Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tim Smole

Implementacija brezstičnih pametnih kartic z varnostnim elementom v oblaku

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Viljan Mahnič

Ljubljana 2015

Rezultati diplomskega dela so intelektualna lastnina avtorja. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Proučite možnosti za implementacijo brezstičnih pametnih kartic na mobilnih telefonih z operacijskim sistemom Android. Pri tem se osredotočite na rešitev, ki bo za hranjenje varnostnega elementa in procesiranje ukazov uporabljala strežnik v oblaku. Izdelajte ogrodje take rešitve in podrobneje opišite komunikacijo med terminalom (čitalnikom kartic), mobilnim telefonom in oblakom. Ogrodje naj bo zasnovano tako, da bo omogočalo enostavno prilagoditev različnim protokolom za komunikacijo med terminalom in pametnimi karticami.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Tim Smole, z vpisno številko 63110255, sem avtor diplomskega dela z naslovom:

Implementacija brezstični pametnih kartic z varnostnim elementom v oblaku

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Viljana Mahniča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 5. september 2015 Podpis avtorja:

Kazalo

Seznam uporabljenih kratic

AID Application identifier. Identifikator aplikacije, namenjen naslavljanju te aplikacije.

APDU Application protocol data unit. Komunikacijski protokol za prenos podatkov med pametno kartico in bralnikom pametnih kartic.

API Application programming interface. Programski vmesnik.

EEPROM Electrically erasable programmable read-only memory. Električno izbrisljiv programirljiv bralni pomnilnik.

FID File identifier. Identifikator datoteke, namenjen naslavljanju te datoteke.

HCE Host-based card emulation. Emulacija kartice.

HTTPS Secure Hypertext Transfer Protocol. Protokol za varno komunikacijo v računalniških omrežjih.

JWT JSON web token. Žeton, ki se uporablja pri vodenju uporabniške seje.

MicroSD Bliskovni pomnilnik majhnih dimenzij.

NFC Near field communication. Nabor standardov za brezstično komunikacijo mobilnih naprav na kratki razdalji.

RAM Random-access memory. Bralno-pisalni pomnilnik.

REST Representational state transfer. Protokol za prenos podatkov preko interneta.

ROM Read-only memory. Bralni pomnilnik.

SIM Subscriber identification module. Kartica za identifikacijo uporabnika v mobilnem omrežju.

Povzetek

V zadnjem desetletju se vedno pogosteje srečujemo s pojmom pametna kartica. Gre za pomnilniške kartice, katerih vsebino podatkov je mogoče spreminjati tudi med samo uporabo. Pametne kartice zato poznajo nabor ukazov za dostop, branje in spreminjanje vsebine. Danes se take kartice uporabljajo na veliko različnih področjih. Najpogostejši primeri uporabe so bančne kartice, osebne izkaznice, kartice zvestobe in kartice javnega prevoza.

Želja po zmanjšanju števila pametnih kartic, ki jih uporabljamo vsakodnevno, in uporaba pametnih telefonov, je na trg pripeljala novo rešitev. S tehnologijo, ki jo uporabljajo pametni telefoni, je mogoče simulirati pametne kartice. Pametni telefoni bi tako v naslednjih letih lahko zamenjali pametne kartice.

Cilj diplomskega dela je bil predstaviti eno od rešitev, ki nadomesti že obstoječe pametne kartice z uporabo pametnega telefona in oblaka, brez potrebe po dodatnem nameščanju programske opreme na že obstoječe čitalnike pametnih kartic.

Ključne besede: Pametna kartica, virtualna pametna kartica, varnostni element, oblak, Android, Android HCE.

Abstract

In the last decade we are often confronted with term smart card. Smart cards are usually plastic cards with integrated circuit that enables storage of data. The term *smart* indicates that the data on the card can be modified after initial manifacturing. Today this kind of cards are used in many different areas. The most known use cases are credit cards, loyalty cards, biometric passports, transportation cards and so on.

Our wish to reduce the number of cards in our wallets and the rise of smart phones has brought new solutions. With technologies that smart phones offer it is possible to simulate and emulate virtual smart cards. Smart phones therefore could reduce or even replace the number of plastic smart cards in the next couple of years.

The goal of this bachelor thesis if to present a solution that could replace plastic smart cards with the use of smart phones and cloud, without modifying or upgrading any software which is already installed on every day used card readers.

Keywords: Smart card, virtual smart card, secure element, cloud, Android, Android HCE.

Poglavje 1

Uvod

Pod pojmom pametna kartica si nas večina predstavlja plastične kartice, ki jih pogosto uporabljamo v vsakodnevnem življenju. Kljub temu da plastične kartice, ki jih uporabljamo že več desetletij, spadajo med pametne kartice, je pomembno poudariti, da danes na trg prihaja vedno več navideznih pametnih kartic, ki same po sebi nimajo oblike, temveč se izvajajo znotraj enega ali več računalnikov. Pojem pametna kartica torej ni vezan na samo obliko in realizacijo kartice, temveč na uporabnost, standarde in protokole, ki se izvajajo nad osebnimi podatki, ki so bili včasih zapisani na fizični kartici, dandanes pa večkrat kar na pametnem telefonu ali kakšnem drugem računalniku.

Razlog, da so navidezne pametne kartice čedalje bolj priljubljene, se skriva v vzponu pametnih telefonov. Večina prodanih mobilnih telefonov spada med pametne telefone. Delež uporabnikov s pametnimi telefoni že predstavlja večino. Med pametnimi telefoni se dviguje tudi delež takšnih, ki podpirajo tehnologijo NFC, le ta pa je pomembna za realizacijo brezstične navidezne pametne kartice.

Ključnega pomena za uspešno komunikacijo med pametno kartico in bralnikom kartice je standardiziran protokol. Pri implementaciji navidezne pametne kartice je pomembno upoštevati, da je na trgu veliko čitalnikov kartice z že nameščeno programsko opremo, ki sledi standardnim protokolom. Standard ne opisujejo le komunikacije med pametno kartico in bralnikom, temveč

tudi samo datotečno strukturo pametne kartice, varnostne mehanizme ter obdelavo podatkov, shranjenih na sami pametni kartici. Potrebno se je zavedati, da obstaja veliko različnih standardiziranih protokol, za uspešno komunikacijo pa morata tako pametna kartica kot terminal slediti istemu protokolu. Za implementacijo navidezne pametne kartice je zato izbira protokola ključnega pomena.

Trenutno najbolj tehnološko napreden standard, ki ga podpira vedno več pametnih kartic, je standard Cipurse. Gre za odprtokodni protokol, ki ga je vzpostavil OSPT (Open Standard for Transportation) Alliance [16] za zadovoljitev potreb po varni komunikaciji med pametnimi karticami in čitalniki pametnih kartic. Standard Cipurse poleg varne komunikacije zagotavlja preprosto datotečno strukturo posamezne aplikacije ter omogoča izvajanje večih aplikacij na eni kartici. Pomembno je poudariti, da se je Cipurse standard uveljavil tako med plastičnimi karticami kot tudi navideznimi pametnimi karticami. Tako plastične kot navidezne kartice imajo podatke shranjene v varnostnem elementu, le ti pa so lahko realizirani na različne načine.

Pri plastičnih pametnih karticah je varnostni element integrirano vezje, ki pametnim karticam omogoča visok nivo varnosti pri shranjevanju podatkov. V kolikor želimo simulirati plastično pametno kartico z mobilnim telefonom, mora tudi telefon zagotoviti ustrezno varnost podatkov. Na izbiro imamo več različnih tipov varnostnega elementa, kot so kartica *SIM*, vdelani varnostni element, *MicroSD*, ... [18]

S prihodom operacijskega sistema Android KitKat 4.4, je Google razvijalcem omogočil uporabo knjižnice HCE (Host-based Card Emulation) [15], ki omogoča emulacijo navidezniih kartic brez varnostnega elementa. Rešitev omogoča razvoj aplikacij za plačevanje z mobilnimi telefoni, ki so neodvisne od proizvajalcev mobilnih telefonov in mobilnih operaterjev. Kljub temu pa knjižnica HCE ne rešuje problema varnega shranjevanja podatkov in dostopa do njih, ampak prepušča implementacijo oziroma simulacijo varnostnega elementa razvijalcu.

Namen diplomske naloge je bralcu predstaviti kako implementirati navi-

dezno pametno kartico z uporabo knjižnice HCE, ki kot varnostni element uporablja oblak. Mobilni telefon bo predstavljal posredovalni člen med terminalom in varnostnim elementom shranjenim v oblaku. Posredoval bo vsak prejeti ukaz terminala v oblak ter odgovor oblaka posredoval nazaj terminalu. Zaradi različnih standardiziranih protokolov, katerim sledijo pametne kartice in bralniki kartic, bo naš oblak predstavljen kot ogrodje, ki omogoča enostavno razširitev in prilagoditev na posamezne protokole.

Poglavje 2

Varnostni element

Za implementacijo navidezne pametne kartice je obvezno poznavanje delovanja plastičnih kartic.

