Diplomaterv

Programozható chipkártyák nyújtotta biztonság

Berta István Zsolt

Konzulens:

Dr. Vajda István

Híradástechnikai Tanszék

Budapesti Műszaki és Gazdaságtudományi Egyetem

2001.

Diplomaterv kiírás

Berta István Zsolt

A programozható chipkártyák hitelkártya méretű biztonságos mikroszámítógépek, alkalmazásuk új területeket nyithat meg biztonságtechnikai alkalmazások esetén. Ugyanakkor kis erőforráskészletük komoly kihívást támaszt, ez különösen jelentős lehet kriptográfiai műveleteket igénylő alakalmazásokban.

Tekintse át a téma magyar és a külföldi szakirodalmát, majd ismertesse a programozható kártyák által nyújtott új lehetőségeket! Vázolja fel programozásuk főbb lehetőségeit, módszereit, kihívásait. Térjen ki a platform jellegzetesebb problémáira, biztonsági veszélyeire!

A megismert eljárások, és technológiák ismeretében tervezzen meg egy mintarendszert, amely demonstrálja a programozható chipkártyák által nyújtott újdonságokat, különös tekintettel azok adatbiztonsági alkalmazására. Vizsgálja meg, a tervezett rendszer mennyivel nyújt többet egy hagyományos (generikus) chipkártyákkal megvalósított rendszerrel szemben! A problémát biztonságtechnikai szempontból közelítse meg, s gondolja át a kulcsgondozás témakörét!

Készítse el a rendszertervet, majd implementálja a szoftvert PC-n, illetve amennyiben a szükséges hardver rendelkezésre áll, valósítsa meg a rendszert chipkártyán is.

Nyilatkozat

Alulírott Berta István Zsolt, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

.....

Berta István Zsolt

Abstract

Programmable smart cards are small security-oriented microcomputers. Although they have been present in the market for many years now, their exact area of application is still subject to research.

The author gives a detailed background about these cards in this paper. A card is not only discussed by itself, but together with its environment: the terminal, the network resources and the user.

A brief overview of today's programmable cards is given, but focus is laid on the Java Card specification, which is one of the most popular smart card programming environments. Various features of the Java Card are discussed, especially those in connection with security, the main power of smart cards.

In this paper three applications for programmable smart cards are presented. The first application is an elliptic curve cryptography engine for a Java smart card. In this case the programmable smart card is used as a prototype to test new algorithms in smart card environment.

The second application uses the smart card to store the profile of a user of a heterogeneous system. The card plays an important role in user authentication, but in this system not only the user is authenticated. The smart card also checks the identity of the terminal and protects the user's interests by denying certain information toward the insecure (or possibly malicious) terminal. In this application the programmable smart card is used as a platform for a security oriented software. The algorithm it runs is so complex that implementations other than software are totally out of the question.

The third application is not a pioneer by any means. It does not break into new areas of cryptography for smart cards, and does not explore unknown areas of complex smart card applications either. It is a simple, but very useful program, that gives extra security in SSH challenge and response authentication. The cardlet for this low-resource machine was developed in the Java Card language, and thus it was integrated into the world of high level programming languages.

Tartalomjegyzék

1. Bevezetés

A programozható chipkártyák, valójában biztonságos mikroszámítógépek, amelyek megjelenésükkel új teret nyitnak az adatbiztonsági alkalmazások számára. Már több éve jelen vannak a piacon, mind számuk, mind jelentőségük fokozatosan nő. Ennek ellenére, a mai napig nem terjedtek el olyan mértékben, hogy kiszoríthatnák elődeiket, az úgynevezett hagyományos kártyákat.

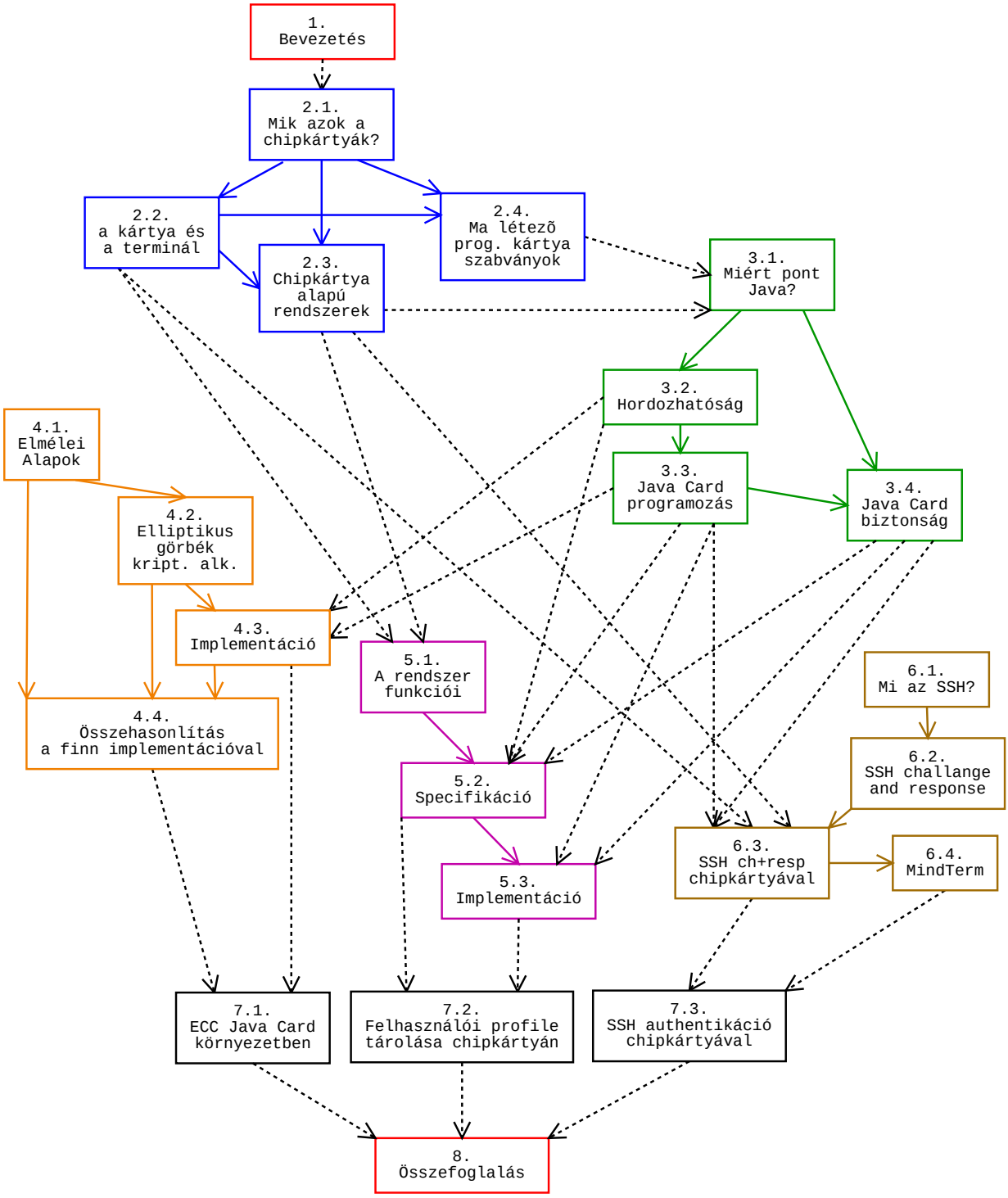
Ennek fő oka, hogy mind a mai napig nem tisztult le, mennyiben nyújtanak többet a hagyományos kártyáknál. Igaz, számos új tulajdonságuk, lehetőségük ismert, pontos képességeik azonban még a mai napig is kutatások tárgyát képezik. Diplomatervem keretében én is ezen témával foglalkoztam, és a következő fejezetekben az e téren szerzett tapasztalataimat, elért eredményeimet ismertetem.

Először, a 2. fejezetben a programozható kártyák témakörét járom körül. Beszélek általánosságban a chipkártyákról, majd bemutatom a hagyományos chipkártyák és a programozható chipkártyák között lévő különbséget. Áttekintem a programozható kártyák ma ismert programozási módszereit, alkalmazási lehetőségeit is. Később, a 3. fejezetben a Java Card programozási környezetet mutatom be, amely a ma létező három nagy chipkártyák programozására alkalmas környezet egyike. Szólok előnyeiről, hátrányairól, gyenge pontjairól, valamint sajátos biztonsági problémáiról is.

Az ezt követő három fejezetben kiragadok egy-egy alkalmazási lehetőséget, majd az erre elkészített implementációmat mutatom be. A 4. fejezetben azon programomat mutatom be, amely az elliptikus görbék elméletén alapuló nyilvános kulcsú kriptográfiát valósítja meg Java Card környezetben. Ezen témával már az Országos Tudományos Diákköri konferencián is sikerrel szerepeltem.

A második példaalkalmazás, a [33] által bemutatott ötleten alapul, és szintén a hagyományos kártyák képességeit meghaladó módon használja ki a chipkártyát. Az 5. fejezet ezen szoftverről, specifikációjáról és implementációs nehézségeiről szól. A fejezet végén az implementációs nehézségekről, felmerülő problémákról, valamint szükséges tervezői döntésekről írok.

Végül, a 6. fejezetben részletesen bemutatom azon programomat, amely challenge and response alapú SSH bejelentkezést tesz lehetővé chipkártya segítségével, és komoly lépést jelent a gyakorlati hasznosítás irányába.



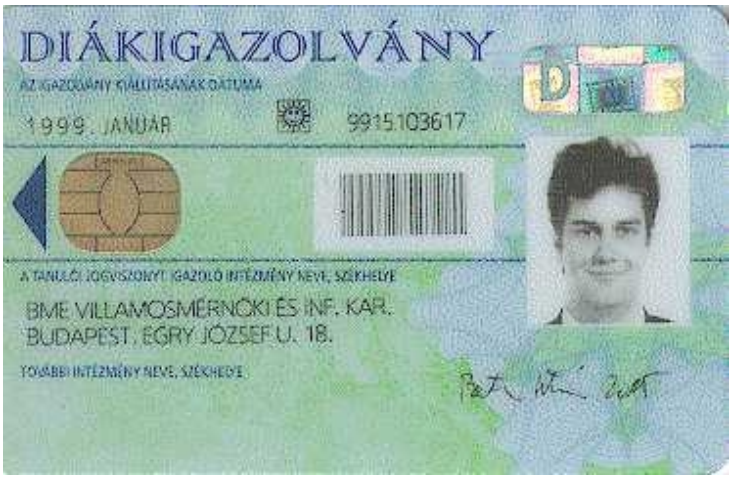
1. ábra. Függőségi viszonyok a diplomaterv fejezetei között

Irodalmi és technológiai háttér

2. Hagyományos és programozható chipkártyák

2.1. Mik azok a chipkártyák?

Chipkártya alatt olyan műanyag kártyát értünk, amelyen mikrochipet helyeztek el (2. ábra). A kártyán információkat elsősorban e chip segítségével tárolhatunk, de a chip mellett létezhet a kártyán egyéb információtárolási lehetőség is (például mágnescsík, dombornyomás, hologramm, vagy a kártyatulajdonos aláírása).



2. ábra. Egy chipkártya

2.1.1. Az első kártyák

A ma chipkártyának nevezett eszközök ősei az 1950-es években megjelent (és mind a mai napig) széles körben alkalmazott mágneskártyák. Az ezeken a kártyákon elhelyezett mágnescsík néhány száz byte adat mágneses úton való tárolására alkalmas. Az 1970-es években merült fel mikroelektronikai áramkörök alkalmazásának lehetősége. A francia Bull cég 1979-ben készítette el első mikroprocesszorral is rendelkező kártyáját. Ezen az „ősi” kártyán

a processzor és a memória még külön chipben helyezkedtek el (2.1.4. fejezet). Később, a 80-as években, a technológiai fejlődés lehetővé tette minden alkatrész egyetlen mikrochipre való integrálását.

A chipkártyák széles körben elterjedtek, hiszen számos előnnyel rendelkeznek a mágneskártyákkal szemben. Kevésbé érzékenyek mágneses zavarokra, valamint jóval biztonságosabbak, hiszen a kártya processzora kontrollálja az írási-olvasási műveleteket. A telefonkártyák között már régóta a chipkártyáké a vezető szerep, de manapság kezdenek megjelenni a bankok konzervatív világában is pénzkiadó automaták (ATM) és chipkártya alapú home banking megoldások képében.[1]

2.1.2. A chipkártyák három generációja

Ha azt mondjuk, „chipkártya" (smart card), akkor csupán azt mondtuk, hogy a kártyán mikrochip található. Arról nem szóltunk, hogy ez a mikrochip mekkora intelligenciát biztosít a kártya számára. E fejezetben a chipkártyák három csoportját, „generációját", mutatom be, melyek nemcsak időben követik egymást, de a kártyában lévő intelligencia szempontjából is három egymást követő szinten helyezkednek el.

1. A legegyszerűbb chipkártyák az ún. *memóriakártyák*. Ez esetben a kártyán lévő IC lényegében egy memóriachip. A terminál közvetlenül hozzáfér a chip tartalmához: olvashatja, írhatja. Ezek a kártyák a logikai biztonság szempontjából nem jelentenek komoly áttörést a mágneskártyákhoz képest, – az általuk nyújtott logikai biztonság egy floppy disk-évelekvivalens.

2. Jelentős előrelépést jelentett a *generikus kártyák* megjelenése. Ezen eszközök logikai és esetleg kriptográfiai módszerek segítségével védik a rajtuk tárolt adatokat. [49], [34], [2] Az adatok file-okba, azok pedig file-rendszerbe szerveződnek. A kártyán definiálhatunk felhasználókat, és azok jogosultságait. Megadhatjuk, mely felhasználóknak milyen módszerrel szükséges azonosítani magukat – PIN kód, challenge and response (C függelék) –, illetve mely file-okhoz, könyvtárakhoz milyen módon (írás, olvasás, inkrementálás, stb.) férhetnek hozzá. A legújabb generikus kártyák szimmetrikus és nyilvános kulcsú kriptográfiai módszerekkel is védhetik az adataikat. Ezen kártyák nevében a „generikus" szó arra utal, hogy a kártyagyártó által legyártott eszköz egy perszonalizációs folyamaton keresztülmenve kerül a végfelhasználó kezébe. A

perszonalizáció arra szolgál, hogy a kártyakibocsájtó a gyártó által elkészített általános célú (generikus) kártyát saját céljaihoz igazítsa. Nem szükséges tehát minden célra külön kártyát gyártani, a termelés legutolsó fázisa a perszonalizálás (felhasználók, filerendszer, kulcsok, jogosultságok) kártyára való felvitele a kártyakibocsájtóhoz kerülhet, míg a gyártó általános célú kártyákat hozhat létre.
Eleinte az volt a vélekedés, hogy a generikus kártyák nagyon drágák, és a legtöbb célra sokkal olcsóbb lesz a memóriakártyák használata. Azonban a perszonalizáció gyártásról való leválasztása új utakat nyitott meg a tömegtermelés előtt. A nagy példányszámnak köszönhetően a generikus kártyák ára olyannyira lecsökkent, hogy mára szinte teljesen kiszorították a memórikártyákat. Gyakran olyan célokra is használnak generikus kártyát, ahol nem is lenne rá szükség.

3. A *programozható kártyák* képezik a chipkártyák harmadik generációját. Ezen eszözök – a generikus kártyák minden tudása mellett – képesek felhasználói programok futtatására. Ez azt jelenti, hogy – akár a kártya élete során – letölthetünk rá újabb alkalmazásokat, s ezeket rajtuk futtathatjuk. Ez nemcsak azt eredményezi, hogy bármikor kiterjeszthetjük a kártya által nyújtott szolgáltatásokat, hanem azt is, hogy jóval nagyobb intelligenciával ruházhatjuk fel őket, mint a generikus kártyákat. Míg egy generikus kártya esetén csupán az adatok struktúráját, és azok hozzáférési lehetőségeit állíthatjuk be a perszonalizáció során, egy programozható kártya segítségével bármilyen védelmi mechanizmusok és műveletek implementálhatóak, határtalan teret nyitva ezzel a kártykibocsájtóknak.
   A programozható kártyák jelenleg igen drágának minősülnek. Ugyanakkor, figyelembe véve az általuk nyújtott lehetőségeket, valamint az irányukba jelentkező igényt mind a banki alkalmazások, mind a mobil kommunikáció, mind pedig az elektronikus kereskedelem terén, várható, hogy áruk a tömegtermelésnek köszönhetően jelentősen csökkenni fog. Ezen dolgozat célja a programozható kártyák által nyújtott új lehetőségek vizsgálata. Ennek megfelelően, a későbbiekben a programozható kártyákkal foglalkozom, és amennyiben erről külön szó nem esik, „chipkártya” alatt a dolgozat további részében programozható kártyát értek.

Fontosnak tartom annak hangsúlyozását, hogy a generációk közti határ korántsem éles. A memóriakártyák és a generikus kártyák egyaránt adattárolásra szolgálnak, csupán az utóbbi csoport kevésbé enged közvetlen hozzáférést az adatokhoz a külvilág számára.

A generikus kártyák és a programozható kártyák közti határ sem határozott. Egyes generikus kártyák jelentős kriptográfiai képességekkel rendelkeznek, és védelmi mechanizmusaik is roppant rafináltak lehetnek. A fő különbség – amely egyben jól definiált határvonalat is jelent – az, hogy amíg a generikus kártyák processzorán csakis az a kód futhat, amelyet gyártójuk engedélyezett, és rájuk telepített, a programozható kártyák felhasználó által írt kódot is végrehajthatnak.

2.1.3. A kártya belseje

Mint azt a mellékelt ábra mutatja (3. ábra), a chipkártya a következő fő alkatrészekkel rendelkezik: [16, 2. fejezet, 16. oldal]

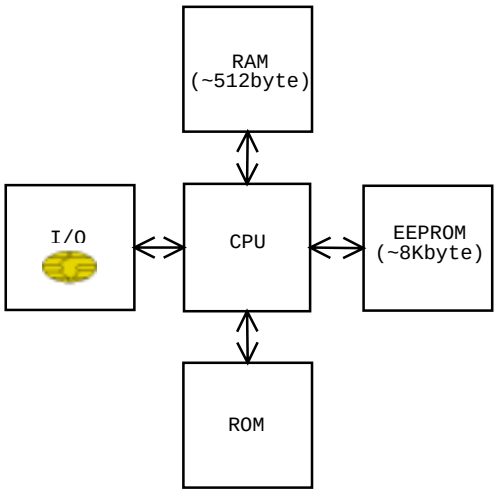
1. CPU[1]
2. Háttértár (EEPROM)
3. Memória (RAM)
4. I/O egység

Ennek alapján a chipkátyát számítógépnek tekinthetjük. Valójában nem is egyszerű számítógépekről van szó, hanem különleges, biztonságos számítógépekről. A biztonság ezen kártyák fő erőssége. Ez kiderül blokkvázlatukból is (3. ábra). A fő különbség a Neumann-féle architektúrájú számítógépek és a chipkártyák között, hogy az utóbbi esetben az I/O perifériák nem kapcsolódnak közvetlenül a belső buszokra.

Chipkártyák esetében a külvilág (I/O periféria) és a adattároló egység (EEPROM) között egy döntő logika helyezkedik el, amely a bemenetet megszűrheti, felülbírálhatja. Az ábrán látható egységek egyetlen chipben helyezkednek el, nincsenek közöttük buszok, amelyeket egy külső támadó esetleg lehallgathat. A chip pedig olyan mikroelektronikai technológiák [34] segítségével készült, hogy minél nehezebb legyen belőle információkat a kártya felnyitása esetén kinyerni.

A kártyát kívülről egynek és oszthatatlannak szeretnénk tekinteni, amelyből információkat csakis a kontaktusokon keresztül nyerhetünk ki. A kontaktusok jelentik a kártya egyetlen kapcsolatát a külvilággal. Ez fontos, hiszen a chipkártyák biztonságának egyik legjelentősebb pillére az, hogy csakis egy jól definiált és jól ellenőrzött interfésszel rendelkeznek a külvilág felé.

[1] A chipkártyák CPU-ja tipikusan 8 bites, de létezik nagyobb is. Ma 32 bites a legnagyobb. [41]



3. ábra. Egy chipkártya belsejének blokkvázlata

2.1.4. A chipkártyatechnológia korlátai

A fenti technológia (2.1.3. fejezet) ugyanakkor korlátokat is támaszt. Azzal, hogy minden alkatrésznek (3. ábra) egyetlen chipbe kell kerülnie, a chip bonyolulttá válik, valamint igen komoly hőelvezetési problémák jelennek meg. Ez erősen bekorlátozhatja a kártya órajelét, számítási és tárolókapacitását. [5]

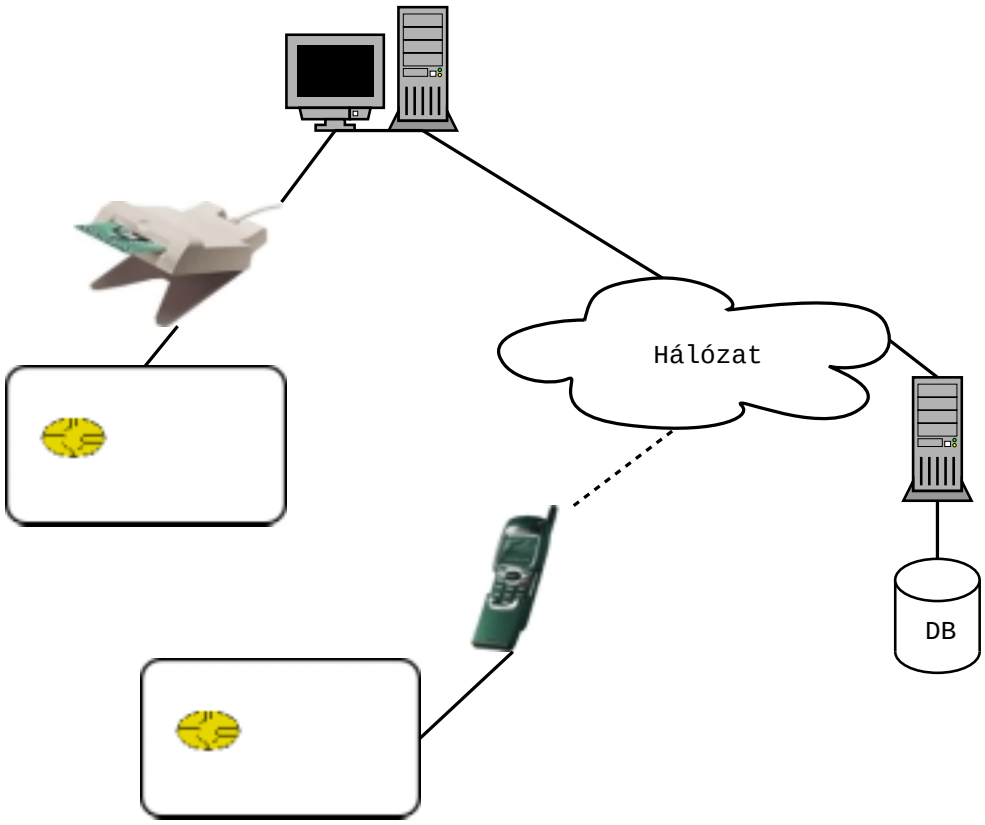
A gyártók másik kritikus problémája a kártya előállítási költségének minimalizálása. Ez azért különösen jelentős, hiszen rendkívül nagy példányszámú gyártásról lehet szó.

2.2. A kártya és a terminál

2.2.1. Mit nevezünk terminálnak?

A chipkártya input-output műveleteket csakis a mikrochip kontaktusain végezhet. Miközben ez a szűk interfész a kártya adatainak biztonságát szolgálja, megakadályozza a felhasználót azok közvetlen elérésében. A kártya "fogyatékos számítógépnek" tekinthető, amely nem képes önállóan kommunikálni még a saját felhasználójával sem, így ezt egy másik eszköz, az ún. terminál biztosítja.

A terminál elég sokféle lehet. Érthetünk alatta banki automatát (ATM), de lehet szó mobiltelefonról, set-top-boxról, de akár PC-ről is. Lehetséges, hogy a terminál csak a felületet biztosítja a kártya számára (5. ábra), de gyakori, hogy a terminál egyben hálózati kapcsolatot is biztosít (4. ábra). Tipikus eset (például mobiltelefonok esetében a SIM, azaz



4. ábra. A chipkártya szerepe a hálózatban

Subscriber Identity Module kártya, amikor a chipkártya az előfizető azonosítására szolgál, s a terminál a kártya behelyezésével válik anonim terminálból az előfizetőt képviselő eszközzé.

Akármilyen alakú és tudású is a terminál, roppant fontos szerep jut neki mind funkcionális, mind biztonsági szempontból. Csakis ő biztosíthatja ugyanis a kártya számára:

- a felhasználói felületet,
- a hálózati kapcsolatot,
- a tápfeszültséget (!) és
- az órát[2].

[2]Igaz, a hitelkártya formájú chipkártyák nem rendelkeznek saját órával és tápfeszültséggel, de léteznek olyan biztonságtechnikai mikroszámítógépek, amelyek igen. Ilyen például a Dallas Semiconductor terméke, az iButton. (D függelék és [15])

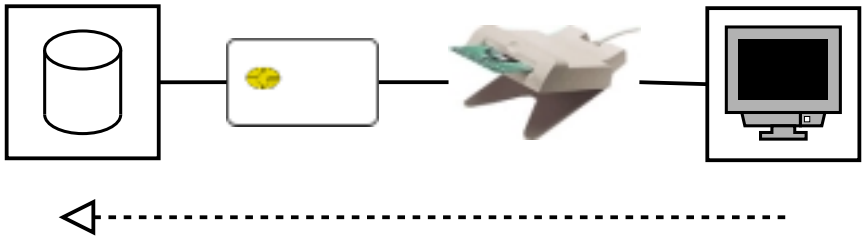
5. ábra. Egyenleglekérdező eszköz digitális pénztárca alkalmazáshoz (Mondex)

Mivel ezen eszközök egy "hagyományos" számítógép esetén a gép elválaszthatatlan részei, a chipkártya biztonságtechnikai analízis során nem tekinthető minden esetben számítógépnek, gyakran figyelembe kell venni olyan támadásokat is (pl. a terminál részéről), amelyek egy asztali gép esetén fel sem merülhetnek. [40]

2.2.2. Intelligencia

Az adatokat akkor védhetjük biztonságosan, hogyha megtagadunk hozzájuk minden közvetlen hozzáférést, és csakis bizonyos műveleteket engedélyezünk feléjük. Így nemcsak biztonságban tudhatjuk őket, de konzisztenciájukat is megőrizhetjük. Ehhez szükségünk van egyfajta intelligenciára, amelynek segítségével ezen műveleteket definiálhatjuk, valamint őket végrehajthatjuk.

Megfigyelhető, hogy ez az intelligencia eredetileg a terminálban volt, majd fokozatosan mozdult el a kártya felé, egyre közelebb és közelebb az adatokhoz (6. ábra).



6. ábra. Az adatokat kezelő intelligencia eltolódása a terminálától az adatok irányába

A mobiltelefonok esete bizonyos szempontból speciálisnak mondható (2.2.3). Ez esetben ugyanis a felhasználó magával hordozza a terminált is, nemcsak a kártyát, így a terminál biztonságáról jóval erősebb feltételezések tehetők. Itt érdekes verseny zajlik a chipkártya-gyártók, valamint a mobiltelefongyártók között: Mindkét fél a saját berendezését próbálja meg minél több intelligenciával felruházni, és azt elérni, hogy a jövőben az ő oldalán legyenek az intelligencia által nyújtott szolgáltatások.

Ezt mutatja az intelligens mobiltelefonok terjedése (pl. naptár, szövegszerkesztő, stb. funkciók), valamint az egyre szaporodó programozható SIM kártyák (pl: SIM ROCK, GemXpresso SIM). Ugyanez a verseny vezet azokhoz furcsa esetekhez, amikor ugyanazon szolgáltatások (pl. telefonkönyv) léteznek mind a telefonban, mind a kártyában, és a felhasználó soha nem találja meg semmijét sem.

2.2.3. A kártya – célpont vagy támadó?

Amikor a kártyákat passzív eszközöknek tekintettük, a „chipkártyás” alkalmazás fejlesztése a kártya azonosítását, majd a kártyából adatok kinyerését jelentette. A tervező a terminál szemszögéből nézte a kártyát, célja a kártya, valamint a kártyához tartozó felhasználó azonosítása volt. A legfontosabb szempont az volt, hogy a kártyát ne lehessen hamisítani: a támadó ne legyen képes ugyanolyan kártyát csinálni.

Ahogy a kártyákban egyre több intelligencia halmozódott fel, egyre több érzékeny adat is megjelent bennük. A „chipkártyás” alkalmazást fejlesztő programtervező új feladattal szembesült: az intelligens kártya képes lehet a terminált is azonosítani. A tervezőnek a rendszert meg kell vizsgálnia a kártya szemszögéből is, s biztosítania azt, hogy a támadó ne adhassa ki magát terminálnak, s ne tudjon a kártyából információkat kicsalni. Az új

kihívások egyik oka a felmerülő új lehetőségek tárháza, a másik pedig az, hogy egyre több szereplője van egy chipkártyás tranzakciónak (12. ábra), és ez új biztonsági problémákat vet fel [40].

Attól függően, hogy az alkalmazás mi célt szolgál, a terminálnak és a kártyának más és más biztonsági kritériumoknak kell eleget tennie.

- Beléptető rendszer esetében továbbra is a fő cél a kártya felismerése, és a csaló kártyák kiszűrése. Itt jelenik meg az a szempont, hogy a kártyákból a "kulcs" információt minél nehezebb legyen a támadónak kinyernie.
- Adatokat és szolgáltatásokat tartalmazó kártya esetén roppant lényeges szempont az is, hogy a kártyát esetleg ellopó támadó ne férjen hozzá a kártya bizalmas tartalmához. Nem új biztonsági probléma merült fel, csupán arról van szó, hogy immár képesek vagyunk védekezni egy új - memóriakártyák esetén védhetetlen - veszély ellen is.
- Mobiltelefonos alkalmazások esetében egyéb feltételezések is tehetők. A telefont az átlagos felhasználó ritkán választja el a kártyától, és a legritkább esetekben helyezi bele ismeretlen telefonokba. Itt felesleges lenne folyamatosan ellenőrizni a terminált, sokkal egyszerűbb struktúrájú védelem is elegendő.

2.2.4. Az „olvasó"

A kártya egy úgynevezett kártyaolvasón (7. ábra) keresztül kapcsolódik a terminálhoz. Ezen roppant megtévesztő elnevezésnek történelmi okai vannak, hiszen a mágneskártyák, illetve memóriakártyák esetében a terminálhoz illesztő egység valóban olvasta a kártyát. Intelligens chipkártya esetében erről szó sincs.

A chipkártyaolvasó ugyanis a kártyákat sem olvasni, sem írni nem tudja. A terminál az olvasón keresztül mindössze „kérést" küldhet a kártyának, amely azt „megfontolja", s vagy elfogadja/megválaszolja, vagy pedig visszautasítja. Az írás-olvasás modell pusztán memóriakártyák esetében igaz, de generikus kártyák esetében is alkalmazható. Igaz, ez esetben a kártya ezen műveleteket meg is tagadhatja, és az íráson és olvasáson kívül számos más lehetőség is szóbajöhet (inkrementálás, következő rekord kiválasztása, stb.).

