FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Knihovna pro komunikaci s NFC kartami
Student:	Maroš Popovič
Vedoucí:	Ing. Zdeněk Balák
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je vytvořit opensource knihovnu, která usnadní komunikaci s NFC kartami od výrobce NXP pro platformu Android. Nativní řešení funguje na principu výměny bytového pole, kde každý bit vyjadřuje nějakou vlastnost popsanou v dokumentaci jednotlivých karet.

- Finální řešení by mělo uživateli knihovny poskytnout mnohem konkrétnější a pohodlnější rozhraní a ušetřit ho vlastní implementaci protokolu.
- Knihovna musí fungovat i v offline podmínkách (nebude obsahovat žádnou online licenci nebo jakoukoliv komunikaci přes internet)
- Knihovna poskytne základní funkce (zapsat data, přečíst data, smazat data)

1. Seznamte se s nativním rozhraním pro komunikaci s NFC kartami na platformě Android
2. Udělejte rešerši existujících řešení.
3. Naimplementujte řešení pro karty Mifare Plus EV1 a Mifare Desfire EV1
4. Řešení řádně otestujte

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 11. února 2022 v Praze.

Bakalárska práca

KNIHOVNA PRO KOMUNIKACI S NFC KARTAMI

Maroš Popovič

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedúci: Ing. Zdeněk Balák
9. mája 2022

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov kódu

Ďakujem Ing. Zdeňku Balákovi za vedenie tejto práce, priateľský prístup, a taktiež za to, že si našiel čas na spoločné stretnutia, z ktorých som vždy odchádzal o niečo múdrejší. Ďakujem Quanti s. r. o. za príležitosť pracovať na zmysluplnej bakalárskej práci. Ďakujem Bc. Milošovi Popovičovi, Jakubovi Kuchejdovi a Robinovi Pospíšilovi za spätnú väzbu a za podporu pri tvorbe práce. Ďakujem mojej rodine za podporu pri štúdiu.

Vyhlásenie

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 9. mája 2022 .....

Abstrakt

Cieľom tejto bakalárskej práce je návrh a implementácia open-source knižnice na komunikáciu s NFC kartami od výrobcu NXP pre mobilný operačný systém Android. Finálne riešenie práce je schopné komunikovať s kartou MIFARE DESFire EV1. Ovláda základné príkazy na manipuláciu a šifrovanie dát. Ďalšie príkazy sú jednoducho rozšíriteľné.

Knižnica je vytvorená v jazyku Kotlin. Konkrétne príkazy, komunikujúce s kartou, sú implementované podľa súboru príkazov špecifických pre MIFARE DESFire EV1. Tento súbor príkazov je založený na výmene dát vo forme bajtového poľa.

Prínosom práce je uľahčenie používania týchto príkazov. Užívateľ knižnice nepotrebuje komunikovať pomocou výmeny bajtového poľa, a taktiež nemusí mať o konkrétnych príkazoch podrobnú znalosť. Zároveň je zabezpečené to, že na plnú funkcionalitu knižnice nepotrebuje zariadenie pripojenie k internetu.

Klíčová slova vývoj open-source knižnice, offline použiteľnosť, NFC komunikácia, Android, MIFARE DESFire EV1, IsoDep, Kotlin

Abstract

The goal of this bachelor's thesis is design and implementation of open-source library for communication with NFC cards from the manufacturer NXP for mobile operating system Android. Final solution of the thesis is able to communicate with card MIFARE DESFire EV1. It consists of basic commands that manipulate and encrypt data. Additional commands are easily expandable.

The library is created in Kotlin programming language. Specific commands, communicating with the card, are implemented by the command set defined for MIFARE DESFire EV1. This command set is based on data transfer in the form of byte array.

The contribution of this thesis is to make the usage of the command set easier. User of the library doesn't have to transfer data in the form of byte array. Also, the user does not need to have detailed knowledge about particular commands. The library doesn't require any internet connection in order to be fully used.

Keywords open-source library development, offline usability, NFC communication, Android, MIFARE DESFire EV1, IsoDep, Kotlin

Zoznam skratiek

SDK	Software Development Kit
RFID	Radio Frequency Identification
NFC	Near Field Communication
NDEF	NFC Data Exchange Format
URI	Uniform Resource Identifier
MIME	Multipurpose Internet Mail Extensions
AES	Advanced Encryption Standard
DES	Data Encryption Standard

Úvod

Využívanie bezkontaktných čipových kariet sa v dnešnej dobe považuje v mnohých situáciách za samozrejmosť. Stretneme sa s nimi počas platby v obchodoch, cestovania mestskou hromadnou dopravou či pri elektronickom odomykaní dverí. Bezkontaktné čipové karty ponúkajú riešenie, ktoré poskytuje rýchly a šifrovaný prenos dát s dôrazom na ochranu osobných údajov užívateľa. K ich rastúcej popularite prispela aj pandémia ochorenia COVID-19, počas ktorej bola bezkontaktná komunikácia odporúčaným preventívnym opatrením.

Bezkontaktné čipové karty sú hlavnou tématikou tejto práce. Práca sa zaoberá analýzou, návrhom a implementáciou knižnice pre komunikáciu Android zariadenia s NFC kartou MIFARE DESFire EV1. Je určená pre vývojárov, ktorí s touto knižnicou môžu vyvíjať aplikácie komunikujúce s kartou, bez potreby hlbšej znalosti jej komunikačného protokolu. Výstup práce zároveň pomôže s tvorbou aplikácií, ktoré si vyžadujú „offline“ podmienky, ako napríklad v rozvojových krajinách, kde pripojenie k internetu nebýva samozrejmosťou.