Ko so se prvič pojavile plastične pomnilniške kartice, je bilo na njih možno zapisati vsebino le ob proizvodnji. Kasnejše spreminjanje vsebine ni bilo mogoče, možno je bilo le branje. Takšne kartice so zato potrebovale mehanizem za dolgotrajno hranjenje podatkov. Branje podatkov je potekalo s pomočjo bralnika kartic ali terminala. Kmalu za tem so se pojavile pametne plastične kartice, ki so omogočale spremembo zapisane vsebine tudi po proizvodnji. Za dostop in spremembo vsebine so zato potrebni ukazi, ki jih izstavljajo terminali. Za izvedbo samega ukaza potrebuje pametna kartica procesor. Ker pa sama kartica nima lastnega napajanja, pridobi električno energijo kar iz terminala. Sprva so za prenos ukazov potrebovale fizični stik s terminalom (stične pametne kartice), danes pa se najpogosteje uporablja prenos preko NFC protokola, zato danes govorimo o tako imenovanih brezstičnih pametnih karticah.

Sama fizična zgradba pametnih kartic je predpisana s standardom ISO/IEC 7810 [13]. Podrobno pa jo tudi opisuje diplomsko delo Mihe Zorca, *Implementacija brezstičnih pametnih kartic na napravah NFC* [19]. Za razumevanje tega poglavja pa je pomembno naslednje. Obvezni sestavni del pametnih kartic je tako imenovan varnostni element. Gre za integrirano vezje,

sestavljeno iz procesorja, delovnega pomnilnika (angleško Random-access memory, krajše RAM), trajnega pomnilnika (angleško Read-only memory, krajše ROM) in električno izbrisljiv programirljiv bralni pomnilnik (angleško Electrically Erasable Programmable Read-Only, krajše EEPROM). Omogoča dolgotrajno in varno shranjevanje podatkov. Podpira varnostne mehanizme, ki onemogočajo dostop do podatkov brez poznavanja ključev za dostop. Ker ključi za dostop običajno niso znani uporabnikom, je neželjeno spreminjanje podatkov napadalcem onemogočeno.

Rešitev za varnostni element je veliko. Pametni telefoni običajno nudijo vsaj dve. Mi pa si bomo pogledali, kaj je potrebno za simulacijo varnostnega elementa v oblaku.

Poglavje 3

Komunikacija med terminalom in oblakom

Komunikacija med terminalom in pametno kartico poteka po protokolu APDU (Application Protocol Data Unit, slovensko podatkovna enota aplikacijskega protokola), ki je definiran s standardom ISO/IEC 7816-4 [14]. Vsak APDU korak je sestavljen iz ukaza in odgovora, ki se navezuje na izvedbo pravkar prejetega ukaza. APDU ukaz vedno izstavi terminal in ga posreduje pametni kartici oziroma pametnemu telefonu. Le-ta nato ukaz interpretira ter izvede, terminalu pa odgovori z APDU odgovorom.

Kot že rečeno standard ISO/IEC 7816-4 [14] predpisuje le zgradbo APDU ukaza in odgovora ter najbolj osnovne APDU ukaze, ki jih morajo podpirate vse implementacije tega standarda. Implementacije pa običajno definirajo širši nabor APDU ukazov za manipulacijo datotek oziroma njihovih podatkov. Prav tako definirajo APDU odgovore glede na uspeh izvedbe ukaza.

Tabela 3.1: Zgradba APDU ukaza

Glava ukaza (obvezno)	**Telo ukaza (neobvezno)**
CLA INS P1 P2	Lc Data Le

Vsak APDU ukaz je torej predstavljen z najmanj štirimi bajti. Vsa polja

Tabela 3.2: Zgradba APDU odgovora

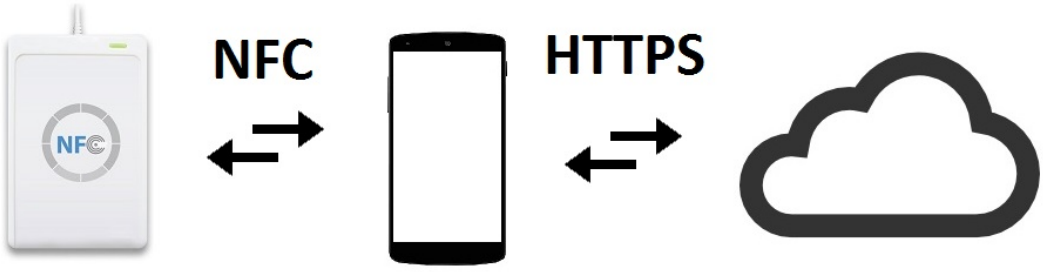
Telo (neobvezno)	**Rep (obvezno)**
Data	SW1 SW2

razen Data predstavljajo en bajt, medtem ko polje je Data lahko predstavljen tudi z več kot enim bajtom. Polje INS določa vrsto ukaza. Za nas bo v nadaljevanju najpomembnejši ukaz SELECT, ki ima vrednost INS polja 0xA4. V nadaljevanju si bomo nekoliko bolj podrobno pogledali zakaj je za nas ta ukaz zanimiv, še prej pa si oglejmo kako se APDU ukazi prenašajo med terminalom in pametno kartico.

Brezstične pametne kartice in navidezne pametne kartice na napravah NFC običajno prejmejo APDU ukaz preko protokola NFC, ki omogoča izmenjavo manjše količine podatkov na kratko razdaljo (običajno do nekaj centimetrov). APDU ukaz se nato izvede znotraj varnostnega elementa ter preko NFC protokola posreduje terminalu APDU odgovor. V tem primeru se varnostni element in NFC čitalnik fizično nahajata na isti napravi (plastični kartici ali mobilnem telefonu).

Naša implementacija se bo nekoliko razlikovala, saj imamo varnostni element v oblaku. Razdelili jo bomo na dva dela. Prvi del predstavlja komunikacijo med terminalom in mobilnim telefonom. Drugi del pa potek komunikacije med mobilno napravo in oblakom.

Pametni telefon bo z uporabo knjižnice HCE preko protokola NFC prejel ukaz APDU. HCE omogoča obdelavo prejetega ukaza, mi pa ga bomo namesto obdelave brezžično posredoval na strežnik, ki predstavlja naš navidezni varnostni element. Posredovanje APDU ukazov iz HCE v oblak poteka po protokolu HTTPS [8], ki omogoča varen prenos podatkov. Glede na APDU ukaz se na strežniku ustvari ustrezen APDU odgovor, ki se vrne mobilnemu telefonu kot HTTPS odgovor. Mobilni telefon nato preko protokola NFC posreduje odgovor APDU nazaj na terminal. Poleg posredovanja APDU ukazov bo mobilni telefon zadolžen za vzdrževanje seje navidezne pametne kartice



Slika 3.1: Prikaz komunikacije med terminalom, mobilno aplikacijo in oblakom.

na oblaku.

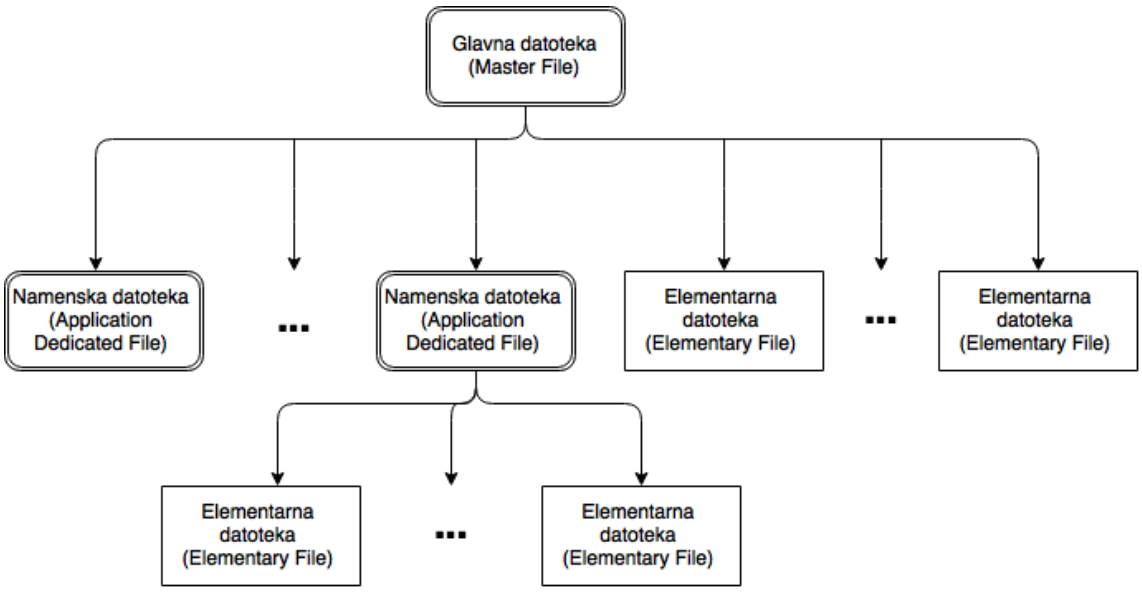
Preden se poglobimo kako tako komunikacijo implementirati, si poglejmo samo datotečno strukturo pametnih kartic, nad katerimi se vršijo APDU ukazi.

Poglavje 4

Datotečna struktura

Datotečna struktura pametnih kartic je predpisana s standardom ISO/IEC 7816 [14]. Standard loči dve vrsti datotek: namenske datoteke (dedicated file) in elementarne datoteke (elementary file). Korenski namenski datoteki pravimo tudi glavna datoteka (master file) in je tudi edina namenska datoteka, ki je obvezna. Znotraj glavne datoteke imamo eno ali več aplikacijskih namenskih datotek (application dadicated file), ki predstavljajo posamezno aplikacijo na kartici (v nadaljevanju aplikacije). Namenske datoteke vsebujejo nič ali več elementarnih datotek.