Programozható kártyák esetében viszont gyakran nincs is szó írásról vagy olvasásról. Itt műveleteket kérhetünk a kártyától, és ezen műveletekhez csatolhatunk paramétereket. Kér-

7. ábra. Kártyaolvasó - ez kapcsolja a össze a PC-t a smart carddal.

hetünk a kártyától olyat, hogy lásson el digitális aláírással egy bitsorozatot, kódoljon valamit, fizessen ki egy számlát, vagy emeljen négyzetre egy számot. Ilyenkor az írás/olvasás jelentősége rendkívül csekély, és általában nem is szokás adatokhoz közvetlen hozzáférést engedélyezni. A kártyákban felhalmozódó intelligencia azon folyamat eredménye, melyben az adatok védelme és az adatokról való döntés egyre távolabb kerül a kártyát kezelő terminálától (6. ábra), és egyre közelebb magukhoz az adatokhoz, vagyis bekerül kártyába. Amint az adatokat csakis a kártya által felkínált műveleteken keresztül érhetjük el, egy, az objektum-orientált világból jól ismert fogalom, az encapsulation (egységbe zárás) közelébe jutunk (3.3.2. fejezet).

2.2.5. Kommunikáció a kártyával

A terminál és a kártya közti kommunikációt az ISO7816-4 szabvány [21] írja le. A kártyával való kommunikáció alapegysége az APDU (Application Protocol Data Unit). A kommunikáció tipikusan félduplex (bár a szabvány lehetőséget biztosít duplex kommunikációra is): a terminál egy APDU-val elindít egy műveletet a kártyán, amely a művelet végeztével válaszol(hat). Az ISO 7816 szabvány leegyszerűsítve a 8. ábrán látható parancs-APDU struktúrát fekteti le. [34]

CLA	INS	P1	P2	LC	Data field	LE

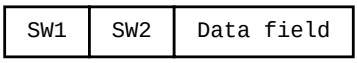
8. ábra. A kártyának küldött APDU szerkezete

Az egyes utasítások utasításosztályokba (CLA) sorolhatók (pl: szabványos ISO utasítások, GSM utasítások stb), az utasítás (INS) kódja pedig a csoporton belül választja ki az utasítást, tehát kódjaik együtt azonosítják a végrehajtandó műveletet. P1 és P2 opcionális

paraméter, ezeket követheti az adatok hosszát jelző byte, maga az adatmező, majd a várt válasz hossza.

A 8. ábrán az általános parancs APDU látható. Ebből bizonyos tagok elhagyhatók. Előfordulhat például, hogy az APDU nem tartalmaz adatmezőt, amely ilyenkor az LC paraméterrel együtt elhagyható. Szintén elhagyható az LE mező, amennyiben a terminál nem vár választ az APDU-ra.

A kártya válasza (9. ábra) adatokat s két státusz szót tartalmaz. Amennyiben a parancs APDU nem tartalmazott LE mezőt, úgy a válasz APDU-nak sem lehet adatmezeje.



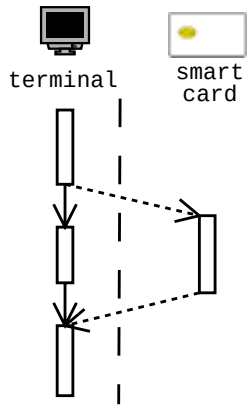
9. ábra. A válasz-APDU szerkezete

Megfigyelhető, hogy sem az APDU, sem pedig a válasz APDU esetén nem könnyű előre megmondani a keret hosszát. Például, amikor a terminál fogadja a kártya válaszát, csupán abból tudja előre megmondani, milyen hosszú lesz az, hogy hány byte választ kért. Amennyiben a kártya nem pont annyi byte-ban válaszolt, mint amennyit a terminál várt, roppant kellemetlen helyzetek adódhatnak. Ezen protokoll a magas szintű programozási nyelvekkel nehezen működik együtt, s beágyazása komoly munkát igényel.

Mivel a kártya intelligenciával rendelkezhet, elvileg tekinthetnénk a folyamatot két független számítógép kommunikációjának is. Csakhogy a szabvány szerint minden esetben a terminál a kezdeményező, a kártya feladata pedig a terminál kéréseinek megválaszolása. Így a folyamat sokkal inkább függvényhívás-szerű (10. ábra) – a terminálon futó program bizonyos részei a kártyán helyezkednek el. Mint ahogy az az ábrából kitűnik (10. ábra), a terminál és a kártya párhuzamosan működhetnek. Ez azt jelenti, hogy a terminál folytathatja a működését a kártya válaszára várva. Csakhogy ez a valóságban elég nehézkes, ugyanis a terminál azért fordult a kártyához, mert valamilyen speciális információra van szüksége, méghozzá olyanra, amilyennel ő nem rendelkezik. Így általában szükség lenne ezen információra ahhoz, hogy a terminál folytathassa a működést.

2.2.6. Tranzakciókezelés

További sajátságos probléma, hogy a felhasználó bármikor megszakíthatja a folyamatban lévő műveletet a kártyának az olvasóból történő kihúzásával, s így annak adatai inkonzisztens állapotban maradhatnak. Ez ellen olyan terminálokkal szokás védekezni, melyek a



10. ábra. A terminál a kártyához fordul

kártyát elnyelik, és csak a művelet végén adják vissza. Azonban ez a felhasználókból gyakran bizalmatlanságot vált ki. Gondos programozással megvalósítható tranzakció kezelés a kártyán (de a hardware támogatás jelentősen megkönnyíti a munkát). Ez az adatbáziskezelők világából kölcsönzött kifejezés azt jelenti, hogy a rendszer a kritikus műveletekről garantálja, hogy vagy teljesen befejeződnek, vagy pedig (ha valamiért ez nem lehetséges) a rendszer visszajut a tranzakció előtti állapotába.

2.3. Chipkártya alapú rendszerek

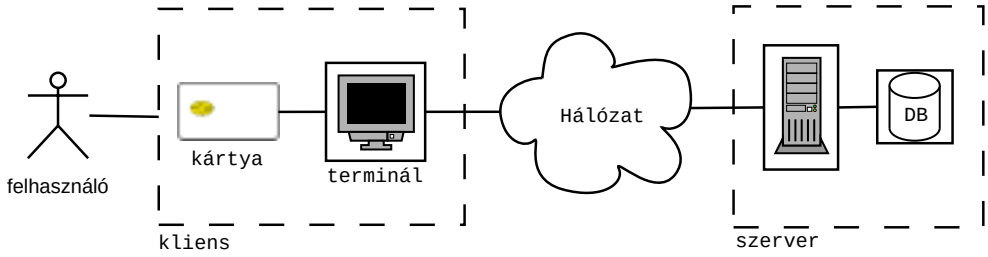
Mint azt láthattuk, a chipkártya önállóan életképtelen és használhatatlan. Értelme csakis egy nagyobb rendszer egy komponenseként lehet. E fejezetben a chipkártya szerepét – erősségeit és gyenge pontjait – fogom elemezni annak tükrében, hogy a kártya egy nagy egész egy kis részét alkotja. Mindezt először műszaki szempontból teszem, majd gazdasági-jogi környezetében vizsgálom a kártyát.

2.3.1. Kártya szerepe a rendszerben

A 2.2.1. fejezetben láthattuk (4. ábra), hogy a chipkártya akár egy igen bonyolult rendszer komponensét is alkothatja. Az következőkben ezt leegyszerűsítjük, absztrakttá tesszük (11. ábra).

Tételezzünk fel egy tipikus alkalmazást! A felhasználó valamilyen távoli szolgáltatást vesz igénybe. Ehhez két komponensre mindenképpen szükség van: a felhasználóra és a szolgáltatásra. A szolgáltatáshoz általában szükség van egy adatbázisra, valamint egy számítógépre,

amelynek segítségével a felhasználó az adatbázist használhatja.



11. ábra. Chipkártya - a hálózat egy eleme

Mivel távoli szolgáltatásról van szó, a felhasználó nem ülhet le közvetlenül a szerver számítógép mellé – hálózaton kapcsolódik hozzá. Ahhoz, hogy a hálózaton kommunikálni tudjon, szüksége van egy terminálra is. Itt merül fel az kérdés, hogyan ellenőrzi a távoli szerver, hogy tényleg a jogosult felhasználó ül-e a terminál előtt. Nyilvánvalóan azonosítania kell valamilyen módon. Akkor végezheti jól a dolgát, ha az azonosítás három fajtájából (B) legalább kettőt függetlenül alkalmaz. A chipkártya használata tulajdon alapú azonosítási módszer.

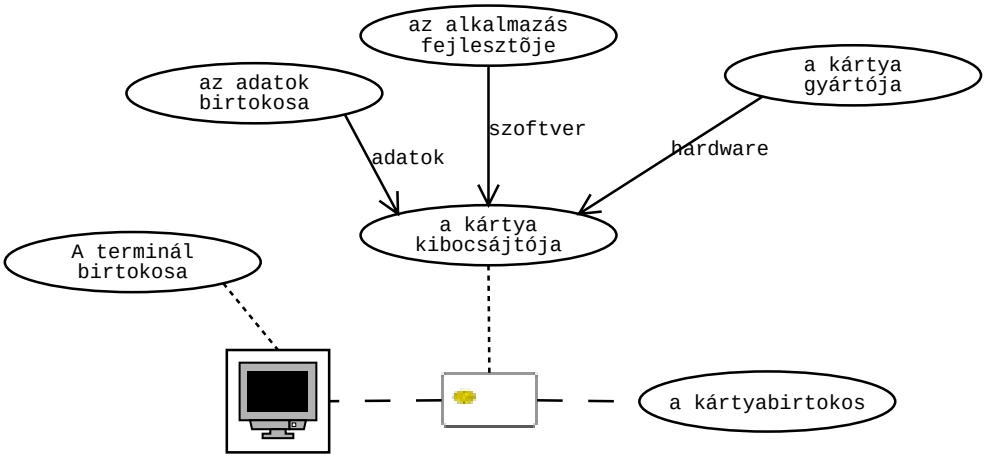
A felhasználó be kell, hogy helyezze a chipkártyáját a terminál kártyaolvasójába, és a terminál a chipkártya és a felhasználó jelszava segítségével azonosíthatja magát a távoli rendszer felé. Így a felhasználó hozzáférhet a távoli szolgáltatáshoz.

A chipkártyákat a ma létező rendszerekben elsősorban felhasználó-azonosításra vagy bizalmas adatok tárolására használják:

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

2.3.2. Egy kártyás rendszerben érintett különböző felek

Egy chipkártya alapú rendszer tipikus szereplői a kártya gyártója (card manufacturer), kibocsátója (card issuer) és birtokosa (card holder). Ezek közül az első kettő egybeeshet. További szereplő lehet a terminálok tulajdonosa (terminal owner); ez például a telefonkártya szolgáltatás esetén rendszerint a telefontársaság (amely egyben a kártya kibocsátója is), de például bankkártyák esetén lehetőség van más bank termináljainak, vagy elektronikus kártyaelfogadóhelyek (Electronic Point of Sale) használatára. Tovább bonyolíthatja a helyzetet, hogy a kártya birtokosa nem feltétlenül egyezik meg a kártyán lévő adatok tulajdonosával (data owner); jó példa erre a hitelkártya, ahol a kártyán lévő adat tulajdonosa a bank, hiszen csak ő tudja ezen adatokat megváltoztatni. Programozható kártyák esetén nem feltétlenül egyezik meg a kártya gyártója a kártyán futó programok készítőivel. Sajnos a bemutatott felek érdekei gyakran különböznek, és számos lehetőségük nyílik egymás megtámadására. [40]



12. ábra. Egy chipkártyás rendszer szereplői

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

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

2.4. Ma létező programozható chipkártya szabványok

2.4.1. Java Card

A Sun Microsystems a Java nyelvet azért alkotta meg, hogy legyen egy platformfüggetlen nyelv, amellyel a fejlesztett alkalmazások a legkülönbözőbb architektúrájú gépeken futnak. A specifikáció nyílt, letölthető, hozzáférhető [45], bárki írhat Java Card programot. Sőt, letölthető hozzá emulátor is, amellyel még PC-n ki lehet próbálni az alkalmazást. Számos nagy kártyagyártó rendelkezik Java Card implementációval. (D függelék)

A Java appletek - mivel Internetes terjesztésre találták ki őket - kellően kicsik. Mivel egy intelligens smartcard tekinthető számítógépnek is (2.1.3), természetesen vetődik fel a kérdés, hogy miért ne lehetne ez a platform is Java kompatibilis. A válasz is természetes: Egyrészt, mert a java byte-kódot generál, amit egy, az adott gépen futó virtuális gép (java VM), értelmez, s ez eredendően lassú. A smartcardok processzora így is jelentősen lassabb, mint a PC-ké, és ez mindig is így lesz [5]. Másrészt a Java nyelv túl komplex ahhoz, hogy programjait smartcardokon futtassuk. Maga a java virtuális gép is rengeteg helyet foglal el.

A Java Card API is az objektum orientált szemléletet követi. Minden egyes applet, amit a kártyára letöltünk, egy-egy objektum, amely önálló életet él. Rendelkezik attribútumokkal és metódusokkal. Akárcsak a webes appletek esetében, vannak kitüntetett metódusai, amelyek bizonyos időpontokban meghívódnak (3.3.1). Létezik egy kitüntetett metódus (*process*), amely argumentumként egy APDU-t (2.2.5) kap.

A Java Card appletek szervesen illeszkednek a Java elképzelésekhez és az objektum orientált világhoz, és rendelkeznek a magasszintű Java nyelv erejével és hordozhatóságával. Természetesen eredeti formájában a Java nem alkalmas smart cardra. A Java Cardokon

alkalmazott Java az igazinak egy leegyszerűsített változata (3.3.3). A programozó bizonyos speciális csomagokat használhat, melyek támogatják a kártyákba gyakran beépített kripto-koprocesszorok használatát. A másik változtatás pedig, hogy a kártyákon nem Java byte-kód fut, hanem gyártóspecifikus kód, amit egy kártyaspecifikus konvertáló programmal állíthatunk elő.

A később bemutatott alkalmazás Java Card nyelven készült, így a Java Cardnak e diplomatervben lényegesen nagyobb szerep jutott, mint az alábbi programozási környezeteknek. A teljes 3. fejezetet neki szenteltem, ott jóval bővebb leírást adok róla.

2.4.2. MULTOS

A MULTOS lényege az, hogy megoldást ad több teljesen különálló alkalmazás futtatására és elkülönítésére egyetlen kártyán. Itt nem a nyelvet rögzítették le, hanem csupán az operációs rendszert. A MULTOS egy olyan smartcardos operációs rendszer, amely definiálja egyrészt az operációs rendszernek a kártyán futó applikációk felé tanusított viselkedését, másrészt az applikációknak a kártya operációs rendszere felé tanusított viselkedését. Bárki fejleszthet MULTOS alá C, assembly vagy akár Java nyelven - persze egy bizonyos kártyán futó MULTOS alá. A MULTOS pedig kártyagyártóknak eladja a MULTOS specifikációját, hogy azok megírhassák saját kártyájukra a MULTOS-t. [31]

2.4.3. Smart Card for Windows

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

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

adtam [7]. A WinCard-ról a legfrissebb információk a `www.microsoft.com/smartcard` oldalon találhatóak.

2.5. A programozható kártyák előnyei

Új technikai lehetőségek

A programozható kártyák számos új technológia alkalmazását lehetővé teszik. Tekintsük át őket!

- Kölcsönös autentikáció (30. ábra)
- Tetszőleges biztonsági mechanizmusok implementálhatóak
- A kártya szolgáltatási köre piacra bocsájtás után is bővíthető
- Prototípus alkalmazások készítése ([3])
- Felhasználó személyiségi jogainak fokozottabb védelme ([7, 4.3. fejezet])

Ha a chipkártya szolgáltatási körét zártnak tekintjük, a programozható kártya nem tud semmivel sem többet a hagyományos kártyáknál. Ha elképzelünk egy kártyás alkalmazást, tudunk olyan hardware-t tervezni, amely pont az alkalmazás funkcióit valósítja meg. Sőt, a hardware-es megvalósítás gyorsabb is lesz.

A programozható kártya csupán annyival tud többet a hagyományos kártyáknál, hogy a kártya feladata a élete során bármikor megváltoztatható, bővíthető. Előnyeinek nagy része nem is annyira technikai (hiszen bármilyen szoftveres funkciót megvalósíthatunk hardware-esen is), inkább gazdasági.

Gazdasági előnyök

- Lehetségessé válik az alkalmazásfejlesztés pusztán egy PC és egy kártyaolvasó segítségével. Mivel a piacra kerüléshez nem szükséges IC-t tervezni vagy gyártani, kb. 200 USD értékű beruházással bármilyen kártyás applikáció kártya oldali szoftvere elkészíthető. Ez növelheti a versenyt, és kisebb fejlesztők új ötleteit is kivitelezhetővé teszi.

- Mivel IC gyártás nem szükséges a termék elkészítéséhez, lerövidül egy kártya piacra-kerülési ideje.
- Nem szükséges minden célra célhardvert gyártani, így tömegtermelés lehetséges. Ez leszoríthatja az árakat.

3. A Java Card technológia

3.1. Miért pont Java?

A Sun Microsystems által kifejlesztett Java Card specifikáció fő célja az, hogy csökkentse a programozható kártyákba való befektetés kockázatát, és ezzel elősegítse a technológia fejlődését. A Java nyelv kártyákon való megjelenésének egyik fő célja tehát a platformfüggetlenség, de számos biztonsági szempont érvényesítését is lehetővé teszi. [43]

3.2. Hordozhatóság, platformfüggetlenség

3.2.1. A kártya belső felépítése

A kártya hardware-én fut annak operációs rendszere. Az operációs rendszer bizonyos szolgáltatásai kívülről is elérhetőek (pl. titkosítás, filekezelés), a funkciók többi részét a cardletek használják ki. (13. ábra) Az operációs rendszeren futhatnak natív applikációk is, de ezek közül Java Cardok esetében csupán egynek van jelentősége: a Java virtuális gépnek (JVM vagy JCVM). A virtuális gépen számos – méretben és funkcióban különböző – cardlet[3] futhat. Elképzelhető, hogy a cardletek csupán statikusan egymás mellett élnek, de akár dinamikusan is együttműködhetnek.

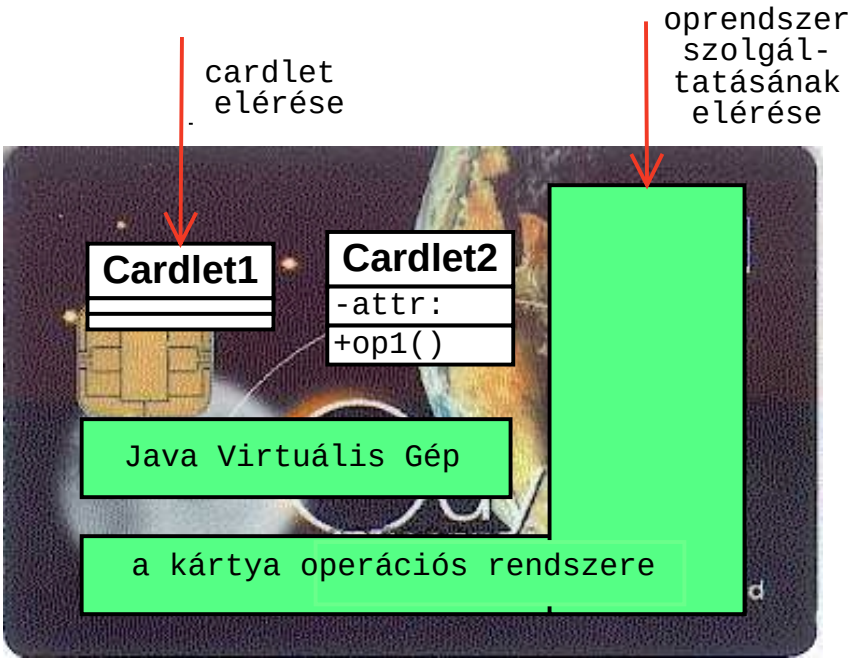
Minden applet-cardlet rendelkezik egy ún. AID (applet identifier) kóddal, amelynek segítségével hivatkozni lehet rá. A terminál először kiválasztja a cardletet, majd neki küldhet APDU-kat (2.2.5. fejezet).

3.2.2. A specifikáció

A Java Card specifikáció három részből áll: [13, 3.1. fejezet]

Java Card 2.1 Virtual Machine (JCVM) Specification, amely a Java programozási nyelv, valamint a Java virtuális gép elemeinek egy részhalmazát definiálja a chipkártyákon való haszálathoz.

[3]Sajnos a szakirodalomban a Java kártyán futó alkalmazás elnevezése következetlen. Nevezik alkalmazásnak, (holott ez nemcsak Java nyelven jelent mást, de pl. [37] az alkalmazást Java Card környezetben is a cardlettől való elválasztásra használja) de hívják appletnek és cardletnek is. Az utóbbi két elnevezés nem áll távol egymástól, így ezen dolgozatban ezeket használom. Elsősorban akkor nevezem őket cardletnek, ha a kártyán való futásukat akarom kihangsúlyozni.



13. ábra. Egy Java Card belseje

Java Card 2.1 Runtime Environment (JCRE), amely a Java kártyák futás közbeni viselkedését adja meg, vagyis a kártya operációs rendszerének a cardletek felé tanúsított viselkedését definiálja.

Java Card 2.1 Application Programming Interface (API) Specification, amely a chipkártyás környezetben elérhető szabványos Java osztályokat és csomagokat definiálja (3.3.4. fejezet).

3.2.3. A szoftverhordozhatóság gazdasági vonatkozásai

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

A Java Card platformfüggetlenséget hoz a chipkártyák világába, és ezzel védi a fejlesztőket a kártyagyártóktól. Így a fejlesztők kiszolgáltatottsága csökken, hiszen sokkal könnyebben tudnak hardware-t váltani, amennyiben szükséges. Mivel a technológia kortántsem

kiforrott, a kártyaváltásnak elég sok indoka lehet:

- A hardware-ről kiderül valamilyen hiba
- Nagyobb memóriájú/teljesítményű hardware szükséges
- A kártyagyártó megszünteti a hardware forgalmazását
- A kártyagyártó az árral próbálja zsarolni a fejlesztőket

A Java Card olyan nyilvános specifikáció, amelynek több implementációja létezik, működését sokan tanulmányozzák [29], [35]. Így a Java Cardra fejlesztők sokkal kevésbé válnak kiszolgáltatottá a specifikáció tervezőivel szemben.

További előnyök:

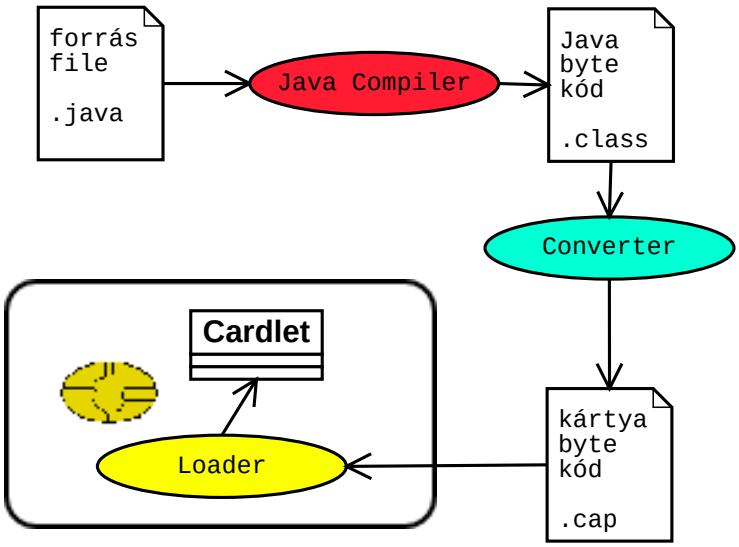
- A fejlesztés/tesztelés jelentős része PC-n is elvégezhető
- Már meglévő, nem kártyára fejlesztett kódok is átültethetőek Java Cardra
- A programozóknak nem kell új nyelvet megtanulni
- A cardletek könnyen telepíthetőek a kártyákra, így a szoftver piacra kerülésének ideje jelentősen lecsökken

3.2.4. Java Card fejlesztés

Hogyan történik egy Java Card fejlesztés? (14. ábra)

1. Elkészítjük a forráskódot. Egy vagy több osztályt hozunk létre, ahol a fő osztály a *javacard.framework.Applet* leszármazottja.
2. A forráskódot lefordítjuk byte kóddá. Egy Java Card program byte kódja ingyenes eszközökkel (pl. JDK) elkészíthető.
3. Az appletet konvertálni kell egy - a kártyagyártótól származó - konvertáló programmal. Ez kiszűri a nem Java Card konform elemeket, és az adott kártyatípus kódjára[4] konvertálja a programot.

[4]A Java Card platformon ugyanúgy interpreter fut, mint sima Java esetében. Ugyanakkor ez a kód kicsit más . Kisebb lesz, a felesleges elemek elmaradnak belőle, továbbá a gyártó egyéb optimalizálásokat is végezhet a hatékonyság további növelése érdekében. Például a Schlumberger Cyberflex kártyán a Java Card Virtual Machine utasításai megmaradtak ugyan, de az optimalizálás során átszámozásra kerültek. [37, D függelék]



14. ábra. Egy Java Card fejlesztés

4. Ez egy az egyben feltölthető a Java Cardra, majd a megfelelő APDU segítségével meg kell hívni az install metódusát. Ezután az applet működőképes – kész APDU-k fogadására.

3.2.5. Programfuttatás

1. A felhasználó behelyezi a kártyáját az olvasóba. A terminál bekapcsolja a kártyát, és a kártyán egy *reset* művelet hajtódik végre. A kártya kiadja magából az ATR (answer to reset) üzenetet. [21]

2. A terminál küld egy Select Applet APDU-t a kártyának, ezzel megmondja, melyik cardletet kívánja használni. [37]

3. A Select Applet APDU hatására meghívódik a cardlet *select* metódusa

4. Ezek után minden beérkező APDU-t a cardlet kap meg. (Kivéve azon szabványos APDU-kat, amelyeket a kártya kezel le. Például egy másik *select* APDU-t.)

5. Másik cardlet kiválasztása esetén meghívódik a cardlet *deselect* metódusa, de ha a felhasználó eltávolítja a kártyát, erre a kártya operációs rendszerének aligha van esélye. Így erre a fejlesztés során nem célszerű építeni.

3.2.6. A mai piac főbb Java kártyái

E fejezetben pusztán felsorolom a mai piac néhány fontosabb Java Card kompatibilis kártyáját. További információ a D függelékben található.

- Schlumberger – Cyberflex (31)
- Gemplus – GemXpresso, Gemplus GemXpresso SIM
- Bull – Odyssey
- Bull – SIM Rock
- Oberthur – GenerIC
- Dallas Semiconductor – iButton

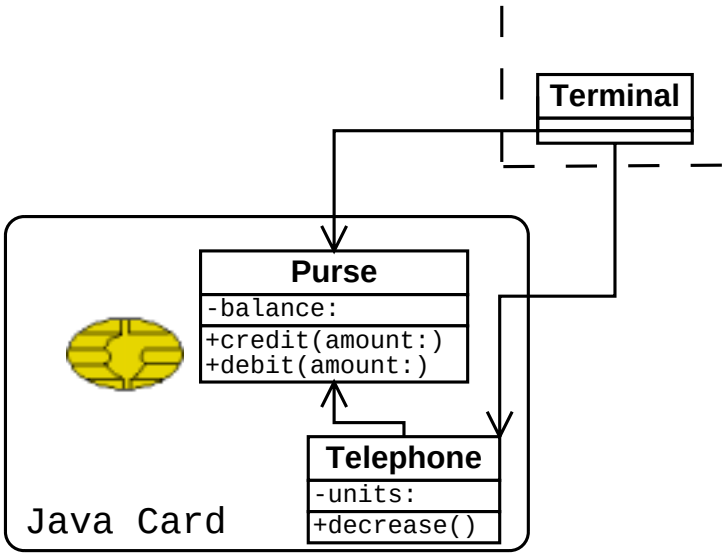
3.3. Java Card programozás

3.3.1. Objektum-orientált szemlélet

A Java Cardok nyelve a Java nyelvre alapul, használja annak platformfüggetlenségét és objektum-orientált szemléletét. A kártyán objektumok foglalnak helyet, minden cardletben egy a *javacard.framework.Applet* leszármazottja. A terminál objektumai – igaz, nem közvetlenül, de – használhatják a kártya objektumait, és a kártya objektumai is megmeghívhatják egymás szolgáltatásait, metódusait. (15. ábra)

A *javacard.framework.Applet* leszármazott objektum egyes metódusai szolgálnak az alkalmazás belépési pontjaként, pont úgy, mint a webes appletek esetében.

A cardlet működés módja lényegében a következő: miután az applet feltöltésre kerül, a rendszer meghívja annak *install* metódusát. Az *install* általában létrehozza az applet egy példányát, s meghívja a *register* metódust. Futtatáshoz ki kell választani az adott appletet a megfelelő APDU-val, ekkor a rendszer meghívja annak *select* metódusát. A Java Card appletnek nincs sokféle eseménykezelője, hiszen csak egyféle „közönséges” esemény történhet vele: a futtatás. Ilyenkor a process(APDU) metódus hívódik meg. Ez megkapja paraméterként a teljes APDU-t, s feldolgozhatja azt.



15. ábra. A kártyán futó osztályok akár a program szerves részét is alkothatják

Az alábbiakban felsorolom a Java Card appletek néhány fontosabb[5] beépített metódusát, valamint bemutatok egy példa Java Card forráskódot. (16. ábra)

install: Statikus metódus, amely az osztály feltöltése után hívódik meg. Tipikus esetben az install a konstruktort hívja meg a megfelelő paraméterekkel, és példányt hoz létre az osztályból. A Bull Odyessey kártya esetében az install hívást automatikusan a feltöltőprogram végzi, a Schlumberger Cyberflex Access (5.3.1) kártya esetében ezt a fejlesztőnek külön kell megtennie, s így lehetőség van egy osztályhoz több példányt is létrehozni.

konstruktor: Az osztály megpéldányosítását végzi a hagyományos objektum-orientált Java szemléletnek megfelelően.

register: Egy *javacard.framework.Applet* leszármazott objektumot regisztráltat be a cardletek közé. Tipikus esetben a konstruktor hívja meg.

select: A cardlet kiválasztásakor hívódik meg. A fejlesztés során arra használhatjuk, hogy inicializáljunk bizonyos értékeket, mielőtt a külvilágot az applet közelébe engednénk. Például bekapcsolás után nem lesz aktuális applet, ilyenkor az appletet újra ki kell választani a használathoz. A *select* segítségével detektálhatjuk például, ha valamilyen tranzakció megszakadt.

[5]Teljes listát pl. [12] tartalmaz.

process: Ez talán a legfontosabb mind közül, hiszen ez az applet eseménykezelője. Kiválasztás után minden hívást a *process* kaphat, tipikusan a *process* tartalmazza az applet működő mechanizmusát, logikáját. A *process* tartalma tipikusan egy hosszú *switch-case* utasítás. [12]

```
import javacard.framework.*;

public class MyApplet extends javacard.framework.Applet
{

private byte attr;

private MyApplet()
{
   attr = 0;
   register();
}

public static void install( APDU apdu )
{
   new MyApplet();
}

public boolean select()
{ return true; }

public void process( APDU apdu )
{
   /*
     Itt altalaban eldobjuk a nem nekunk szolo APDU-kat,
     es valaszolunk a nekunk szolokra.
     Tipikusan az APDU INS mezeje szerint van itt egy elagazas.
   */
}

}
```

16. ábra. Egy tipikus Java Card program felépítése

A tapasztalat azt mutatja, hogy az objektum orientált szoftverek lassabbak a hagyományos szoftvereknél. A chipkártyák „gyenge” processzorán a sebesség lényeges szempont. Miért objektum-orientáltak mégis a chipkártyák?

Több oka van:

- Az objektum-orientált szoftverek sokkal könnyebben karbantarthatók, könnyebb egyes részeiket lecserélni. A szoftver kevésbé a programozótól, sokkal inkább a programtervezőtől függ.