V prvej časti bude analyzovaná technológia NFC a jej využitie v operačnom systéme Android. Taktiež bude analyzovaný komunikačný protokol karty MIFARE DESFire EV1, ktorého príkazy podrobne rozoberieme v daných podkapitolách. Druhá časť obsahuje rozbor existujúcich riešení, vrátane knižnice od firmy NXP. Jej nedostatky v práci s „offline“ podmienkami boli motiváciou pre vznik tejto práce a vytvorenie vlastnej knižnice v programovacom jazyku Kotlin. Týmto bude uzavretá teoretická časť a nasledovať bude praktická časť, v ktorej si popíšeme návrh, implementáciu a testovanie vytvorenej knižnice.

Aktuálnosť riešenia spočíva v tom, že je plne funkčné v podmienkach bez internetu a jeho používanie si nevyžaduje žiadnu licenciu. Zároveň je vytvorené v jazyku Kotlin, ktorý je momentálne odporúčaný jazyk na vývoj Android aplikácií. Je modernejšou alternatívou k jazyku Java, ktorá bola dlhodobo jedinou technológiou na vývoj pre Android.

Kapitola 1
Ciele práce

Cieľom práce je vytvoriť open-source knižnicu, ktorá uľahčí komunikáciu s NFC kartami pre platformu Android. Natívna komunikácia funguje na základe výmeny bajtového poľa, pomocou ktorého sa posielajú jednotlivé inštrukcie, popísané v dokumentácii kariet.

Knižnica musí poskytovať základné inštrukcie na komunikáciu s kartou, ako je zápis dát, čítanie dát a odstránenie dát. Taktiež musí byť schopná plne fungovať aj bez pripojenia na internet. Riešenie má byť schopné komunikovať s kartami typu MIFARE DESFire EV1 a MIFARE Plus EV1. Celkový výsledok práce má za cieľ zjednodušiť vývoj Android aplikácií, ktoré potrebujú komunikovať s týmito NFC kartami.

V rámci dosiahnutia týchto cieľov je nutné analyzovať ako funguje technológia NFC pre Android a akým spôsobom je možné s kartami komunikovať. Taktiež je úlohou vytvoriť rešerš existujúcich knižníc. Najdôležitejšou úlohou je na základe tejto analýzy navrhnúť, implementovať a otestovať finálne riešenie.

Kapitola 2
Analýza

2.1 Využité technológie

Pozrieme sa na technológie, ktoré úzko súvisia s témou bakalárskej práce. Začneme so všeobecnými pojmami ako RFID, NFC či bezkontaktná čipová karta. Postupne sa dostaneme ku konkrétnejším konceptom, ako je štruktúra NFC kariet značky MIFARE a rozhranie NFC pre Android.

2.1.1 RFID

Radio Frequency Identification alebo Vysokofrekvenčná identifikácia je technológia, ktorá vznikla za účelom rozpoznávania tovaru. Jedná sa konkrétne o štítok používajúci elektromagnetické pole. Štítok po aktivácii impulzom z RFID čítačky prenesie svoje uložené informácie, pričom väčšinou ide o unikátne identifikačné číslo, patriace konkrétnemu tovaru. [1, 2]

Táto technológia nadväzuje ako rozšírenie na systém čiarových kódov. Na rozdiel od čiarových kódov, RFID štítok nemusí byť pri skenovaní viditeľný, prenos dát funguje na väčšiu vzdialenosť a obsah štítku sa dá jednoducho aktualizovať. Nevýhodou je, že výroba týchto štítkov je drahšia. Vývoj technológie RFID iniciovala spoločnosť WalMart. [2]

Obr. 2.1 Bezpečnostný RFID štítok [3]

Obr. 2.2 Bezkontaktná platba kartou [4]

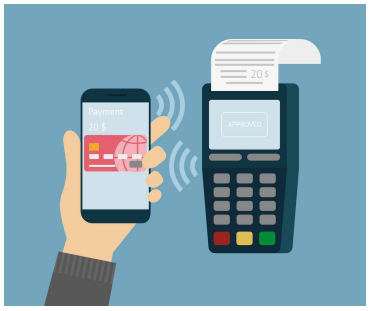
- V súčasnej dobe sa RFID využíva napríklad na zabezpečenie tovaru v obchodoch. Etiketa je nalepená priamo na tovar, pričom v prípade krádeže sa po prechode etikety zabezpečovacím systémom spustí alarm. [5]

- Ďalším príkladom použitia RFID je bezkontaktná platba. Platobné karty ako MasterCard a VISA obsahujú RFID čip, ktorý vlastníkovi karty umožňuje vykonať platobnú transakciu jednoduchým priblížením karty ku platobnému terminálu. Nahradzuje to potrebu vkladania karty do terminálu. [6]

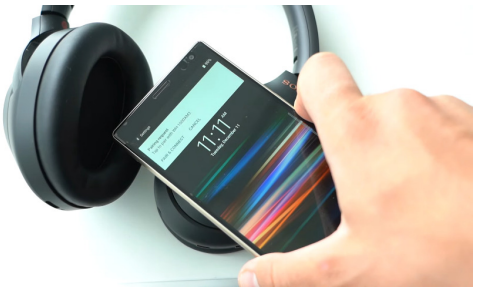
- RFID štítky majú svoje využitie aj mimo obchodnej sféry. Napríklad na stopovanie času bežcov počas hromadných športových podujatí, na sledovanie kufrov na letisku či na odomykanie inteligentných zámkov. [7]