Elementarne datoteke ločimo glede na njihovo uporabo. Poznamo notranje in delovne elementarne datoteke. Notranje elementarne datoteke se uporabljajo za pravilno delovanje kartice in so manj pomembne za implementacijo navidezne pametne kartice. Glavnega pomena so delovne elementarne datoteke. V njih so shranjeni podatki, nad katerimi izvršujemo ukaze (v nadaljevanju le elementarne datoteke).



Slika 4.1: Prikaz datotečne strukture pametne kartice. Elementi obrobljeni z dvojno črto predstavljajo namenske datoteke, z enojno obrobo pa elementarne datoteke.

4.1 Naslavljanje datotek

Do datotek imamo možni dve vrsti dostopa. Prva možnost je implicitno - preko kazalcev na trenutno izbrani datoteki, druga pa eksplicitno - preko naslavljanja s pomočjo enolično določenih datotečnih identifikatorjev.

Kazalca na izbrani datoteki sta vedno dva. Prvi kaže na trenutno izbrano namensko datoteko. V primeru, da ni izbrana še nobena aplikacija, kaže kazalec na glavno datoteko. Drugi kazalec kaže na trenutno izbrano elementarno datoteko, ki pa ni nujno nastavljen.

Pri eksplicitnem naslavljanju vedno uporabimo ukaz *SELECT*, kateremu podamo identifikator datoteke. Ločimo dve vrsti identifikatorjev: AID (Application Identifier, slovensko identifikator aplikacije), ki pripada le aplikacijskim namenskim datotekam in FID (File Identifier, slovensko datotečni identifikator), ki pripada vsem datotekam. AID je predstavljen s šestnajstimi bajti, FID pa z dvema. Zaradi lažje berljivosti jih bomo pisali kar v šestnajstiški obliki. FID 0x3F00 vedno predstavlja naslov glavne datoteke, preko FID-a 0x3FFF dostopamo do trenutno izbrane aplikacijske namenske datoteke, FID 0xFFFF pa je rezerviran za morebitno prihodnjo uporabo.

FID elementarnih datotek mora biti enolično določen znotraj aplikacije, pri čemer velja, da se mora prav tako razlikovati od FID-a gostujoče aplikacije. Lahko pa se ujema z datotečnim identifikatorjem elementarne datoteke, ki pripada drugi aplikaciji. Glavna datoteka vsebuje tako aplikacijske namenske datoteke kot elementarne datoteke, zato je potrebno poskrbeti, da se FID-i aplikacijskih namenskih datotek razlikujejo od FID-ov elementarnih datotek na tem nivoju datotečnega drevesa.

4.2 Varnostni mehanizmi

Da so na pametni kartici lahko shranjeni občutljivi podatki, do katerih uporabnik oziroma napadalec ne sme imeti dostopa, je potrebno zagotoviti varnostne mehanizme, ki zagotavljajo avtentikacijo in avtorizacijo terminala ter avtentičnost prispelega APDU ukaza.

Terminal in kartica se avtenticirata s pomočjo ključev in APDU ukazov *GET CHALLANGE* in *EXTERNAL AUTHENTICATE* (včasih imenovan tudi *MUTUAL AUTHENTICATE*). Ključe za avtentikacijo vsebujejo namenske datoteke, ni pa potrebno, da jih imajo. Kadar namenska datoteka ne vsebuje ključev za avtentikacijo, avtentikacija ni mogoča. Kadar so ključi prisotni, govorimo o varni namenski datoteki. V tem primeru so varne tudi vse vsebovane elementarne datoteke.

Za uspešno avtentikacijo mora terminal dokazati poznavanje vrednosti ključa, ki ga vsebuje namenska datoteka, nad katero se želimo avtenticirati. Po uspešni avtentikaciji se med terminalom in namensko datoteko vzpostavi varen kanal, po katerem se prenašajo kriptirani APDU ukazi. Večina implementacij pozna več vrst enkripcije APDU ukazov. Razlikujejo se po stopnji enkripcije. Nekatere vrste kriptirajo celoten ukaz, nekatere le podatkovni del telesa ukaza, nekatere pa ukaza niti ne kriptirajo, le vzdržujejo varen kanal z računanjem novih kriptirnih ključev ob vsakem novem APDU ukazu oziroma odgovoru. V primeru različnih stopenj enkripcij lahko zahtevamo kakšna naj bo minimalna stopnja za določen ukaz. Stopnjo minimalne enrkipcije določimo s pomočjo pravil varnega sporočanja (angleško Secure Messaging Rules, krajše SMR), kjer za vsak ukaz definiramo pravilo, kakšna je še minimalno sprejemljiva enkripcija prejetega ukaza in kakšna naj bo enkripcija odgovora kartice. V primeru, da je enkripcija prejetega ukaza oziroma zahtevana enkripcija odgovora nižja od minimalne stopnje enkripcije definirane v pravilih varnega sporočanja, kartica ukaza ne izvrši in terminalu posreduje status napake.

Tako kot za avtentikacijo se tudi za avtorizacijo uporabljajo ključi. Vsakemu ključu lahko omogočimo in onemogočimo izvršitev določenih ukazov.

Pravice so določene s tabelo pravic dostopa (angleško Access Rights Table, krajše ART). Vrstica tabele pripada ključu, stolpci pa predstavljajo posamezne ukaze. Pred vsako izvršitvijo ukaza se nato v tabeli pravic dostopa preveri, ali je za ključ, s katerim smo avtenticirani, dovoljena izvršitev ukaza. Če ključu ne pripadajo pravice, se ukaz ne izvrši, terminalu pa se posreduje status napake.

4.3 Vrste elementarnih datotek

Standard ISO/IEC 7816 [14] glede na organizacijo podatkov znotraj datotek loči različne tipe elementarnih datotek. Podatki so lahko neorganizirani, takrat govorimo o binarni datoteki. Lahko pa so predstavljeni kot sosledje neodvisnih zapisov (angleško Records). Zapisi so lahko fiksne ali spremenljive dolžine. Sestavljajo pa lahko linearno ali ciklično strukturo.

V nadaljevanju se ne nameravamo spuščati v podrobnosti implementacije posameznih elementarnih datotek, saj standard ISO/IEC 7816 [14] ne predpisuje implementacije in obdelave struktur različnih elementarnih datotek. Prav zaradi tega so se ustanovile organizacije, ter ustvarile standarde, ki datotečne strukture definirajo nekoliko bolj podrobno. Prav tako definirajo nabor in zgradbo APDU ukazov za obdelavo podatkov ter podrobneje opisujejo kriptirne in varnostne mehanizme. Med bolj znanimi standardi oziroma implementacijami standarda ISO/IEC 14443 [12] so *CIPURSE [16], Calypso [7], MIFARE [11] in biometrični potni listi.* Potrebno je poudariti, da organizacije standarde konstantno dopolnjujejo in izboljšujejo, zato ima lahko določena implementacija več različnih verzij, še več pa jih lahko pričakujemo v prihodnosti.

Ogrodje navidezne kartice

Preprosto implementacijo oziroma ogrodje za kasnejšo razširitev glede na standard, ki želimo, da ga navidezna kartica podpira, prikazujejo izseki programske kode od 4.1 do 4.4:

Listing 4.1: Implementacija navidezne pametne kartice.

```
/*
*   Zaradi preglednosti so metode za nastavljanje in pridobivanje
*   atributov razreda izpuščene. Prav tako so izpuščene metode za
*   obdelavo APDU ukazov.
*/
// Card.java
```

```
public class Card
{
    /* datotecna struktura kartice */
    private MasterFile masterFile;

    /* podatki o trenutni seji */
    private CardRam cardRam;
}
```

Listing 4.2: Razred MasterFile.

```
// MasterFile.java
public class MasterFile extends FileAdf
{
    /* aplikacije, ki jih vsebuje glavna datoteka */
    private FileAdf[] fileAdfs;
}
```

Listing 4.3: Implementacija namenske datoteke bi izgledala na sledeči način.

```
// FileAdf.java
public class FileAdf extends FileEf
{
    /* identifikator aplikacije */
    private byte[] applicationID;

    /* vrednosti kljucev */
    protected Key[] keys;

    /* elementarne datoteke, ki jih vsebuje aplikacija */
    protected File[] FileEfs;
}
```

Listing 4.4: Implementacija elementarne datoteke bi izgledala na sledeči način.

```
// FileEf.java
public abstract class FileEf
{
    /* datotecni identifikatior */
    protected short fileID;

    /* podatki o minimalni enkripciji prejetih APDU ukazov in
       odgovorov */
    private SMR secureMessegingRules;

    /* podatki o pravicah za izvedbo APDU ukaza */
    private ART accessRightsTable[];
}
```

Bodimo pozorni, da je razred *FileEf* abstraktni razred in ne vsebuje nobene spremenljivke, namenjene shranjevanju podatkov. Ravno zaradi razlik v organizaciji podatkov bomo prepustili implementacijo shranjevanja podatkov konkretnim razredom, ki razširjajo razred *FileEf*.

Poglavje 5

Implementacija

Sedaj, ko razumemo osnovno zgradbo pametnih kartic in njihovo poenostavljeno implementacijo, si lahko pogledamo kako bi izgledalo izvrševanje APDU ukazov na posamezni navidezni kartici.