- Chipkártyák esetén különösen fontos a rajta futó szoftver robusztussága. Nemcsak „debuggolni”, vagy a program futását egyéb módon megfigyelni nem lehet (épp a kártya biztonsági mechanizmusai miatt), de még az adatokat konzisztens állapotba állítani sem biztos, hogy lehetséges. Ha egy kártya „lefagy”, használhatatlanná válik. A robusztusság biztosítására ma chipkártyákon – sajnos – egyetlen lehetőségünk átekinthető terveket készíteni, valamint jó fordítóprogramot használni. Megjegyzem, helyesebb volna szoftverhelyesség-bizonyítási módszereket alkalmazni, de ezek a mai imperatív programozási nyelvek esetén roppant nehézkesek.
- Az objektum-orientáltság manapság divatos dolog.
- A Java Card specifikációt a Sun fejlesztette ki, amely jelentős összeget fektetett a Java nyelv kifejlesztésébe. Üzleti érdeke azt diktálja, hogy a Javát favorizálja, ne pedig pl. a Microsoft objektum-mentesített Visual Basic-ét (2.4.3. fejezet). [4].
- Az objektum orientált nyelvek bizonyos védelmet (encapsulation, information hiding) is biztosítanak az adatok számára, valamint segítségükkel könnyebb megfogni a több együttműködő alkalmazás problémakörét is.

3.3.2. Egységbe zárás – encapsulation

Az encapsulation és az information hiding az objektum-orientált programozás két lényeges gondolata. Az objektumok adatmezőit elrejtjük a külvilág elől, megtiltjuk a közvetlen hozzáférést hozzájuk, elérésük, változtatásuk csakis az objektum metódusain, műveletein keresztül lehetséges. A szoftvertervezés ezen gondolatkört elsősorban a könnyű tervezés és az olcsó karbantartás miatt vezette be.

Chipkártyák esetében mindez biztonságtechnikai funkciókat is kaphat. Az intelligens kártya ugyanis nem más, mint egy olyan eszköz, amely elrejti az adatokat a külvilág elől, és elérésükhöz bizonyos műveleteket kínál fel, amelyeket a külvilág APDU-k segítségével hívhat meg. Látszik, hogy hasonló gondolatról van szó.

Képzeljünk el egy telefonkártyát! [7, 2.3.4.]Tekintsük ezt egy objektumnak. Attribútumának tekinthetjük a rajta lévő egységek számát, melyen két műveletet értelmezhetünk:

1. Egységek számának lekérdezése
2. Egységek számának csökkentése n-nel, amely sikeres vagy sikertelen lehet

Látható, hogy nem definiáltunk olyan műveletet, amellyel a kártyán lévő egységek számát növelhetnénk. Így egy esetleges támadó bármilyen jogosultságokhoz jut is, bármilyen kulcsot vagy jelszót szerez is meg, nem tudja a kártyán lévő egységek számát megnövelni. Ezt azzal értük el, hogy nem engedtünk közvetlen teljes hozzáférést az egységek számához, hanem bizonyos műveletekkel elfedtük azt. A műveletek helyes megválasztása azt biztosította, hogy az – eredetileg konzisztens állapotban lévő – attribútumok értéke csakis érvényes irányba változhat.

3.3.3. A Java egy részhalmaza

A Java nyelv kifejlesztésének egyik célja beágyazott rendszerekben való alkalmazás, a másik pedig a weblapokon megjelenő programocskák (appletek) készítése volt, amelyek intelligens funkciókat vittek az addig javarészt statikus weblapokba. Ehhez képest gyökeresen más kihívást támasztott a Java Cardok esete. Míg egy Web-böngészőt futtató PC gyors processzorral rendelkezik, amely képes lehet appleteket hatékonyan futtatni, egy chipkártya néhány Mhz-en járó 8 bites processzora és néhány kilobyte memóriája merőben más sebességviszonyokat jelentenek. Így a Java Cardok nyelve, bár szintaktikailag Java, valójában sokkal közelebb áll a C-hez. [4]

Egy Java Card applet forráskódját nézegetve feltűnik, hogy a kód alacsony szintű. Kevés az objektum-létrehozás, s rengeteg a bitművelet és shiftelés. Látszik a kódon, hogy a programozónak minden órajel ütemért meg kell küzdeni. A Java nyelven nem ilyen filozófiával szokás programozni. [3]

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

Különben lehetséges volna, hogy az egyik cardlet megtámadja a másikat úgy, hogy felemészti a kártyán lévő összes memóriát, és amikor a másik memóriát kér, nem kap. Ilyen módon lehetne adatait inkonzisztens állapotba vinni. Mivel a gondos Java Card programozó lefoglal minden memóriát az applet installációjakor, ilyen támadások ellen védekezhet.

Szintén nem támogatja a Java Card a thread-eket. A program csakis egyetlen szálon futhat. Mint később láthatjuk, ennek biztonsági okai is vannak.

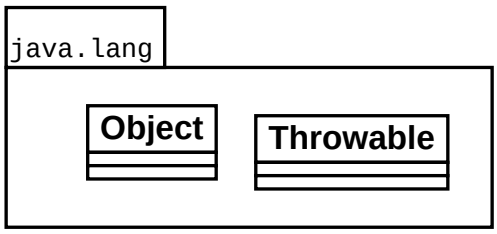
A Java Card specifikáció támogatja a tranzakciókezelést (2.2.6). A programozó megadhat olyan blokkot, amely kívülről atominak minősül. Tranzakció közben a kiírandó adatok egy pufferben tárolódnak, s csak akkor kerülnek ténylegesen kiírásra, ha a tranzakció véget ér. Sajnos minden puffer mérete véges, és ha betelik, akkor a tranzakciókezelés nem működik. Ennek elkerülésére gondos programozás szükséges.

3.3.4. A szabványos Java Card osztályok

Az alábbiakban rövid leírást adok a Java Card osztályok csomagjairól, főbb osztályairól. Célom a jelentősebb osztályok megnevezése, valamint a Java Card filozófia ismertetése. Hely hiányában nem törekszem teljességre, csupán a megfelelő irodalomra hivatkozom. [43], [45], [13, 3.6. fejezet]

A *java.lang* csomag

E csomag funkciója teljesen megegyezik a Java *java.lang* csomagjáéval. Szerepe a megfelelő ősosztályok biztosítása (17. ábra). Definiálja a *java.lang.Object* osztályt, amely minden osztály őse, valamint a *Throwable* osztályt, melyből az Exception-ök származnak. Definiál aritmetikai kivételt, tömbindexelési kivételt, stb. Fő különbség a *java.lang*-tól, hogy jóval kevesebb osztály tartozik bele, így az API jóval karcsúbb lehet.



17. ábra. A *java.lang* csomag két ősosztálya

A *javacard.framework* csomag

Ez a Java Card-ok talán legfontosabb csomagja. Benne foglalnak helyet a Java Card programozás legfontosabb osztályai. (18)

Applet: Ez minden cardlet főosztályának őse, jelentősége hasonlít a *java.applet.Applet*-éhez. Ő adja a cardlet interfészét a külvilág felé. (3.3.1)

A programozó közvetlenül ritkán használja, hanem főappletét leszármaztatja belőle (MyApplet, 16. ábra).

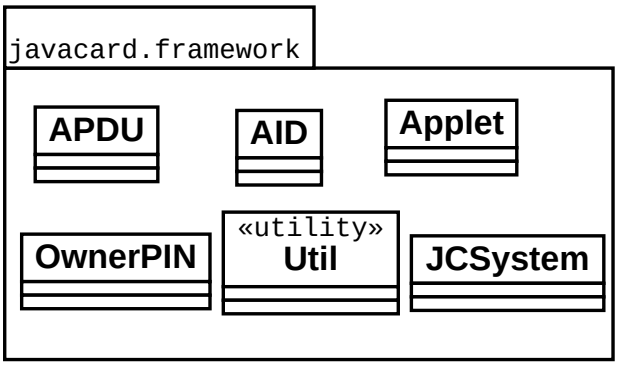
APDU: Szintén roppant jelentős osztály. Az *Applet process* metódusa kapja meg paraméterként (16. ábra), majd általában tömbbé alakítja és úgy hivatkozik az egyes mezőire (8. ábra).

AID: Applet Identifier – A JCRE minden egyes applethez létrehoz egy ilyen objektumot, majd ezt lehet tőle elkérni, ha egy appletre hivatkozni akarunk. [45]

OwnerPIN: A PIN kódok nyilvántartására, kezelésére használt applet. Tartalmazza a kódot, azt, hogy hányszor próbálkozhat a felhasználó, és hogy már hányszor rontotta el. Jelentősége nem nagy, de kellemes segítség lehet a programozó számára.

Util: Bizonyos műveletek elvégzésére használatos utility-jellegű osztály, statikus metódusokkal. Itt helyezkedik el egy csomó konverziós rutin (pl. 1 db short-ot 2 db byte-tá konvertál), memóriamásoló rutin, stb.

JCSystem: A *System*-nek megfelelő osztály. Az applet futtatásának irányítása és erőforrásmenedzsment a feladata. Csak statikus metódusokkal rendelkezik.

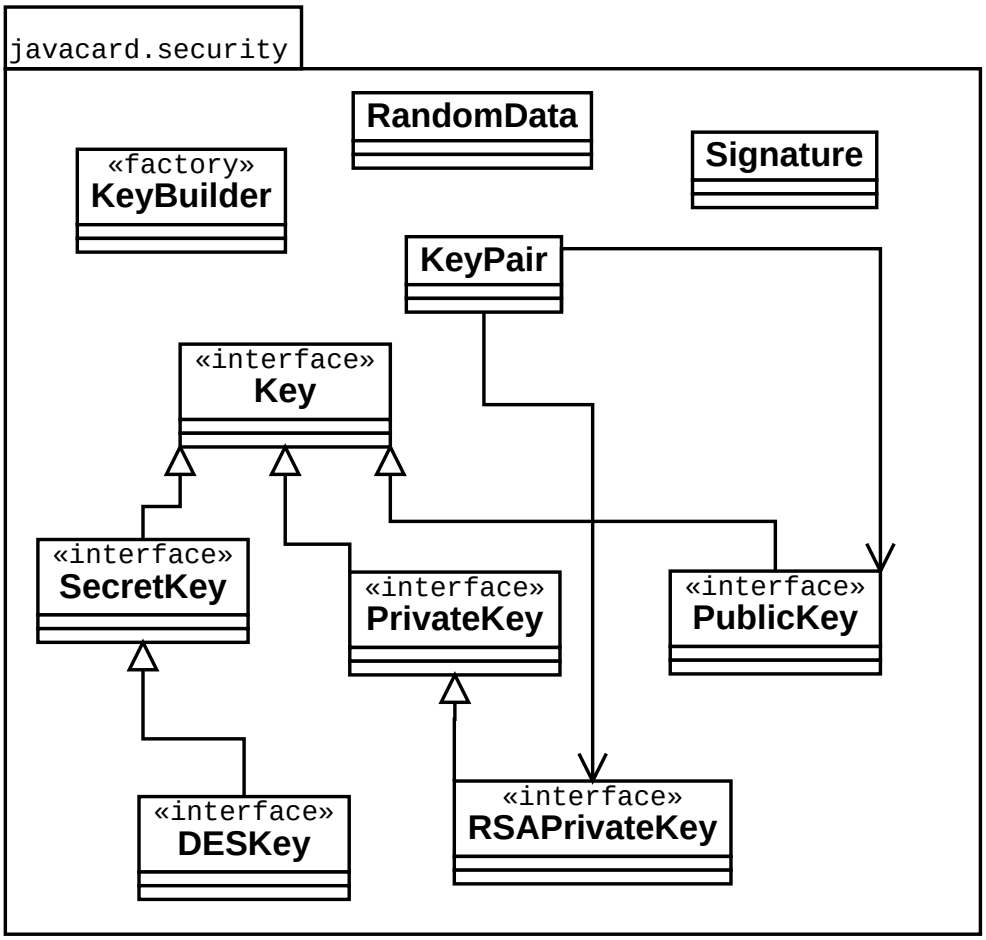


18. ábra. A *javacard.framework* főbb osztályai

A javacard.security csomag

Ez a csomag főként interfészeket és interfészekkel operáló osztályokat tartalmaz. Ez készíti elő a kriptográfia kártyafüggő megvalósítását, ez biztosítja az absztrakt interfészeket, melyeken keresztül a kártyafüggő kriptográfiai műveletek elérhetőek lesznek.

Vannak implementált osztályok (pl: *RandomData*, amely véletlen számsorozatot ad vissza), vannak interfészek (*Key*, melynek leszármazottai a *PrivateKey* és a *PublicKey*, amelyek a konkrét algoritmusok kulcsinterfészeinek ősei), és vannak olyan osztályok is, mint a *KeyPair*, melyek igaz, hogy már implementáltak, de interfészeken hajtanak végre műveleteket. (19 ábra)[6].



19. ábra. Részlet a *javacard.security* csomagból

A *javacardx.crypto* csomag

Ez a csomag egyedül a *javacardx.crypto.Cipher* osztályt tartalmazza a specifikáció szerint, amely az összes kódolóalgoritmus közös ősét képezi. Lényegében interfészdefiníciót és konstansokat tartalmaz. A *Cipher* leszármazottai implementálják majd a 19. ábrán látható interfészeket.

[6]Megjegyzés: az ábrán látható, hogy a *KeyPair* a nyilvános kulcsú algoritmustól függetlenül használható, működhet pl. RSA-val és ECC-vel egyaránt.

3.4. Java kártyák biztonsága

A Java Card számos biztonsági mechanizmussal rendelkezik. Ezeknek egy része a specifikációból ered, egy másik része a Java nyelvből, harmadik része kriptográfiai protokollokból, a negyedik pedig a kártya operációs rendszeréből, hardware-éből.

3.4.1. Fizikai biztonság

A Java Card fizikai biztonsága nem kell, hogy különbözzön a hagyományos smart card biztonságától. Lényeg, hogy a kártya felnyitásával ne lehessen belőle további információkat kinyerni (2.1.3. fejezet).

3.4.2. Operációs rendszer biztonság

A kártya operációs rendszerének biztonságosan kell tudni futtatni több különálló alkalmazást, biztosítani kell, hogy minden filehoz csakis a megfelelő jogosultságokkal lehessen hozzáférni, stb. A legfontosabb feladat a megfelelő futási környezet biztosítása a JCVM számára.

3.4.3. A Java nyelv

Mint azt a későbbiekben láthatjuk (3.3.3. fejezet), chipkártyákon nem a teljes Java került megvalósításra, csupán annak egy részhalmaza. Az így kapott nyelv sok szempontból közelebb áll a C-hez, mint a Java-hoz. Mégis, számos biztonsági elem létezik, amelyeket a Java nyújt ([37, 1. fejezet]):

A referenciakonstrukció Java nyelven nem megengedett – Míg C-ben és assemblyben a memória bármely része megcímezhető, Java nyelven csak azon memóriaterületeket érhetjük el, melyekre pointerünk van. Ezen memóriaterületek is csak azon típus szerint érhetőek el, amely pointerrel rendelkezünk. [35]

A programozó nem készíthet saját memória-allokációs rutinokat [7] – vagyis a kártya memóriájából csakis az operációs rendszer allokációs műveleteivel lehet memóri-

[7] Persze amennyiben a programozó saját memory poolt hoz létre magának (lefoglal egy nagy tömböt), abból nyugodtan adhat objektumainak memóriaszeleteket. Ez teljesen szoftveres megoldás, és csak egy cardleten belül működik. Lásd még: 4.3.5. fejezet.

ára szert tenni. Mivel a kártya nem engedi közel az alkalmazásfejlesztőt a memóriához, nem ad lehetőséget ilyen támadásra.

A Java fordító hibaellenőrzése – Pl: A fordító ellenőrzi minden hivatkozáskor, hogy a hivatkozás megfelelő típusú változóra vagy osztályra mutat-e.

Hozzáférésvédelem minden metódushoz, tagváltozóhoz – A Java nyelv rendelkezik bizonyos hozzáférésvédelmi mechanizmusokkal. Definiálhatjuk, hogy egyes elemek milyen szintű védelemmel rendelkeznek (*public, protected, private,* package).

Nem maradhatnak inicializálatlan változók, és a fordító időnként arra is felhívja a figyelmetm, ha *null* pointerre akar hívatkozni a programozó.

Jól definiált alaptípusok – Definiálták, milyen tagváltozónak mi az alapértelmezés szerinti kezdőértéke, és hogy egyes műveleteknél mikor történhet túlcsordulás. A Java alaptípusok mérete nyilvános és architektúrafüggetlen, és a precedenciák is jóldefiniáltak.

Metódus-szintű védelem – Az egyes metódusok csupán saját lokális változóikhoz férhetnek hozzá, a stack többi részéhez nem.

3.4.4. A Java nyelv csonkítása

Bizonyos elemeket kiemeltek a Java nyelvből, és chipkártyák esetében nem használhatjuk őket. Ezek egy részének teljesítmény-okai, másik részének biztonsági okai vannak.

Nincsenek thread-ek – Az utóbbi kategóriába sorolhatjuk a többszálú programok tiltását is. [18] A Java Card program csakis egyetlen szálon futhat. Ez bizonyos szempontból a program robusztusságát is elősegíti. Hagyományos Java nyelven is létezhetnek platformfüggőségi problémák különböző operációs rendszerek ütemező algoritmusainak eltérése miatt.

Nem lehetséges a dinamikus osztálybetöltés – Míg Java nyelven ilyen előfordulhat, a JCVM ezt nem hagyja. Konvertálás pillanatában minden Java osztálynak léteznie kell.

Nincs garbage collection – Szintén biztonsági óvintézkedés volna a garbage collection eltávolítása is. Azzal, hogy a programozónak le kell foglalnia a program elején minden

memóriát, bizonyos támadások lehetetlenné válnak. (3.3.3. fejezet) Sajnos, a konvertáló program nem kényszeríti ki ezt, így ez a védelem éppen az ellenkezőjére sült el: a programban bárhol megbújhatnak memory leak-ek, felderítésük szinte lehetetlen. A garbage collection hiánya ebben a formában rontja a Java Card-ok biztonságát. [17]

Kivételkezelés – A Java nyelv nagy erőssége pl. a C++ nyelvvel szemben a kivételek korrekt kezelése. A fordítóprogram ellenőrzi, hogy a dobott kivételeket valahol el is kapják. Java Card nyelven ez nem így működik, ugyanis itt a kivételdobás másképp működik. Java nyelven úgy dobunk kivételt, hogy:

```
throw new Exception();
```

Ez azt jelenti, hogy objektumot hozunk létre, és azt dobjuk el. Java Card nyelven nem történhet objektumlétrehozás (hiszen nem tudnánk elpusztítani).

```
ISOException.throwIt(0x5444);
```

Ez egy statikus metódus meghívását jelenti. Nem Java kivételdobás történik, ezen kivételek elkapása, felderítése Java módszerekkel nem lehetséges. A programot áttekinthetetlenné, karbantarthatatlanná teszi, hogy akárhol megbújhat egy kivételdobás, amely felfordíthatja az egész program működését, esetleg a kártyát is inkonzisztens állapotba hozhatja.

Hozzáférési lehetőség a natív kódhoz – A teljesítményért cserébe feláldozzuk a platformfüggőséget, és lehetőséget biztosítunk egy támadó számára, hogy a kártya hardveréhez nyúlva új lehetőségeket, exploitokat használjon ki. [18]

3.4.5. A konvertáló és ellenőrző program

Java bytekódból kártyaspecifikus bytekódot állítunk elő (14. ábra). Ellenőrizhetjük, a programozó tényleg a Java Cardok leszűkített Java nyelvét használta-e, valamint egyéb kártyaspecifikus vizsgálatokat is végezhetünk. Kiszűrhetünk nem megengedett nyelvi elemeket, rossz helyen lévő memóriaallokációkat is.

A konvertálóprogram végül elláthatja cardletünket digitális aláírással is.

3.4.6. Aláírt cardletek

Ha cardletet akarunk feltölteni a kártyára, az ellenőrizheti az aláírás hitelességét. Ilyen módon elérhetjük azt, hogy csakis "megbízható" appletek kerülhetnek fel a kártyára, olyanok, amilyeneket egy konvertáló a specifikációnak megfelelően konvertált és egy ellenőrző megvizsgált és helyesnek talált.

A cardletek aláírása roppant hatékony védelmet jelentene – csakhogy az esetek egy jelentős részében semmit nem ér. Ugyanis:

- A kártya lehet, hogy nem rendelkezik kriptográfiai képességekkel, így nincs esélye az aláírást ellenőrizni.
- Az aláírás szimmetrikus kulccsal történik (pl: a Cyberflex [37] esete), és a kulcs párja ott van a fejlesztőeszközben.
- Minden jól működik, és csakis bevizsgált appletek kerülhetnek fel a kártyára.
  Ez esetben elveszítettük azt az előnyt, hogy a kártya számítógépként működik, és feltölthetünk rá szoftvereket. Számos cég nem fog rendelkezni a szükséges kulcsokkal (vagy anyagi eszközökkel), hogy cardletét aláírhassa (vagy bevizsgáltathassa), így ezeket kizárjuk a versenyből. Holott a programozható kártyák egyik nagy előnye éppen a piaci verseny élénkítése volt (2.5. fejezet).

3.4.7. A virtuális gép védelme

Mint az előző fejezetekben láthattuk, a Java fordító és a konvertáló program igen sok védelmi mechanizmussal rendelkezik. Sajnos, hiába ellenőriz bizonyos hivatkozásokat, címzéseket a fordító és a konvertáló program, előfordulhat, hogy a támadó a konvertáló program outputját támadja meg. Módosítja a kártyafüggő byte-kódot, és azt tölti fel a kártyára.

Védhet ez ellen az appleten lévő digitális aláírás is, de – az előző fejezetben láthattuk – ez bizonyos esetekben nem sokat ér. Ilyenkor egy nem legális byte kód kerül a kártyára, amelyben a támadó a vezérlést megpróbálhatja bizonyos olyan JCVM-assembly kódrészletekre terelni, amelyekkel a kártyát megtámadhatja.

A szakirodalomban léteznek próbálkozások a JCVM specifikáció elvi helyességének igazolására [35], de sajnos ezek csak a specifikáció elég kis részére terjednek ki. Elsősorban a típusellenőrzés problémakörét próbálják körüljárni, valamint a Java és a Java Card hozzáférésvédelmi mechanizmusainak érvényesítését gondolják végig.

3.4.8. Applet firewall

Az applet firewall feladata annak biztosítása, hogy az egyes appletek csakis az engedélyezett objektumokhoz férhessenek hozzá. Célja nem csupán az, hogy az appleteket elválassza egymástól, hanem az is, hogy az engedélyezett együttműködést lehetővé tegye. A programozó megadhatja, mely appletek mely más appleteknek milyen szolgáltatásait vehetik igénybe, a firewall célja pedig az, hogy semmilyen más interakció ne mehessen végbe. [13, 9. fejezet] Az applet firewall a Java Card egyetlen dinamikusan működő védelmi mechanizmusa. [29]

3.4.9. Megoldatlan probléma – együttműködő appletek

Mint láthatjuk, a Java Card igen sokrétű biztonságtechnikai mechanizmusokkal rendelkezik. Mégis előfordulhat az is, hogy mindez nem elegendő. Tételezzük fel, hogy az eddigi fejezetekben felvázolt minden biztonságtechnikai mechanizmus működik, a programozók az appleteket helyesen programozták le, azok a specifikáció szerint működnek, kiskapukat, biztonsági lyukakat nem tartalmaznak. Az appletek közti együttműködések jól definiáltak, és páronként ellenőrzésre kerültek.

Mégis előfordulhat, hogy az egyik fél – a többi appletet, és azok működését ismerve – olyan információkat következtethet ki, amelyek nem tartoznak rá, és ezzel megsértheti a felhasználó személyiségi jogait, illetve az üzleti partner bizalmas adatait. Erre mutat példát [8].

Felhívnám a figyelmet arra, hogy több együttműködő applet esetén komoly elméleti hiányosságokkal rendelkezünk, és nincs megfelelő módszerünk bizonyos [40] által vázolt támadási lehetőségek kivédésére.

Példaimplementációk bemutatása

4. Elliptikus görbékre alapuló kriptográfia Java Card környezetben

A matematikusok már évszázadok óta foglalkoztak az elliptikus görbékkel, pusztán az elmélet esztétikai szépsége miatt. Az 1980-as években fordult először a kriptográfusok figyelme is az elliptikus görbék felé, mivel ezek segítségével rengeteg véges csoportot lehet gyártani, melyekben a diszkrét logaritmus probléma segítségével erős kriptográfia hozható létre [27]. Az azóta eltelt mintegy 15 év tapasztalatai alátámasztani látszanak ezt az elgondolást, de mivel nincs semmilyen elméleti bizonyíték a módszer biztonságára, így sokan úgy tartják, hogy ez a technológia még mindig nem érett meg az ipari és egyéb alkalmazásra. Ezért minél több gyakorlati tapasztalatra van szükség arról, hogy az ECC (Elliptic Curve Cryptography) valójában mennyire biztonságos.

Sietni kell azonban a tapasztalatszerzéssel. Alapvetően két fajta veszéllyel kell szembenéznünk az aszimmetrikus, nyilvános kulcsú kriptográfia terén. Az első veszély abból fakad, hogy lényegében egyetlen nyilvános kulcsú titkosítási eljárás terjedt el széles körben, az RSA. Ennek biztonságára vonatkozóan sincsen elméleti garanciánk, mégis lényegében a teljes PKI (Public Key Infrastructure – nyilvános kulcsú infrastruktúra) erre épül. Ha egy napon valaki találna egy hatékony algoritmust az RSA törésére, ennek beláthatatlan következményei lennének. Nagyon jó lenne tehát, ha lennének más nyilvános kulcsú kriptográfiai módszereink is. Erre pedig jelenleg az ECC a legesélyesebb.

A másik veszély a törésre felhasználható számítási kapacitás ugrásszerű növekedése. Ez egyrészt az olcsó és nagy teljesítményű számítógépek széleskörű elterjedésének köszönhető, másrészt annak, hogy ezek a számítógépek egyre nagyobb mértékben hálózatba vannak kötve. Ennek megfelelően az utóbbi évek leglátványosabb törései nem szuperszámítógépek, hanem párhuzamosan működő munkaállomások segítségével történtek. [10] Ez a jövőben várhatóan egyre inkább így lesz, mivel a nagy teljesítményű munkaállomások az idő jelentős részében nincsenek használatban, így nagy számítási kapacitás marad kihasználatlanul.

Szintén viszonylag új szempont a kriptográfiában, hogy olyan algoritmusokra van szükség, amelyek a chipkártyák adta szűk erőforrás-keretek között is működőképesek.

Az ECC annyiban jelenthet megoldást ezekre a problémákra, hogy jelenlegi ismereteink szerint ugyanazt a biztonsági szintet lényegesen kisebb kulcsokkal tudja elérni, mint az RSA. [18] Ez abból fakad, hogy az ECC alapját képező ECDLP (Elliptic Curve Discrete Logarithm Problem) törésére az általános esetben nem ismeretes exponenciálisnál hatékonyabb algoritmus, míg az RSA törésére igen (bár persze ez sem polinomiális). Ebből nem csak az következik, hogy ugyanazon biztonsági szint eléréséhez az ECC-nek kisebb kulcsokra van szüksége, hanem az is, hogy ahogy a számítási kapacitás növekedésével a kulcsméreteket növelni kell, ez a különbség tovább fog nőni. A kisebb kulcsméret lehetővé teszi a chipkártyás implementációt is. Mint látni fogjuk azonban, nem egyértelmű, hogy a kisebb kulcsok következtében a műveletigény is csökken.

4.1. Elméleti alapok

A Fermat-sejtés bizonyításának kapcsán nemrég az érdeklődés középpontjába kerültek az elliptikus görbék, ugyanis a több évszázados sejtés bizonyításához meglepő módon ezek jelentették a kulcsot. [36] Azonban az elliptikus görbék tanulmányozása már igen nagy múltra tekint vissza. [20], [24], [27]

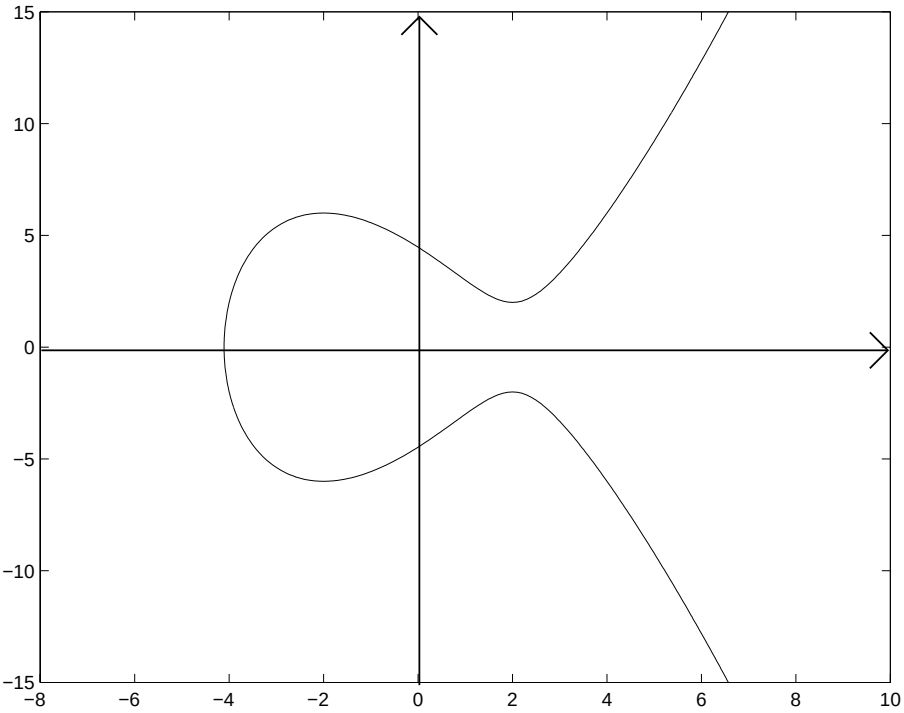
Elliptikus görbét az általános esetben az alábbi egyenlet definiál:

$$y^2 + axy + by = x^3 + cx^2 + dx + e \tag{1}$$

Itt az a_i együtthatók egy adott K testből valók, csakúgy, mint az x és y változók. Ilyenkor az (1) egyenlet megoldásai az E elliptikus görbe K test fölötti pontjait szolgáltatják. Ezek halmazát $E(K)$ jelöli. A legszemléletesebb $K = \mathbb{R}$ (valós) eset, így először ez kerül bemutatásra.

