagy modulusú algebrák esetén nagyon türelmesnek kell lennie.

Az ECC nagy problémája a megfelelő véletlen EG generálása, valamint a nyílt szöveg erre történő bijektív ráképezése. Eleve sok számolással jár eldönteni egy tetszőleges pontról (jelen esetben véges egész), hogy eleme-e a diszkrét EG-nek. A komplexebb számítási

igényért cserébe, sokkal kisebb az elvárhatóan erős kulcs mérete. Egyre több szoftverben cserélik le a matuzsálem RSA-t az ECC-re, ez alól nem kivételek a smart kártyás alkalmazások sem. Megszoktuk, hogy egy-egy rejtjelezés mikroszámítógépes létjogosultságát a JK 1.2.2-es specifikáció adekvát interfészeivel igazoljuk: **ECKey**, **ECPrivateKey**, **ECPublicKey**.

6.4. Titkosító protokollok smart kártyákon

A smart kártyákon alkalmazott leggyakoribb általános kriptográfiai metodikák után, vizsgáljunk most meg két konkrét protokollt. Az egyik mindennapjaink maroktelefonon bonyolított üzenetváltásának titkosságáért felelős. A másik protokoll a hiteles, ugyanakkor nyomon követhetetlen adatkezelést teszi lehetővé.

6.4.1. Titkosítás a GSM kommunikációban

Alapvető igény a GSM kommunikációval szemben, hogy az adatátvitel mások számára (kémhálózat, üzleti ellenfél) lehallgathatatlanul folyjon. A klasszikus telekommunikációban sok ilyen visszaélés történt, főleg a politika részéről, mindenki életében ismerős a „Nem telefontéma!” vagy hasonló megjegyzés. Ennek a telefonok hangátvivő csatornáihoz kötődött általános bizalmatlanság az oka. Pedig a GSM szabvány előírja az éteren áthaladó információ titkosítását. E titkosítás logikai vázát mutatjuk most be.

A titkosítás a mobilkészülék és a központ között zajlik. Az IP protokollhoz hasonlóan az adatok keretekben (frame) közlekednek. A keret mérete 228 bit, mindegyiket egy 22 biten ábrázolt, egyedi szekvenciaszám (frame number) indexel. Lényegében 3 darab speciális algoritmus használunk; az A3 egy azonosító, az A8 egy viszonykulcs generáló, az A5 család tagjai (A5/1, A5/2, A5/3) pedig kriptográfiai eljárások. Az első két algoritmust a SIM kártya tartalmazza, az A5 valamelyikét a mobiltelefon ismeri, modelltől függően. A szolgáltatónál egy azonosító központ (AuC – Authentication unit Center) működteti az A3 és A8 algoritmusokat. Minden telefonbeszélgetés az azonosítással kezdődik. Az AuC-tól kapott *R* véletlen számból és a SIM kártyán tárolt egyedi, „kártyába zárt” kulcsból (*Ki*) az A3- mal egy válaszüzenetet állítunk elő, ez a SRES. Mivel az azonosító központ is ismeri a

Ki kulcsot – minden hozzá tartozó SIM kártyáét ismeri – és a saját maga generálta *R* titkot, ellenőrizni tudja a SRES helyességét. Jó SRES érkezése esetén az azonosítás megtörtént. Egymástól függetlenül az A8 algoritmussal 64 bites viszonykulcsot generálnak (*Kc*) a SRES-ből. A SIM kártya ezt adja át a mobiltelefonnak, az AuC pedig a hálózati központnak, így a kommunikációban résztvevő felek – vigyázat, nem a két telefon! – ismernek egy közös kulcsot, ami sohasem járt az éterben. Minden keretet „átfuttatunk" a kiválasztott A5 algoritmuson, mely a rejtjelezéshez a keret szekvenciaszámát és a *Kc*-t használja, a központban aztán az átérkezett adat desifrírozásra kerül. Sajnos itt, de csakis itt, lehetőség van a lehallgatásra, azért a szolgáltató és az ügyfél kölcsönös bizalma rendkívül fontos.

Pillantsunk most bele az A5/1-es algoritmus belsejébe. 1987-ben publikálták Franciaországban, az LFSR (linear feedback shift register – lineáris visszacsatolt léptetőregiszter) rejtjelezők családjába tartozik. Működését titokban akarták tartani, de mint oly sok más rébusz, végül ez is az Interneten kötött ki. Feltörésére, érthetően, sok kísérlet született. Az eddig legjobbnak bizonyult törés, a beszélgetés első 2 percének analizálása után, az adatfolyam maradék részét valós időben fejti meg.