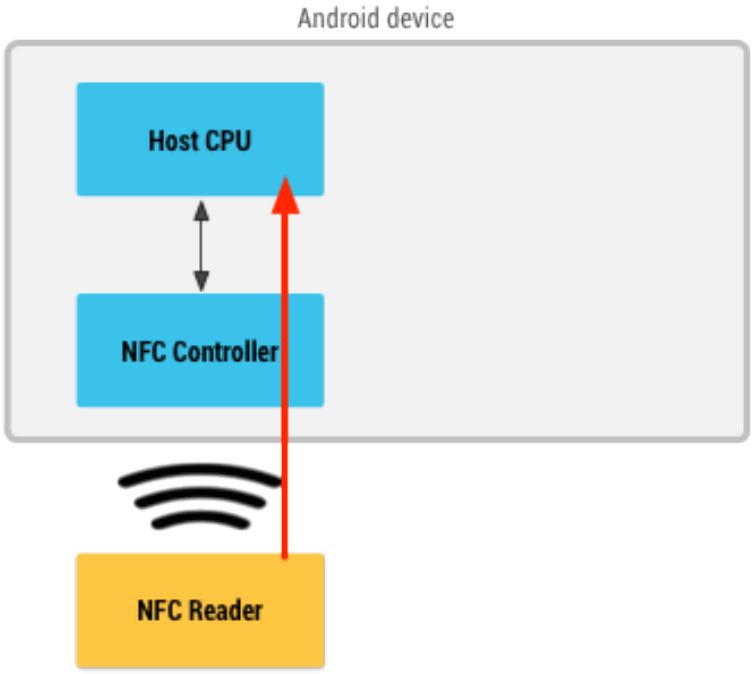
V nadaljevanju se bom predvsem osredotočil na samo implementacijo Androidne aplikacije. Podrobneje bomo opisali:

- kako pravilno izbrati navidezno kartico v oblaku
- kako ji dostaviti APDU ukaze in odgovore vrniti terminalu
- kako voditi sejo posamezne navidezne kartice, ki se nahaja v oblaku, tekom NFC komunikacije med terminalom in androidno aplikacijo nameščeno na mobilnem telefonu

Na koncu bom prikazal kako implementirati spletne storitve, ki jih potrebuje androidna aplikacija za uspešno delovanje. Kljub temu da standard ISO/IEC 7810 [13] ne definira same implementacije izvajanja APDU ukazov in njihovih odgovorov, bom v zaključku poglavja na kratko opisal kako bi lahko realizirali izvajanje prispelega ukaza nad navidezno kartico.

5.1 Android aplikacija

Večino mobilnih naprav, ki imajo nameščen operacijski sistem Android in vgrajeno NFC anteno, že omogoča emulacijo NFC emulacijo kartic (angleško Host-based Card Emulation, krajše HCE) [10]. Gre za izvajanje navidezne pametne kartice znotraj okolja Android. Do verzije Android 4.4 [6] je varnostni element, kjer poteka emulacija, najpogosteje predstavljal kar čip SIM (Subscriber Identity Module). Verzija Android 4.4 pa je prinesla rešitev, ki varnostnega elementa ne potrebuje in s tem omogoča androidnim aplikacijam direktno NFC komunikacijo s terminalom.



Slika 5.1: Prikaz komunikacije med terminalom in emulacijo kartice brez varnostnega elementa.

Naša aplikacija bo predstavljala eno navidezno pametno kartico v oblaku, sestavljena pa bo iz dveh komponent. Prva komponenta bo aktivnost (angleško *Activity* [2], kjer se bo uporabnik lahko registriral oziroma avtenticiral v primeru, da uporabniški račun že ima. Druga komponenta pa bo storitev

(angleško *Service*) [5], ki se bo izvajala v ozadju in skrbela za komunikacijo med terminalom in oblakom. V tem poglavju si bomo torej pogledali, kaj potrebuje androidna aplikacija za uspešno komunikacijo s terminalom in kako tako aplikacijo prilagodimo za naše potrebe, da bo prejete APDU ukaze posredovala pravi navidezni kartici v oblak.

Ob začetku izvajanja aplikacije se bo na zaslonu prikazala aktivnost, medtem ko se HCE storitev še ne bo izvajala. Pomembno je, da omogočimo uporabo HCE storitve šele po validaciji uporabniškega računa. Ob uspešni uporabniški prijavi se bo HCE storitev zagnala, aktivnost bo uporabnika obvestila o uspehu prijave ter se zaključila. Od tega trenutka dalje se bo storitev izvajala v ozadju, dokler je uporabnik ročno ne zaustavi ali ugasne telefona. V praksi bi uporabniku omogočili odjavo iz aplikacije ter v tem procesu tudi sami zaustavili storitev, vendar z namenom preprostosti naše aplikacije tega ne bomo demonstrirali.

5.1.1 Android HCE

Android aplikacija, ki želi komunicirati z terminalom preko NFC protokola mora implementirati abstraktne metode razreda *HostApduService*, ki je sposoben sprejemati APDU ukaze, ki jih pošilja terminal. Abstraktni razred *HostApduService* razširja razred *Service* (slovensko storitev), zato večkrat konkretni implementaciji razreda *HostApduService* pravimo kar HCE storitev.

Storitev je Android komponenta, ki se izvaja v ozadju in za delovanje ne potrebuje grafičnega vmesnika. Na ta način je omogočena hitrejša komunikacija s terminalom, saj uporabniku ni potrebno zagnati aplikacije preden prisloni mobilni telefon k terminalu.

Terminal torej komunicira s HCE storitvijo, ki je sestavni del mobilne aplikacije. Takih aplikacij, ki poslušajo za APDU ukaze je lahko na isti mobilni napravi več. Ko mobilna naprava zazna prihod novega APDU ukaza, mora operacijski sitem poskrbeti, da izbere ustrezno aplikacijo, oziroma ustrezno HCE storitev. Problem rešimo tako, da vsaki HCE storitvi določimo, na ka-

tere AID identifikatorje se bo storitev odzivala. Komunikacija se tako prične s prihodom "SELECT AID" APDU ukaza. Glede na AID operacijski sistem Android izbere ustrezno aplikacijo, naslednji APDU ukazi pa se nato posredujejo isti HCE storitvi, vse dokler ne prispe nov "SELECT AID" ukaz, ali pa se povezava prekine z oddaljitvijo mobilne naprave od terminala.

5.1.2 Datoteka AndroidManifest.xml

Tako kot vsaka androidna aplikacija mora tudi naša aplikacija vsebovati datoteko *AndroidManifest.xml* [1]. V datoteki se nahajajo bistveni podatki o aplikaciji, ki jih potrebuje operacijski sistem za pravilno izvajanje aplikacije. Podatkov, ki jih je potrebno navesti za pravilno delovanje aplikacije, je veliko, v nadaljevanju pa se bom osredotočil le na tisti del datoteke *AndroidManifest.xml*, ki je specifičen za aplikacije, ki simulirajo navidezne kartice.

Ker naša aplikacija komunicira s terminalom preko protokola NFC ter z oblakom preko protokola HTTPS, moramo v datoteko *AndroidManifest.xml* navesti dovoljenja za uporabno NFC antene in dostop do interneta. Prav tako moramo v datoteko *AndroidManifest.xml* navesti komponente, ki jih uporablja aplikacija. V našem primeru so to HCE storitev in aktivnosti za registracijo in prijavo uporabnika. Deklaracija HCE storitve se od deklaracije običajne storitve razlikuje le v tem, da kot meta podatek dodamo referenco na XML datoteko, kjer se nahajajo AID identifikatorji, na katere se bo HCE storitev odzivala.

Listing 5.1: Prikaz deklaracije dovoljenj in HCE storitve v datoteki AndroidManifest.xml

```
<!-- AndroidManifest.xml -->
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    ...
    <!-- dovoljenja aplikacije -->
    <uses-permission android:name="android.permission.INTERNET" />
```

```
    <uses-permission android:name="android.permission.NFC" />

    <!-- registracija HCE storitve -->
    <service android:name=".ApduProcessor" android:exported="true"
             android:permission="android.permission.BIND_NFC_SERVICE">
        <intent-filter>
            <action
                android:name="android.nfc.cardemulation.action.HOST_APDU_SERVICE"/>
        </intent-filter>

        <!-- referenca na datoteko, kjer so zapisani meta podatki
            za HCE storitev -->
        <meta-data
            android:name="android.nfc.cardemulation.host_apdu_service"
                  android:resource="@xml/apduservice"/>
    </service>
    ...
</manifest>
```

Listing 5.2: Meta podatki shranjeni v datoteki na katero kaže deklaracija v datoteki AndroidManifest.xml