4.1.1. Valós elliptikus görbék

A valós esetben az (1. ábra) egyenlet bal oldalán szereplő xy-os és y-os tag lineáris helyettesítéssel eliminálható. Ez abból is látszik, hogy az xy-os tag a görbe forgatásának, az



20. ábra. Egy elliptikus görbe a valós számok felett

y-os tag pedig y irányú eltolásnak felel meg az $x - y$ síkon. Vagyis egy forgatással és egy eltolással ezektől meg lehet szabadulni. Ezután a görbe egyenlete a következő alakot ölti:

$$y^2 = f(x) \tag{2}$$

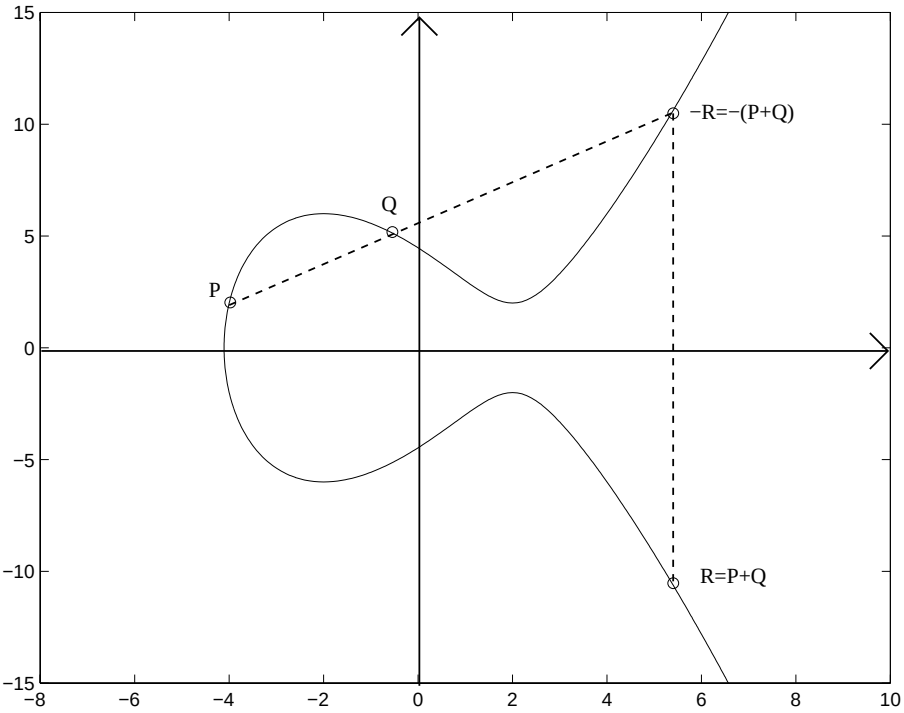
ahol $f(x)$ egy harmadfokú polinom. Sőt, azt is fel lehet tételezni, hogy

$$f(x) = x^3 + ax + b$$

alakú. Azt is meg szokták követelni, hogy ennek a polinomnak ne legyen többszörös gyöke. (Ez tulajdonképpen annak felel meg, hogy ha az egyenletet $F(x, y) = 0$ alakra hozzuk, az F függvény legyen sima, vagyis akárhányszor differenciálható.) Egy ilyen egyenlettel definiált görbe grafikonja látható a mellékelt ábrán (20. ábra).

Az elliptikus görbék kriptográfiai alkalmazhatósága a rajtuk értelmezhető csoport-struk-

túrából fakad. Ehhez először is definiálni kell egy kétváltozós műveletet a görbe pontjai között. Nevezzük ezt összeadásnak, mivel kommutatív csoportok esetén általában az additív írásmód a szokásos! Az összeadást a következőképpen definiálhatjuk: Legyen adva a görbén két pont: P és Q. Tekintsük a rajtuk átmenő egyenest. Ennek az egyenesnek általában pontosan három közös pontja van a görbével; a harmadik közös pontot jelölje $-R$. Tükrözzük $-R$-et az y tengelyre. Az így kapott pont szintén rajta van a görbén, jelölje ezt R. Ekkor definiáljuk az összeget így: $P + Q = R$. Erre alábbb példát is láthatunk (21. ábra).



21. ábra. Egy görbe két pontjának összeadása

Számos probléma van azonban ezzel a definícióval. Például mi van akkor, ha a P és Q pontok egymásnak az x tengelyre vonatkozó tükörképei? Ebben az esetben ugyanis a rajtuk átmenő egyenes nem metszi többször a görbét. E probléma kiküszöbölésére ki kell egészítenünk a görbét egy új ponttal, melyet O-val jelölünk, és úgy gondolhatunk rá, mint a sík y irányú végtelen távoli pontjára. Mint később kiderül, ez a pont lesz a csoport neutrális eleme. [6], [24],[14]

Térjünk vissza az összeadás definíciójához! Most már tudunk válaszolni arra a kérdésre, hogy mi legyen két olyan pont összege, melyek egymásnak az x tengelyre vett tükörképei.

Ugyanis az O pont lesz a két ponton átmenő egyenesnek a görbével vett harmadik közös pontja. Ha még azt feltesszük, hogy az O pontnak az x tengelyre vett tükörképe önmaga, akkor azt kapjuk, hogy a fenti szabály alkalmazásával a két pont összege éppen az O pont. Ezek szerint - tudván, hogy az O lesz a csoport nulleleme - két ilyen pont egymás inverze. Ez indokolja azt is, hogy az összeadási szabály fenti leírásánál miért használtuk az R és $-R$ pontokra ezt a jelölést: e két pont ugyanis tényleg egymás inverze. Az, hogy miért pont így definiáljuk az O pontot, kicsit mesterkéltnek tűnhet. Az O pont definíciójának precíz körüljárását leírja pl. [3], [6], [14]

Külön meg kell még vizsgálni azt az esetet, amikor a két összeadandó pont megegyezik. Ilyenkor összekötő egyenesükön értsük a görbének az adott pontban vett érintőjét. (Ilyen nyilván van, hiszen a görbe sima.) Hasonlóképp az is előfordulhat, hogy a P és Q pontokat összekötő egyenes nem metszi el egy harmadik pontban is a görbét, hanem például a P pontban érinti azt. Ilyenkor a $-R$ pont megegyezik a P-vel.

Két pont összegét lehet tisztán algebrai úton, képletekkel is definiálni. Tegyük fel, hogy a két összeadandó pont koordinátái $P(x_1, y_1)$ illetve $Q(x_2, y_2)$. Az összeg koordinátái ekkor:

$$\begin{aligned} x_3 &= s^2 - x_1 - x_2 \\ y_3 &= s(x_1 - x_3) - y1 \end{aligned} \tag{3}$$

A fenti képletben szereplő s éppen a görbe P és Q pontját összekötő egyenes meredeksége.

$$s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}, \; ha \, P \neq Q \\ \frac{3x_1^2 + a}{2y_1}, \; ha \, P = Q \end{cases}$$

A képletekből jól látszik, hogy ha a két pont egymás inverze, akkor 0-val kellene osztani; ez felel meg annak, hogy a végtelen távoli pontot kapjuk eredményül.

Több szempontból is nagy jelentőségű az a tény, hogy az összeg koordinátáit egy képlet segítségével ki tudtuk fejezni az összeadandók koordinátáiból. Először is, a képleteket a geometriai személet segítségével lehet ugyan levezetni, de innentől fogva már ezek a képletek helyettesítik a geometriai szemléletet, hiszen ezekben már minden benne van, ami a csoport struktúrára jellemző. Ez fontos, hiszen kriptográfiai rendszerünket így nem geometriai, hanem algebrai eszközökkel tudjuk definiálni. Még sokkal lényegesebbé válik

ez a szempont véges testek feletti elliptikus görbék esetében, hiszen ott már nem lehetséges a geometriai szemléletre támaszkodni.

4.1.2. Véges testek feletti elliptikus görbék

Kriptográfiai alkalmazásokban a végtelen ($\mathbb{R}$, $\mathbb{C}$, stb.) testek azért nem megfelelőek, mert számítógépekkel eredendően csak véges pontossággal tudunk számokat tudunk tárolni, és ez például valós számok használata esetén számos problémához vezethet. Ennek elkerülésére számolunk inkább véges test felett.

A véges testek elemszáma mindig prímhatvány, a továbbiakban ezt $q = p^r$ jelöli. Bizonyítható, hogy q elemszámú testből lényegében csak egy van; ennek jele $GF(q)$. Egy test karakterisztikáján azt a legkisebb c pozitív egész számot értjük, amire teljesül, hogy minden testelemet c-szer saját magával összeadva 0-t kapunk. Egy p^r elemszámú test esetén $c = p$. Véges testek multiplikatív csoportja (a $GF(q) \setminus \{0\}$ halmaz) mindig ciklikus, egy generátorelemét g jelöli. [22]

Ezután a kis bevezető után rátérhetünk a véges testek fölötti elliptikus görbék vizsgálatára. Az a módszer, ahogy a valós esetben az (1) egyenletből 3 tagot eliminántunk, véges testek fölött nem mindig alkalmazható. Pontosabban, csak akkor alkalmazható, ha a test karakterisztikája nem 2 vagy 3. A 2-karakterisztikájú eset a legbonyolultabb: ilyenkor választhatunk, hogy az xy-os és x^2-es tagot ejtjük ki, vagy pedig az y-os és az x-es tagot. (Azonban mindezen tagokat egyszerre nem tudjuk kiejteni.) Az előbbi esetet szuperszingulárisnak, az utóbbit nem-szuperszingulárisnak nevezzük. A nem-szuperszinguláris esetben azt is fel lehet tételezni, hogy az xy-os tag együtthatója 1. 3-as karakterisztika esetén ki tudjuk ejteni az xy-os és az y-os tagot, de a jobb oldalt már nem tudjuk tovább egyszerűsíteni. Összefoglalva tehát az egyenletek a következők:

$p = 2$, szuperszinguláris eset: $y^2 + ay = x^3 + bx + c$

$p = 2$, nem-szuperszinguláris eset: $y^2 + xy = x^3 + ax^2 + b$

$p = 3$: $y^2 = x^3 + ax^2 + bx + c$

$p > 3$: $y^2 = x^3 + ax + b$

Mivel az összeadásra vonatkozó képleteket is a (2) egyenletből vezettük le, így $p = 2$ illetve $p = 3$ esetén ezek sem érvényesek. Kanyarodjunk vissza ezért az általánosabb (1) egyenlethez. Ebből is levezethetjük a megfelelő képleteket, melyek azonban korábbi

formuláinknál jóval bonyolultabbak lesznek, ezért ezeket itt nem ismertetjük. Azonban azt vegyük észre, hogy mivel a görbe el van tolva és forgatva, így az (x, y) pont inverze nem az $(x, -y)$ pont lesz, hanem (az (1) egyenlet jelöléseivel) $(x, -ax - b - y)$. [14, 3. fejezet]

A véges testek fölötti elliptikus görbék elméletében fontos szerepet játszik annak meghatározása, hogy a görbének hány pontja van. Jelölje ezt N. Az affin (tehát nem végtelen távoli) pontok tekintetében persze legfeljebb q^2 pont jöhet szóba, de N ennél lényegesen kevesebb. Ugyanis x egy konkrét értéke mellett a görbe egyenlete egy másodfokú egyenlet y-ra. Ezért $N \leq 2q$. Azonban nem minden másodfokú egyenlet oldható meg a véges testek fölött, így N valójában még ennél is kevesebb. Hasse 1934-es tétele értelmében $N \sim q$, pontosabban N és $q + 1$ eltérése $2\sqrt{q}$-val becsülhető fölülről. Ebből következően könnyen lehet egy adott görbén véletlenszerűen pontokat keresni: veszünk egy tetszőleges x-et, és ehhez körülbelül 50% eséllyel tudunk megfelelő y-t találni. Érdekes azonban, hogy általánosságban nem ismeretes determinisztikus polinomidejű algoritmus egy adott görbén pontok megtalálására.

4.2. Elliptikus görbék kriptográfiai alkalmazása

4.2.1. ECDLP

1985-ben V. Miller és N. Koblitz egymástól függetlenül javasolták az elliptikus görbék pontjain értelmezett diszkrét logaritmus probléma (ECDLP) kriptográfiai alkalmazását. [27], [24] Ennek lényege a következő. Adva van egy E elliptikus görbe egy véges test felett, tovább egy $P \in E$ pont. Legyen N a P pont rendje, vagyis a legkisebb olyan pozitív egész szám, amire $NP = O$. (NP jelentése: a P pontot N-szer összeadjuk saját magával.) Legyen ezen kívül adva még egy Q pont. A feladat: találni egy olyan $0 < k < N$ természetes számot, amire $Q = kP$, feltéve, hogy ilyen szám létezik.

Ez valóban megegyezik a diszkrét logaritmus problémával, csupán a jelölés más. Ha ugyanis a csoport-operációt nem összeadásnak, hanem szorzásnak tekintenénk, akkor a feladat egy olyan k szám keresése lenne, melyre $Q = P^k$, amit így is jelölhetnénk: $k = \log_P Q$.

Természetesen sokkal kellemesebb a modulo p maradékosztályokkal számolni, mint egy véges test feletti elliptikus görbe pontjaival. Joggal merül fel tehát a kérdés: miért éppen egy ilyen, viszonylag nehezen kezelhető csoportban kell definiálni a diszkrét logaritmus problémát? A válasz az, hogy természetesen a diszkrét logaritmus problémának tetszőleges csoportban van értelme, azonban az elliptikus görbék segítségével olyan csoportot sikerült

definiálni, amiben a diszkrét logaritmus probléma megoldására nem ismeretes exponenciálisnál alacsonyabb időigényű algoritmus. Pontosabban, az elliptikus görbék egyes eseteire ismeretesek szubexponenciális törő algoritmusok, ezért nem mindegy, kriptográfiai rendszerünkben milyen görbét vagy görbéket használunk. A legnagyobb ilyen jellegű eredmény 1993-ból származik. Menezes, Okamoto és Vanstone adott egy módszert , melynek segítségével a szuperszinguláris görbéken értelmezett ECDLP viszonylag hatékonyan oldható meg. [26]

Azonban az elliptikus görbék túlnyomórészt nem szuperszingulárisak, így továbbra is nyitott kérdés, hogy van-e az ECDLP-re általánosságban használható szub-exponenciális algoritmus. Jelenleg úgy tűnik, hogy nincs, ezért az ECC rendszerek viszonylag kis kulcsmérettel nagy biztonságot tudnak elérni. Ezzel együtt persze, mint azt a későbbiekben látni fogjuk, a bonyolult számítások nagyban rontják a módszer hatékonyságát.

4.2.2. A véges test

A gyakorlati megvalósítás szempontjából nagyon lényeges kérdés, hogy mi az a véges test, amely fölött az elliptikus görbét értelmezzük. Rendszerint vagy egy $GF(p)$ prímrendű testet szoktak használni, vagy pedig egy $GF(2^r)$ 2-karakterisztikájú testet. Az előbbi nagy előnye, hogy egyszerűen lehet benne számolni, hiszen a szokásos módon lehet a műveleteket végezni, csupán arra kell figyelni, hogy az eredmény ismét 0 és $p-1$ között legyen. Az utóbbi viszont jobban illeszkedik a számítástechnikában alkalmazott 2-es számrendszerhez, így számos művelet gyorsabban végezhető el.

$GF(2^r)$ elemeinek reprezentálására több különböző módszer is használatos. Ezek közül a legszélesebb körben a polinom-alapú ábrázolás terjedt el. Ennek lényege, hogy tekintünk egy r-edfokú irreducibilis polinomot $GF(2)$ felett; jelöljük ezt f-fel. Ezután $GF(2^r)$ elemeire úgy tekintünk, mint legfeljebb $r-1$ fokú, $GF(2)$ feletti polinomokra. [19]

Az összeadás a polinomoknál megszokott módon történik, csak éppen arra kell figyelni, hogy a 2-karakterisztika miatt egy tetszőleges számot (vagy egy tetszőleges polinomot) önmagához adva 0-t kapunk. (Emiatt például a kivonás megegyezik az összeadással.) Két testelemet úgy kell összeszorozni, hogy polinomként összeszorozzuk őket, majd a kapott polinomot modulo f vesszük. Vagyis elosztjuk maradékosan f-fel, és a maradék (amely szintén egy legfeljebb $r-1$ fokú polinom) lesz a szorzás eredménye. Bebizonyítható, hogy így tényleg testet kapunk, és mivel a 2^r elemszámú test lényegében egyértelmű, így kész vagyunk $GF(2^r)$ konstrukciójával. A polinomokat gyakran az együtthatóikból képezett r

hosszúságú 0-1 sorozatokkal adják meg. Léteznek kutatási eredmények a szorzás további gyorsítására ún. optimális normál bázis reprezentációval. [30]

Ezennel végére értünk az ECC világába tett elméleti kirándulásnak, a továbbiakban ECC implementációmról lesz szó. A A függelékben olvasható néhány ismert ECC-re épülő protkoll részletes leírása.

4.3. Implementáció

Saját ECC implementációmat kettős céllal hoztam létre. Egyrészt jobban meg kívántam ismerni az ECC technológiát, és különféle méréseket akartam elvégezni, másrészt létre akartam hozni egy chipkártya alapú ECC megvalósítást. Végül egy olyan programot hoztam létre, amely egyaránt fut PC-n és Java Cardon, vagyis ugyanaz a forráskód, valamint ugyanazon Java osztályok futnak mindkét platformon.

Mivel a Java Cardot csakis Java nyelven programozhattam, a fejlesztést Java nyelven végeztem. A Java [44] egy magasszintű nyelv, amely nagyfokú platformfüggetlenséget tesz lehetővé. Ugyanebből következik, hogy egy Java program korántsem optimális egy adott hardverre.

A Java Card nyelv számos erős megszorítást tartalmaz a Java nyelvhez képest (3.3.3), s így programomban is a nyelvnek csupán egy részhalmazát használhattam. Célom nem valamely platformon való nagy sebesség elérése volt, hanem az, hogy alkalmazásom működjön a kártya adta szűkös erőforráskeretek között is.

Igaz, a chipkártya mind sebesség, mind tárkapacitás terén alulmúlja az asztali számítógépeket, mindkét paraméterük jelentősen fejlődött az utóbbi időben. [5] A fő hangsúlyt a biztonság megvalósíthatóságára (polinomiális komplexitás) és az algoritmikus hatékonyságra fektettem, nem pedig a hétköznapi értelemben vett sebességre.

Implementációm továbbá elsősorban prototípus értékű, amely demonstrálni kívánja a Java Card technológia jelen fejlettségi szintjét, számítási erejét, nem pedig egy majd kereskedelmi forgalomba kerülő termék.

4.3.1. A Java Card technológiából adódó korlátok

A kártya hardveréből adódó korlátozások

Megvalósíthatóság terén fő korlátnak a kártya szűkös memóriája bizonyult. Nemcsak a

program forráskódja kellett, hogy a kártyán elférjen, de a kártya szűkös memóriájának tartalmaznia kellett a program által használt adatokat is. Ezek közt helyet kell szorítani mindannak a nem kevés változónak is, amelyet a program futása közben használok. Így az előtt a választási lehetőség előtt álltam, hogy ha bonyolultabb (és ettől gyorsabb) algoritmust írok, nem marad memóriám megfelelő méretű testet választani a kellő biztonság eléréséhez. (Programomban a naiv eljárásokat használtam a műveletek megvalósítására, és appletem ennek ellenére kitette a kártya memóriájának több, mint felét.)

Komoly korlátot jelent egy chipkártyás alkalmazás esetén a sebesség is, de ez a megvalósíthatóság tényét nem befolyásolta. Kriptográfiai alkalmazás esetén gyakorlati jelentősége csupán hardveresen támogatott műveleteknek lehetséges. Amennyiben az ECC a későbbiekben elterjed, várható, hogy a megjelenő ECC-t támogató kripto-koprocesszorok jelentősen megnövelik a számítási sebességet. (Mint az tapasztalható volt a DES és RSA chipek esetében.) [39]

A Java Card specifikációból adódó korlátozások

A fő korlátozást itt is a memória jelentette. A Java specifikációtól eltérően a Java Card ugyanis nem támogatja a garbage collectiont. Míg Javában azon objektumok, amelyek elérhetetlenné válnak, önmaguktól megsemmisülnek, Java Card nyelven ez nem történik meg. Nincs mód továbbá a lefoglalt memória explicit felszabadítására sem (mint ahogy ilyet Java nyelven sem tehetünk). A specifikáció szerint egyáltalán nincsen garancia arra, hogy a memória, amelyet lefoglalunk, valaha is felszabadul. [43]

4.3.2. Algoritmikus megoldások

Úgy döntöttem, a $GF(2^m)$ testben valósítom meg az ECC algoritmust a kártyára. Ennek fő oka az volt, hogy számítógépekkel viszonylag könnyebb bitvektorokat feldolgozni, mint a $GF(p)$ testben számolni. Egyrészt az elvégzendő műveleteket a bináris világban kell elvégezni, másrészt az algoritmus is jóval tömörebb, egyszerűbb lett. Szintén a kis méretű programkód miatt döntöttünk a polinomiális reprezentáció mellett.

Aritmetika

A kód méretének csökkentése végett tömör, spártai egyszerűségű algoritmusokat választottam, akkor is, ha ez nem a leggyorsabb kódot eredményezte. A következő műveleteket valósítottam meg a testelemek között (m az input testelem(ek) hosszát jelenti):

összeadás: a bitenkénti kizáró vagynak felel meg (polinomiális reprezentációban ez a triviális megoldás), $O(m)$ lépésben elvégezhető.

elforgatás (shiftelés): A balra való shiftelés a polinomok világában x-szel való szorzást jelent, a jobbra való forgatás pedig x-szel való maradékos osztást. Az implementációban ez úgy jelenik meg, hogy minden egyes bitnek értékül adjuk a jobbra/balra mellette lévőt, majd (balra shiftelés esetén) modulust képzünk. Költsége $O(m)$.

modulusképzés: Meg akarom kapni a P polinom értékét modulo M. Ez a művelet elvégezhető $GF(2^m)$ felett egyszerű összeadással is, amennyiben a P foka ($deg(P)$) megegyezik M fokával. Ha $deg(M) > deg(P)$, nincsen szükség modulusképzésre, de ha $deg(M) < deg(P)$, a modulusképzés bonyolult művelet lenne. Programomban én úgy intézem, hogy ez utóbbi eset ne fordulhasson elő. Ha $deg(M) = m$, minden polinomot $m + 1$ biten reprezentálunk, ahol a testbe tartozó polinomok esetében a MSB mindig 0. Amennyiben MSB nem 0, szükség van modulusképzésre. Nem engedem meg, hogy egy polinom fokszáma m fölé nőjön, ha elérné m-et, azonnal képzem a modulusát. Ezzel a módszerrel azt értem el, hogy egy P polinomról $O(1)$ lépésben eldönthető, hogy esetében szükség van-e modulusképzésre. Ezek után $O(m)$ lépésben képzem a modulusát (ha szükséges). Ez nem biztos, hogy a leghatékonyabb megoldás, de a memóriával nem bánhattam pazarlóan. A modulusképzés költsége összességében $O(m)$.

szorzás: Az írásbeli szorzás algoritmusának megfelelően valósítottam meg. A szorzás eredményét egy akkumulátorban tárolom, melynek kezdeti értéke 0. Ha P és Q polinomokat szeretném összeszorozni, végigiterálok Q bitjein. Ha az aktuális bit 1, akkor P-t hozzáadom az akkumulátorhoz. Ha a bit 0, nem csinálok semmit. Minden iteráció végén megszorzom P polinomot x-szel. (Azaz balra shiftelem eggyel.) Ha szükség van rá, modulust képzek. Összeadásokat ($O(m)$) végzek el m-szer. Az összköltség $O(m^2)$. (Ez persze még jelentősen gyorsítható.)

maradékos osztás: Az írásbeli osztás műveletét implementáltam. Tegyük fel, hogy P polinomot akarom Q polinommal elosztani, az osztás eredménye kerül majd E-be, a maradék pedig R-be. Első lépésként megszorzom Q-t x^k-nal ($Q' = Q * x^k$), úgy, hogy $deg(P) = deg(Q')$ teljesüljön. Így Q' utolsó k helyiértéke 0 lesz. Ezután a következőket végzem el k-szor: Ha $deg(P') = deg(Q')$, akkor $P' := P + Q$ és $E := E + x^k$, de a feltételtől függetlenül elforgatom Q polinomot jobbra és k-t csökkentem eggyel. Ami-

kor k nulla lesz, akkor P'-ben keletkezett a maradék. Költsége: shifteléseket $O(m)$ és összeadásokat ($O(m)$) végzek el legfeljebb m-szer. Így az összköltsége $O(m^2)$.

osztás, azaz inverzzel való szorzás: Két részből áll. Először inverzet képzek az Euklideszi algoritmus [19] segítségével. Ezután szorzok $O(m^2)$. Az euklideszi algoritmusban $O(m)$ darab maradékos osztást ($O(m^2)$) végzek. Így az összköltség $O(m^3)$.

negálás: minden P bináris polinom negáltja önmaga.

A következő műveleteket definiáltuk a görbe pontjai alkotta csoportban:

P **pont összeadása egy tőle különböző** Q **ponttal:** Legköltségesebb művelete a testelemek közt elvégzett osztás, így költsége $O(m^3)$.

P **pont összeadása saját magával:** Itt is az osztás a legköltségesebb művelet. Költsége: $O(m^3)$.

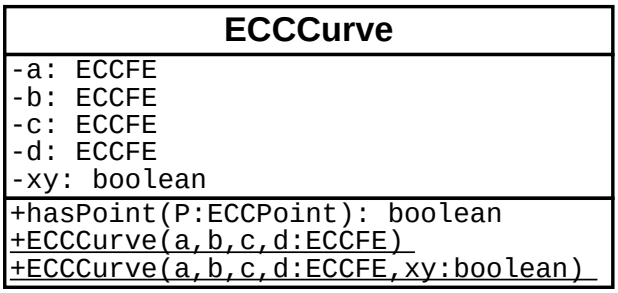
P **pont szorzása** k **egész számmal:** Mivel erre közvetlen képlet nem létezik, összeadásokkal kell megvalósítani az előbbi két művelet segítségével. Mivel a naiv megoldás (k darab összeadás) lépésszáma k hosszával exponenciálisan arányos lépésszámú algoritmust eredményezne, kicsit ravaszabbnak kell lennünk. Az általam választott megoldásban kiszámítom a $P = l(i)$ értékeket, ahol $l(i) = 2^i$, majd k bináris reprezentációját felhasználva összeadásokkal állítom elő $k * P$-t: $O(log(k))$ összeadás. Hogyha egy akkumulátort használok, és k MSB-jétől az LSB-je felé haladunk, és egyes bit esetén hozzáadom P értékét az akkumulátorhoz, valamint továbblépéskor megduplázom az akkumulátort, akkor mindez megoldható konstans méretű tárban is, ami a chipkártya szűkös erőforráskészletét figyelembevéve nem utolsó. A művelet algoritmikus komplexitása $O(m^3 log(k))$. Figyelembe véve a Hasse-tétel (4.1.2. fejezet) azon következményét, miszerint egy $GF(q)$ feletti görbe pontjainak száma jól közelíthető q-val, feltehetjük, hogy $k \leq q$, ugyanis a görbe pontjainak számánál nagyobb k-nak nem lenne értelme. Így $log(k)$-t közelíthetjük m-mel, tehát a kapott költség: $O(m^4)$.

4.3.3. A használt osztályok bemutatása

A programom megtervezéséhez az UML módszertant [32] használtam, működési elvét, szerkezetét is ennek segítségével mutatom be. Az elkövetkezőkben olyan részleteket közlök

UML diagrammomból, melyeken bizonyos – a lényeghez nem tartozó – attribútumokat, objektumokat nem tüntettem fel.

A görbe (*ECCCurve*)



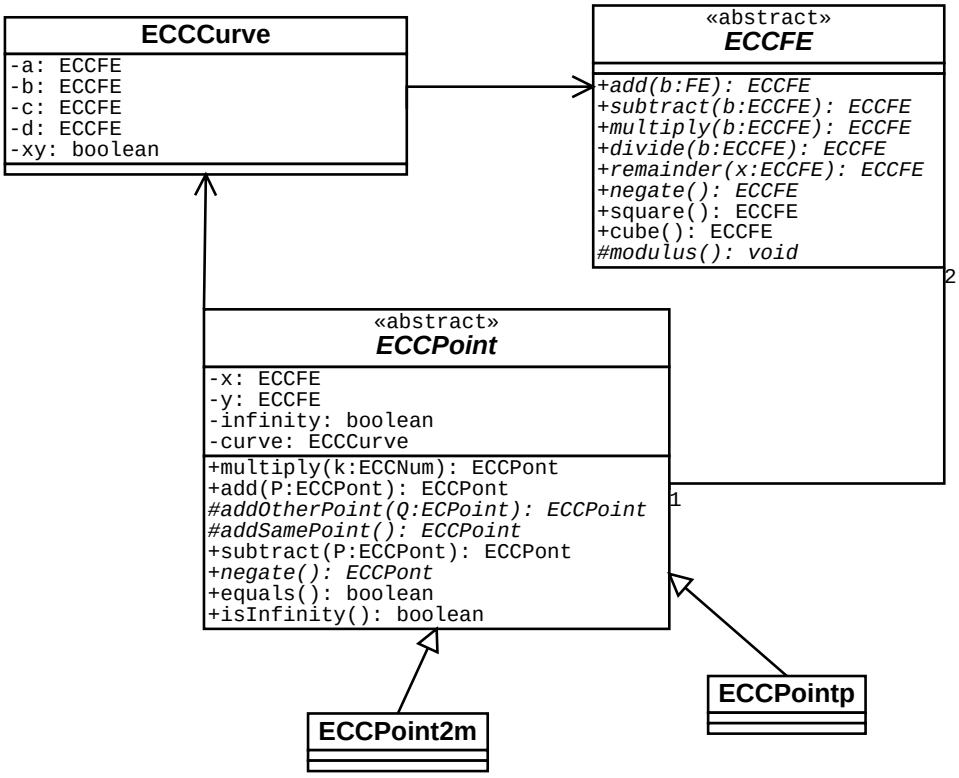
22. ábra. *ECCCurve* – a görbe

Az *ECCCurve* osztály egy Galois test felett értelmezett görbét jellemez. A lehető legáltalánosabb modell elérése érdekében x minden hatványa rendelkezhet együtthatóval, továbbá az egyenlet rendelkezhet xy-os taggal. A görbe egyenlete így: $y^2 + xy = ax^3 + bx^2 + cx + d$ lehet, de belőle az xy tag el is hagyható. Egy görbe objektum nem tudja, pontosan milyen test felett használjuk, ettől függetlenül működik, hiszen ő csak az *ECCFE* (Field Element) általános osztályt látja.

A görbe egy pontja (*ECCPoint*)

Az *ECCPoint* is absztrakt objektum, a gyakorlatban csak belőle származtatott konkrét osztályokat, *ECCPoint2m* és *ECCPointp* pont objektumokat hoztam létre. Eredeti tervem az volt, hogy egyetlen *ECCPoint* osztály lesz, teljesen független a testtől, amelyben a számolok. Később hatékonysági okok miatt választottam szét a $GF(p)$ és $GF(2^m)$ feletti pontokat. Az *ECCPoint* közös interfészt biztosít ezek számára, és elrejti a testet az ECC rendszert használó osztályok elől.

Minden pont rendelkezik egy x és egy y koordinátával. Ezek a koordináták a test elemei, tehát *ECCFE* osztályú objektumok. Komoly problémát jelent ECC rendszerekben az O pont, a görbe pontjaiból álló csoport neutrális eleme. A kis méretű és egyszerű kód végett a naiv megoldást választottam: egy boolean értéket vettem fel, ez tárolja, hogy az adott pont az O pont-e. Amennyiben ez a boolean attribútum igaz értéket vesz fel, az x és y koordináták nem rendelkeznek értelemmel. (Sajnos a helyüket a memóriában felszabadítani nem lehet.)