A mobilkommunikáció során folyamatosan keletkezik az adat, ezeket valós időben folyamatosan kell titkosítanunk, a tejes szöveg átfogó sifrírozására nincs mód. Kézenfekvő megoldás egy már említett OTP-n alapuló technika. Az LFSR eljárások a nyílt szöveget egy regiszterforgatásokkal előállított végtelen hosszú (valójában csupán determinisztikusan generatív) kulccsal XOR-olják össze. A kulcsot előállító rendszert szakszóval kulcsfolyam-generátornak nevezzük. Az egyvégtében előállított kulcsnak az elérhető legnagyobb entrópiájúnak kell lennie, különben a módszer gyengén titkosít. Az A5/1 kulcsfolyam-generátorának lelkét három léptetőregiszter (R1 19 bit, R2 22 bit, R3 23 bit) képezi, ezeket kezdetben a 64 bites A5/1-es kulcs megfelelően hosszú darabjaival inicializáljuk. A kulcsfolyamba R1, R2 és R3 vezető bitjeinek XOR-ja kerül. A regiszterek bizonyos, állapottól függő részhalmazát minden kiszámolt bit után elforgatjuk. A legbaloldalibb bit elvész, jobbról néhány, a specifikációban lefixált helyiértékű bitek összege (valójában XOR-ja) jön be. Azt, hogy mikor melyik regisztert kell elforgatni, az MCC (Majority Clock Control) függvény vezérli. Az MCC leszámolja a regiszterekben szereplő nullák és egyesek gyakoriságát. Azok a regiszterek kerülnek forgatásra, melyek középső bitje gyakrabban fordul elő a regiszterek együttes bináris tartalmában. Belátható, hogy az algoritmus, a regisztermeretekhez viszonyított leghosszabb periódusú kulcsfolyamot állítja elő, ami a nagy entrópia miatt fontos. Az A5/1 IC szinten is nehézség nélkül

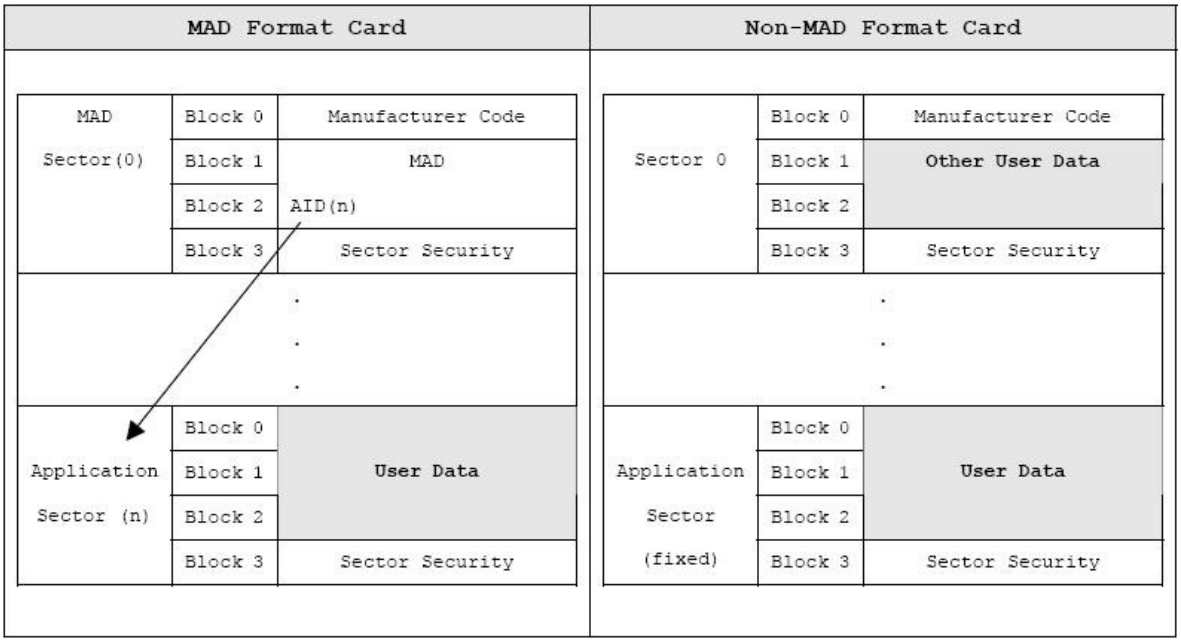
implementálható, ami, mobiltelefonokról lévén szó, valószínűleg nagyban segítette elterjedését, végül szabvánnyá válását.