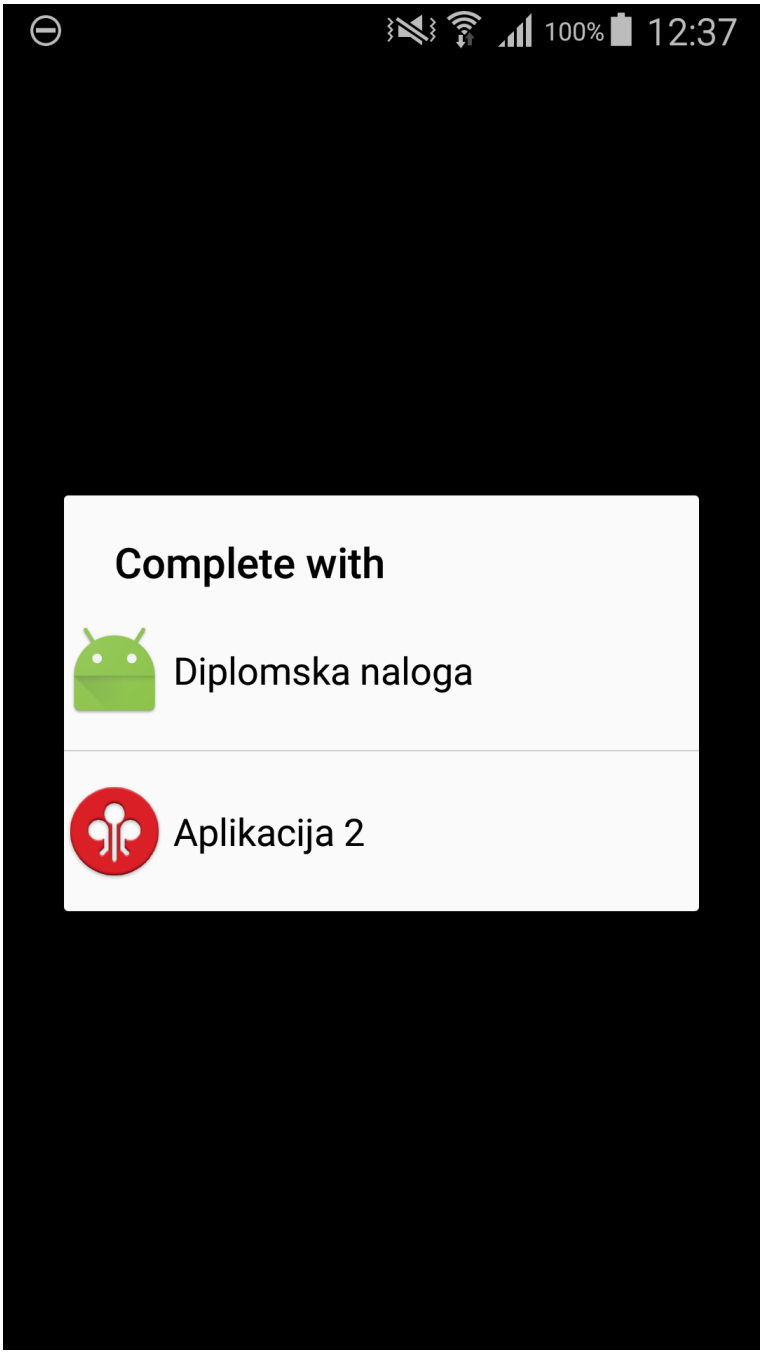
```
<!-- apduservice.xml -->
<host-apdu-service
    xmlns:android="http://schemas.android.com/apk/res/android"
           android:description="Diplomska naloga"
           android:requireDeviceUnlock="true">

    <!-- AID identifikatorji, na katere se bo HCE storitev odzivala
        -->
    <aid-group android:description="Opis AID identifikatorja"
               android:category="other">
        <aid-filter android:name="ADF1"/>
```

```
        <aid-filter android:name="ADF2"/>
    </aid-group>
</host-apdu-service>
```

Potrebno je poudariti, da operacijski sistem Android aplikaciji onemogoči dostop do NFC modula, kadar je zaslon mobilne naprave ugasnjen. V primeru da je zaslon prižgan vendar pa mobilna naprava ni odklenjena, je dostop aplikacije do NFC modula odvisen od vrednosti atributa *-apdu-service>* značke. Privzeta vrednost atributa je logična nič (angleško false), kar pomeni, da bo storitev prejela APDU ukaz ne glede na to, ali je zaslon odklenjen ali ne. V našem primeru pa bomo zaradi varnostnih razlogov od uporabnika zahtevali, da pred uporabo storitve odklene zaslon.

Najpomembnejša značka v datoteki *apduservice.xml* je *<aid-group>*. V njej navedemo seznam vseh AID identifikatorjev, na katere se bo HCE storitev odzivala.

Tu je potrebno omeniti, da se na isti napravi lahko nahaja več HCE storitev, ki imajo registrirane iste AID identifikatorje. Ob prispelem ukazu ŠELECT AID"je zato potrebno izbrati pravo HCE storitev. Operacijski sistem Android razreši konflikt tako, da za izbiro aplikacije vpraša uporabnika.



Slika 5.2: Prikaz dialoga za razrešitev AID konflikta.

Preden pa se podrobneje poglobimo v samo implementacijo naše rešitve, si poglejmo, kaj pričakujemo od uporabnika pred prvo uporabno naše storitve.

Vsak uporabnik mora pred uporabo naše storitve ustvariti svoj uporabniški račun. Grafični vmesnik za kreiranje uporabniškega računa bo del androidne aplikacije, ki bo prav tako ključni element pri posredovanju prejetih APDU ukazov, ki jih izstavi terminal. Zaradi lažje predstave bomo v nadaljevanju predpostavili, da ima vsak uporabnik v oblaku shranjeno natanko eno navidezno kartico. Kreiranje navidezne kartice se bo izvedlo kar v procesu ustvarjanja uporabniškega računa. Kartica bo tako vezana na uporabniški račun, s katerim se bo uporabnik prijavil v androidno aplikacijo.

5.1.3 Vodenje uporabniške seje

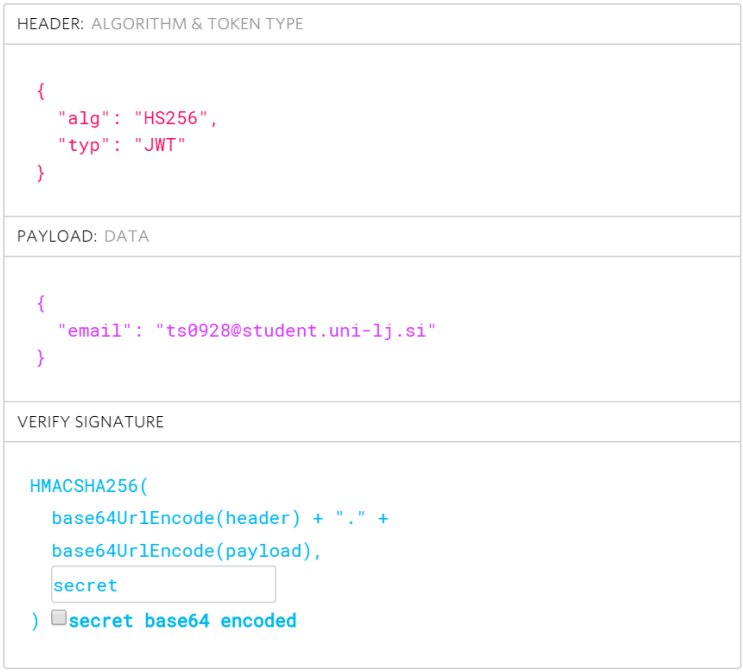
Ker protokol HTTPS ne shranjuje stanja, je potrebno za vodenje uporabniške seje poskrbeti z naprednejšimi mehanizmi. V svetu spletnih aplikacij se največkrat srečujemo s sejnimi piškotki (angleško Session Cookie). Gre za spletni piškotek, ki ga strežnik ob uporabniški avtentikaciji posreduje odjemalcu, v njem pa je zapisan enolični identifikator uporabniške seje. Spletni brskalnik shrani spletni piškotek in ga z vsakim naslednjih zahtevkom posreduje strežniku.

V našem primeru pa odjemalčevo aplikacijo predstavlja androidna aplikacija nameščena na mobilnem telefonu in ne spletni brskalnik. Androidne aplikacije običajno ne shranjujejo spletnih piškotkov, vendar pa se njihov način vodenja uporabniške seje ni razlikuje bistveno. Trenutno najpogostejša rešitev je uporaba žetonov (angleško Token). Žeton zgradi strežnik ter ga ob uporabniški avtentikaciji posreduje odjemalcu, le ta pa ga shrani v lokalni pomnilnik. Žeton nato odjemalec pošlje v vsakem zahtevku kot del glave HTTPS zahtevka (angleško HTTPS Header). V našem primeru bomo uporabili žeton, angleško imenovan JSON Web Token, ali krajše JWT [17].

JWT žeton je definiran s specifikacijo RFC 7519 [9]. Sestavljen je iz glave (header), telesa (payload) in podpisa (signature). Vsak od treh delov je pravzaprav JSON objekt (od tu tudi ime), zakodiran s kodirnim sistemom Base64, ter s piko združen v en sam niz. V glavi se nahajajo podatki o kriptirnem algoritmu in tipu žetona. V telo zapišemo poljubne parametre.

Običajno je teh parametrov malo, saj želimo minimizirati velikost HTTPS zahtevkov, vsaj eden od parametrov pa enolično določa uporabnika. V našem primeru bo to kar elektronski naslov uporabnika, ki ga je uporabnik navedel pri registraciji. Podpis pa predstavlja niz šifriran s kriptirnim algoritmom zapisanim v glavi. Niz, ki vstopa v kriptirni algoritem je sestavljen iz glave JWT žetona, podatkov in skrivnosti oziroma soli (angleško salt), ki jo pozna le strežnik. Na ta način strežnik lahko preverja veljavnost JWT žetona.