2.1.2 NFC

Near Field Communication alebo skrátene NFC je technológia, ktorá umožňuje rýchlu bezdrôtovú výmenu dát medzi dvoma zariadeniami v bezprostrednej blízkosti – do 10 centimetrov. Je rozšírením technológie RFID a zároveň kombinuje rozhrania[1] čipových kariet a ich čítačiek do jedného. Zaručuje spätnú kompatibilitu s už existujúcimi riešeniami, vzniknutými na základe RFID. NFC sa využíva primárne v mobilných zariadeniach. [8]



Obr. 2.3 Platba mobilom využívajúca NFC [9]



Obr. 2.4 Párovanie mobilu so slúchadlami nablízko pomocou NFC [10]

- Najrozšírenejším využitím je bezkontaktná platba v obchodoch, kde sa použitím mobilného telefónu dá zaplatiť pomocou tzv. **emulácie karty**. Telefón sa v tomto prípade správa ako fyzická platobná karta. Jedná sa o pohodlnú alternatívu k plateniu v hotovosti či kartou, ktorá môže byť aj bezpečnejšia. Android aj Apple ponúkajú, v prípade krádeže telefónu, možnosť odstránenia platobných údajov na diaľku. [8, 11, 12]

- Ďalší typ NFC využitia je vhodný na **komunikáciu s NFC štítkom**, najčastejšie v podobe zápisu do NFC karty, alebo čítania z NFC karty. Aktívne zariadenie čaká na kontakt s NFC štítkom. Po priložení štítku prečíta informácie v ňom uložené, spracuje ich a následne na nich reaguje. Jedná sa napríklad o odomknutie hotelovej izby pomocou čipovej karty namiesto klasického kľúča alebo o dobíjanie kreditu na čipovej karte mestskej hromadnej dopravy. Tomuto typu využitia sa venuje táto bakalárska práca. [8, 12]

- V neposlednom rade ponúka NFC technológia tzv. **peer to peer mode** – režim rovného s rovným. Využíva sa na prenos informácií medzi dvoma NFC zariadeniami. Jedná sa napríklad o odoslanie súboru z jedného mobilného telefónu do druhého. V modernejších modeloch slúchadiel sa tento režim dá použiť na urýchlenie procesu prepojenia s mobilným telefónom. Rýchly „dotyk“ týchto NFC zariadení nahradí potrebu vyhľadávania slúchadiel v nastaveniach mobilného telefónu. NFC zabezpečí spárovanie, prenos zvuku už prebieha využitím technológie Bluetooth. [12]

[1] Norma ISO/IEC 14443-3

2.1.3 Bezkontaktná čipová karta

Bezkontaktná čipová karta je karta, ktorá na komunikáciu s čítačkou nevyžaduje fyzický kontakt. Pozostáva z čipu a antény, zalisovanými do plastového obalu. Má svoje uplatnenie v situáciách, ktoré si vyžadujú rýchly, šifrovaný prenos informácií či bezpečné uchovanie osobných údajov: v preprave mestskou hromadnou dopravou, kde funguje ako elektronická peňaženka, v kreditných a debetných kartách na uskutočňovanie bezkontaktnej platby, alebo na miestach, kde je potrebná identifikácia osôb. Ich popularita neustále rastie, pričom tento nárast zapríčinila aj pandémia koronavírusu, počas ktorej bolo v rámci prevencie silne odporúčané nahradiť platbu v hotovosti jej bezkontaktnou alternatívou. Bezkontaktné čipové karty obsahujú aj identifikačný RFID čip.

Nízko-úrovňová bezkontaktná komunikácia medzi kartou a čítačkou je definovaná podľa normy ISO/IEC 14443, ktorej rozhranie podporuje aj technológia NFC. Túto normu podporuje 80% všetkých bezkontaktných čipových kariet. Najrozšírenejšie typy kariet podporujú taktiež normu ISO 7816-4, ktorá definuje súbor príkazov na prácu s kartou – zápis, čítanie či odstránenie dát. ISO 7816-4 pôvodne vznikla len pre klasické, „kontaktné“, karty. Táto práca sa venuje kartám od výrobcu MIFARE, ktoré okrem podpory ISO 7816-4 definujú vlastné príkazy. Jedná sa o výrobcu jedných z najrozšírenejších typov bezkontaktných čipových kariet. [13, 14, 15, 16, 17, 18, 19]

Obr. 2.5 Symbol čítačky bezkontaktných čipových kariet [20]

2.1.4 Čipy MIFARE

MIFARE je obchodná značka mikročipov vlastnená Holandskou firmou NXP Semiconductors. Tieto mikročipy obsahujú technológiu na bezkontaktnú komunikáciu, využívanú v bezkontaktných čipových kartách MIFARE. Používa šifrovacie štandardy AES, DES a dnes už prelomený Crypto1. Výrobca tvrdí, že produkty sú spoľahlivé a časom osvedčené, čo potvrdzuje aj fakt, že do roku 2022 bolo predaných viac ako 12 miliárd rôznych druhov kariet s bezkontaktným mikročipom, využívaných v infraštruktúrach vo viac ako 750 mestách po celom svete. Do dedikovanej online komunity pre MIFARE sa pridalo tisíce obchodných partnerov. MIFARE produkty ponúkajú nové riešenia, pričom každá nová verzia čipovej karty je spätne kompatibilná s jej predchodcami. V prípade potreby modernizácie existujúcej infraštruktúry zákazníka, sa jedná o značné uľahčenie prechodu z nižšej úrovne na vyššiu. Počas svojej úspešnej existencie sa stali MIFARE produkty riešením s dobrou reputáciou medzi vývojármi aplikácií, ponúkajúce funkčné a flexibilné možnosti práce s bezkontaktnými technológiami. Značka svoj dosah postupne rozširuje aj v rámci mobilných aplikácií pre telefóny podporujúce NFC či v oblasti cloud služieb. Nízko-úrovňová komunikácia bezkontaktných čipových kariet od MIFARE je v súlade s normou ISO/IEC 14443. [19]