Az A5/2-es az egyszerűbb hardverrel felvértezett mobiltelefonokhoz készült, értelemszerűen gyengébb az A5/1-nél. Az A5/3 az A5/1-nek egy továbbfejlesztett változata, mely minden eddiginél nagyobb biztonságot ígér.

7. Mifare Card Issuer

Nézzünk egy konkrét kártya író-olvasó készüléket. Az általam kipróbált berendezés egy Promag termék, melynek neve Mifare Card Issuer PCR 310. Ez az író-olvasó rádióhullámokkal kommunikál a kártyákkal. Frekvenciája 13,56 MHz, 2 cm távolságból már képes érzékelni a lapkát. A számítógéphez USB porton keresztül csatlakoztathatjuk. A mellékelt CD tartalmaz leírást és drivert a készülékhez, valamint programokat, amelyekkel manipulálni tudjuk a kártyán lévő adatokat. Létezik csak olvasásra használható berendezés is, de mi most ezzel nem foglalkozunk.

Az író-olvasó segítségével a kártyákat kétféle formátumúvá tudjuk alakítani. Létezik MAD és Non-MAD formátum. MAD típus esetén a kártya 0. szektora egy ún. Mifare applikációs könyvtárat (Mifare Application Directory; MAD) tartalmaz, ami arról ír le információt, hogy melyik applikációs szektornak mi az AID-ja (Application ID). Non-MAD esetén a Non-MAD Sector szám közvetlenül megadja az applikációs szektor számát.