Potrebno je poudariti, da so vsi podatki, ki se nahajajo v glavi in telesu žetona, dostopni in berljivi uporabniku. Zato moramo biti previdni, da kot parametre v telo žetona ne podajamo vsebinsko občutljivih podatkov.



Slika 5.3: Prikaz zgradbe JWT žetona pred kodiranjem s sistemom Base64.

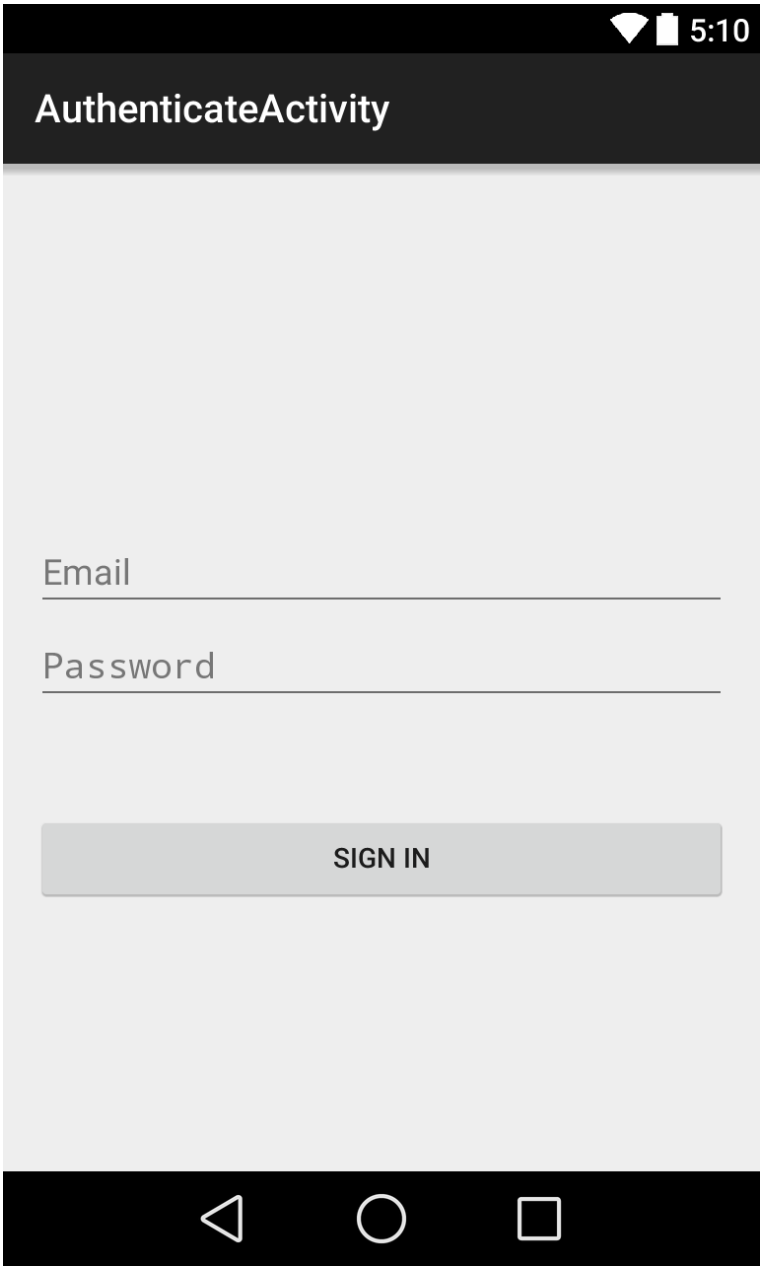
5.1.4 Kreiranje uporabniškega računa

V nadaljevanju bom opisal kako bi izgledala implementacija aktivnosti za kreiranje novega uporabniškega računa. V praksi običajno ločimo proces registracije od procesa avtentikacije uporabnika, saj ob kreiranju uporabniškega

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbW
FpbCI6InRzMDkyOEBzdHVkZW50LnVuaS1sai5zaSJ9.
TEr-2IWOGMf7MEQ1ovlLsHBA-YfUnUme8rGx2XWrBLc
```

Slika 5.4: JWT žeton zakodiran s sistemom Base64.

računa običajno želimo izvedeti več informacij o uporabniku. Da pa bi bila naša rešitev čim bolj preprosta, bom v nadaljevanju prikazal aktivnost, ki se uporablja tako za registracijo novega uporabnika, kot tudi za avtentikacijo že obstoječih uporabniških računov. Aktivnost od uporabnika zahteva vnos elektronskega naslova in gesla, ki jih nato posreduje strežniku. Strežniška aplikacija bo sprva poiskala uporabnika z danim elektronskim naslovom v podatkovni bazi in validirala poslano geslo v primeru, da uporabnik z danim elektronskim naslovom že obstaja. V primeru, da uporabnik z danim elektronskim naslovom še ne obstaja, bo strežniška aplikacija ustvarila nov uporabniški račun ter s tem tudi navidezno pametno kartico in ju shranila v podatkovno bazo. V kolikor je bila avtentikacija oziroma registracija uspešna bo strežnik androidni aplikaciji posredoval JWT žeton. Aplikacija nato žeton shrani v lokalni pomnilnik, saj ga mora poslati na strežnik ob vsakem prispelem APDU ukazu.



Slika 5.5: Prikaz aktivnosti AuthenticateActivity. Aktivnost vsebuje vnosno polje za elektronski naslov in geslo ter gumb, ki sproži proces pošiljanja uporabniških podatkov na strežnik.

Listing 5.3: Razred AuthenticateActivity.

```
// AuthenticateActivity.java
public class AuthenticateActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sign_in);

        final Button loginButton = (Button)
            findViewById(R.id.sign_in_btn);
        final EditText emailInput = (EditText)
            findViewById(R.id.email_input);
        final EditText passwordInput = (EditText)
            findViewById(R.id.password_input);

        loginButton.setOnClickListener((v) -> {
                List<NameValuePair> params = new
                    ArrayList<NameValuePair>(2);
                params.add(new BasicNameValuePair("email",
                    emailInput.getText().toString()));
                params.add(new BasicNameValuePair("password",
                    passwordInput.getText().toString()));
                new AuthenticateTask(getApplicationContext(),
                    params).execute();
            }
        });
    }
}
```

Operacijski sistem Android vedno zažene aplikacijo v eni niti. Ob kliku na gumb za avtentikacijo naša aplikacija posreduje podatke strežniku preko

HTTPS protokola ter čaka na odgovor strežnika. Od trenutka ko kliknemo na gumb pa vse do trenutka ko prispe odgovor strežnika bi bil zaslon neodziven. Da bi to preprečili moramo zahtevek strežniku posredovati v svoji niti. Najenostavnejša rešitev za izvajanje dolgotrajnejših operacij je razširitev abstraktnega razreda *AsyncTask* [3], kjer je potrebno prepisati abstraktno metodo *doInBackground*, ki se izvede v posebni niti, ob klicu metode *execute*. Dobra lastnost razreda AsyncTask je ta, da vsebuje metodo *onPostExecute*, ki se izvede v glavni niti aplikacije. Že samo ime metode pove, da se izvede po zaključku dolgotrajne operacije. V tej metodi bomo uporabnika obvestili o uspešni prijavi ter v ozadju pognali HCE storitev.

Razširitev razreda *AsyncTask* bomo v nadaljevanju uporabili še za pošiljanje APDU ukazov na strežnik ter za ponastavitev seje virtualne kartice na strežniku.

Izsek programske kode 5.4 prikazuje razred AuthenticateTask, k i v samostojni niti posreduje uporabniške podatke na strežnik. Iz HTTPS odgovora prebere JWT in ga shrani v privatni pomnilnik. Po opravljeni dolgotrajnejši operaciji obvesti uporabnika s prikazom napisa, prične z izvajanjem HCE storitve in konča aktivnost.

Listing 5.4: Razred AuthenticateTask.

```
// AuthenticateTask.java
public class AuthenticateTask extends AsyncTask<Void, Void, Void>
{
    private Context mContext;
    private List params;

    public AuthenticateTask(Context ctx, List params)
    {
        this.mContext = ctx;
        this.params = params;
    }

    @Override
```

```
protected String doInBackground(Void... voids)
{
    try
    {
        HttpPost request = new HttpPost(Constants.CLOUD_URL +
            "/login");
        request.setHeader("Content-Type",
            "application/x-www-form-urlencoded");
        request.setHeader("Accept", "application/json");
        request.setEntity(new UrlEncodedFormEntity(params));
        HttpClient httpclient = new DefaultHttpClient();
        HttpResponse response = httpclient.execute(request);

        String jwt =
            response.getHeaders("Authorization")[0].getValue();
        saveJwtInPrivateStorage(jwt);
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
    return null;
}

@Override
protected void onPostExecute(Void param)
{
    super.onPostExecute(param);
    mContext.startService(new Intent(mContext,
        ApduProcessor.class));
    Toast.makeText(mContext, "You are now logged in!",
        Toast.LENGTH_LONG).show();
}
```