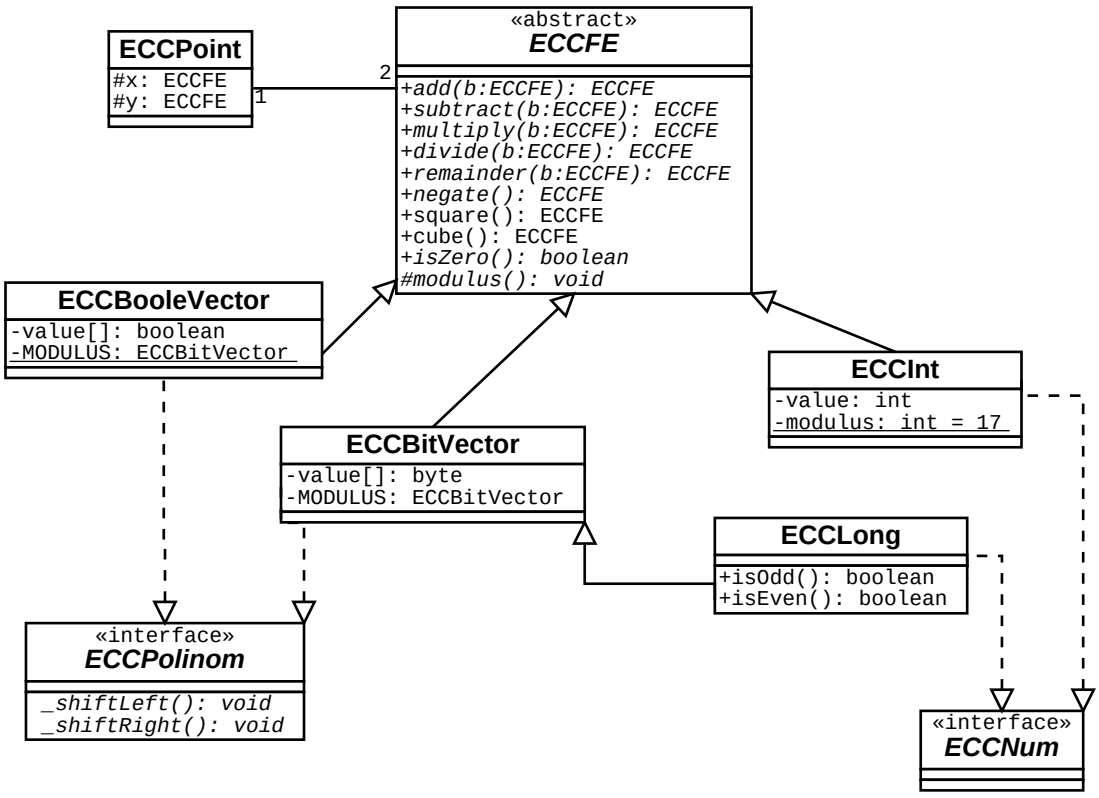
23. ábra. *ECCPoint* – a görbe egy pontja

Egy Galois test eleme (*ECCFE*)

A rendszer kulcsfontosságú osztálya az *ECCFE* absztrakt osztály, amely egy Galois test aritmetikáját írja le.

ECCFE példányokat természetesen soha nem hoztam létre, helyette belőle leszármaztatott osztályokat használtam, amelyek egy konkrét Galois testet írnak le. Ezeket két csoportra oszthatjuk: Egész számokra és polinomokra.

Az *ECCInt* és az *ECCLong* egész számok, s a $GF(p)$-beli aritmetikának megfelelően működnek. Az előbbi a java int típusa segítségével dolgozik, így a test méretét az int felülről korlátozza 32 bitben. Ennek az osztálynak kriptográfiai jelentősége természetesen nincsen, prototípusoknak hoztam őket létre, a tesztelést segítették. Az *ECCLong* osztály tetszőleges hosszúságú pozitív egész ábrázolására képes. Mivel a $GF(2^m)$-béli aritmetika mellett döntöttem – ugyanis ezt ítéltem hatékonyabbnak – ezt az osztályt teljesen nem valósítottam meg. Néhány egyszerűbb műveletére szükség volt viszont az *ECCPoint.multiply* művelet implementálásakor. Ilyen volt a kettővel való osztás (jobbra shiftelés) és a páratlanság vizsgálata. Ezen okok miatt származtattam végül az *ECCBitVector* osztályból.



24. ábra. *ECCFE* – a véges test egy eleme

Az *ECCPolinom* interfészt implementáló két osztály, az *ECCBooleVector* és az *ECCBitVector* mindössze a polinom tárolásának és elérésének módjában különbözik egymástól.

Polinomok és azok bináris reprezentációi

Az *ECCBooleVector* és az *ECCBitVector* egyaránt tetszőleges hosszúságú polinomokkal képesek dolgozni. A különbség köztük az, hogy míg az *ECCBooleVector* osztály a Java nyelv beépített boolean típusát használja a polinom bitjeinek tárolására, az *ECCBitVector* számokat használ e célra.

Először csupán az *ECCBooleVectort* valósítottam meg, mivel ezt ítéltem hatékonyabbnak. Ezzel a módszerrel a polinom egyes bitjei közvetlenül megcímezhetőek, és a boolean típussal gyors műveletvégzés lehetséges. Ugyanakkor hibája, hogy mivel a Java a boolean változókat 1 byte-on tárolja, így 1 bit információval 8 bitet foglal el. Ez azt is jelentheti, hogy sok bit mozgatásakor (pl.: átmásolás, összeadás) jelentős adatterületet kell megmozgatnunk.

Az *ECCBitVector* esetén a Java byte típusát használtam, itt minden byte minden bitjét képes voltam kihasználni adataim számára. Egy polinom mozgatásakor sokkal kisebb

adatterületet kellett írni-olvasni, így a chipkártya processzora sokkal kevesebb memóriahozzáférést kellett, hogy végrehajtson. Ugyanakkor az egyes bitek elérése sokkal drágább lett, hiszen a byte-ról le kellett választanunk a felesleges biteket. A Java nyelv támogatja ugyan a bitműveleteket, de támogatása az alacsony szintű nyelvekéhez képest igen szegényes. Nem rendelkezik a C-éhez hasonló unsigned típusokkal, továbbá bitműveleteket csakis int változók között lehet elvégezni. A Java Card specifikáció rosszallóan nyilatkozik az int típusról, s megvalósítását opcionálisnak tartja. Szerencsére az általam használt Odyssey I kártya támogatja az int típust, de felhívja a figyelmet arra, hogy megvalósítása korántsem hatékony, hiszen a kártya csupán 8 bites processzorral rendelkezik.

A Java Card specifikáció mindennek ellenére tartalmaz bitműveleteket, de hogy megvalósításuk hogyan történik, arról nincsen pontos képem. *ECCBitVector*t használva a mozgatás és az összeadás sebessége jelentősen csökken, viszont az egyes bitek tesztelésének ideje, valamint a szorzás és osztás ideje jelentősen megnő.

4.3.4. Tesztadatok ismertetése

Az általam kifejlesztett ECC implementációt össze kívántam vetni mások eredményeivel. Ezért olyan adatokat választottam, amelyek nyilvánosan ismertek, és sokan dolgoztak már velük. Megfeleltek ezen szempontoknak a Certicom honlapján (`http://www.certicom.com`) ECC Challenge néven kibocsájtott kihívások. Ezen feladványokkal a Certicom - mint egy elliptikus görbéken alapuló kriptográfiai technológiát kereskedelmi termékekbe integráló cég - demonstrálni kívánja az ECC erejét. Különböző nehézségű (kulcsméretű) kihívásokat támasztott a közönségnek, s megfelelő pénzdíjat ajánlott fel a kódok feltörőinek. [10], [11]

Választásom az ECC2-109-es görbére esett a következő okokból:

- Gyakorlati szempontból is megfelelő erősségűnek tartjuk, hiszen ilyen méretű kihívást a mai napig senki nem oldott meg.
- Elméleti szepontból az erőssége megfelel a 1024-bites RSA-nak.
- $GF(2^m)$ feletti feladványról van szó, tehát az általunk hatékonyabbnak ítélt aritmetikával dolgozik.
- Kulcsmérete elég kicsi ahhoz, hogy az általunk hozzáférhető chipkártya memóriája elegendő a kódolás elvégzéséhez.

4.3.5. Java Card implementáció

Lévén a Java Card erőforráskészlete egyértelműen szerényebb a PC-énél, a chipkártyán való futás volt a szűk keresztmetszet. Tudtam jól, hogy szoftveresen megvalósított algoritmusom, amelyet ráadásul magasszintű nyelven írtam, nem lehet versenyképes semmilyen hardver segítségével megvalósított algoritmussal. Így célom az volt, hogy:

1. Programom algoritmikus szempontból hatékony legyen (tehát a test méretének legfeljebb polinomjával legyen arányos a végrehajtási idő). Ez azt jelenti, hogy elméletileg bármekkora probléma esetén kivárható, amíg a kártya elvégzi a kódolást.
2. A kártya fizikailag képes legyen végrehajtani a programot.

Ezek után a fő probléma a memóriagazdálkodás megszervezése maradt. Két gondot kellett orvosolni:

1. A kártya memóriája kicsi (7040 byte).
2. A Java Card specifikáció nem teszi lehetővé a Java-hoz hasonló memóriakezelést. (3.3.3. fejezet) Ez azt jelenti, hogy nem lehetséges dinamikusan objektumokat létrehozni és megsemmisíteni. A programozó köteles minden memóriát az applet konstruktorában lefoglalni.
3. A Java nyelv nem teszi láthatóvá a programozó számára közvetlenül a memóriát. Így nem lehetséges közvetlenül egyes memóriacímekre írni. (3.4.3. fejezet) Tehát az assemblyben szokásos memóriakezelés sem lehetséges.

Ki kellett dolgoznom egy módszert, amely segítségével az applet élete elején lefoglalom a memóriát, majd rendelkezésére bocsájtom a metódusoknak, amelyek használják azt, majd annak befejeztével visszanyerjük a memóriát.

Egyik megoldás erre a célra, hogy szoftveresen létrehozok egy heap-et, amelyből én gazdálkodhatok. Metódusaim igényelhetnek belőle területet, majd miután nem kell nekik, visszaadják azt. Mivel programunk során ki kívántam használni a Java nyelv magasszintű típusait, a heap nem egy véletlen elérésű byte tömb lenne, hanem cellákból állna, amelyek megfelelnének a Galois test elemeinek. Tehát *ECCFE*-leszármazott objektumokból állna. A heap használatát végül elvetettem.

A másik megoldás, amelyet végül választottam, egy regiszterkészlet definiálását jelentette. Objektumaim létrejöttükkor lefoglalnak maguknak *ECCFE* és *ECCPoint* objektumokat (illetve azok leszármazottait), majd a későbbiek során csakis ezen objektumokat használják a számítások elvégzésére. Később nem foglalok le magamnak új adatterületet, és ezen regiszter-objektumokat sem pusztítom el, hanem tartalmukat írom felül. Természtesen a regisztereknek nincsen semmi közük a kártya processzorának regisztereihez, hiszen egyrészt Java nyelven programozva nem is láthatjuk azokat, másrészt méretük akár többszáz byte is lehet.

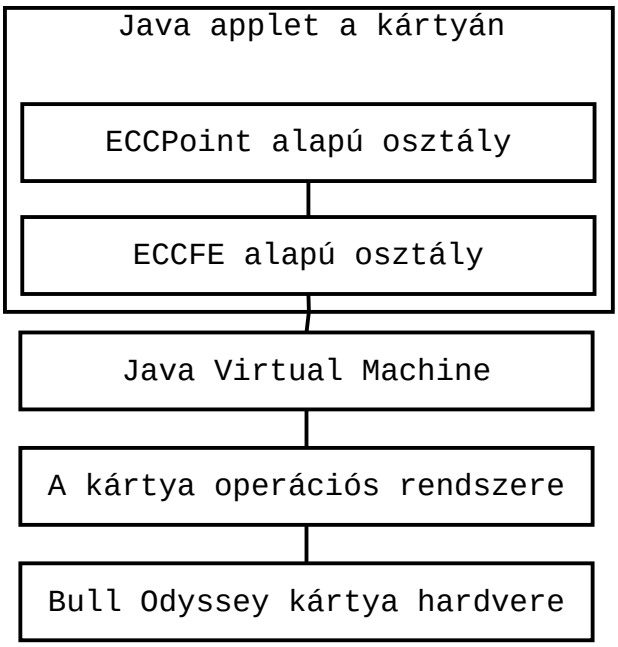
Azért választottam ezt a megoldást, mert ez logikailag is biztosítja, hogy a kártya memóriája nem fogyhat el. Az ellen is véd például, hogyha az első megoldásban az egyik művelet végén elfelejtenénk felszabadítani valamelyik segédváltozó által használt memóriát. Ilyen hibát teszteléssel lehetetlen lenne kimutatni, hiszen ha n futást megvizsgáltunk, lehetséges, hogy csak az $n+1$. futtatás után fogyna el a kártya memóriája.

Készítettem általános célú pszeudo-regisztereket is, olyanokat is, amelyek bizonyos speciális feladatokat látnak el. A *GF*-eredményregiszter például arra szolgált, hogy a test műveletei ezen regiszterbe írják be eredményüket, s később innen lehet majd azt kiolvasni. Nem egyeztek meg a regiszterek típusai sem. Többségük a test elemei közt való műveletek elvégzésére szolgált, s itt töltötte be segédváltozó szerepét. De voltak olyan regiszterek is, amelyek nem testelem-regiszterek, hanem pont-regiszterek voltak (nem *ECCFE*-leszármazottak, hanem *ECCPoint* leszármazottak).

Érdekes helyzet állt elő: Magukat a műveleteket a programot futtató gép (vagy kártya) processzora végzi el a processzor regiszterei közt. A JVM ezen regisztereket elfedi a programozó elől, és byte-okat, objektumokat tesz láthatóvá. Ezen byte-ok felett hoztam létre ECCFE-alapú pszeudoregisztereket, és a Galois-test műveleteit ezek között végeztem el. A felsőbb szint felől pedig elfedtük a Galois-test műveleteinek implementációját, s létrehoztunk ECCPoint-alapú pontregisztereket. Az ECC implementációnkat használó program elől próbáltunk elfedni minél többet a pontregiszterekből. Mindössze egyetlen egy, a pontok eredményregisztere látható kívülről, s az eredményt belőle kell kiolvasni.

Futási eredmények

A kártyára feltelepítettem az ECC2-109 paraméterű ECC Challenge-t. Méréseim eredménye értelmében egy elemi műveletet (vagyis két pont összeadását) a rendszer 7-8 perc alatt végezte el. Hasse tétele (4.1.2. fejezet) értelmében körülbelül 100 összeadás elvégzésénél többre nem lesz szükség egy kódoláshoz. Ez összesen 11 órát jelentene. (Hangsúlyozom,



hogy implementációm a technológia jelen szintjét demonstráló prototípusnak készült, nem pedig kereskedelmi forgalomba kerülő terméknek.)

Az ECC rendszer egy Java Card appletbe beágyazva képes elvégezni egy ECC kódolást, de sebessége olyan kicsi, hogy a gyakorlatban csakis akkor alkalmazható, ha feltétlenül szükséges a chipkártya által nyújtott biztonság kihasználása.

4.3.6. PC implementáció

Bár, programomat azon fő szempont szerint fejlesztettem, hogy működjön Java Cardon, ugyanaz a Java kód sikeresen lefut PC-n is. Egy közönséges JDK segítségével lefordítható, és bármilyen Java programba beágyazható. PC-n így nem jelentkeznek a Java Card sebesség- és méretkorlátozásai, tehát tetszőleges méretű test felett dolgozhatunk. A 32-bites processzor viszont az int típust kezeli hatékonyan, így mivel mi a Java Card miatt a byte és short típusokat használtam, a program PC-s Java kódnak nem optimális.

Hogyha programomat PC-re fejlesztettem volna, legalább a sebesség szempontjából kritikus részeket valamely alacsony szintű nyelven írtam volna meg.

Futási eredmények

PC-s környezetben is az ECC2-109-es kihívással teszteltem. Itt a mért eredmény természetesen sokkal jobb: A rendszer egy kódolást, vagyis egy pontnak konstanssal való szorzását

(kb 100 összeadás) 28 másodperc alatt végzett el.

PC-s környezetben a program alkalmas gyakorlati haszálatra is, bár elsősorban nem PC-re készült, így közel nem a leggyorsabb implementáció.

4.3.7. Teljesítmény szempontjából kritikus pontok

Két olyan pontot találtam, amely a teljesítmény szempontjából igen kritikusnak ítélhető. Az egyik ilyen a testben való osztás (inverzzel való szorzás), a másik pedig a pont konstanssal való szorzása. Ráadásul e két művelet egymásba ágyazódik, tehát algoritmikus komplexitásuk összeadódik.

A testelemek közt végzett osztás két részből áll: inverzképzésből és szorzásból. Az inverzképzést euklideszi algoritmussal végzem, amely igen hatékony. Ugyanakkor az euklideszi algoritmus is szorzásokból, összeadásokból és maradékos osztásokból áll. Amennyiben ezen műveleteket felgyorsítom, maga az inverzképzés is gyorsul.

A szorzást és maradékos osztást sokkal gyorsabban végezhetném el, ha az egybefüggő 0 területeket nem egyesével, hanem egyben shiftelném végig. Azt is megtehetném, hogy nem végzek minden egyes elforgatás után modulusképzést. A szorzás elvégzése után egy legfeljebb $2m$ fokszámú polinomot kapok. Ebből a modulust maradékos osztás elvégzésével kapnám.

A szorzást ráadásul sokkal hatékonyabb lenne elvégezni optimális normál bázis reprezentációval. [30] Hogy ezt nem így teszem, annak egyik oka, hogy ilyenkor az összeadás lassul le, másik pedig, hogy az algoritmus biztonsága is megszenvedheti. Másik gyorsítási lehetőség az lenne, ha ismernénk az alaptest multiplikatív csoportjának egy generátorelemét, hiszen így a szorzást tulajdonképpen összeadásként tudnám értelmezni.

Sokkal nehezebb probléma a pont egész számmal való szorzásának gyorsítása. Figyelembe véve, hogy nem ismeretes közvetlen képlet kP kiszámítására, a szorzás csakis összeadásokkal valósítható meg. Ezt figyelembe véve jelen algoritmusom igen hatékonynak mondható. Ugyanakkor konstansszorosára gyorsíthatom például azzal, hogyha a 2^iP értékeket előre kiszámolom, és egy táblázatban tárolom. Ekkor is $O(log(k))$ műveletet kell elvégezni, de a műveletek száma durván felére csökken, ugyanis jelentős mennyiségű pontduplázást kiváltunk. Ezzel a módszerrel csak akkor érünk el eredményt, ha egy P pontot viszonylag sokszor használunk, tehát sok k számmal kell megszoroznunk. Figyelembe véve, hogy P igen gyakran nyilvános kulcs, amellyel viszonylag sok kódolást kell elvégezni, ezen gyorsítási

lehetőség egyáltalán nem irreális. Nehezen megvalósítható viszont a mai memóriaméretek mellett chipkártyán, hiszen $m*log(k)$, tehát $O(m^2)$ méretű táblázatot vagyunk kénytelenek tárolni.

4.4. Összehasonlítás a Helsinki Műszaki Egyetemen készült implementációval

Kutatásomat, fejlesztésemet 2000 szeptemberében végeztem. Akkor még nem volt tudomásom a 2000 áprilisában készült, majd később publikált finn Java Card platformon működő ECC implementációról. [18], [17] A Helsinki Műszaki Egyetemen készült implementáció sok szempontból hasonlít a miénkhez, sok szempontból viszont gyökeresen más. E fejezetben e két implementációt fogom összevetni. Egyéb chipkártyás ECC implementációról nincs tudomásom.

Tommi Elo és Pekka Nikander egy már létező, tesztelt PC-s Java ECC implementációt alakítottak át Java Card platformra. Céljuk egy ECDSA-ra (A függelék) épülő azonosítási rendszer kialakítása volt, melyben a kártyára nemcsak kódolási, hanem kulcsgenerálási szerep is hárult. Egyrészt az ECC kis méretű kulcsait kívánták használni olyan architektúrában, amely sok kulcsot igényel, másrészt pedig az RSA-énál jóval egyszerűbb és gyorsabb kulcsgenerálási lehetőséget akarták kihasználni smart card segítségével.

Ők szembekerültek a $GF(p)$ vagy $GF(2^m)$ döntéssel (4.3.2), de ők – velem ellentétben – a $GF(p)$ mellett voksoltak. Döntésüket azzal indokolták, hogy $GF(p)$ esetre rendelkeztek egy letesztelt PC-s Java implementációval, valamint úgy vélték, igaz, hogy $GF(2^m)$ kis erőforráskészletű gépek esetében egyszerűbb algoritmusokat igényel, de a Java nyelv elfedi az alacsony szintű programozás nagy lehetőségeit a programozó elől.

Így döntöttek a $GF(p)$-beli aritmetika mellett. Megtehették, hiszen az ő kártyájuk (Schlumberger Access – egy Java kompatibilis, de kripto-koprocesszorral nem rendelkező kártya) 16 kilobyte memóriával rendelkezik, míg az általam akkor használt kártya mindössze 8 kilobyte-tal bír. A $GF(p)$-beli algoritmusok pedig nemcsak komplexebbek, de jóval nagyobb helyet foglalnak is el a kártyán.

Hasonlóan sok problémájuk akadt a garbage collection hiányával, ők a Java *BigInteger* típusa helyett egy *MutableBigInteger* osztályt fejlesztettek ki, amely illeszkedik a Java Card világhoz. Mutable, vagyis nem új objektumban képzi az eredményt, mint a BigInteger,

hanem ő maga “változik” át az eredményt kéző *MutableBigInteger*-ré. Állításuk szerint *MutableBigInteger* kódját egy már létező C implementációból vitték át Java Card-ra.

A két implementáció teljesítményét nehéz összehasonlítani, mivel ők nem közölnek teljes mérési eredményeket – legalábbis kártyára vonatkozót nem. Pusztán összeadások, szorzások és inverzépzések idejét közlik, 50-bites, 100 bites és 192-bites ECC problémákon. Ennek ellenére látható, hogy 50 bit felett nem tudnak inverzképzési eredményt felmutatni, holott az inverzképzés kriptográfiai műveletek elvégzéséhez szükséges. 50 bit nemcsak ma, de a Certicom Challenge-ek kibocsájtásakor sem, volt kriptográfiai szempontból érdekes problémának tekinthető.

Az én ECC implementációm 109 bites ECC problémákkal is megbírkózik, amely már olyan méretű, hogy ekkorát feltörni a mai napig senkinek sem sikerült.

5. Felhasználói profile tárolása intelligens kártyán

E fejezetben egy példaalkalmazás specifikációja kerül bemutatásra, amely kihasználja a programozható kártyák nyújtotta új lehetőségek egy részét. A rendszer alapötletét [33] és [25] képezte, de – lévén, hogy az említett irodalmak egy nagy cég lehetőségeit vázolták fel – jelentős egyszerűsítéseket kellett elvégeznem a modelljükön. További változás, hogy a fent nevezett irodalom egy komplex rendszert ír le, különféle terminál típusokat említ, és heterogén struktúrában gondolkozik. Munkámban elsősorban a kártyára, valamint a kártyának a rendszerben betöltött szerepére koncentráltam, a rendszer többi komponensére kisebb hangsúlyt fektettem.

5.1. A rendszer funkcióinak ismertetése

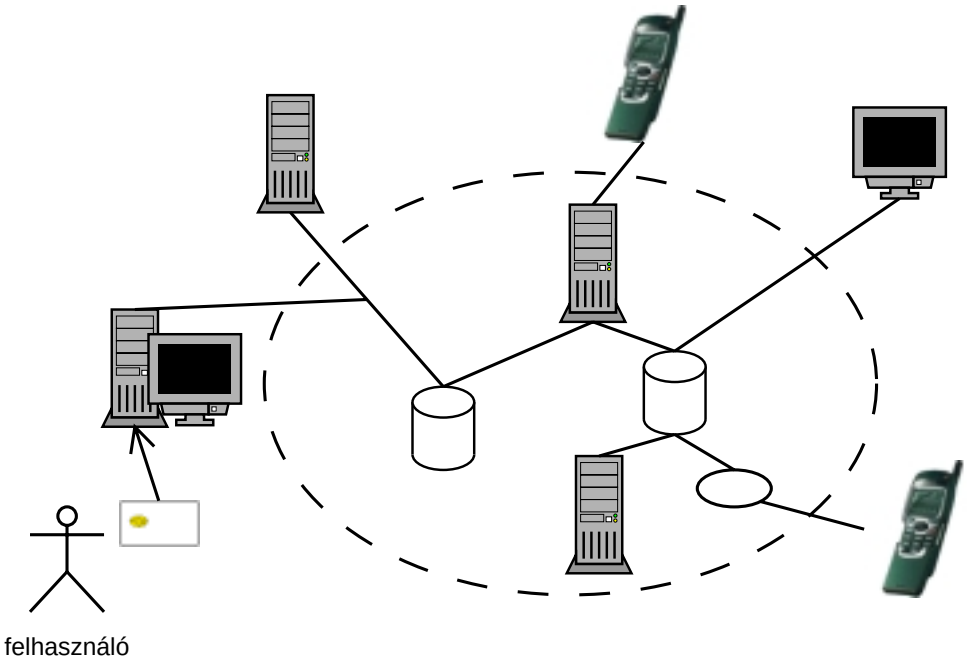
5.1.1. Új felhasználói igények

A felhasználók egyre többet utaznak, mozognak, egyre „nomádabbakká" válnak. Tevékenységük egyre kevésbé egy számítógépre korlátozódik, látókörük és hatáskörük kiszélesedett. Amellett, hogy hálózatba kötött gépeket használnak, vagy az Interneten dolgoznak, új és új készülékek, rendszerek, terminálok jelennek meg, melyek tovább növelik a rendszer komplexitását, heterogenitását.

Manapság kezd teljesen általánossá válni az intelligens mobil telefonok, PDA-k használata, de – elsősorban a tengerentúlon – kezdenek előtérbe kerülni a set-top-boxok is. A felhasználók oldalán egyre erősödő igény jelentkezik: különféle termináljaikról egységesen szeretnék látni a hálózat által felkínált szolgáltatásokat, alkalmazásokat. (25. ábra)

Attól függetlenül, hogy éppen milyen készülékről lépnek be a rendszerbe, ugyanazon alkalmazásokat szeretnék elérni, és ugyanazon adatokkal szeretnének dolgozni. Ilyen igény például, hogy a mobiltelefon és a PC segítségével ugyanazon postaládát olvashassák, és ugyanazon címlistát (address book) használhassák, valamint eltárolt leveleiket egyformán olvashassák. [33]

Figyelembe véve, hogy a PC és a mobiltelefon különböző célt szolgál, azt a célt kitűzni, hogy például ugyanazon szövegszerkesztő program fusson rajtuk, nyilvánvalóan ostobaság. Nemcsak számítási kapacitásukban és memóriájukban különböznek, de felhasználói felületük is gyökeresen más. A mobiltelefon billentyűzete nem arra szolgál, hogy regényt írjanak rajta, és kijelzőjével szemben is egészen más követelmények érvényesülnek, mint egy PC



25. ábra. Heterogén rendszer, heterogén terminálok

képernyőjével szemben. Ugyanakkor, – a példánál maradva – egy mobiltelefon felhasználó számára lassan természetes igénnyé kezd válni, hogy ha (pl. emailben) kap egy szövegfile-t, abba beleolvashasson, majd módosíthassa vagy továbbküldje.

Az egyik probléma tehát a különböző terminál típusokon futó különböző alkalmazások kérdése. A másik pedig az, hogy a mai alkalmazások erősen perszonalizálhatók, rengeteg beállítás szükséges hozzájuk. El kell érniük a felhasználó bizonyos adatait: nevét, lakcímét, bankszámlaszámát, digitális pénztárcáját, stb. Mindemellett el kell érni azon adatokat is, amelyekkel a felhasználó dolgozni szokott: kedvenc weboldalainak címét, ismerőseinek, rokonainak és üzletfeleinek címét, telefonszámát, valamint bizonyos FTP site-okat, stb. Az érintett adatok jelentős része bizalmas (pl. hitelkártya kódja), továbbítása csakis bizonyos biztonsági óvintézkedések mellett célszerű.

Amíg a felhasználók bizonyos terminálokhoz szorosan kötődnek, a fenti adatokat természetszerűleg a terminálban érdemes tárolni. Csakhogy a tapasztalat azt mutatja, hogy a felhasználók egyre nomádabbakká válnak: munkahelyükön vándorolhatnak gépek között, időnként otthoni PC-jükről szeretnének dolgozni, de utazás közben a mobiltelefonjukról is el szeretnének érni bizonyos adatokat. Nem is beszélve arról, hogyha bizonyos berendezéseket le kívánunk cserélni, minden adatot át kell vinni az új terminálra a régiről. A felhasználók részéről mutatkozó nomaditás új feladatokat ró ki az alkalmazásfejlesztőkre.

5.1.2. Három lehetséges megoldás

Gondoskodnunk kell a felhasználói profil, valamint a felhasználók számára szükséges alkalmazások elérhetőségéről. Erre három megoldás kínálkozik:

1. Minden terminálra telepítsünk fel minden alkalmazást, valamint vigyük fel minden felhasználó minden adatát! Ebbe még belegondolni is rossz. Egy felhasználócsoport számára szükséges alkalmazások még telepíthetőek egy-egy teminálra, de ha a rendszeben n terminál és m felhasználó létezik, ez $n * m$ db profil karbantartását jelenti, amely kivitelezhetetlen, ha n és m nagy számok.
   Felmerülnek továbbá komoly biztonsági kérdések is: a felhasználói profil bizonyos részei bizalmasok, bizonyos terminálok biztonsága pedig igencsak megkérdőjelezhető. Érzékeny információkat minél kevesebb helyen kell tárolni, minél jobban védve.

2. Legyen a rendszerben egy központi szerver, érjék el azt a felhasználók! Ez esetben a terminálokon csupán a szerverre való bejelentkezéshez szükséges információk lennének rajta, minden további információ a szerverről érkezne. Csakhogy a központi szerverrel szemben több igen komoly probléma merül fel:

   (a) Ha sok felhasználó mind egyazon szervert akarja elérni, s vele nagy sávszélességű kommunikációt folytat, nagyon leterhelheti azt.

   (b) Nem minden terminháltípus esetén lehetséges a nagy sávszélesség (pl mobiltelefon), sőt, bizonyos terminálok nem lehetnek folyamatosan elérhetőek.

   (c) A központi szerver ún. single point of faliure (SPF) lehet, amely kitűnő célpontjává válhat denial of service (DOS) támadásoknak.

   (d) Ha az érzékeny információk folyamatosan a hálózaton mozognak, gondoskodni kell megfelelő rejtjelezésükről is, amely jelentős számítási kapacitást igényelhet.

3. Tároljanak a felhasználók minden információt maguknál egy adathordozón, és ezzel telepítsék fel a terminált! Itt is igen komoly gondok vannak. Ha minden információt tárolni akarunk, a mai adathordozók közül egyedül a CD (vagy DVD) lemez jöhet szóba. A floppy disk vagy a chipkártya csupán elenyésző mennyiségű adatot képes tárolni, a mágnesszalagon nem tudunk kellő sebességgel keresni, a merevlemez pedig nagy és igen érzékeny. CD lemez viszont nem helyezhető igen sok terminháltípusba (pl mobiltelefon, palmtop).

Mivel a fenti három pont közül önállóan mindhármat elvethetjük, [33] és [25] egy olyan megoldással álltak elő, amely ezek valamely kombinációjából áll.

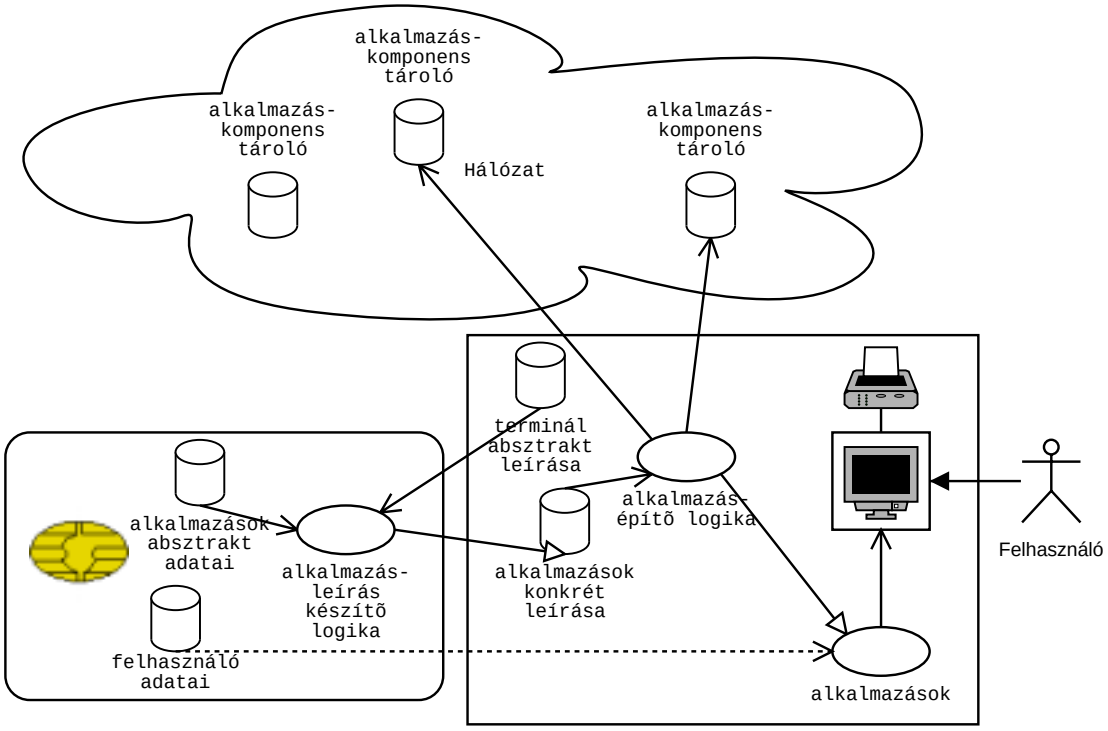
5.1.3. Egy negyedik lehetőség

Az előző fejezet 3. pontjában a felsorolt adathordozók közül egyedül a chipkártya tett eleget egy nagyon fontos követelménynek: szinte minden fontos platformon elérhető. PC-n kártyaolvasó, vagy notebookokba beépített olvasó segítségével, mobiltelefonon SIM kártya olvasó segítségével, de számos egyéb platform is képes lehet chipkártyákat olvasni: banki automaták, boltban kártyás fizetéshez szüksége automaták, utcai telefonkészülékek, de set-top-boxok is.