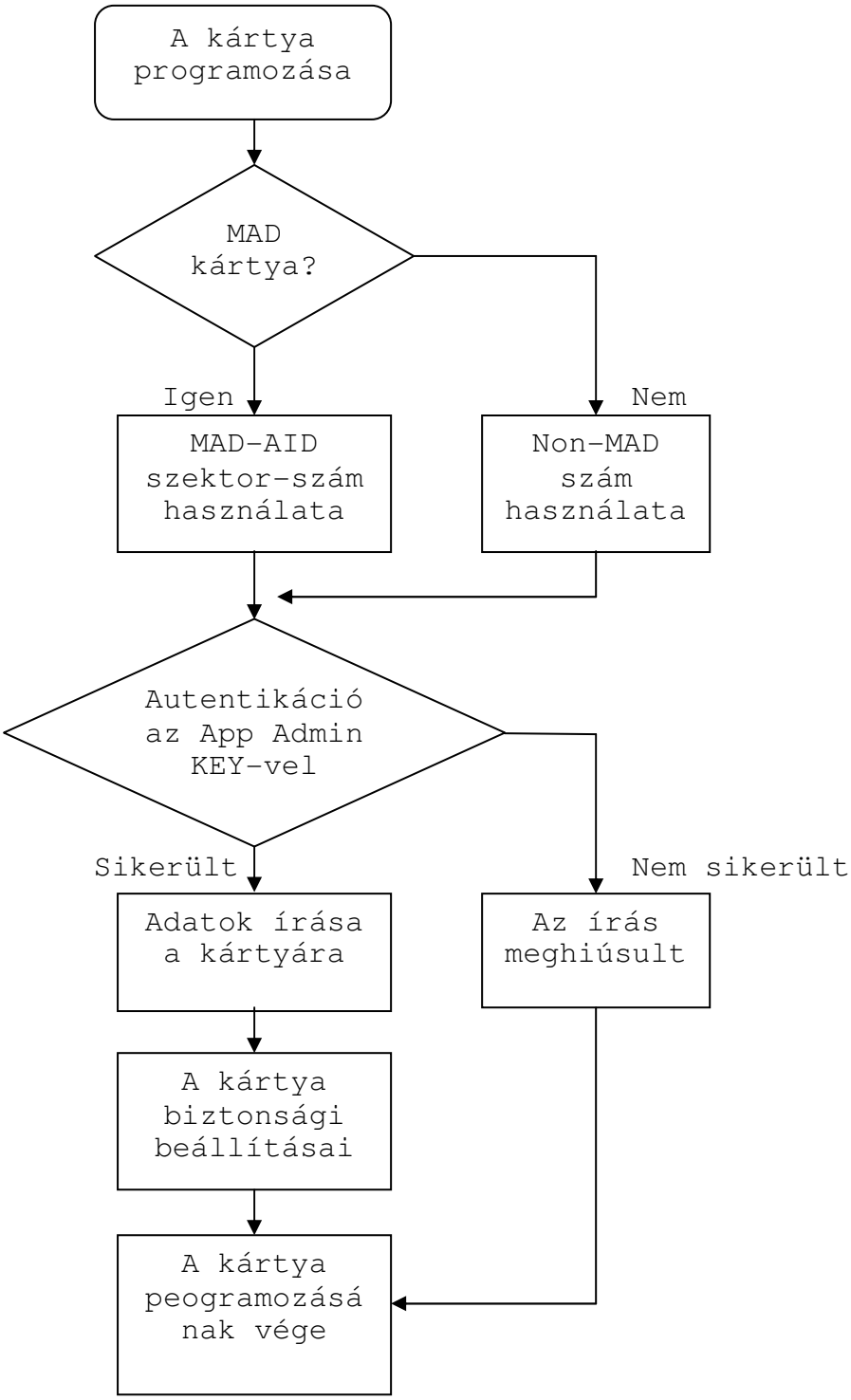


3. ábra

Mifare Card típusok

A Sector Security mezők a biztonságért felelősek. A MAD szektoron lévő mezőben található egy állandó kulcs, amelynek mindig ugyanaz az értéke, és csak olvasásra használható, valamint egy MAD Admin Key névre elkeresztelt olvasásnál és írásnál egyaránt szerepet játszó kulcs, amit MAD üzemmódban használunk. Az applikációs szektor Security mezőjében szintén két kulcs szerepel. Az egyik az App Key, amely az olvasó kulcsa ahhoz, hogy olvassa az applikációs szektort. Ez a kulcs tehát a hitelesítéshez (autentikáláshoz) szükséges. A másik az App Admin Key, aminek segítségével adatokat írhatunk a kártyára.

A Renew Card programmal Non-MAD típusúvá formázhatjuk, míg az MF700 Mifare Card Issuer program segítségével MAD formátumúvá alakíthatjuk, illetve adatokkal láthatjuk el a kártyáinkat. Ennek megtétele előtt azonban a Configure gombra kattintva beállíthatjuk a fent említett kulcsokat, adhatunk meg jelszót, amit minden indításkor bekér a Card Issuer program, illetve választhatunk titkosítási módszert. Összesen öt különböző titkosítás közül választhatunk, arról azonban nincs információ, hogy melyik lehetőség melyik módszert takarja. A következő folyamatábra azt mutatja, miként írhatunk adatokat a kártyára a program segítségével.



4. ábra

Kártyák „programozása” Mifare Card Issuer segítségével

8. Összefoglalás

A dolgozat általános képet ad a smart kártyák széles világáról. Segítségével bepillantást nyerhetünk a fejlődésük szakaszaiba, láthatjuk, mely területeken használják már ma is ezeket az eszközöket. Szól néhány szót az ISO/IEC 7816-os szabványról is, amely a smart kártyák alacsony szintű karakterisztikájára ad ajánlást, valamint a Java Card-okról, amelyek mai JK 2.2.2-es specifikációt használják. Bemutattuk a legfőbb titkosítási algoritmusokat, amelyek elengedhetetlenek ahhoz, hogy adataink biztonságban maradjanak – akár a kártyáinkon. Végül bemutatásra került egy komplett kártya író-olvasó eszköz, amelyet a hozzá mellékelt programokkal lehet használni.

A smart kártyák egyre szélesebb körben terjednek napjainkban, így nem kétséges, hogy komoly jövője lesz ennek a világnak. Már ma kártyák segítségével azonosítjuk magunkat egyre több munkahelyen, telefonálunk, pénztároló eszközként használjuk, és ki tudja, mi mindenre lesznek még használatosak a jövőben.

9. Felhasznált irodalom

Virasztó Tamás: Titkosítás és adatrejtés, NetAcademia Kft., 2004

Nyékyné Gaizler Judit: Java 2 Útikalauz programozóknak 1.3 I-II., ELTE TTK Hallgatói alapítvány, 2001

ISO 7816/1-10 szabvány, http://www.iso.org/

JK 2.2.2-es specifikáció, http://java.sun.com/products/javacard/specs.html

Az elliptikus görbe pontjaiból alkotott csoport, http://www.biztostu.hu/oktatas/kriptologia/ECC_3_csopoermuvelet.htm

Az A3, A5 és A8 kriptográfiai algoritmusok forráskódja, http://www.mirrors.wiretapped.net/security/cryptography/algorithms/gsm/

Intelligens Kártya Fórum, http://www.njszt.iif.hu/ikf/

AES, Advanced Encryption Standard, http://www.biztostu.hu/mod/resource/view.php?id=211

Algoritmusos adatvédelem, http://indy.poliod.hu/program/oktinf/info3.html

Mifare Card Issuer PCR310 programmer for MF700 Reader User's Manual

The DES Algorithm Illustrated, http://www.aci.net/Kalliste/des.htm