```
    private void saveJwtInPrivateStorage(String jwt)
    {
        SharedPreferences settings =
            mContext.getSharedPreferences("USER_DATA", 0);
        SharedPreferences.Editor editor = settings.edit();
        editor.putString("JWT", jwt);
        editor.commit();
    }
}
```

5.1.5 HCE storitev

Abstraktni razred *HostApduService* [4] vsebuje dve abstraktni metodi - *processCommandApdu* in *onDeactivated*, ki sta ključnega pomena za implementacijo navidezne pametne kartice.

Posredovanje APDU ukazov oblaku

Metoda *processCommandApdu* se izvede, ko storitev prejme APDU ukaz terminala. Kot parameter dobi APDU ukaz, ki je predstavljen s tabelo števil, vsako število pa je predstavljeno z osmimi biti (enim bajtom). Terminalu lahko APDU odgovor posredujemo kot parameter, ki ga vrača metoda *processCommandApdu*, ali pa s klicem podedovane metode *sendResponseApdu*, kjer kot parameter metodi podamo kar APDU odgovor. Ker pričakujemo, da bo naša izvedba navidezne pametne kartice, zaradi latence internetne povezave nekoliko počasnejša, bomo APDU odgovore terminalu posredovali s pomočjo metode *sendResponseApdu*, metoda *processCommandApdu* pa v tem primeru ne bo vračala ničesar, oziroma vrednost *null*.

Naša implementacija metode *processCommandApdu* bo zelo preprosta. Ustvarila bo objekt tipa *ExecuteApduOnCloudTask*, ki bo preko protokola HTTPS strežniku poslal APDU ukaz in počakal na njegov odgovor. Odgovor

bo nato vrnil HCE storitvi. Ker bo objekt tipa *ExecuteApduOnCloudTask* izvajal dolgotrajnejšo operacijo, je pomembno, da se izvaja v lastni niti. Odgovor strežnika bomo tako pridobili v drugi niti kot teče HCE storitev, zato potrebujemo poseben mehanizem za obveščanje storitve o prispelem APDU odgovoru.

Najenostavnejša rešitev za obveščanje storitev o dogodkih je uporaba objektov tipa *Intent* in *BroadcastReceiver*. Operacijski sistem Android omogoča komponentam, oddajanje objektov tipa *Intent*, ki jih nato prejme objekt tipa *BroadcastReceiver*. *Intent* igra v operacijskem sistemu Android več vlog, v tem primeru pa bomo objekt tega tipa uporabili za prenos manjše količine podatkov. Ko ustvarimo objekt tipa *Intent*, mu običajno podamo ime akcije, ki jo predstavlja, ter kratko informacijo, ki jo bo nosil. Glavno vlogo pri komunikaciji pa ima objekt tipa *BroadcastReceiver*. Prijavimo ga na prejemanje *Intentov* ena ali večih akcij. Razred *BroadcastReceiver* vsebuje abstraktno metodo *onReceive*, ki se sproži ob prejetju novega *Intenta* z akcijo na katero je objekt prijavljen. Za pošiljanje objekta tipa *Intent* pa uporabimo metodo *sendBroadcast* razreda *Context*, ki ji *Intent* podamo kot parameter, le-ta pa poskrbi, da bodo vsi objekti tipa *BroadcastReceiver*, ki *Intent* z dano akcijo pričakujejo, *Intent* tudi prejeli.

V našem primeru bo *Intent* vseboval APDU odgovor, predstavljal pa bo akcijo "BROADCAST RESPONSE APDU". *BroadcastReceiver*, ki je v našem primeru predstavljen kar kot anonimni razred znotraj HCE storitve, pa bo iz *Intenta* prebral APDU odgovor in ga s pomočjo metode *sendResponseApdu* posredoval terminalu.

Listing 5.5: Razred ExecuteApduOnCloudTask odgovoren za pošiljanje APDU ukaza na strežnik.

```
//ExecuteApduOnCloudTask.java
public class ExecuteApduOnCloudTask extends AsyncTask<byte[],
    Void, Void>
{
```

```
private Context mContext;

public ExecuteApduOnCloudTask(Context ctx)
{
    this.mContext = ctx;
}

@Override
protected Void doInBackground(byte[]... apdus)
{
    byte[] responseApdu = sendApduToCloud(apdus[0]);
    sendResponseToHceService(responseApdu);

    return null;
}

private byte[] sendApduToCloud(byte[] apdu)
{
    String requestApdu = Utils.byteArrayToHexString(apdu);

    SharedPreferences settings =
        mContext.getSharedPreferences("USER_DATA", 0);
    String jwt = settings.getString("JWT", "");

    HttpClient httpClient = new DefaultHttpClient();
    HttpPost httpPost = new HttpPost(Constants.CLOUD_URL +
        "/execute");
    httpPost.setHeader("Content-type", "text/plain");
    httpPost.setHeader("Authorization", jwt);
    StringEntity entity = new StringEntity(requestApdu,
        "UTF-8");
    httpPost.setEntity(entity);
```

```
        HttpResponse response = httpClient.execute(httpPost);
        String responseApdu = response.getEntity().getContent();
        return Utils.hexStringToByteArray(responseApdu);
    }

    private void sendResponseToHceService(byte[] responseApdu)
    {
        Intent intent = new Intent("BROADCAST_RESPONSE_APDU");
        intent.putExtra("RESPONSE_APDU",
            Utils.byteArrayToHexString(responseApdu));
        mContext.sendBroadcast(intent);
    }
}
```

Ponastavitev seje virtualne kartice

Druga abstraktna metoda se imenuje *onDeactivated*. Izvede se v dveh primerih. V prvem primeru se izvede, ko je NFC povezava prekinjena, bodisi izključitev NFC modula ali pa oddaljitev mobilne naprave od terminala. V drugem primeru pa se izvede kadar je operacijski sistem Android prepoznal nov "SELECT AID" ukaz, ki naslavlja drugo HCE storitev. Tu je pomembno poudariti, da se metoda *onDeactivated* ne izvede v primeru AID konflikta, če konflikt razrešimo z izbiro iste HCE storitve.

Kot sem že opisal v prvem poglavju, pametne kartice vsebujejo delovni spomin RAM, v katerem so shranjeni podatki vezani na trenutno sejo. Ker plastične kartice nimajo lastnega električnega napajanja, se vsebina delovnega spomina izgubi ob odmiku plastične kartice od terminala. S tem se tudi zaključi seja kartice. Naša navidezna pametna kartica bi lahko vzdrževala isto sejo skozi celoten obstoj, kar bi bilo lahko v veliko primerih prednost navideznih pametnih kartic. Ker pa se želimo držati standardov in omogočiti istočasno uporabo navideznih in plastičnih kartic na terminalih z že nameščeno programsko opremo, moramo zagotoviti, da se bodo podatki, vezani na sejo,

prav tako ponastavili na navidezni kartici. Metoda *onDeactivated* nam bo omogočila ravno to. Ob klicu metode bomo na strežnik poslali HTTPS GET zahtevek, ki bo virtualni kartici ponastavil vsebino v delovnem spominu.

Listing 5.6: Razred ResetSessionOnCloudTask odgovoren za ponastavitev seje virtualne kartice v oblaku.

```
//ResetSessionOnCloudTask.java
public class ResetSessionOnCloudTask extends AsyncTask<Void, Void,
    Void>
{
    private Context mContext;

    public ResetSessionOnCloudTask(Context ctx)
    {
        this.mContext = ctx;
    }

    @Override
    protected Void doInBackground(Void... voids)
    {
        SharedPreferences settings =
            mContext.getSharedPreferences("USER_DATA", 0);
        String jwt = settings.getString("JWT", "");

        HttpClient httpClient = new DefaultHttpClient();
        HttpGet httpGet = new HttpPost(Constants.CLOUD_URL +
            "/resetSession");
        httpGet.setHeader("Authorization", jwt);

        HttpResponse response = httpClient.execute(httpGet);
        return null;
    }
```

```
}
```

Listing 5.7: HCE storitev kot povezovalni člen med terminalom in oblakom.

```
// ApduProcessor.java
public class ApduProcessor extends HostApduService
{
    /**
     * Receives apdu response that are broadcasted from
         ExecuteApduOnCloudTask and sends it to NFC reader
     */
    private final BroadcastReceiver apduResponseFromCloudReceiver =
        new BroadcastReceiver()
    {
        @Override
        public void onReceive(Context context, Intent intent)
        {
            String action = intent.getAction();
            if (action.equals("BROADCAST_RESPONSE_APDU"))
            {
                String apduResponse =
                    intent.getStringExtra("RESPONSE_APDU");
                sendResponseApdu(Utils.hexStringToByteArray(apduResponse));
            }
        }
    };

    @Override
    public void onCreate()
    {
        super.onCreate();
        IntentFilter filter = new IntentFilter();
        filter.addAction("BROADCAST_RESPONSE_APDU");
```

```
        registerReceiver(apduResponseFromCloudReceiver, filter);
    }

    @Override
    public byte[] processCommandApdu(byte[] apdu, Bundle extras)
    {
       new
           ExecuteApduOnCloudTask(getApplicationContext()).execute(apdu);
    }