A chipkártya számos további előnnyel is rendelkezik. Sokkal többet nyújt, mint egy floppy disk, mert képes megvédeni az adatokat, és akár kriptográfiai műveleteket is végezhet. Egy igen komoly hátulütője van: a kis tárkapacitása. Itt kombináljuk össze a chipkártyákat az előző fejezet 2. pontjával: a hálózati szerverekkel.

Vagyis, a kártya tartalmazza az érzékeny adatokat, nagy adatmennyiség esetén pedig valamely hálózati szolgáltatásra hivatkozik, illetve annak eléréséhez biztosítja a szükséges jelszót vagy kulcsokat. A chipkártyát a felhasználó a zsebében hordja magával, és ha be kíván jelentkezni egy terminálról a rendszerbe, behelyezi a kártyáját. Azonosítja magát a kártya felé, majd a terminál elkéri a kártyától a szolgáltatások listáját. Annak függvényében, hogy a felhasználó és a kártya mennyire bíznak a terminálban, bizonyos szolgáltatásokat, illetve bizonyos információkat a kártya megtagadhat a terminálról.

Az alkalmazásoknak a terminálra való telepítésére Pellegrini, Potonniée és Marvie [33] egy ún. unified application bootstrap kialakítását javasolják. Ez esetben a kártya a lehetséges alkalmazások egy absztrakt, terminál típus-független leírását tartalmazza. E leírás különféle részeire különféle megbízhatósági szintű, illetve típusú terminálok eltérő jogosultsági szintekkel rendelkezhetnek. A terminál az alkalmazásleíráshoz hozzácsatolja annak eszközfüggő részét, majd a kártya és a terminál közösen telepítik fel az alkalmazást a terminálra bizonyos hálózati erőforrásokról. (26. ábra) Ezen diplomaterv a chipkártyák szerepére összpontosít, az alkalmazások absztrakt leírásokból való legyártásával nem foglalkozik. Erre pl. [25] vázol fejt ki egy lehetőséget. (26. ábra)



26. ábra. Hogy települnek a szolgáltatások a terminálra?

5.1.4. Mit tudjon a rendszer?

Képzeljünk magunk elé egy heterogén rendszert (25. ábra)! A felhasználó egy terminálról jelentkezik be, amely lehet PC, de lehet mobiltelefon vagy set-top-box vagy akár PDA is. Terminálja segítségével a hálózatban lévő szervereken különféle szolgáltatásokat vesznek igénybe. A hálózat nem feltétlenül egységes, különböző részei különböző cégek vagy intézmények felügyelete alá eshetnek. A hálózat minden komponense nem lehet mindig elérhető, nincsen központi szerver, ahova be lehetne jelentkezni, a bejelentkezést javarészt a chipkártya kell, hogy levezényelje.

- Ha ellopják a kártyát, ne férhessen hozzá a tolvaj a rajta lévő adatokhoz!
- Ne lehessen a felhasználót hamis terminálokkal megtéveszteni!
- Ne lehessen a rendszert kártya nélkül használni!

5.1.5. A kártya szerepe a rendszerben

- Felhasználó beléptetése – a kártya azonosítja a felhasználót és a terminált, valamint ő is azonosítja magát a terminál felé. Mivel a chipkártya számítógép, az egyszerű jelszónál sokkal biztonságosabb mechanizmusokat is használhat.
- Felhasználói profil tárolása és továbbítása. Vagyis a chipkártya a hálózatot váltja ki – legalábbis részben. Segítségével rendelkezésre áll a felhasználó összes adata, beállítása, mindig, amikor csak szükség van rá.
- Felhasználói profil védelme. Ez azt jelenti, hogy a kártya nem adja ki magából a szükséges adatokat, csakis akkor, ha a felhasználó felhatalmazza erre.
- Távoli rendszerekbe való bejelentkezés dinamikus jelszavakkal. Távoli rendszerbe való bejelentkezés esetén csakis a tudás alapú azonosítás (B függelék) használható. Ha a szerver mindig ugyanazon bitsorozat alapján engedi be a felhasználókat, ezt egy támadó felveheti és később visszajátszhatja. A megoldás az, ha ez a bitsorozat mindig változik. (pl: challenge and response módszer, időfüggő jelszavak, stb.) Itt a felhasználóra hárul az éppen aktuális jelszó kiszámítása, amely számítógép (vagy chipkártya) segítsége nélkül nem reális lehetőség. Erre példát mutatok a későbbiekben (6. fejezet).
- A felhasználó által elérhető szolgáltatások telepítése. A kártya nemcsak a felhasználó, hanem a rendszer érdekeit is érvényesítheti. Pl. megmondhatja a terminálnak, hogy az adott felhasználó mely szolgáltatásokat használhat, és melyeket nem.
- A kártya végezhet a felhasználó utasítására titkosítási műveleteket, vagy készíthet digitális aláírást.

5.2. Specifikáció

5.2.1. Általános megfontolások

A rendszernek elsősorban a kártyán elhelyezkedő részeivel kívántam foglalkozni, PC oldalon a már létező eszközökre építettem (web böngésző, FTP kliens, stb.), vagyis nem volt célom sem alkalmazások írása, sem programomnak alkalmazásokhoz való finomhangolása.

A rendszernek elsősorban a chipkártyás oldalát dolgoztam ki, de – korlátos erőforrásaim miatt – pár dolgot itt is a feladat határain kívül kellett helyeznem.

A rendszer kártyás oldalát Java Card technológiára kívántam építeni. Egyrészt, mert ez egy nyílt specifikáció, amely igen sok implementációval rendelkezik (D függelék), másrészt pedig mert ezen specifikációt jól ismerem, s Java Card programozási környezetben már sokat dolgoztam. [3]

Mivel a Java Card specifikációnak különböző változatai léteznek, próbáltam a Java Card API azon elemeire építeni, melyek mindegyikben megvannak (gondolva itt a Java Card 2.0 és 2.1 közti jelentős különbségekre). Továbbá nem használtam ki az általam választott hardware (5.3.1. fejezet) bizonyos speciális lehetőségeit, hogy programom kártyafüggetlen maradhasson.

PC oldalon is Java nyelven dolgoztam, kihasználva annak platformfüggetlenségét, így alkalmazásom nem csak egyetlen platformon működőképes.

5.2.2. A rendszer szűkítése

Mivel a rendelkezésemre álló idő, illetve erőforrások mind korlátosak voltak, továbbá a diplomaterv célja nem "a tökéletes rendszer" megtervezése, hanem a programozható chipkártyák egy lehetséges alkalmazásának bemutatása volt, a rendszeren bizonyos szűkítéseket végeztem:

1. Nem foglalkoztam az alkalmazások absztrakt leírásának elkészítésével, illetve az alkalmazások absztrakt komponensekből való összeállításával, majd az alkalmazásnak különböző platformokra való telepítésével. Az én cardletem nem "alkalmazások", hanem 'szolgáltatások" elérésére szolgál, a szolgáltatásokkal pedig igyekeztem a megvalósíthatóság határain belül maradni. Ugyanakkor célom volt, hogy ne akadályozzam meg az esetleges továbbfejlesztést sem. Az általam kifejlesztett cardlet illetve terminálprogram ismert hálózati illetve PC-s szolgáltatásokhoz (pl: WWW, FTP, SSH) csatlakozik – lehetőleg számos platformon létező – ismert kliens programok segítségével (5.3.1. fejezet).

2. Az 5.1.3. fejezet szerint különböző megbízhatóságú terminéltípusok létezhetnek. Egymástól eltérő architektúrájú rendszerekre egy külső támadónak betörni különböző nehézségű lehet (pl: semmiképp sem nevezhető azonos biztonságúnak egy Win-

dows 98-at futtató számítógép egy Windows NT-s workstationnel vagy egy mobiltelefonnal), de egyes terminál típusokat tekinthetünk kisebb vagy nagyobb mértékben megbízhatónak amiatt is, mert például más-más cég kezében vannak. Én három biztonsági szintet definiáltam (5.2.4. fejezet). Úgy véltem, ennek segítségével megfelelően demonstrálhatom a programozható chipkártyák képességeit, de sem a rendszer komplexitását nem növelem meg túlzottan, sem pedig a kártyán nem foglalok le elviselhetetlenül sok helyet a kulcsok számára.
Az általam készített specifikációban a három biztonsági szinthez 1-1 RSA [19] nyilvános kulcs tartozik. Ugyanakkor könnyen elképzelhetőnek tartom, hogy egy gyakorlati alkalmazásban több kulcs szerepeljen. Egy bizonyos szám felett kezelhetetlenül sok kulcs lenne a kártyán. Ilyenkor nemcsak gondozásuk lenne költséges, de az általuk elfoglalt hely is igen jelentős lenne a kártya méretéhez képest. (Sőt, mint az kiderült, meglepően sok helyet foglalhatnak el (5.3.2. fejezet).)
A mérnöki szempontból helyes megoldás egy tanúsítványokra alapuló rendszer lenne. ([19]) Ilyenkor a terminál elküldené a nyilvános kulcsát a kártyának, s mellékelné hozzá a tanúsítványt, amely igazolja, hogy az előbb felmutatott nyilvános kulcs egy a rendszerben regisztrált kulcs. A kártya feladata az lenne, hogy ellenőrizze a tanúsítványt, majd egy kihívás segítségével meggyőződjön róla, hogy a terminál tényleg rendelkezik-e a nyilvános kulcs titkos párjával.
E tanúsítvány-rendszerből egy már bevezetett és szabványos rendszer használata lenne célszerű, amely egy külön feladatot képezne. Ezt a részt nem kívántam a diplomaterv keretein belül megvalósítani.

3. Az általam fejlesztett szoftverben a kártyán szolgáltatások foglalnak helyet. A szolgáltatások a cardlet részét képezik, és annak kártyára való feltöltésekor kerülnek a kártyára, és a cardlet letörlésekor tűnnek el. Egy esetleges gyakorlati alkalmazás esetén lényeges lenne körüljárni a cardletek telepítésének és eltávolításának problémakörét is. Érdekes kérdések merülnek fel:

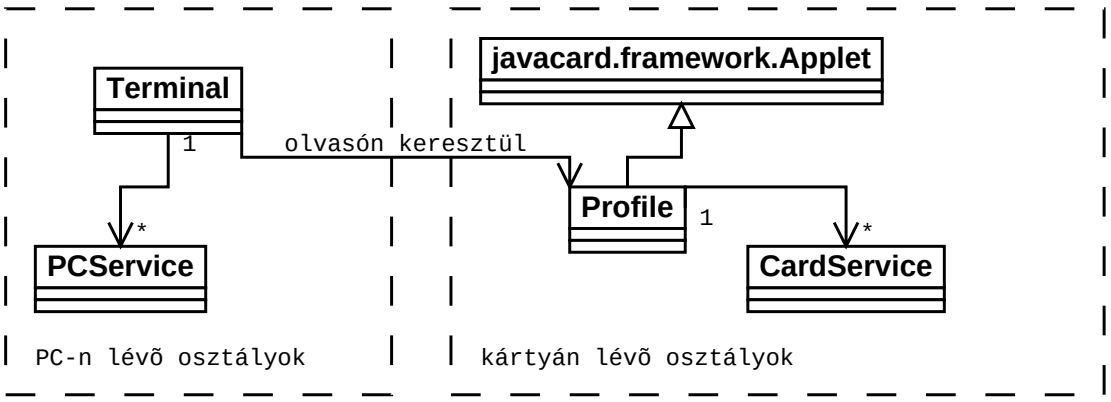
   - Hogyan települ fel a szolgáltatás a kártyára?
   - Hogyan tűnik el a szolgáltatás a kártyáról?
   - Mikor, illetve milyen tanúsítványok felmutatása esetén változhat meg egy szolgáltatás tartalma?
   - Kinek legyen joga szolgáltatást telepíteni?
   - Kinek legyen joga szolgáltatást letörölni?

- Mi történik, ha letörlünk egy szolgáltatást? (A 3.3.3. fejezet szerint a Java Card specifikáció nem biztosít lehetőséget memória felszabadítására, így ezen problémát szoftveresen kell megkerülni.)
- Mi történik, ha nincs elég hely a kártyán?

A fentiek számos új biztonságtechnikai problémát vetnek fel, melyek megoldása a diplomaterv keretén belül nem volt célom. Alkalmazásommal elsősorban a chipkártyatechnológia lehetőségeit kívántam demonstrálni, és ezen célt a fenti problémakör körüljárása nem közelítené meg jobban.

5.2.3. A használt osztálystruktúra bemutatása

A rendszer legkomplexebb alkotóeleme a *Profile* osztály volt, amely a kártyán foglal helyet. A *Profile* leszármazottja a *javacard.framework.Applet*-nek, ez alkotja a cardlet interfészét a külvilág felé. A cardlet rendelkezik szolgáltatások egy listájával, melynek elemeihez a *Profile* osztály hozzáférhet.



27. ábra. A PC-n és a kártyán lévő osztályok együttműködése

A *Profile*, mint minden cardlet, megfelel a korábbiakban már bemutatott, 16. ábrán vázolt szerkezetnek. Legfontosabb metódusa a *process*, amely az APDU INS mezőjét figyelve eldönti, milyen műveletet szükséges végrehajtani. Ez után a *process* meghívja a cardlet megfelelő metódusát.

A kívánt metódus ellenőrzi, hogy a kártya megfelelő állapotban van-e, és ha igen, ellenőrzi a feladathoz szükséges jogosultságokat, majd elvégzi a kért feladatot. A kártya állapotának elsősorban a kölcsönös autentikáció során van szerepe. Itt elkülöníthetünk különböző

fázisokat (terminál kért-e már challenge-et, válaszolt-e már rá, azonosította-e magát a terminál, azonosította-e magát a felhasználó, stb.), amelyeket később (29. ábra) részletesen ismertetek.

A *Profile* felelőssége:

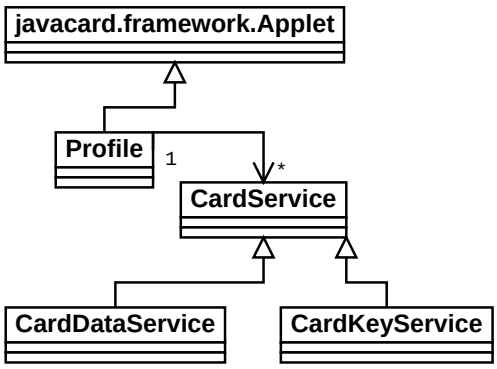
1. A terminál azonosítása
2. A cardlet azonosítása, amennyiben a terminál igényli
3. A felhasználó azonosítása, ha a terminál azonosítása sikeres volt
4. A rendszer szolgáltatásainak létrehozása
5. Kérés esetén a szolgáltatásokhoz hozzáférés biztosítása
6. A szolgáltatások felé az adott biztonsági szint (5.2.4. fejezet) jelzése

5.2.2 szűkítései következtében a *Profile* szolgáltatástelepítési része nincsen kidolgozva, de a későbbiekben ez is a *Profile* része lenne. A rendszer jelen állapotában a szolgáltatásokat nem lehetséges dinamikusan telepíteni, hanem a rendszerben értelmezett szolgáltatások a cardlet élete során statikusan léteznek. A *Profile* konstruktora telepíti ezen szolgáltatásokat, ő tölti fel őket adatokkal is.

A kártyán természetesen nem maguk a szolgáltatások foglalnak helyet, csupán az elérésükhöz szükséges információ, illetve annak védelmi mechanizmusai, hogy akárki ne férhessen hozzá. A fenti (27. ábra) struktúrán látható, hogy a szolgáltatáslistának megfelelő osztályok megtalálhatóak mind a kártya, mind a terminál oldalon.

Csakhogy sem tartalmuk, sem funkciójuk nem egyezik meg. A kártyán lévő objektumokban (*CardService* leszármazottai) a szolgáltatás eléréséhez szükséges adatok találhatóak meg (pl: szolgáltatás helye, usernév, jelszó), míg a terminálban lévő objektumokból (*PCService* leszármazottai) pont ezen információk hiányoznak. Ami ezen objektumokban benne van, az az a tudás, hogy a kártyából kinyert információk segítségével hogyan lehet a szolgáltatásokat elérni.

Természetesen mind a *CardService*, mind *PCService* absztrakt osztályok, a konkrét implementációkat leszármazottaik tartalmazzák. A *CardService* esetében az implementációkat az alapján választottam szét, hogy az, amit védünk kulcs vagy adat (28. ábra).



28. ábra. A kártyán lévő osztálystruktúra

Ha a *Profile*-hoz kérés érkezik, amelyben a terminál meg kívánja tudni valamely szolgáltatás valamely adatát (pl. a `ural2.hszk.bme.hu` című gépre való fileelérés-hez szükséges felhasználónevet), akkor a *Profile* továbbadja a kérést az adott szolgáltatásnak. Az adott szolgáltatás felelőssége az adatok kiadásáról dönteni. A szolgáltatás megkérdezheti a *Profile*-tól az adott azonosítási szintet, majd ennek függvényében fogadhatja el vagy tagadhatja meg a kérést.

5.2.4. Azonosítási szintek

Háromféle azonosítási szintet különítettem el annak alapján, hogy a terminál mennyire megbízható:

A: Anonim. Ebben az esetben a *Profile* és a *CardService*-ek megtagadnak minden olyan információt a terminálótól, amelynek segítségével a felhasználó személyazonossága meghatározható lenne. Nem férhet hozzá senki a felhasználó nevéhez, lakcíméhez, de az FTP usernevéhez sem az egyes szolgáltatások elérésekor. Lehetséges viszont a bookmarkjainak elérése (WWWService), és esetleg lehetséges digitális pénz vagy mikrofizetési módszerek segítségével fizetnie (de bakkártyaszám segítségével már nem).

B: Biztonságos. Feltételezzük a terminálról, hogy megbízható, és hajlandóak vagyunk felfedni a felhasználó személyazonosságát. Ez esetben hajlandó a kártya kiadni az FTP usernevеket, de a jelszavakat nem, mert a terminál esetleg nyílt csatornán továbbíthatja őket, vagy a cache-éből esetleg később kinyerhetők lehetnek. Lehetséges viszont SSH (6.3. fejezet) segítségével belépni távoli rendszerekbe, mert itt használhatunk challenge and response azonosítást, és jelszavunk nem kerül veszélybe.

C: Teljesen nyílt. Maximálisan megbízik a kártya a terminálban. Hajlandó magából kiadni bármilyen információt (akár jelszavakat is), kivéve az RSA kulcsokat, hiszen azokat tökéletesen használhatjuk a kártya segítségével is.

Amennyiben a kártya megtagadja a hozzáférést valamely adatelemhez, akkor az hozzáférhetetlenné válik mindenki számára. Természetesen a megbízhatatlannak vélt terminál elől titkoljuk, de nem juthat hozzá maga a felhasználó sem, hiszen az adatokat ő is csak a terminálon keresztül érhetné el.

5.2.5. A kapcsolat felépülése

A kártya olvasóba való behelyezésekor az olvasó megvizsgálja, tényleg a rendszerhez tartozike a kártya. Ezt challenge and response azonosítással teheti meg (44. ábra): Minden, a rendszerhez tartozó kártya rendelkezik egy titkos kulccsal, s minden terminálban benne van a hozzá tartozó nyilvános kulcs. Ez az azonosítás a kártya állapotán nem változtat.

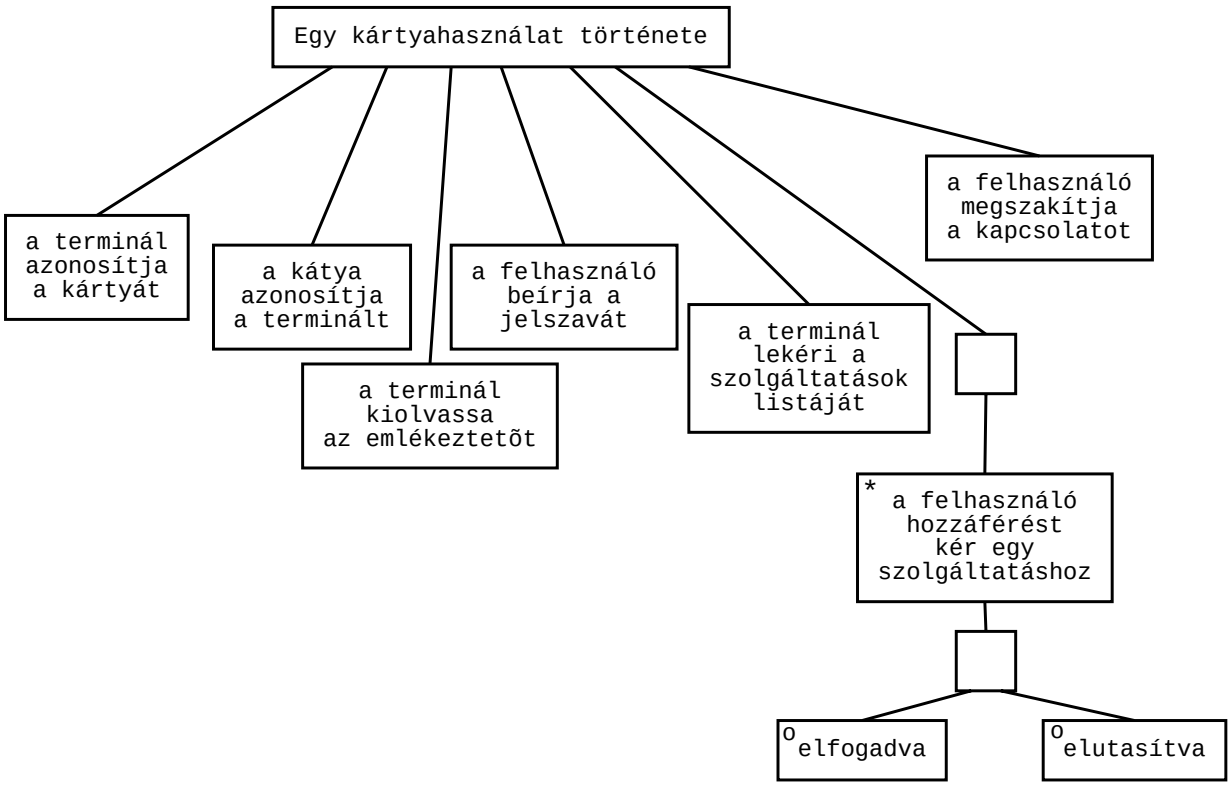
Ezután a terminál közli a kártyával, hogy azonosítani szeretné magát. Ez is challenge and response módszerrel történhet. Az azonosítás kérésekor ki is kell választania egyet a 3 azonosítási szint (5.2.4. fejezet) közül, amelyiken az azonosítást kéri. Mivel a kommunikációban mindig a terminál a kezdeményező (2.2.5. fejezet), ez csak két APDU segítségével történhet: az első APDU-val a terminál kér egy kihívást a kártyától, majd kiszámítja annak válaszát, és a második APDU-val visszaküldi azt.

Ezen a ponton a terminál és a kártya biztosak lehetnek egymás kilétében. A terminál tudja, hogy tényleg egy, a rendszerhez tartozó kártyát lát, a kártya pedig tudja, hogy egy a rendszerhez tartozó terminál olvasójába helyezték bele, valamint azt is tudja, hogy az olvasó mennyire megbízható (A, B vagy C megbízhatósági szint).

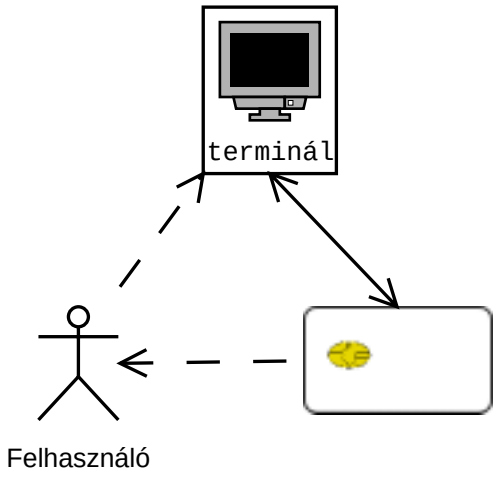
A terminál és a kártya kölcsönösen azonosítják egymást. Ezt jelöltem folytonos vonallal (30. ábra). Szaggatott vonallal jelöltem, amikor a szereplők a harmadik fél közbenjárásával azonosítják egymást, mint ahogy az a következőkben látható.

Ekkor következik be a felhasználó azonosítása. Itt azonban van egy kis probléma. A chipkártya ugyanis nem rendelkezik input-output eszközökkel, így nemcsak hogy a felhasználó azonosítására nem képes, nem is tud közvetlenül jelt adni neki, ha a terminált nem véli elég biztonságosnak.

Itt alkalmaztam [7, 4.2.2. fejezet] által leírt protokollt. A felhasználó tárol a kártyában egy "emlékeztetőt". (Nem nevezem jelszónak, hogy ne legyen összekeverhető azzal, amivel



29. ábra. A kapcsolat felépülése



30. ábra. A rendszer szereplői azonosítják egymást

a felhasználó azonosítja magát a kártya felé.) Az “emlékeztetőt” a felhasználó bármikor, bármely terminálon megváltoztathatja. Kiolvasni viszont csak akkor lehet az emlékeztetőt, ha a terminál sikeresen azonosította magát a kártya felé, különben a kártya nem ad engedélyt annak elérésére.

A protokoll a következő:

1. A terminál azonosítja magát a kártya felé.
2. A terminál elkéri a kártyától az "emlékeztetőt" (ezt a kártya csak az 1. lépés után adja ki magából)[8]
3. A terminál kiírja az emlékeztetőt a képernyőjére.
4. A felhasználó elolvassa az emlékeztetőt, és felismeri, ha az az övé.
5. Ha a felhasználó a saját emlékeztetőjét ismerte fel, beírhatja a jelszavát.

Mint ahogy azt láthatjuk (30. ábra), a felhasználó nem azonosítja közvetlenül a terminált, de értesül róla, ha az megbukott a kártya azonosításán, s akkor nem írja be a jelszavát/PIN kódját. Így – közvetve bár – azonosítja a terminált. Szintén szaggatott nyilat rajzoltam a felhasználó és a kártya közé. A kártya ugyanis csak a terminál segítségével képes a felhasználót azonosítani. Az alábbi táblázatban (1. táblázat) láthatjuk, végül ki azonosít kit.

ki\kit	**felhasználót**	**terminált**	**chipkártyát**
felhasználó	x	igen	nem
terminál	nem	x	igen
chipkártya	igen	igen	x

1. táblázat. Ki azonosít kit?

Egyedül a felhasználó nem azonosítja a chipkártyát, hiszen ő vette ki az imént a zsebéből, mielőtt a terminálba behelyezte. Nem azonosítja továbbá a terminál a felhasználót. Ez közvetlenül elég nehezen lehetséges, hiszen minden terminálba nem építhetünk be biometriai (B függelék) azonosítási rendszereket. Jelen esetben a szolgáltatásokat úgyis csak a chipkártya segítségével lehet igénybe venni, a kártya pedig nem ad ki magából semmit, míg a felhasználó be nem írta a jelszavát. A terminál tehát nem azonosítja a felhasználót, de a felhasználónak be kell írnia a jelszavát, különben semmilyen szolgáltatáshoz nem férhet hozzá a kártyán.

[8]Mivel különböző biztonsági szintek vannak, helyes volna, ha a felhasználónak is különböző emlékeztetője és jelszava lenne minden biztonsági szinthez. Amennyiben tanúsítvány rendszert alkalmaznék, a kártyának kellene a tanúsítványt valamilyen biztonsági szintre besorolni. Ugyanakkor a felhasználó memóriáját sem célszerű túlterhelni, nem is beszélve arról, hogy az emlékeztetőnek csakis akkor lehet értelme, ha kevés van belőle, és így a felhasználó gyakran változtathatja.

5.2.6. Milyen kulcsok vannak a kártyán?

1. Az a kulcs, amely bizonyítja, hogy a kártya a rendszerhez tartozik
2. Az annak ellenőrzésére szolgáló nyilvános kulcs, hogy a terminál valóban a rendszerhez tartozik-e (5.2.2. fejezet)
3. A felhasználó kulcspárja titkosításhoz, illetve digitális aláíráshoz
4. A különféle szolgáltatásokba való bejelentkezéshez szükséges kulcsok (pl SSH kulcspár)

5.3. Implementáció

5.3.1. A használt technika bemutatása

A kártya – Schlumberger Cyberflex Access

A 5. fejezetben bemutatott rendszert a Schlumberger „Cyberflex Access 3C" kártyájára (31. ábra) fejlesztettem. A Cyberflex a Java Card specifikáció egy referencia-implementációja. Schlumberger saját bevallása szerint a „Cyberflex az első olyan kereskedelmi forgalomban lévő kártya, amely ötvözi a Java Card technológiát erős kriptográfiával" [38].

31. ábra. Cyberflex Access: Java Card technológia + kriptográfia

A kártya 16 kilobyte EEPROM memóriát tartalmaz, amelyből 13506 byte használható fel fileok és appletek számára. A processzora feltehetőleg 16 bites, legalábbis a kártya dokumentációja [37] erősen a Java short típusának használatát javasolja a byte típussal szemben. Sajnos a gyártó a dokumentációban nem adott meg ilyen jellegű információkat, bár ez nem is feltétlenül cél: egy Java Card programnak nem egy adott hardverre kell optimálisnak lennie, hanem célja éppen a kártyafüggetlenség.

A kártyán GPOS (General Purpose Operating System) operációs rendszer van, valamint egy Solo fedőnévre hallgató Java virtuális géppel is rendelkezik. A kártya (illetve a rajta futó JVM) nem támogatja a Java 16 bitnél hosszabb egészeit, vagyis az int-et és a long-ot.

A Cyberflex szimmetrikus kulcsú titkosítási eljárások közül a kártya támogatja a DES-t és a tripleDES-t, az asszimetrikus kulcsúak közül pedig az RSA-t, maximum 1024 bites blokkmérettel. Emellett ismeri az SHA-hash függvényt. A kártya a Java Card 2.0-s specifikációnak felel meg, vagyis nem a legújabbnak, a 2.1.1-nek. [45] Így igen sok olyan, a kártya dokumentációjában favorizált vonás van, melynek használatától eltekintettem, ugyanis a Java Card 2.1-es specifikációnak nem felelnek meg. Ilyen például az RSA kulcs objektum file-nak való megfeleltetése. Ez roppant kellemes volna, hiszen a kulcsot így ki lehetne cserélni az applet megváltoztatása nélkül is, és a kulcs így a lehető legkisebb helyet foglalná el. Sajnos, mivel a 2.1-es specifikációban file-ok nem léteznek, ez a technológia nem felel meg a jelen fejlődési iránynak.