2.1.4.1 MIFARE Classic

Bezkontaktná čipová karta MIFARE Classic je zariadenie, ktoré funguje ako jednoduché pamäťové úložisko. Karta disponuje niekoľkými algoritmami na riadenie prístupu k uloženým informáciám. Z dôvodu spoľahlivosti, kombinovanou s nízkymi nákladmi na výrobu, je karta používaná v mnohých mestských infraštruktúrach. Milióny čipov typu MIFARE Classic vieme nájsť napríklad v rámci mestskej hromadnej dopravy v Európe, Amerike či Ázii. [21]

2.1.4.1.1 Prelomenie šifry Crypto1: Problémy s kartou nastali približne v roku 2008, keď Holandský parlament vydal varovanie pred používaním bezpečnostných kľúčov, implementovaných v čipe MIFARE Classic. Stalo sa tak po tom, čo nemeckí experti na kryptografiu Henryk Plötz a Karsten Nohl prelomili Crypto1 – šifru vytvorenú priamo pre MIFARE. Crypto1 tak bolo možné prelomiť v priebehu niekoľkých minút na klasickom stolnom počítači. Ďalším problémom bolo generovanie náhodného čísla, ktoré karta generuje a využíva počas šifrovanej komunikácie. Tento generátor bol veľmi jednoducho manipulovateľný tak, aby po každom vyžiadaní nového náhodného čísla, vrátil číslo presne zvolené hackerom, čo má za následok porušenie bezpečnosti. O nejaký čas na to skupina výskumníkov z vysokej školy Radboud University v holandskom meste Nijmegen dokázala pomocou metód reverzného inžinierstva prelomiť a skopírovať kompletný obsah čipovej karty, využívajúcej čip MIFARE Classic. S obsahom karty dokázali jednoducho manipulovať. Taktiež bolo možné ho naklonovať do inej karty, čo predstavovalo napríklad možnosť duplikovať karty slúžiace na bezkontaktné odomykanie hotelových izieb. Firma NXP preto oficiálne odporúča kartu MIFARE Classic vymeniť za kartu s vyšším stupňom bezpečnosti, ako napríklad jeho potomka MIFARE Plus EV1 či MIFARE DESFire EV1. [21, 22, 23]

2.1.4.2 MIFARE DESFire EV1

MIFARE DESFire EV1 je bezkontaktná čipová karta, ktorá bola vytvorená v roku 2008. V dnešnej dobe sa jedná o jeden z najrozšírenejších typov čipových kariet. Pomenovanie „DESFire“ naznačuje jej hlavné charakteristiky. „DES“ značí vylepšenú úroveň bezpečnosti v podobe algoritmov 3DES alebo AES, slúžiace na šifrovanie prenášaných dát. Autentifikácia užívateľa pozostáva z trojpriechodového kryptografického protokolu. Slovo „Fire“ je obrazným pomenovaním, znamenajúcim pokrok v rýchlosti, spoľahlivosti a bezpečnosti karty. Oproti predošlým generáciám kariet, môže DESFire EV1 slúžiť držiteľovi karty na viac účelov naraz. Jediné zariadenie môže využívať na cestovanie v mestskej hromadnej doprave, stravovanie sa v jedálni či autorizáciu na získavanie prístupu do miestností. DESFire čip disponuje systémom na zálohovanie dát, ktoré môžu byť po novom ukladané do 32 rôznych súborov v 28 rôznych aplikáciách. Zvýšenú flexibilitu pamäti na karte zaručuje aj to, že veľkosti súborov sú definované až v momente ich vytvárania. Čip taktiež obsahuje mechanizmus, ktorý je schopný po detekcii straty spojenia karty s bezkontaktnou čítačkou zachovať integritu uložených informácií – buď čiastočným uložením dát, alebo úplným zneplatnením prerušenej transakcie. Samozrejmosťou je aj plná kompatibilita s čítačkami využívajúcimi technológiu NFC. [13, 24]