    @Override
    public void onDeactivated(int reason)
    {
        new
            ResetSessionOnCloudTask(getApplicationContext()).execute();
    }
}
```

5.2 Oblak

Nazadnje si oglejmo še strežniško aplikacijo, ki ima dostop do podatkovne baze, kjer so trajno shranjene virtualne kartice in uporabniški računi.

5.2.1 Varnostni element

Za prejemanje APDU ukazov in ponastavljanje seje virtualne kartice ima oblak izpostavljeni dve REST (krajše za Representational State Transfer) spletni storitvi. Prav tako potrebujemo storitev, ki ustvari uporabniški račun oziroma prijavi uporabnika v sistem, v primeru da uporabniški račun že ima.

Da bo komunikacija med mobilno napravo in strežnikom resnično varna moramo za aplikacijski protokol uporabiti protokol HTTPS [8]. Gre za protokol, ki se uporablja za varno komunikacijo v omrežjih kot je internet. Protokol za prenos podatkov pošilja HTTP zahtevke po kriptiranem kanalu, tako da poslanih podatkov med prenosom ni mogoče prebrati ali spremeniti. Na ta način ustvarimo varen komunikacijski kanal za pošiljanje APDU ukazov do našega varnostnega elementa.

Za uporabniško avtentikacijo smo poskrbeli z zahtevo po prijavi uporabnika v sistem pred uporabo aplikacije ter pošiljanjem JWT žetona ob vsakem dostopu do oblaka. Na ta način preprečimo uporabniku dostop do tujih kartic.

Ena od nalog varnostnega elementa je trajno hranjenje podatkov. Oblaku to omogočimo z uporabo podatkovne baze. Najpreprostejši način shranjevanja virtualnih kartic je s pomočjo serializacije, kjer objekt zapišemo v obliki niza ter ga shranimo v podatkovno bazo. Ko kartico ponovno potrebujemo iz podatkovne baze preberemo niz in ga z deserializacijo ponovno pretvorimo v objekt.

5.2.2 Implementacija

Kot sem omenil že na začetku diplomske naloge je implementacij standarda ISO/IEC 7816 [14] veliko. Različne implementacije zahtevajo različne da-

totečne strukture pametnih kartic, ki so si med sabo podobne, vendar pa se v podrobnostih kjub temu razlikujejo. Prav tako definirajo različne nabore ukazov in varnostne mehanizme. Pri sami implementaciji navidezne kartice bi se morali odločiti kateri standard želimo podpreti.

Naša rešitev predstavlja ogrodje, ki omogoča enostavno razširitev in prilagoditev na različne standarde.

Primer rešitve prikazuje vmesnik *CloudApduProcessor*, ki vsebuje metodo *processApdu*, katero bi morale implementirati APDU procesorji posameznih implementacij (recimo *CipurseApduProcesor* ali *MifareApduProcesor*, ...). Ob prejetem HTTPS zahtevku bi nato poiskali kartico uporabnika, ter glede na implementacijo kartice izbrali pravi procesnik.

Listing 5.8: Prikaz vmesnika CloudApduProcessor.

```
public interface CloudApduProcessor
{
    public byte[] processApdu();
}
```

Poglejmo si še kaj je potrebno za pravilno izvedbo celotne rešitve, ki ni odvisno od implementacije kartice in APDU procesnika. Kot vemo, potrebuje naša strežniška aplikacija dve spletni storitvi *execute* in *resetSession*. V obeh storitvah je potrebno validirati JWT žeton in preveriti ali je uporabnik res tisti za katerega se izdaja. Nato sledi pridobivanje kartice iz podatkovne baze. Kartico pridobimo tako, da sprva poiščemo uporabnika, le-ta pa ima shranjeno referenco na svojo kartico.

V spletni storitvi *execute* nato sledi obdelava APDU ukaza, ki nam vrne APDU odgovor. Pomembno je, da po obdelavi ne pozabimo shraniti kartice nazaj v podatkovno bazo, saj lahko APDU ukaz posodobi podatke na kartici. Na zadnje posredujemo rezultat ukaza še androidni aplikacijo kot HTTPS odgovor strežnika.

V spletni storitvi *resetSession* z namenom preglednosti odgovora androi-

dni aplikaciji ne posredujemo, čeprav bi v praksi to običajno storili. Namesto obdelave APDU zahtevka v metodi *resetSession* kartici le ponastavimo objekt, v katerem so shranjeni podatki o trenutni seji, ter jo shranimo nazaj v podatkovno bazo.

Listing 5.9: Prikaz implementacije spletnih storitev execute in resetSession.

```
@Path("/cloudApduProcessor")
public class CloudApduProcessorWebServices
{
    @Inject
    private CardManager cardManager;

    @POST
    @Path("execute/")
    @Consumes(MediaType.TEXT_PLAIN)
    @Produces(MediaType.TEXT_PLAIN)
    public String execute(@HeaderParam("Authorization") String jwt,
        String apduRequest)
    {
        User user = userManager.validateAndFindUser(jwt);
        Card card = cardManager.findCardForUser(user);

        byte apdu = Utils.hexStringToByteArray(apduRequest);

        CloudApduProcessor apduProcessor;

        if(card instanceof CipurseCard)
        {
            apduProcessor = new CipurseApduProcesor(card, apdu);
        }
        else if(card instanceof MifareCard)
        {
```

```
            apduProcessor = new MifareApduProcesor(card, apdu);
        }
        else
        {
            ...
        }

        byte[] responseApdu = apduProcessor.processApdu();

        cardManager.saveCard(card);

        return responseApdu;
    }

    @GET
    @Path("resetSession/")
    public void resetSession(@HeaderParam("Authorization") String
        jwt)
    {
        User user = userManager.validateAndFindUser(jwt);
        Card card = cardManager.findCardForUser(user);

        card.setCardRam(new CardRam());

        cardManager.saveCard(card);
    }

}
```

Poglavje 6

Zaključek

Naša rešitev predstavlja enostavno implementacijo virtualne kartice, ki se nahaja v oblaku. Aplikacija, nameščena na mobilni napravi, omogoča uporabniku prijavo v sistem in služi kot posredovalni člen med terminalom in oblakom.

Naša aplikacija omogoča uporabniku uporabo le ene virtualne kartice. Z malo spremembe pa bi uporabnik lahko v oblaku hranil mnogo kartic različnih standardov. Taka implementacija oblaka, ki bi omogočala uporabniku hraniti več kartic, bi lahko služila kot API razvijalcem androidnih aplikacij, ki potrebujejo enostavno rešitev varnostnega elementa.

Varnostni element v oblaku bi tako lahko uporabljalo veliko aplikacij, sama integracija pa bi bila tako prepuščena razvijalcem samim. Kot smo pokazali, je integracija v posamezno mobilno aplikacijo s pomočjo knjižnice HCE zelo preprosta. Vseeno pa je potrebno poudariti, da uporaba varnostnega elementa v oblaku ni vedno idealna rešitev za vse vrste problemov. Predstavljena rešitev ne bi bila primerna, kjer mobilen prenos podatkov ni mogoč ali pa mora biti čas od oddanega APDU ukaza do prejema odgovora minimalen.

Kljub temu pa obstaja veliko primerov uporabe, kjer čas ni ključnega pomena. HCE knjižnica odpira nove možnosti implementacije navideznih pametnih kartic. Z vedno večjo dostopnostjo do hitrega mobilnega interneta,

postaja naša rešitev implementacije v oblaku bolj uporabna in privlačna.

Literatura

[1] Android dokumentacija za datoteko AndroidManifest.xml. http://developer.android.com/guide/topics/manifest/manifest-intro.html.

[2] Android dokumentacija za razred Activity. http://developer.android.com/reference/android/app/Activity.html.

[3] Android dokumentacija za razred AsyncTask. http://developer.android.com/reference/android/os/AsyncTask.html.

[4] Android dokumentacija za razred HostApduService. https://developer.android.com/reference/android/nfc/cardemulation/HostApduService.html.

[5] Android dokumentacija za razred Service. http://developer.android.com/reference/android/app/Service.html.

[6] Android KitKat 4.4. https://developer.android.com/about/versions/kitkat.html.

[7] Calypso functional specification. https://www.calypsonet-asso.org/sites/default/files/010608-NT-CalypsoGenSpecs15.pdf.

[8] Dokument RFC 2818. http://tools.ietf.org/html/rfc2818.

[9] Dokument RFC 7519. https://tools.ietf.org/html/rfc7519.

[10] Host-based Card Emulation. https://developer.android.com/guide/topics/connectivity/nfc/hce.html.

[11] MiFare Protocol Guide. http://www.metratec.com/fileadmin/docs/en/documentation/metraTec_MiFare_Protocol-Guide.pdf.

[12] Standard ISO/IEC 14443. http://en.wikipedia.org/wiki/ISO/IEC_14443.

[13] Standard ISO/IEC 7810. http://en.wikipedia.org/wiki/ISO/IEC_7810.

[14] Standard ISO/IEC 7816. http://en.wikipedia.org/wiki/ISO/IEC_7816.

[15] Smart Card Alliance. Host card emulation (hce) 101. 2014.

[16] Laurent Cremer. Cipurse: An open standard for next-generation fare collection solutions. In *IT-Trans, IT Solutions for Public Transport*, number Session 1, 2012.

[17] Michael Jones, Paul Tarjan, Yaron Goland, Nat Sakimura, John Bradley, John Panzer, and Dirk Balfanz. Json web token (jwt). 2012.

[18] Marie Reveilhac and Marc Pasquet. Promising secure element alternatives for nfc technology. In *Near Field Communication, 2009. NFC'09. First International Workshop on*, pages 75–80. IEEE, 2009.

[19] M. Zorec and V. Mahnič. *Implementacija brezstičnih pametnih kartic na napravah NFC: diplomsko delo.* M. Zorec, 2014.