A kártya támogat bizonyos tanusítvány-formátumokat, illetve tanusítványok file-ban való tárolását, de ezen tulajdonságaival – mint a kártyán elhelyezkedő file-okkal általában – a fent említett okokból nem foglalkoztam.

A kártya több előre definiált felhasználót tartalmaz, amelyek azonosíthatják magukat a kártya felé PIN vagy challenge and response módszerrel.

A fejlesztőeszköz

A Schlumberger „Cyberflex Access SDK 3C" névre hallgató kártyás fejlesztőeszközét használtam, amely Microsoft Windows NT alatt fut. A Schlumberger fokozatosan kezd támogatni egyéb platformokat is, például a kártyához egy ún. Linux Starter Kit ingyenesen letölthető a Schlumberger honlapjáról.

A fejlesztőeszköz számos segédprogrammal rendelkezik: Megtalálható rajta APDU-küldő, cardletfeltöltő, filekezelő, perszonalizáló, valamint egy COVE nevezetű segédprogram, amely kulcsoknak a kártyára való feltöltésére szolgál.

A fejlesztőeszköz támogatja az együttműködést több nagy szoftvergyártó cég programjaival (Netscape, Internet Explorer), lehetőséget biztosít kártyáknak ezen alkalmazások számára való perszonalizálására. A Cyberflex Access 3C telepítő CD-n létezik egy program, amely Windows 2000 bejelentkezést tesz lehetővé kártya segítségével.

A fejlesztőeszközhöz mellékeltek terminál-oldali szoftverkönyvtárakat, amelyek segítségével elérhetjük a kártyát C++ (Visual C++ 6.0) illetve Java nyelven. Választásom az utóbbira esett.

A terminál

A terminálprogramot Java (Java 1.1) nyelven készítettem, hiszen ez az egyik legköltségkímélőbb módja a többplatformos alkalmazások fejlesztésének. További jelentős előny, hogy Java nyelven elég jelentős kriptográfiai támogatás létezik, például a *java.math.BigInteger* osztály igen jól használható az RSA algoritmus megvalósítására (5.3.2. fejezet).

A Java nyelv mellett szólt az is, hogy a Schlumberger adott a Cyberflex Access SDK 3C-hez egy Java API-t, melynek segítségével a kártyát Java nyelvből magas szintű utasításokkal érhettem el. Sajnos ez a Java API a Windows egy DLL file-ján keresztül kezeli az olvasót, tehát natív kódú elemeket használ, amely a platformfüggetlenség rovására megy. Ugyanakkor a Schlumberger Linux alatt is nyújt támogatást a Cyberflex kártyákhoz, tehát feltételezem, a kártyához adott Java osztályok más platformokon is működőképesek. (Megjegyzem, Java nyelven a soros portot, ahova a kártyaolvasó csatlakozik, csakis natív kód közbeiktatásával lehet kezelni, "pure Java" módon semmiképp.)

Kliens programok

Mint azt korábban (5.2.2. fejezet) kifejtettem, nem állt szándékomban egy felhasználó összes lehetséges kliensprogramját elkészíteni. Ez – valljuk be – nem is reális lehetőség. Egyrészt, hogy a feladatnak határt szabjak, másrészt, hogy a rendszer működőképes legyen, már meglévő kliensprogramokra támaszkodtam.

Jelen rendszeremben három szolgáltatásfajta létezik: WWW, FTP és SSH. Az előbbi kettőre a Windowsban amúgy is jelen lévő Internet Explorert használtam, melynek command line paraméterként könnyen átadhatom az URL-t és a felhasználónevet, jelszót. Klienseim csupán demonstratív célt szolgálnak, nem tartottam feladatomnak a kliensek biztonságossá tételét, amely magába foglalná pl.:

- A kliens cache-ének ürítését
- Letiltani, hogy a kliens eltárolja a jelszavakat
- Meggátolni, hogy bizalmas információk (pl.: jelszó) megjelenhessenek a képernyőn a felhasználó akarata ellenére

Az SSH kliensnek a később bemutatásra kerülő MindTerm (6.4. fejezet) programot használtam, amely a svéd MindBright cég nyílt forráskódú pure Java SSH kliense [28], melynek forráskódját is módosítottam. Itt ugyanis nem kapja meg a program a felhasználó titkos

kulcsát paraméterként, hanem csupán hajlandó az SSH kliens számára a nyilvános kulcsot felmutatni, vagy a titkos kulccsal kódolni. Ez utóbbit nem adja a program ki magából, még a C (5.2.4. fejezet) azonosítási szinten sem.

5.3.2. Nehézségek

Kulcsok generálása

A fejlesztés során igen sok problémám volt az RSA kulcsokkal, illetve előállításukkal. Kulcsokat ugyanis igen sok helyen kellett használnom, ráadásul mindenütt másfajta formátumban. Próbálkoztam meglévő RSA kulcsok használatával, de sajnos saját generálóprogramot kellett írnom, hogy a kulcsokat az összes szükséges formátumban előállíthassam.

A következő helyeken volt szükségem RSA kulcsokra:

1. Kártyán kódolás, dekódolás
2. PC-n, a terminálprogramban
3. PC-n, az SSH kliensben
4. Unixos szerveren, az SSH szerver oldalon

Kártyán kódolnom, dekódolnom gyakran kellett. Szükség volt ezen műveletekre minden challenge and response azonosításkor, de fontos volt digitális aláíráskor és rejtjelezéskor is. PC-n teljesen hasonló volt a helyzet. Az SSH szerver oldalon megvolt a challenge and response azonosításhoz való nyilvános kulcs, az SSH klienssel pedig igazolnom kellett a szerver felé a titkos kulcs meglétét.

Kulcsokra az alábbi formátumokban volt szükségem:

1. Kártyán a nyilvános kulcsok tárolása egyszerű volt. Ún. modulus-exponent formában kellett megadni őket, tehát először a modulust (0x80 byte), majd a nyilvános kitevőt (kb. 3 byte). A kártya specifikációjának értelmében [37] a titkos kulcs megadható hasonlóképpen, de megadható ún. CRT (Chinese Remainder Theorem – kínai maradék tétel, [23]) formában is. Sajnos, az egyik, a kártyához adott CD-n lévő file-ból kiderült, hogy a modulus-exponent formában megadott titkos kulcsok nem működnek helyesen, tehát a CRT formára kell támaszkodni.

1. a q prímszám
2. a p prímszám (e két prímszám szorzata a modulus ($p * q = m$))
3. a q-nak p-re vett inverze ($\frac{1}{q} mod(p)$)
4. a titkos kitevő (d) $q-1$-re vett modulusa ($d\, mod(q-1)$)
5. a titkos kitevő $p-1$-re vett modulusa ($d\, mod(p-1)$)

32. ábra. Egy RSA CRT (Chinese Remainder Theorem) kulcs felépítése

A CRT formátumú kulcs szerkezete fent (32. ábra) látható. A CRT eljárás segítségével a dekódolás vagy aláírás jelentősen gyorsítható, de, mint az az ábrából kiderül, szükség van hozzá a p és q prímekre. Így – tulajdonképpen – csakis a kulcs birtokosa tudja használni [9, 17].

2. PC-n szerencsés helyzetben voltam: Mivel a szoftvert én írtam, a kulcsformátumokról saját hatáskörben tudtam dönteni. Mivel egyes helyeken modulus-exponens formában volt szükség a kulcsra, máshol pedig CRT formában, kénytelen voltam saját kulcsgeneráló programot készíteni, amely előállította a kulcsot az összes szükséges formában.

3. Az SSH kliensben tulajdonképpen nem volt szükség kódolásra, illetve ami már benne volt, azt eltávolítottam belőle, s magát a kódolást a kártya végezte. Szükség volt viszont a nyilvános kulcsra is, mert azt fel kell mutatni a szerver felé. (6.1)

4. Szerver oldalon a nyilvános kulcsra volt szükség modulus exponens formában, decimálisan.

Szerencsére a kulcsgenerálás nem volt olyan nehéz feladat, hiszen a java.math.BigInteger hatékonyan támogatja az RSA-hoz szükséges funkciókat. Emellett készítettem egy RSA kódoló-dekódoló programot is, hogy a különböző platformokon és operációs rendszereken futó RSA-k azonosságát ellenőrizhessem.

Kulcsok betáplálása

Az előzőekben a megfelelő formátumú kulcsok generálásának problémakörét ismertettem. Csakhogy nemcsak a kulcsok létrehozása okozott komoly gondot, hanem azoknak a kártyára

való felvitele is. Több lehetőség mutatkozott, és nem volt számomra egyértelmű, melyik az optimális.

A Cyberflex kártya a következő lehetőségeket biztosítja a kulcsok elérésére:

1. Rendelkezik a Cyberflex Access SDK 3C egy COVE (Cryptographical Object Viewer and Editor) nevezetű segédprogrammal, amellyel a kulcsokat bizonyos formátumokban generálhatjuk, majd a kártyára feltölthetjük. Ezután a kulcs egy speciális formátumú file-ként jön létre a kártyán, amelyből a cardletben létrehozhatunk egy kulcs-objektumot.
   Ezt a lehetőséget bizonyos 5.3.1. fejezetben leírt okok miatt elvetettem. Megjegyzem, a COVE nagyon hasznos eszköz lehet az esetben, ha kifejezetten Cyberflex vagy Cryptoflex kártyához fejlesztünk szoftvert, és a kártyának az operációs rendszer által felkínált kriptográfiai szolgáltatásait használjuk.

2. A kulcsot feltölthetem a kártyára valamilyen saját formátumú file-ként is, majd beolvashatom. A saját formátumú file kisebb helyet foglal el, mert nincsenek benne fejlécek, de ez a méretkülönbség nem jelentős. Akárcsak az előző esetben, itt is jelentős előny, hogy a kulcs nem része a cardletnek, tőle függetlenül cserélhető. Ugyanakkor rendelkezik az előző hátrányával is (5.3.1. fejezet).

3. A kulcsot a cardlet egy attribútumaként veszem fel, majd a cardlet egyik saját művelete segítségével inicializálom. Ez esetben a kulcs a cardlet részét képezi, tőle elválaszthatatlan. Így a cardlet változtatása esetén a kulcsot is újra fel kell tölteni, az új cardlet inicializálását követően. Elegáns megoldás volna, de roppant megnehezítené és költségessé tenné a fejlesztést.

4. A kulcsok a cardlet egy attribútumában tárolom, és konstanssal töltöm fel. Ez egy egyszerű, "buta" megoldás, az ún. quick and dirty filozófiát követi. Kész termék esetén feltétlenül célszerű a 3. pont megoldását alkalmazni, de a fejlesztés, tesztelés jelentősen egyszerűbben, költésgkímélőbben végezhető.

Kulcsok mérete

Mivel a fejlesztés során az előző fejezet 4. pontja mellett döntöttem, elég sok bajom akadt a kulcsok méretével. Egyrészt a titkos kulcsokat CRT formátumban kellett tárolnom, és

így méretük nagyobb lett, mintha modulus-exponent formában tároltam volna őket. Másrészt, így nemcsak a kulcsok értéke foglalt helyet a kódban, de objektumok is megjelentek, amelyek szintén tartalmazták a kulcsok méreteit.

Előzetes számításokat végeztem, és a tervezett kulcsok (5.2.6. fejezet) így is mind elfértek volna a kártyán, de kiderült, hogy a kulcsok – különösen a titkos kulcsok – az indokoltnál jóval nagyobb helyet foglalnak el. Így a program fő korlátja a tárolható kulcsok száma lett. Gyakorlati alkalmazás esetén létfontosságú volna az előző fejezet 3. pontjára áttérnem, vagy a kulcsgondozás problémakörét alaposabban körüljárni. (7.2. fejezet)

5.3.3. Mi került megvalósításra?

Kölcsönös autentikáció: A kártya és a terminál képesek egymást kölcsönösen challenge and response módszerrel, az RSA algoritmus segítségével azonosítani. Míg a kártya nem győződött meg a terminál és a felhasználó (30. ábra) megfelelő szintű azonosításáról, nem ad ki magából semmilyen információt.

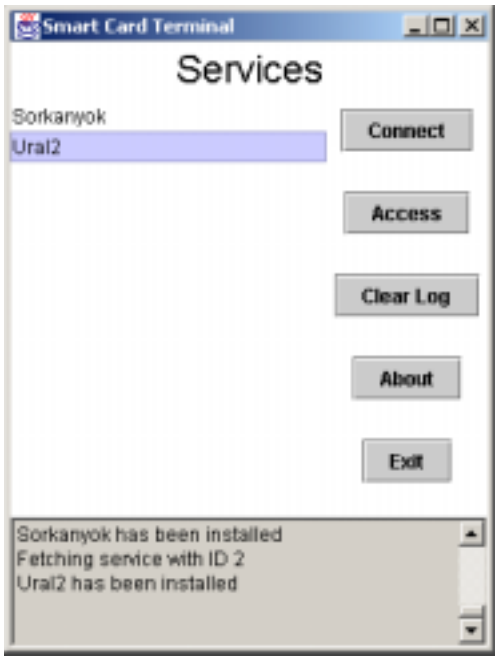
Hozzáférésvédelem: A kártyán futó szolgáltatások ellenőrzik, hogy a megfelelő szintű azonosítás megtörtént-e, mielőtt magukból bármilyen információt kiadnának.

SSH challenge and response azonosítás: Ezt is megvalósítottam a MindTerm program segítségével. Ez jelenleg önálló alkalmazásként működik, és később (6. fejezet) részletes ismertetésre kerül. Önálló blokként akár kereskedelmi alkalmazásként is üzemelhet. Sajnos, a titkos kulcsokból létrehozott objektumok (*javacardx.crypto.RSA_CRT_PrivateKey*) nagyságrendekkel több helyet foglaltak el a kártyán, mint az várható volt, így a két alkalmazást nem tudtam összeintegrálni. A probléma megoldására léteznek ötleteim (7.2. fejezet), de ennek feloldása egy kulcsgondozó keretrendszer kidolgozását igényelné a Schlumberger Java Card platformjára. Ezt a későbbiekben meg kívánom valósítani, hogy izgalmasabb kártya-oldali applikációkkal is foglalkozhassak.

Terminál: Elkészítettem egy terminálprogramot Java platformon (33. ábra), amely képes kilistázni a kártyán lévő szolgáltatásokat, majd képes róluk a megfelelő információkat lekérni, – ha a kártya odaadja neki. Ezen információk birtokában elindíthatja a megfelelő kliensprogramot.

A terminál elkészítésekor nem dédelgettem nagyratörő terveket. Nem kívántam csillogó-villogó alkalmazást létrehozni. Ellenkezőleg. Terveim közt az szerepelt, hogy

az alkalmazásom ne igényeljen többet egy mobiltelefon képernyőjén megjeleníthető felhasználói felületnél. A felhasználó egy listában válogat, majd miután kiválasztja a kívánt szolgáltatást, és megnyomja az “Access” gombot, a szolgáltatás elindul. Terminálomat Windows 2000 platformon teszteltem, de – Java alkalmazás lévén – egyéb platformokon is működőképes. Egyetlen kivétel a kártyát kezelő rész, amely – a Schlumberger által közölt információk szerint – egy Windows DLL-t használ. Ugyanakkor a kártya elérhető Linux alatt is, tehát legfeljebb ezen részeket kell lecserélni.



33. ábra. Termiálprogramom kilistázza a kártyán lévő szolgáltatásokat

6. SSH autentikáció chipkártyával

Ez az alkalmazás elvileg az 5. fejezet részét képezi, mégis külön fejezetbe került. Ennek egyik fő oka, hogy ez az alkalmazás önálló jelentőséggel bír, és kisebb módosításokkal, valamint jogi körültekintéssel (6.4. fejezet) akár kereskedelmi használatra is alkalmas lehet. A külön kezelés másik fő oka, hogy bizonyos problémák merültek fel a két alkalmazás kártyán való összeillesztésekor (5.3.2. fejezet), így jelenleg két független cardletet képeznek.

6.1. Mi az SSH?

Az SSH protokoll arra szolgál, hogy a felhasználók távoli rendszerekbe biztonságosan bejelentkezhessenek. A kliens és a szerver nyilvános kulcsú kriptográfiát használnak egy közös titkos kulcs kiválasztására. Ezután a kommunikáció ezen titkos kulcs segítségével folytatódik. [48]

A protokoll első változatát Tatu Ylönen, finn programozó fejlesztette ki, ma a protocoll az SSH Communications Security kezében van. Az SSH Unix rendszerekben de facto szabvánnyá vált a távoli rendszeradminisztrációra, de létezik Microsoft Windows platformra is. Nemrég biztonsági hibára derült fény az SSH protokollban, és a SSH Communications új protokollt bocsájtott ki, SSH2 vagy SECSH néven. [42] Ezen dolgozat az SSH RSA challenge and response azonosítási rendszerével foglalkozik, annak többi részével nem. Így a fenti változás a dolgozatot – közvetlenül – nem érinti.

Igaz, az SSH a secure shell rövidítése, de a protokoll ma már jóval többet nyújt egy egyszerű shell-nél. Az SSH segítségével lehetséges file-okat mozgatni (SCP), de képes XWindow titkosítására is. Sőt, lehetséges az SSH segítségével ún. tunnel-t kialakítani, melyen keresztül bármilyen TCP/IP kapcsolat átvezethető. Így SSH segítségével védhetünk HTTP, FTP, POP3 vagy akár VNC kapcsolatokat is.

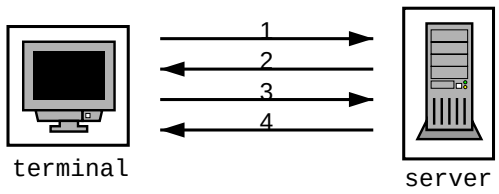
A továbbiakban az SSH protokoll illetve kapcsolatfelépítés nem kerül részletes bemutatásra [47], pusztán az SSH challenge and response felhasználó-azonosításról lesz szó.

6.2. SSH challenge and response azonosítás

6.2.1. Hogyan történik?

Az SSH többféle lehetőséget kínál a felhasználó azonosítására. A legegyszerűbb ezek közül a jelszó (password). Ez egy viszonylag gyenge módszer, hiszen egyrészt rövid, másrészt érzékeny arra, hogyha a felhasználó „gyenge" jelszót választ magának. Az SSH biztosít még lehetőséget rhosts és Kerberos alapú azonosításokra is, de számunkra legérdekesebb a challenge and response (34. ábra, alg:39. ábra):

1. A kliens megkéri a szervert, hogy szeretné azonosítani magát
2. A szerver küld egy véletlen kihívást
3. A kliens a titkos kulcsa segítségével kódolja a kihívást, majd elküldi a szervernek
4. A szerver a kliens nyilvános kulcsa segítségével visszakódolja a kihívást, majd megnézi, azt kapta-e, amit ő küldött a kliensnek a 2. lépésben. Ezután beengedi vagy elutasítja a klienst.



34. ábra. SSH challenge and response azonosítás

6.2.2. Miért jó ez a módszer?

- Az azonosítás nyilvános csatornán történik
- A csatornán nem jelenik meg olyan információ, amelyet egy támadó később sikerrel használhatna fel

Ez azért lehetséges, mert a kliens nem adja ki magából titkos kulcsát. A csatornán csak t és $E_{t,kliens}\{t\}$ jelenik meg, ahol t friss elem (pl. véletlen szám). Nagyon kicsi a valószínűsége

annak, hogy kétszer ugyanaz a kihívás érkezik, és így a válasz ugyanaz lehet. Ha egy támadó fel kíván venni minden kihívást és választ, hatalmas tárkapacitásra lenne szüksége: Ha 1024 bites RSA-ról van szó, összesen 2^{1024} db kihívás és ugyanennyi ehhez tartozó válasz lehetséges. Ennek tárolása – minden józan elképzelés szerint – lehetetlen. A challenge and response módszer segítségével maga a kulcs nem kerül ki a nyilvános csatornára, a kliens csupán igazolja a szerver felé, hogy rendelkezik a vele.

A challenge and response leggyakrabban – így az SSH-ban is – RSA segítségével zajlik le. Tehát a kliens rendelkezik egy titkos RSA kulccsal, s a nyilvános kulcsa meg kell, hogy legyen a szerverben. (Az SSH-1 még nem használ tanusítványokat.)

6.2.3. További védelem

Térjünk most vissza a 11. ábrán vázolt modellünkhöz, melynek releváns része a 34. ábrán látható. A felhasználó a terminál mellett ül, s így kíván a szerverre bejelentkezni. A szerver által a 2. pontban elküldött kihívásra a terminál válaszol, amely tartalmazza a felhasználó titkos kulcsát.

Bár, a challenge and response módszer tekinthető az SSH legbiztonságosabb azonosítási módszerének, jelentős veszélyt hordoz, hogy ha egy támadó betör a terminálra, megkaparinthatja a felhasználó titkos kulcsát, és visszaélhet azzal. Így további védelem szükséges.

Az SSH kliensek általában úgynevezett passphrase-ket használnak a kulcs további védelmére. A felhasználó titkos kulcsát kódolva tárolják, melynek visszaállításához a passphrase szükséges. A passphrase egy vagy több szó, amelyeket a felhasználó akár könnyen fejben tarthat.

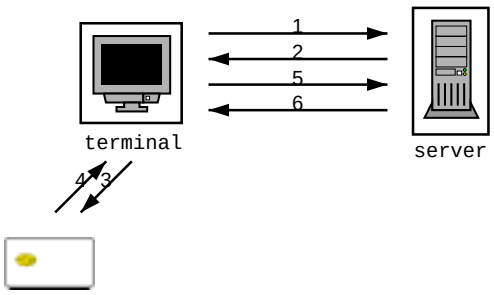
Jelszavak esetén általában „gyengének” nevezzük azokat, amelyek értelmes szavak, és jelentéssel bírnak. Ezek a szavak kis részét képezik az összes lehetséges jelszó halmazának, és a támadó elég könnyen végigpróbálgathatja őket.

A passphrase abban különbözik a jelszótól (password), hogy akár hosszú is lehet. Több szóból állhat, de lehet akár egy verssor vagy egy versszak is. Általában egy lenyomat készül belőle, és ezt használják a kódolt titkos kulcs visszaállítására. A passphrase-t a felhasználónak fejben kell tartania, és be kell gépelnie, ha a titkos kulcsát használni kívánja. A felhasználó azzal a feltételezéssel él, hogy begépelt passhprase-ét a terminál nem rögzíti, használat után elfelejti.

Sajnos ez a feltételezés nem mindig helytálló. Lehet, hogy a rendszergazda rosszindulatú, és összegyűjti a felhasználók passphrase-eit. Az is lehet, hogy egy cracker betört a rendszerbe, és módosította az SSH kliens programot. De az is lehetséges, hogy a rendszer egyszerűen hibás, és a titkos kulcsot ki lehet nyerni belőle valamilyen más módon.

Sajnos, az összes azonosítási mód, amelyet az SSH használ, tudás alapú. (B függelék) A következőkben bemutatásra kerül egy tulajdon alapú azonosítást használó SSH bejelentkezési módszer. Célom az volt, hogy a chipkártya segítségével további védelmet biztosítsak a kulcs számára függetlenné téve azt a konkrét terminálról. Így a tudás alapú azonosítást – részben – tulajdon alapú azonosítássá alakítottam át.

6.3. SSH challenge and response azonosítás chipkártya segítségével

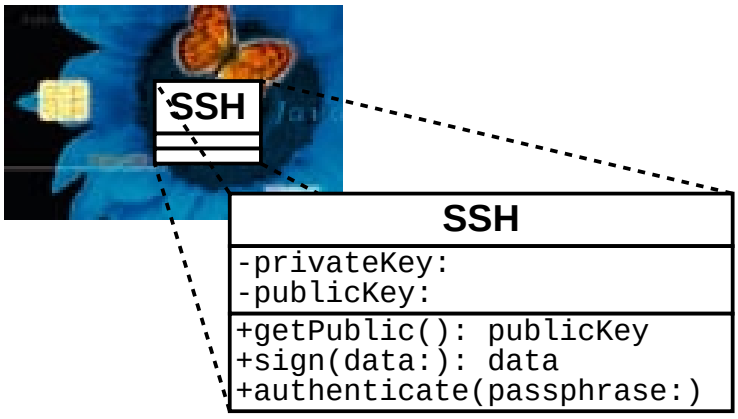


35. ábra. SSH challenge and response azonosítás chipkártya segítségével

Célom tehát az volt, hogy a kulcsot valamilyen módon a kártyához kapcsoljam, ezáltal megnehezítve annak elérését. Amennyiben a kulcs a kártyán foglal helyet, de magát a kódolást a terminál végzi el, nem jutottunk messzebbre, mint az előbb felvázolt módszerek (6.2.3. fejezet). Ha a terminál kiolvashatja a kulcsot a kártyából, és elvégzi a kódolást, továbbra is védtelenek maradunk a csaló, hibás vagy feltört terminálok ellen, amelyek a kulcsot utána nem pusztítják el teljesen.

Jelentőset akkor javíthatunk a rendszeren, hogyha a kulcsot a chipkártya nem adja ki magából soha, semmilyen körülmények között. Hozhatunk erősebb megszorítást is: ne is létezzen olyan művelet, amellyel a titkos kulcsot a kártyából kiolvashatnánk (3.3.2. fejezet). Így egy támadó bármilyen jogosultságot, jelszót szerez is meg a kártyához, a kulcshoz hozzáférni nem tud. A kulcsot csak írni lehessen, olvasni pedig nem.

Ebből természetesen adódik, hogy a kódolást a kártyának kell elvégeznie. Vegyünk tehát egy olyan kártyát, amely képes 1024 bites RSA kódolást legfeljebb néhány másodperc alatt végrehajtani! Erre a célra megfelel a Cyberflex Access kártya, melyet már korábban (5.3.1. fejezet) bemutattam.



36. ábra. SSH cardlet a Cyberflex kártyán

A kártya rendelkezzen egy SSH kulcspárral! Nézzük meg, milyen műveleteket definiáltam a kulcspáron:

- Mivel az azonosításkor a publikus kulcsot fel kell mutatni a szerver felé, biztosítsunk egy metódust, amellyel az kiolvasható.
- Hogy felelni tudjunk az SSH-szerver kihívására, legyen egy metódus, amely a titkos kulcsot alkalmazza a kihívásra.
- A felhasználónak bizonyítania kell a kártya felé személyazonosságát.

Ennek ismeretében vizsgáljuk meg, hogyan játszódhat le a korábban (6.2.1. fejezet) felvázolt challenge and response azonosítás a kártya segítségével (35. ábra)!

1. A kliens megkéri a szervert, hogy szeretné azonosítani magát (ugyanaz)
2. A szerver küld egy véletlen kihívást (ugyanaz)
3. Itt látható az első változás. Míg a 6.2.1. fejezet terminálja birtokában van a titkos kulcsnak, az itt leírt terminál nem. (Így garantálható, hogy a terminál később nem élhet vissza a felhasználó kulcsával.) A terminálnak a kártyához kell fordulnia, ugyanis

csak az rendelkezik a titkos kulccsal, csak az tud válaszolni a kihívásra. Előtte – természetesen – gondoskodnia kell a felhasználó azonosításáról is.

4. A kártya a titkos kulcsa segítségével kódolja a kihívást, majd elküldi a kliensnek.

5. Ez a lépés ismét megegyezik a 6.2.1. fejezet 3. lépésével. A kliens elküldi a választ a szervernek. Igaz, itt a kliens csupán a kártya válaszát továbbítja.

6. A szerver a kliens nyilvános kulcsa segítségével visszakódolja a kihívást, majd megnézi, azt kapta-e, amit ő küldött a kliensnek a 2. lépésben. Ezután beengedi vagy elutasítja a klienst.

Az ábrából (6.3. ábra) és a fenti leírásból látszik, hogy a kliens az azonosítás során pusztán átjátszóként működik. Amit a szervertől kap, a kártya felé továbbítja, amit a kártya visszaad, azt a szervernek küldi el. A kliens az azonosítás után válik terminállá. Ezután a felhasználó által begépelt karaktereket elküldi a szervernek.

Ha a kliens rosszindulatú, továbbra is felvehet mindent, amit a felhasználó végez a szerveren. Esetleg módosíthatja is a felhasználó tevékenységét, bár ennek hatékony kihasználásához egy egyszerű terminálénál sokkal több intelligenciára volna szükség. Ez ellen a chipkártytechnológia nemcsak ma nem tud, de a későbbiekben sem lesz képes védelmet biztosítani.

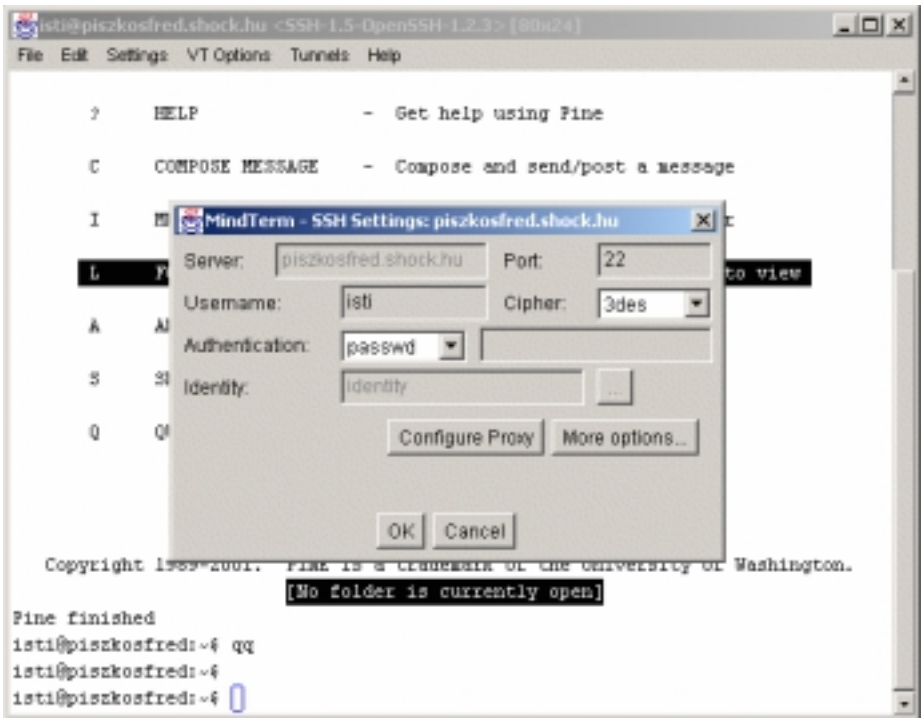
Az ellen viszont hatékony védelmet biztosít, hogy a terminál felvegye az azonosításhoz szükséges kulcsot, hogy azt később egy crackernek odaadja. Nincs ugyanis lehetőség arra, hogy a titkos kulcs a kártyából kikerülhessen, és a terminálba juthasson, holott ez a kulcs szükséges a kihívás megválaszolásához.

6.4. MindTerm - SSH kliens

Mint említettem, az SSH azonosítás implementálásához a Schlumberger termékét, a Cyberflex Access kártyát használtam. Saját SSH kliens kifejlesztését nem tűztem ki célul, hiszen ez jócskán meghaladta volna egyetlen személy lehetőségeit. Így a svéd MindBright cég MindTerm [28] nevű SSH kliens implementációját használtam. (37. ábra)

Azért esett választásom a MindTerm-re, mert:

- Nyílt forráskóddal rendelkezik



37. ábra. Mindterm

- Magánhasználatra ingyenes
- Teljes egészében Java nyelven készült, vagyis platformfüggetlen

A MindTerm-ben az azonosítást végző osztályokat módosítottam, lecseréltem, így értem el, hogy létezzen olyan azonosítási módszer is, amely a chipkártyát használja a kihívás megválaszolására.