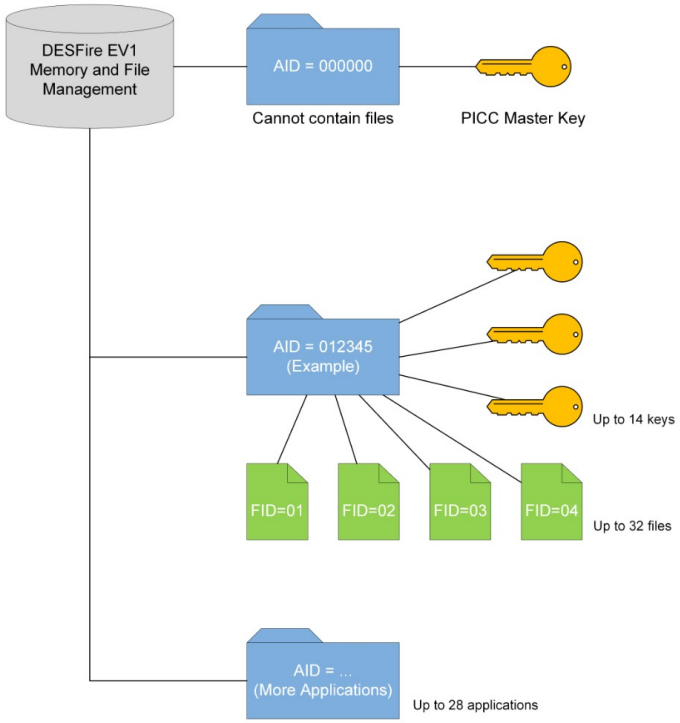
2.1.4.2.1 Organizácia pamäti: Na karte je možné vytvoriť niekoľko aplikácií. Do aplikácií sa dajú ukladať rôzne typy súborov, ako napríklad hodnotový, dátový či záložný. [24]

- **Hodnotový súbor**[2] má v sebe uložené jediné číslo, ktoré môže nadobúdať rôzneho rozsahu hodnôt v závislosti od typu karty. [25]
- **Dátový typ súboru**[3] umožňuje užívateľovi zapísať do súboru ľubovoľné dáta, preto sa podľa mňa javí ako najlepší variant. Dáta musia byť vo forme sekvencie bajtov. [25]

[2] Value file
[3] Standard data file

- Rozšírením dátového typu súboru je **záložný dátový súbor**,[4] ktorý si taktiež ukladá dáta ako sekvenciu bajtov. Ich zápis si vyžaduje extra potvrdzovací príkaz, ktorý potvrdí úplnosť odoslaných dát. Po potvrdení sa dáta považujú za platné. V prípade, že potvrdenie nepríde, napríklad z dôvodu prerušenia spojenia medzi kartou a čítačkou, je zápis neplatný a obsah dát na karte sa nezmení. Záložný dátový súbor si oproti štandardnému dátovému súboru vyžaduje dvojnásobne viac pamäti na karte. [25]

Dáta na karte sa dajú prepisovať, no nevýhodou je, že sa nedajú jednoducho odstrániť. V prípade, že na karte existuje nežiadúci súbor, je možné ho príkazom na „odstránenie“ len deaktivovať. Deaktivácia zakáže používanie súboru, no neuvoľní pamäť, ktorá mu bola pôvodne pridelená. Na znovupoužitie tejto pamäti je potrebné odstrániť kompletný obsah karty príslušným príkazom, ktorý ju vráti do výrobného nastavenia. [25]



Obr. 2.6 Štruktúra čipovej karty MIFARE DESFire EV1. Popisuje možnosť tvorby aplikácií, súborov a ich kľúčov. [26]

2.1.4.2.2 Využitie v praxi: Príkladom karty typu MIFARE DESFire EV1 je moja bezkontaktná čipová karta, používaná na prepravu v mestskej hromadnej doprave mesta Košice. Preskúmaním obsahu sa zistilo, že táto karta po dlhodobom používaní obsahuje 5 aplikácií, 3 hodnotové súbory, 27 dátových súborov a 5 záložných dátových súborov. Po uskutočnení a zaplatení dvoch ciest touto kartou nevznikla žiadna nová aplikácia ani nový súbor. Jediná zmena, ktorú bolo možne odpozorovať, bola zmena nastavenia v jednom hodnotovom súbore, upravujúceho maximálny limit hodnoty z 80 na 150. Súbory majú obsah prístupný na kľúč, takže ich podrobnejšie zmeny nie sú známe.

Ďalším príkladom je moja karta ISIC od FIT ČVUT, ktorá slúži na identifikáciu v priestoroch školy či na využívanie študentských výhod v rôznych organizáciách. Táto karta, na rozdiel od

[4] Backup data file

predošlej, neobsahuje ani po 3 rokoch aktívneho používania žiadne užívateľské dáta. Má len jednu základnú aplikáciu, ktorú získavajú všetky naformátované karty v rámci výrobného nastavenia.[5] Myslím si, že to má dobré opodstatnenie: Na kartách môže z dôvodov obmedzenej veľkosti pamäti vzniknúť len limitovaný počet aplikácií a súborov. Z tohoto dôvodu sa pri práci s kartou môžu niektoré organizácie vyhýbať zápisu svojich údajov priamo do karty. Namiesto toho využívajú unikátne číslo, ktoré obsahuje RFID čip na každej karte, slúžiace na identifikáciu vlastníka karty. Údaje spojené s vlastníkom potom neukladajú do karty, ale do svojich databáz, spolu s týmto číslom. Má to výhodu aj pri strate či poškodení čipovej karty. [14, 24]

2.1.4.3 MIFARE Plus EV1

Príchod tejto bezkontaktnej čipovej karty na trh bol oznámený v roku 2016. Karta bola propagovaná ako vhodná náhrada za MIFARE Classic, zaručujúca jednoduchý a lacný prechod na riešenie s vyššou bezpečnosťou. Plus EV1 umožňuje migráciu systémov, založených na šifrovaní pomocou Crypto1, na systémy zabezpečené pomocou kryptograficky bezpečnejšej šifry AES. Táto špeciálna funkcionalita zabezpečuje to, že zákazníkom stačí zakúpiť si karty MIFARE Plus EV1 bez potreby aktualizácie zvyšku infraštruktúry. Napríklad čítačky vytvorené priamo pre MIFARE Classic ostávajú čiastočne kompatibilné aj s MIFARE Plus. [21, 27, 28] Z tohoto dôvodu obsahuje karta 3 bezpečnostné úrovne:

- **Bezpečnostná úroveň 0:** V tomto stave sa nachádza každá novo vyrobená karta. Na úrovni 0 je potrebné nastaviť algoritmy kľúčov, používaných na šifrovanie karty. Je možné zvoliť algoritmus Crypto1, používaný v MIFARE Classic, alebo bezpečnejší AES, používaný v moderných infraštruktúrach. Taktiež je možné konfigurovať užívateľské dáta. Na konci konfigurácie karty sa dá pomocou príkazu „CommitPerso“ aktualizovať kartu na bezpečnostnú úroveň 1, alebo priamo na bezpečnostnú úroveň 3. [28]

- **Bezpečnostná úroveň 1:** Táto úroveň využíva Crypto1, pričom nezaisťuje úplnú bezpečnosť. Voliteľne ponúka rozšírenie, ktoré zaisťuje bezpečnú autentifikáciu medzi čítačkou a kartou – umožňuje čítačke overiť, že karta nie je falzifikát a skutočne patrí do systému. [28]

- **Bezpečnostná úroveň 3:** Zabezpečuje plnú bezpečnosť – na šifrovanie dát a autentifikáciu využíva šifru AES. Táto úroveň nepodporuje spätnú kompatibilitu s MIFARE Classic infraštruktúrou. [28]

2.1.4.3.1 Rozdelenie do sektorov: Pamäť MIFARE Plus EV1 sa dá rozdeľovať do sektorov. Každý sektor môže byť nastavený na inú bezpečnostnú úroveň. Aplikácie v jednom sektore môžu byť zabezpečené pomocou Crypto1 – na úrovni 1, a aplikácie v inom sektore môžu využívať AES – na úrovni 3. Toto rozdelenie umožňuje spätnú kompatibilitu so staršími čítačkami, ktoré podporujú len Crypto1. Citlivé údaje môžu byť kedykoľvek aktualizované na vyššiu bezpečnostnú úroveň. Táto funkcionalita má zjednodušiť postupnú migráciu zo systémov závislých na rozhraní MIFARE Classic na systémy podporujúce MIFARE Plus. Cieľom je aj minimalizácia nákladov na túto migráciu. [21, 27, 28]

2.1.4.3.2 Riešenie v rámci tejto bakalárskej práce: Pôvodným cieľom tejto bakalárskej práce bolo vytvorenie knižnice s príkazmi pre MIFARE DESFire EV1 a MIFARE Plus EV1. Po návrhu vedúceho práce boli príkazy pre MIFARE Plus EV1 vynechané. Dôvodom bola nízka potreba pre tieto príkazy a zameranie na typ MIFARE DESFire EV1 si vyžadovalo väčšiu pozornosť. Avšak, návrh výslednej knižnice umožňuje jednoduché pridávanie príkazov, takže budúce rozšírenie o príkazy MIFARE Plus EV1 nie je problémové.

[5] Jedná sa o aplikáciu s číslom 0x000000, viz obrázok 2.6

2.1.4.4 MIFARE DESFire EV3

Jedná sa o najmodernejšiu bezkontaktnú čipovú kartu typu DESFire, vydanú v roku 2020. Je spätne kompatibilná so svojimi predkami a ponúka nové, rýchlejšie a bezpečnejšie riešenia pre bezkontaktnú komunikáciu, ako napríklad:

- Súbor nových kryptografických algoritmov,
- zabudovaný časovač transakcie – môže napríklad zabrániť hackerskému útoku „man-in-the-middle“,
- maximálny počet aplikácií je limitovaný len veľkostou pamäti. [29]

2.1.5 Android NFC

Android umožňuje zariadeniam podporujúcim NFC zdieľať informácie malej veľkosti medzi zariadením a NFC štítkom či medzi dvoma rôznymi Android zariadeniami. Tieto informácie môžu byť uložené v širokom spektre formátov. Najpodporovanejší formát pre platformu Android, používaný v NFC komunikácii sa nazýva NDEF – NFC Data Exchange Format. NDEF je štandardizovaný formát, ktorý má za cieľ zjednodušiť prenos ľubovoľných dát. Obrázky, šifrované dáta či rôzne dokumenty dokáže zapuzdriť a odoslať v rámci NDEF správy. Jedná sa o odporúčaný formát pre NFC komunikáciu v Androide. [30, 31, 32]

2.1.5.1 Podporované technológie NFC štítkov

Android NFC podporuje rôzne technológie, ktoré poskytujú nízko-úrovňovú komunikáciu so štítkom podľa istej normy. Každý NFC štítok natívne podporuje aspoň jednu z týchto noriem. Technológie sú implementované vo vlastnej triede a volaním ich metód je možné komunikovať so štítkami. Každá táto trieda dedí z interface `TagTechnology` a musí implementovať jeho abstraktné metódy `connect()` a `close()`. Metóda `connect()` povolí vstupno-výstupné operácie so štítkom, metóda `close()` tieto operácie deaktivuje a uvoľní prostriedky. Metódu, ktorá odosiela a prijíma dáta, interface `TagTechnology` nedefinuje, no väčšina tried, ktoré z `TagTechnology` dedia, ju poskytujú samostatne. Napríklad trieda `IsoDep` obsahuje metódu `transceive(byte[])`, ktorá odošle do štítku pole bajtov a obdrží od neho odpoveď, taktiež vo forme bajtového poľa. Práca so štítkom, ktorého komunikácia je kompatibilná s triedou `IsoDep`, teda často spočíva v sérii príkazov v tomto poradí: Pripojenie pomocou `connect()`, odoslanie a prijatie dát pomocou metódy `transceive(byte[])`, ktorá môže byť volaná viac-krát, a napokon uzavretie spojenia pomocou `close()`. [31, 33] Triedy, ktoré poskytujú nízko-úrovňovú komunikáciu so štítkom:

- `NfcA` - Poskytuje vstupno-výstupné operácie podľa normy NFC-A – ISO 14443-3A
- `NfcB` - Poskytuje vstupno-výstupné operácie podľa normy NFC-B – ISO 14443-3B
- `IsoDep` - Poskytuje vstupno-výstupné operácie podľa normy ISO-DEP – ISO 14443-4
- `Ndef` - Poskytuje operácie pre štítky, ktoré sú formátované na prácu s NDEF dátami
- `NdefFormatable` - Poskytuje operácie na naformátovanie štítku na prácu s NDEF dátami
- `MifareClassic` - Poskytuje vstupno-výstupné operácie pre MIFARE Classic. Na rozdiel od predošlých tried, nemusí byť podporovaná u všetkých Android zariadení. [31]

Niektoré štítky sa dajú umelo naformátovať tak, aby podporovali technológiu NDEF, ktorá programátorovi uľahčí život pri manipulácii s dátami. NDEF vytvára vlastnú štruktúru, ktorá zaobaľuje dáta na karte. Je vhodná na prenos a ukladanie bežných dát, no špecifické operácie, ako vytvorenie aplikácie na štítku, zápisu do súboru či autentifikáciu medzi štítkom a zariadením nepodporuje. V týchto prípadoch je potrebné komunikovať s využitím zložitejších technológií ako `IsoDep` a prenášať informácie pomocou bajtového poľa. [32, 33]

2.1.5.1.1 Podporované technológie MIFARE DESFire EV1: Po nasnímaní NFC štítku Android zariadením, je možné pomocou metódy `getTechList()` zistiť všetky technológie, ktoré štítok podporuje. `getTechList()` vráti list s konkrétnymi triedami. [34] Po nasnímaní štítku MIFARE DESFire EV1 obdržíme nasledujúci reťazec tried:

Výpis kódu 2.1 Výpis NFC technológií podporujúcich komunikáciu s MIFARE DESFire EV1

```
TAG: Tech [android.nfc.tech.IsoDep, android.nfc.tech.NfcA,
android.nfc.tech.NdefFormatable]
```

Z výpisu vidíme, že štítok MIFARE DESFire EV1 podporuje normu ISO-DEP – ISO 14443-4 a NFC-A – ISO 14443-3A. Štítok momentálne nie je schopný pracovať s NDEF dátami, ale je možné ho v prípade potreby naformátovať. Po naformátovaní štítku pomocou metód z triedy `NdefFormatable` a opätovnom nasnímaní obdržíme nasledujúci reťazec tried:

Výpis kódu 2.2 Výpis NFC technológií podporujúcich komunikáciu s MIFARE DESFire EV1 po naformátovaní na podporu NDEF

```
TAG: Tech [android.nfc.tech.IsoDep, android.nfc.tech.NfcA,
android.nfc.tech.Ndef]
```

V tomto prípade vznikne na štítku NDEF štruktúra, ktorá umožní MIFARE DESFire EV1 pracovať s NDEF dátami.

2.1.5.1.2 Podporované technológie MIFARE Plus EV1: Po nasnímaní štítku MIFARE Plus EV1 obdržíme nasledujúci reťazec tried:

Výpis kódu 2.3 Výpis NFC technológií podporujúcich komunikáciu s MIFARE Plus EV1

```
TAG: Tech [android.nfc.tech.IsoDep, android.nfc.tech.NfcA]
```

Vidíme, že MIFARE Plus EV1 podporuje ISO-DEP a NFC-A. Na rozdiel od MIFARE DESFire EV1, štítok nie je možné naformátovať na prácu s NDEF dátami.

2.1.5.2 Android tag dispatch system

„Tag dispatch system“, alebo systém prideľovania NFC štítkov je proces, počas ktorého Android zariadenie naskenuje NFC štítok a vyberie vhodnú aplikáciu, ktorá informácie na štítku spracuje. Má na starosti napríklad to, že po priblížení Android zariadenia k platobnému terminálu sa automaticky otvorí aplikácia mobilného bankovníctva, alebo to, že po priložení zariadenia k NFC slúchadlám, sa informácia na ich štítku spracuje v Bluetooth nastaveniach telefónu, čím sa zabezpečí spárovanie. [30]

V prípade, že v zariadení existuje viacero aplikácií, ktoré sú schopné spracovať daný štítok, zobrazí sa na displeji zariadenia okno, v ktorom si užívateľ môže jednu z týchto aplikácií vybrať. Táto situácia môže nastať napríklad v prípade naskenovania NFC štítku, ktorý obsahuje odkaz na webovú stránku. Ak užívateľ nemá nastavený predvolený prehliadač, zobrazia sa mu všetky nainštalované aplikácie, v ktorých si danú webstránku môže zobraziť. Táto možnosť výberu aplikácie je nežiadúca, a to z niekoľkých dôvodov: za prvé, predlžuje sa celkový čas, ktorý užívateľ strávi načítavaním štítku. Za druhé, užívateľ je povinný pri výbere aplikácie manuálne interagovať so svojím Android zariadením, čo ho môže donútiť prerušiť spojenie so štítkom. [30]