Összefoglalás

7. Tervek, továbbfejlesztési lehetőségek

7.1. Elliptikus görbékre alapuló kriptográfia Java Card környezetben

Az elliptikus görbéken alapuló nyilvános kulcsú kriptográfia fiatal, kevéssé ismert elméletnek számít. Igaz, alapelvei lassan húsz éve ismertek, de - főként bonyolult matematikája miatt - sokkal kisebb elterjedtségnek örvend, mint az RSA. Ugyanakkor, mivel számos előnnyel rendelkezik nagy vetélytársával szemben, szerepe jelentősen megnövekedett, s ma már igen sok szabványban szerepel az ECC is az RSA mellett.

Programomat elsősorban demonstratív céllal írtam meg. Többek közt fel kívántam hívni vele a figyelmet a kis méretű kulcsokkal nagy biztonságot nyújtó ECC algoritmusok chipkártyán való használatára. Programom illeszkedik az ECC matematikai struktúrájához, tehát a Galois-test aritmetikája, és a pontok csoportjának aritmetikája elválasztható egymástól, s bármikor kicserélhető. Demonstrálni kívántam továbbá, hogy a mai (illetve néhány évvel ezelőtti) chipkártyák már képesek ECC műveletek végrehajtására is, bár a sebességük jelenleg igen csekély.

Meg kell jegyeznem, hogy magasszintű, tehát nem hardverközeli, nyelven dolgoztam, mindennemű hardvertámogatás nélkül. Nem tartom kizártnak, hogy chipkártyán célhardverrel való (vagy legalábbis assembly nyelven írt) megvalósítás sebessége igen sokszorosa lehet az enyémnek, még a mai technológia adta lehetőségek közt is.

Figyelembevéve a hardverelemek és a chipkártyák rohamos fejlődését (pl amelyet hatásosan demonstrál a Moore törvény is) igen valószínűnek tartom, hogy a néhány éven belül megjelenő jelentősen gyorsabb kártyákon sokkal gyorsabban fog futni a mi imlementációnk is. [5] Memória szempontjából pedig már ma is léteznek olyan Java Card chipkártyák, amelyek nagy méretű memóriájukból (32k-64k) adódóan számos gyorsítási lehetőség előtt nyitnak teret.

7.2. Felhasználói profile tárolása intelligens kártyán

A diplomatervem keretében készült szoftvert befejeztem. Ugyanakkor mindannyian tudjuk, hogy egy szoftver soha nincsen kész. Számos további ötletem, elképzelésem van új és új funkciók, lehetőségek hozzáadására. Ezek közül néhány fontosabbat ismertetek az alábbiakban.

Kulcsgondozás körüljárása: Legkomolyabb akadály a kulcsok tárolásának és generálásának problémája volt. Szándékomban áll ezen témakört alaposabban körüljárni, hogy későbbi fejlesztések esetén ez ne jelentsen komoly akadályt.
A következő szempontok merültek fel a kulcsgondozás kapcsán:

1. Minél kisebb helyet foglaljanak el a kártyán – mint kiderült, ez egy igen fontos korlát.
2. Minél egyszerűbb legyen őket feltölteni, cserélni, különösen a tesztelés során.
3. Lehetőleg ne kötődjenek (közvetlenül) a cardlethez! Ez nemcsak azt jelenti, hogy a cardletet a kulcsok letörlése nélkül lehessen cserélni, de azt is, hogy a kulcsokat is lehessen cserélni a cardlet letörlése nélkül.

Felmerült lehetőségek:

1. Külön cardlet, együttműködő cardletek – Ez esetben létezne a kártyán egy kulcsszerver cardlet, és amennyiben egy cardletnek kulcsra van szüksége, elkéri tőle. Ebben a cardletben lehetőséget biztosítanék a kulcsok feltöltésére, cseréjére, így nem kellene a programkódhoz nyúlni, ha egy kulcsot változtatni kell.
2. Kulcsok file-ban való tárolása – Ez a lehetőség behódolást jelentene a Schlumberger felé. Elveszíteném a platformfüggetlenséget, de jelentősen egyszerűbb lenne a kulcsokat cserélni – nem kötődnének a cardlethez. Mivel a file-ok is a kártya EEPROM-jában foglalnak helyet, memóriát ezzel nem takarítanék meg. Ugyanakkor egyszerűbb volna a szoftvert karbantartani.
3. „Csak egy maradhat!” – Csak egy kulcs objektumot hozok létre, és használat előtt mindig beletölteném a szükséges kulcs értékét. Ez lassítaná a rendszert, viszont óriási memóriamegtakarítást eredményezhetne. A karbantartást viszont nem könnyítené meg.

E harmadik lehetőség tűnik a legvalószínűbbnek, várhatóan ezt fogom megvalósítani.

Alkalmazások és szolgáltatások összekapcsolása: Jelenleg a terminálprogram „tudja", hogy milyen szolgáltatás milyen kliensprogramot igényel. Többplatformos rendszerben ez nyilván nem működhet. XML[46] adatbázist kívántam a Java terminálhoz kapcsolni, amely minden egyes terminéltípuson más és más volna. Ez adná a terminál absztrakt leírását, lásd: 26. ábra.

További szolgáltatások megvalósítása: Számos új ötletem van további szolgáltatásokra, amelyekkel nagyszerűen kihasználhatnám a chipkártyák lehetőségeit:

- SMB protokoll: jelszavas változata alig különbözik az FTP-től, legalábbis a kártya számára. Megvalósítható lenne az Internet Explorer segítségével. Új lehetőséget nem demonstrálnék vele, ez az egyetlen ok, hogy még mindig nem készült el.
- SSL megvalósítása: Az SSL és az SSH nagyon hasonlítanak egymáshoz, és egyformán elterjedtek, csak másra használják őket. Miért ne lehetne az SSL-t is megvalósítani?
- Valamely – lehetőleg titkosított – levelezési protokoll támogatása. Lehetséges variáció pl. a POP3 protokoll SSH port forwardingon keresztül.
- Felhasználó address book-jának, távoli mailboxának elérése
- Digitális aláírás a kártya segítségével – létezik már ilyen alkalmazásom, mindössze bele kell integrálnom a felhasználói profilt tároló alkalmazással. Sajnos, ez új kulcsok felvitelét jelentené, így feltétlenül szükséges, hogy előtte a kulcsgondozás problémáját megoldjam.

A szolgáltatások hierarchikus struktúrába való foglalása: Jelenleg a felhasználó egy egyszerű listából választhat szolgáltatásokat. Később, ha a lista hosszabb lesz, lehetőséget kell biztosítani arra, hogy a felhasználó (vagy a szolgáltatásokat telepítő) a szolgáltatásokat kategóriákba foglalja.
Legegyszerűbb valamilyen hierarchikus fastruktúrára (pl.: könyvtárrendszer) gondolni. Ez – nagy mennyiségű szolgáltatás esetén – jelentősen megnövelné a felhasználóbarátságot.

Egyéb terminéltípusok elkészítése: Különböző operációs rendszerek alatt különböző szokások léteznek. Van, ahol ikonokat szoktak a felhasználók látni, van, ahol listákat,

van, ahol három dimenzióban szeretnek közlekedni. Mindegyik alá készíthető saját terminál, hogy a felhasználó otthon érezze magát e környezetben.

7.3. SSH autentikáció chipkártyával

Az alkalmazás jelenleg két azonosítási fajtát használ:

- A chipkártya tulajdon alapú azonosítást jelent. (Pontosabban ez visszavezethető egy tudás alapú azonosításra. A kártya rendelkezik a tudással – a kulccsal – , amelyet azonosítani szeretnénk. Az, hogy a kulcs a kártyából kivehetetlen, teszi ezt a kártya tudás alapú azonosítását a felhasználó tulajdon alapú azonosításává.)
- A chipkártya csakis jelszó felmutatása esetén hajlandó elvégezni a kívánt kódolásokat. Ez a felhasználó tudás alapú azonosítását jelenti.

Ugyanakkor, ha jobban megvizsgáljuk, észrevehetjük, hogy a két azonosítás egymástól nem teljesen független. Ha egy támadó feltöri a kártyát, és hozzáfér annak tartalmához, elérheti a kulcsot a jelszó ismeret nélkül is. Növelni lehetne a biztonságot, ha a kártya nem tartalmazná a titkos kulcsot, hanem annak csak egy kódolt változatával rendelkezne; a jelszó pedig nem csupán a titkos kulcshoz való hozzáférést engedélyezné, hanem annak kódolt állapotból való visszaállításához lenne szükséges.

Megérné mérlegelni, mely alkalmazásokhoz lenne szükség a fenti biztonsági mechanizmusra. A chipkártya igen erős védelmet biztosít az adatok számára, és a titkos kulcs megszerzéséhez nem elengendő a kártya ellopása, hanem annak biztonsági rendszerét if fel kel törni. Úgy vélem, ez a módszer elégséges védelmet jelent kereskedelmi alkalmazások esetén, további védekezési mechanizmusokra főként csak ún. hárombetűs szervezetek ellenében volna szükség.

8. Összefoglalás

E diplomaterv első részében részletesen áttekintettem a programozható chipkártyák főbb alkalmazási lehetőségeit. Részletesen bemutattam a Java Card technológiát, amely az egyik legerősebb programozási környezet chipkártya alapú fejlesztésekre. Megkíséreltem átölelni a Java Card platform adatbiztonságának nagy, ellenben kevéssé ismert területét.

Munkám második felében három saját készítésű alkalmazást mutattam be, amelyek Java kártyákra készültek. Az első, egy prototípus, amelyben az elliptikus görbékre alapuló kriptográfia chipkártyás lehetőségeit vizsgáltam meg. Ez az alkalmazás nem tud többet egy – később majd létező – ECC-célprocesszorral ellátott kártyánál, de – mivel az említett eszköz még nem létezik, – alkalmazásommal utat török ezen új algoritmus chipkártyás megvalósítása számára.

Második példaalkalmazásom egy rafinált program a Java Card platformon. Komplex protokollokat implementáltam, bonyolult védelmi mechanizmusokat készítettem, melyek alkalmasak lehetnek mind a kártyabirtokos, mind a kártyakibocsájtó érdekeinek markáns képviseletére. A szoftver speciális célt szolgál, és olyan jellegzetességekkel bír, amelyek miatt csakis programozható kártyán érdemes megvalósítani.

Harmadik alkalmazásom nem új lehetőséget akar felmutatni, s nem új területekre kíván betörni. Rafinált védelmi mechanizmusokat sem tartalmaz. Egyszerűen egy nagyon erős és gyakorlatban is nagyon jól használható, nagy biztonságot nyújtó alkalmazásról van szó, amelyet magasszintű programozási nyelven, ezen új, huszonegyedik századi technológiával készítettem.

Köszönetnyilvánítás

Szeretnék köszönetet mondani témavezetőmnek, Vajda István tanár úrnak az iránymutatásért és a rendszeres konzultációkért. Köszönetet mondok Mann Zoltán Ádámnak, diplomatervezést megelőző tudományos diákköri munkában állandó társam volt. Hálás vagyok Tihanyi Sándornak a LaTex magyarításért, továbbá szeretnék köszönetet mondani Bencsáth Boldizsárnak és az E-Bizlabban dolgozó csapat többi tagjának az ott megteremtett kreatív légkörért.

9. Ábrák, táblázatok, hivatkozások

9.1. Ábrák jegyzéke

9.2. Hivatkozások jegyzéke

[1] Bankkártya NetInfo. Átlépte a 40 milliós határt a Visa chipkártyák száma. `http://www.bankkartya.hu/cikk.cgi?id=00489`.

[2] T. Bennó. Általános célú törzsvásárlói (loyalty) rendszer. Master's thesis, BME, HIT, 1999.

[3] I. Berta and Z. Mann. Elliptikus görbéken alapuló nyilvános kulcsú kriptográfia elemzése chipkártyás és PC-s környezetben. Scientific student circle conference, Budapest University of Technology and Economics, 2000.

[4] I. Berta and Z. Mann. Programozható chipkártyák - elmélet és gyakorlati tapasztalatok. Magyar Távközlés, 2000, vol 4, 2000.

[5] I. Berta and Z. Mann. Smart Cards – Present and Future. Híradástechnika, Journal on C^5, 2000., vol 12, 2000.

[6] I. Berta and Z. Mann. Evaluating Elliptic Curve Cryptography on PC and Smart Card. Periodica Polytechnica, Budapest University of Technology and Economics, 4 2001. (bírálás alatt).

[7] I. Berta and Z. Mann. A hitelesség biztosításának lehetőségei intelligens smart card segítségével. Scientific student circle conference, Budapest University of Technology and Economics, November, 1999.

[8] P. Bieber, J. Cazin, V. Wiels, G. Zanon, P. Girard, and J.-L. Lanet. Electronic purse applet certification, February 2000.

[9] L. Brown. Cryptography and Computer Security - Lectures, 2001. `http://www.cs.adfa.oz.au/teaching/studinfo/csc/lectures/`.

[10] Certicom. Certicom crypto challenge sets new security benchmark for ECC, 9 1999. `http://www.certicom.com/news/99/sep2899.html`.

[11] Certicom. Robert Harley and Team Win 10,000 USD Prize in Certicom's ECC Challenge, April 2000.

[12] Z. Chen. How to write a Java Card applet: A developer's guide. JavaWorld, July, 1999, 1999. http://www.javaworld.com/javaworld/jw-07-1999/jw-07-javacard_p.html.

[13] Z. Chen. Java CardTM Technology for Smart Card: Architecture and Programmer's Guide. Addison-Wesley Pub Co; ISBN: 0201703297, 9 2000.

[14] D. A. Crutchley. Cryptography and Elliptic Curves, 1999.

[15] Dallas Semiconductor. iButton Home Page, 5 2001. http://www.ibutton.com/.

[16] H. Dreifus and J. Monk. Smart cards: guide to building and managing smart card applications. John Wiley and Sons, Inc. ISBN 0-471-15748-1, 1998.

[17] T. Elo. A Software Implementation of ECDSA on a Java Smart Card. Master's Thesis, Department of Computer Science and Engineering, Helsinki University of Technology, 4 2000. http://www.hut.fi/~telo/publications/.

[18] T. Elo and P. Nikander. Decentralized Authorization with ECDSA on a Java Smart Card – A Software Implementation, 2000. http://www.hut.fi/~telo/publications/.

[19] L. Györfi, S. Győri, and I. Vajda. Információ és kódelmélet. Typotex, 2000.

[20] D. Husemöller. Elliptic Curves. Springer, 1987.

[21] ISO. ISO/IEC 7816 Part 4: Interindustry command for interchange.

[22] G. Katona, A. Recski, and C. Szabó. Gráfelmélet, algoritmuselmélet és algebra. BME, 1997.

[23] N. Koblitz. A Course in Number Theory and Cryptography. Springer-Verlag, 1987, 2nd edition, 1987.

[24] N. Koblitz. Elliptic Curve Cryptosystems. Mathematics of Computation, 1996, 2nd Edition, John Wiley & Sons, 1996.

[25] A. Macaire. An Open Terminal Infrastructure for Personal Services. TOOLS Europe 2000, Le Mont-St-Michel (France), 2000.

[26] A. Menezes, T. Okamoto, and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. IEEE Transactions on Information Theory, 39 (1993), 1639-1646., 1993.

[27] V. S. Miller. Use of Elliptic Curves in Cryptography. CRYPTO '85, volume 12, 1985.

[28] MindBright Technology. MindTerm, the java secure shell client, 5 2001. `http://www.mindbright.com/products/mindterm/`.

[29] S. Motré. Formal Model and Implementation of the Java Card Dynamic Security Policy. Gemplus Research Laboratory Avenue du Pic de Bertagne 13881 Gémenos CEDEX, 2000.

[30] Mullin, Onyszchuk, Vanstone, and Wilson. Optimal Normal Bases in $GF(p^m)$. Discrete Applied Math, 1988, vol 12, 1988.

[31] MULTOS. MULTOS – The multi application operating system for smart cards, 9 1999. `http://www.multos.com`.

[32] Object Management Group. UML Resource Page, 5 2001. `http://www.omg.org/technology/uml/`.

[33] M. Pellegrini, O. Potonniée, and R. Marvie. Smart Cards: A System Support for Service Accessibility from Heterogeneous Devices. 9th ACM SIGOPS Eurepean Workshop, Kolding, Denmark, 2000.

[34] W. Rankl and W. Effing. Smart Card Handbook. John Wiley & Sons, 2nd edition, 1997.

[35] A. Requet. A B Model for Ensuring Soundness of a Large Subset of the Java Card Virtual Machine. Gemplus Research Laboratory, Av du Pic de Bertagne, 13881 Gémenos cedex BP 100.

[36] L. Rónyai. Elliptikus görbék és a Fermat-sejtés. Matematikai Lapok, 1995.

[37] Schlumberger. Cyberflex Access Software Development Kit – Programmer's Manual Release 2, 1999.

[38] Schlumberger. Cyberflex Access Home Page, 3 2001. `http://www.cyberflex.com`.

[39] B. Schneier. Applied Cryptography, 1996.

[40] B. Schneier and A. Shostack. Breaking up is Hard to do: Modelling security threats for smart cards. http://www.counterpane.com/smart-card-threats.html.

[41] ScreamingMedia. Motorola Ships Industry's First 32-Bit RISC Java Card 2.1 Technology/Visa Open Platform 2.0 Card. ScreamingMedia, Business Wire, 3 2000. http://industry.java.sun.com/javanews/stories/story2/0,1072,24197,00.ht%ml.

[42] SSH Communications Inc. SSH statement regarding the vulnerability of SSH1 protocol. SSH Home Page, 5 2001. http://www.ssh.com/products/ssh/cert/.

[43] Sun Microsystems. *Java Card 2.0 Language Subset and Virtual Machine Specification*, 1997.

[44] Sun Microsystems. Java(TM) 2 Platform, Standard Edition, v 1.3 API Specification, 5 2001. http://java.sun.com/j2se/1.3/docs/api/index.html.

[45] Sun Microsystems Inc. Java Card (TM) 2.1.1 Application Programming Interface. Sun Microsystems, Inc. 901 San Antonio Road Palo Alto, CA 94303 USA 650 960-1300, 5 2000. http://java.sun.com/javacard.

[46] World Wide Web Consortium. Extensible Markup Language (XML), 1998. http://www.w3.org/XML.

[47] T. Ylönen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. Ssh authentication protocol. Internet Engineering Task Force, Network Working Group, draft-ietf-secsh-userauth-09.txt, 1 2000. http://www.ietf.org/internet-drafts/draft-ietf-secsh-userauth-09.txt.

[48] T. Ylönen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH Protocol Architecture. Internet Engineering Task Force, Network Working Group, draft-ietf-secsh-architecture-07.txt, 1 2001. http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-07.tx%t.

[49] J. L. Zoreda and J. M. Oton. Smart cards. Artech House, 1994.

9.3. A szerző témához kapcsolódó publikációi

- Berta István Zsolt – Mann Zoltán Ádám: *A hitelesség biztosításának lehetőségei intelligens smart card segítségével.* TDK dolgozat, BME, 1999.
- Berta István Zsolt – Mann Zoltán Ádám: *Programozható chipkártyák – elmélet és gyakorlati tapasztalatok.* Magyar Távközlés, 2000. április.
- Berta István Zsolt – Mann Zoltán Ádám: *Smart Cards – Present and Future.* Híradástechnika, Journal on C^5, 2000, december.
- Berta István Zsolt – Mann Zoltán Ádám: *Elliptikus görbéken alapuló nyilvános kulcsú kriptográfia elemzése chipkártyás és PC-s környezetben.* TDK dolgozat, BME, 2000.
- Berta István Zsolt – Mann Zoltán Ádám: *Evaluating Elliptic Curve Cryptography on PC and Smart Card. Periodica Polytechnica*, 2001. Bírálása alatt.
- Berta István Zsolt – Mann Zoltán Ádám: *Programozható chipkártyák kriptográfiai alkalmazása.* Networkshop2001, Sopron, Nyugat-Magyarországi Egyetem.

9.4. Táblázatok jegyzéke

Függelék

A. ECC protokollok

Vizsgáljunk meg néhány példát arra, hogy hogyan lehet az ECDLP-t konkrét kriptográfiai prokollokban használni!

A.1. A Diffie-Hellman protokoll ECC-s változata

Kiindulás:	Ismert E elliptikus görbe és annak egy P pontja.
A:	generál egy k_A véletlen számot
B:	generál egy k_B véletlen számot
$A \to B$*:*	$P * k_A$ (ez A nyilvános kulcsa)
$A \to B$*:*	$P * k_B$ (ez B nyilvános kulcsa)
A:	kiszámítja $(P * k_B) * k_A$-t
B:	kiszámítja $(P * k_B) * k_A$-t
Közös kulcs:	$P * k_A * k_B$

38. ábra. Diffie-Hellman protokoll ECC-vel – A és B kulcsot cserélnek

Vegyük például a Diffie-Hellman [19] protokoll elliptikus görbéken alapuló változatát. (38. ábra) Tegyük fel, hogy E egy $GF(q)$ feletti elliptikus görbe, és P ennek egy pontja. E és P mindenki által ismert. Minden szereplő választ magának egy titkos k kulcsot, mely egy pozitív egész szám. Ezek után mindenki kiszámítja kP-t, és ezt közzéteszi, mint nyilvános kulcsot. Tegyük fel, hogy két szereplő, A és B üzeneteket akarnak váltani valamilyen szimmetrikus kriptográfiai rendszerrel, és ehhez szükségük van egy közös kulcsra, melyet csak ők ketten ismernek. Ha titkos kulcsaik k_A és k_B, akkor $k_A k_B P$ pont megfelel. Ugyanis

A ezt úgy tudja meghatározni, hogy B nyilvános kulcsát, k_BP-t, megszorozza k_A-val, hasonlóan **B** úgy határozza meg, hogy A nyilvános kulcsát, k_AP-t megszorozza k_B-vel. Viszont k_A és k_B ismerete nélkül senki más nem tudja kiszámítani ezt, hacsak nem oldja meg az ECDLP-t.

A.2. Az ElGammal protokoll ECC-s változata

Kiindulás:	Ismert E elliptikus görbe és annak egy P pontja. A el akarja küldeni B-nek M üzenetet.
A:	generál egy k_A véletlen számot
B:	generál egy k_B véletlen számot
$B \rightarrow A$*:*	$P * k_B$ (ez B nyilvános kulcsa)
A:	generál egy l véletlen számot
$A \rightarrow B$*:*	$l * P$ és $M + l(k_B * P)$
B:	kiszámolja $(l * P) * k_B$-t, majd ezt kivonja $M + l(k_B * P)$-ből.
B megkapta:	M

39. ábra. ElGammal protokoll ECC-vel. Titkos üzenettovábbítás

Nézzük most az ElGamal protokoll elliptikus görbéken alapuló változatát. (39. ábra) A kiindulás ugyanaz, mint az előbb, de most tegyük fel, hogy A akar küldeni egy M üzenetet B-nek. Feltesszük, hogy valamilyen módon az üzenet az E görbe pontjaként van elkódolva. Ekkor A választ egy tetszőleges l természetes számot, és elküldi B-nek az lP és $M+l(k_BP)$ pontokat. Ebből l és k_B ismerete nélkül nem lehet visszanyerni az M üzenetet. Viszont B a kapott első pontot megszorozza k_B-vel, és ezt kivonva a második pontból épp M-et kapja.

Természetesen a fenti rendszerek biztonságához egy elég nagy méretű test feletti görbe kell, melynek elég sok pontja van, továbbá a kiválasztott P pont rendjének is nagynak kell lennie.

A.3. ECDSA - digitális aláírás ECC segítségével

ECDSA kulcsot generálunk magunknak.

1. Válasszunk egy E elliptikus görbét!
2. Válasszunk a görbén egy $P \in E$ pontot! Legyen P rendje n!
3. Válasszunk egy kriptográfiailag erős számot $d = [1..n-1]$
4. Legyen $Q = d * P$!
5. A titkos kulcsunk a d szám, nyilvános kulcsunk pedig (E, P, n, Q).

40. ábra. ECDSA kulcsgenerálás

Végezetül megemlítem az ECDSA (41. ábra) módszert, amely elliptikus görbéken alapuló digitális aláírás számítását teszi lehetővé. Erről részletes leírást ad Crutchley [14] és Koblitz[24]. E dolgozat csupán a kulcsgeneráló (40. ábra), aláíró (41. ábra) és az aláírást ellenőrző (42. ábra) algoritmust tartalmazza, azok bizonyítását nem.

Alá akarjuk írni m üzenetet. Titkos kulcsunk a d szám, nyilvános kulcsunk pedig (E, P, n, Q), ahol E egy elliptikus görbe, P annak egy pontja, n P rendje és $Q = d * P$. (40. ábra)

1. Generáljunk egy kriptográfiailag erős k számot! $[1..n-1]$!
2. Számítsuk ki $k * P = (x_1; y_1)$ és legyen $r = x_1 \, mod) n)$!
3. Számítsuk ki $k^{-1} \, mod(n)$-t!
4. Számítsuk ki $s = k^{-1}(h(m)+dr)$, ahol a h függvény az SHA (Secure Hash ALgorithm) algoritmus
5. Ha $s = 0$, térjünk vissza az 1. lépéshez!
6. Megkaptuk m üzenet aláírását: (s, r)

41. ábra. ECDSA aláírás

Ellenőrizni kívánjuk az m üzenet digitális aláírását, amely (s, r). Ehhez szükség lesz az aláíró nyilvános kulcsára! (40. ábra, 41. ábra)

1. Szerezzük meg az aláíró nyilvános kulcsát! (E, P, n, Q) Ellenőrizzük, hogy s és r a 1 és $n-1$ közé esik-e!
2. Legyen $w = s\, mod(n)$! Számítsuk ki $h(m)$-et, ahol h az SHA függvény!
3. Számítsuk ki $u_1 = h(m) * w\, mod(n)$-t! Számítsuk ki $u_2 = r * w\, mod(n)$-t!
4. Legyen $u_1 * P + u_2 * Q = (x_0; y_0)$! Legyen $v = x_0\, mod(n)$!
5. Elfogadhatjuk az aláírást, ha $v = r$.

42. ábra. ECDSA aláírás ellenőrzése

B. A felhaszánlóazonosítás három fő módszere

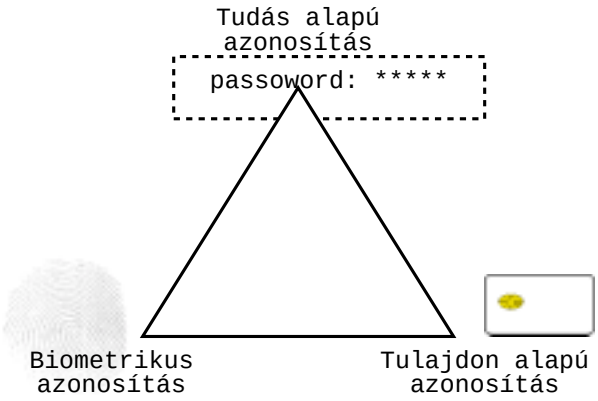
Három módja létezik az azonosításnak (43. ábra):

- Tudás alapú azonosítás – Ez a legegyszerűbb, s gyakran legolcsóbb módja a kliens azonosításának. A kliens azzal bizonyíthatja személyazonosságát, hogy igazolja, valamilyen tudás birtokában van. Ilyenkor feltételezzük, hogy csak ő rendelkezik ezzel a tudással. Egyszerűbb esetben ez a tudás PIN kód vagy jelszó, de lehet akár valamilyen titkos kulcs is. Azonosításkor elkérhetjük a tudást a felhasználótól közvetlenül (bár ilyenkor ki vagyunk téve a lehallgatás veszélyének), de jobb rendszerekben a felhasználónak csupán bizonyítania kell, hogy rendelkezik a tudással, nem kell azt kiadnia magából.
  Sajnos az emberi memória korlátos és hibás. Nem vagyunk képesek nagyon hosszú titkokat megjegyezni, s gyakran felírjuk őket.Ugyanakkor előnye a tudás alapú azonosításnak az olcsóság és az eltulajdoníthatatlanság.

- Tulajdon alapú azonosítás – Szintén gyakran használt módja az azonosításnak. Ilyen esetben a felhasználó birtokában van egy tárgy, s ennek segítségével igazolja személyazonosságát. A legelterjettebb változata a kulcs, amely egy zárat nyit. A chipkártyák szintén olyan tárgyak, amelyek segítségével a személyazonosság könnyen igazolható. Lényeges, hogy a tárgyak, melyekkel a felhasználók azonosítják magukat, egyediek, s lehetőleg másolhatatlanok legyenek.

Sajnos a tárgyak könnyen elveszíthetők vagy ellophatók, így célszerű ezt az azonosítási módot kombinálni a másik kettő egyikével. Szintén fontos, hogy a tárgy ne legyen lemásolható.

- Biometriai azonosítás – Rendkívül jó, de sajnos igen drága módja a személyek azonosításának. Ez esetben az egyes személyek valamely biológiai jellegzetességét ragadjuk meg, s annak segítségével próbáljuk megkülönböztetni a többiektől. Az ujjlenyomatot a rendőrség már régóta alkalmazza erre a célra, de használnak már hangminta alapú azonosítást, retinaleolvasást, aláírásfelismerést és arcfelismerést is.
  A biometriai jellemzőkből általában valamilyen elektromos jel keletkezik, s ez kerül a számítógépben azonosításra. Ha a biometriai jellemző nehezen utánozható is, az elektromos jel később felvehető majd visszajátszható. A visszajátszás a biometriai azonosítási módszerek gyenge pontja.



43. ábra. Felhasználó-azonosítási módszerek

Mivel mindhárom azonosítási módnak vannak gyengéi és erősségei, egyik sem nevezhető tökéletesnek. Mindhárom becsapható, ezért kombinációjuk vezethet igazán jó eredményre. Akkor tekinthetünk egy rendszert biztonságosnak, ha az azonosítás 43. ábrán látható három komponenséből legalább kettőt függetlenül használ.

C. Challenge and repsponse azonosítás

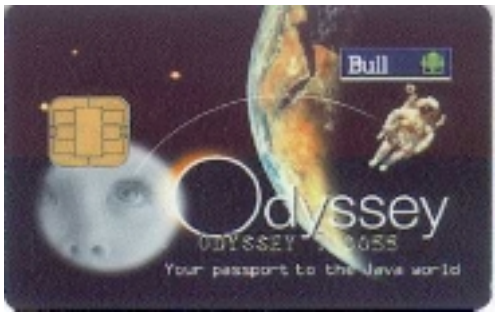
A challenge and response egy olyan módszer, melynek segítségével az egyik fél igazolja, hogy egy titkos kulcs birtokában van, és mindezt anélkül teszi, hogy a kulcsot felmutatná

Kliens->Szerver:	azonosítás kérés
Szerver:	generál egy $r1$ random számot
Szerver->Kliens:	$r1$
Kliens->Szerver:	$E_{t,kliens}\{r1\}$
Szerver:	Ellenőrzi, hogy $E_{ny,kliens}\{E_{t,kliens}\{r1\}\}$ megegyezik-e $r1$-gyel.

44. ábra. Challenge and response azonosítás

a másik félnek. (C) Nyilvános kulcsú challenge and response esetén a szerver úgy képes leellenőrizni a titkos kulcs meglétét, hogy nincs is birtokában annak.

D. Néhány Java Card kompatibilis kártya



Bull Odyssey I: 8 kilobyte EEPROM-mal rendelkezik, kriptográfiai műveleteket nem támogat. Létezik Java Card 2.0 és 2.1-es változata. További információ: [4].

Schlumberger Cyberflex Access: 16 kilobyte EEPROM, az általam használt példány kriptográfiai koprocesszorral is rendelkezett. Támogatja a DES, 3DES, RSA és SHA algoritmusokat. Leírás: 5.3.1. fejezet, [37], [38].

Dallas Semiconductor iButton: Nem kártya formájú, de Java Card kompatibilis egység. Apró gomb formájában kerülhet kulcstartóra, pecsétgyűrűre is. Rendelkezik saját tápegységgel és órával. További információ: [15].