Za ideálnych podmienok sa po naskenovaní štítku vždy spustí aplikácia, ktorá je na spracovanie daného typu dát najvhodnejšia, a to bez potreby užívateľskej interakcie. Preto je potrebné

klásť dôraz na odlišovanie typu dát. Vývojárom mobilných aplikácií s NFC je odporúčané, aby čo najpodrobnejšie špecifikovali dáta a technológie štítkov, o ktoré sa ich aplikácia zaujíma. Týmto spôsobom si po naskenovaní NFC štítku aplikácie s rozdielnym zameraním navzájom neprekážajú a Android dokáže ihneď vybrať tú najvhodnejšiu. [30]

2.1.5.2.1 Štruktúra NDEF správy: NDEF formát umožňuje špecifikovať typ NFC dát s najväčšou presnosťou. Vyplýva to z jeho podrobnej štruktúry. NDEF dáta sú zapuzdrené v správe `NdefMessage`. `NdefMessage` sa skladá z niekoľkých záznamov typu `NdefRecord`. Prvý `NdefRecord` záznam spravidla vždy obsahuje špeciálnu informáciu o celkovom type odosielaných NDEF dát, podľa ktorého sa dáta kategorizujú pre výber aplikácie. Musí obsahovať tieto polia:

Type Name Format: Skrátene TNF. Jedná sa o 3 bity, ktoré len upresňujú, ako interpretovať nasledujúce pole v tomto zozname – Variable lenght type. Podporované, často používané hodnoty TNF sú:

- `TNF_MIME_MEDIA` - nasledujúce pole bude špecifikovať typ MIME
- `TNF_ABSOLUTE_URI` - nasledujúce pole bude špecifikovať typ URI
- `TNF_EMPTY` - celý záznam je prázdny
- `TNF_WELL_KNOWN` - nasledujúce pole bude špecifikovať typ MIME alebo URI v závislosti od oficiálnej špecifikácie nazývanej „Record Type Definition“ [30]

Variable length type: Pole, ktoré popisuje presný typ `NdefRecord` záznamu. V prípade zvolenia `TNF_WELL_KNOWN` v predošlom poli sa využíva špecifikácia „Record Type Definition“, ktorá najčastejšie mapuje tieto hodnoty:

- `RTD_URI` - Dáta NDEF správy obsahujú URI
- `RTD_TEXT` - Dáta NDEF správy obsahujú text vo formáte MIME [30]

Variable length payload: Konkrétne dáta, ktoré chceme zapísať alebo prečítať. Musia mať typ definovaný v predošlých poliach `NdefRecord` záznamu. Keďže `NdefMessage` správa môže obsahovať niekoľko `NdefRecord` záznamov, rozsiahlejšie dáta presahujú tento prvý záznam a sú postupne rozdeľované do ďalších. [30]

2.1.5.2.2 Priebeh analýzy NFC štítku: Proces na analýzu obsahu NFC štítku, „tag dispatch system“, pozostáva z niekoľkých krokov:

1. Rozpoznanie typu obsahu, ktorý sa na štítku nachádza.
2. Zapuzdrenie obsahu do takzvaného „Intentu“. `Intent` je jednoduchý objekt v Androide. Funguje ako správa či žiadosť – zaobaľuje dáta `Intent.data` s akciou `Intent.action`, ktorá má nastať. Aplikácia môže obdržať od operačného systému `Intent` a na jeho základe vykonať požadovanú činnosť či spracovať priložené dáta.[6]
3. Spustenie vhodnej aplikácie na základe Intentu. Aplikácia má k obsahu štítku prostredníctvom Intentu priamy prístup. [30, 35]

2.1.5.2.3 Intent filter: Operačný systém má po vytvorení objektu `Intent`, za úlohu vyhľadať a spustiť aplikáciu, ktorá je schopná tento `Intent` spracovať. V prípade, že má aplikácia záujem o `Intent` istého typu, definuje vo svojom konfiguračnom súbore[7] „Intent filter“. Filtrovať sa dá podľa rôznych kritérií. Po tom, čo aplikácia prejaví svoj záujem, ju operačný systém dokáže vyhľadať a spustiť. Stále platí už spomínané odporúčanie, že aplikácie by mali filtrovať pre čo najkonkrétnejší typ Intentu, ktorý ich zaujíma, aby si navzájom neprekážali. [30, 35]

[6] Táto definícia je pre účely práce trochu zjednodušená. Pre úplnosť: Jedná sa o komunikáciu implicitného Intentu s Android aktivitou danej aplikácie.

[7] AndroidManifest.xml

2.1.5.2.4 Intenty pre NFC: Android tag dispatch system zapuzdruje obsah naskenovaného NFC štítku do Intentu, na základe ktorého sa spúšťa vhodná aplikácia. Definuje tri typy Intentov[8] pre technológiu NFC, zoradených podľa priority. Čím vyššie v poradí, tým podrobnejšie informácie bol Android tag dispatch system schopný zo štítku zapuzdriť:

1. `ACTION_NDEF_DISCOVERED` - NFC štítok obsahuje NDEF dáta, ktoré sú podporovaného typu, viz 2.1.5.2.1.
2. `ACTION_TECH_DISCOVERED` - NFC štítok patrí medzi podporované technológie, viz 2.1.5.1.
3. `ACTION_TAG_DISCOVERED` - NFC štítok neobsahuje NDEF dáta podporovaného typu a nepatrí medzi podporované technológie. [30]

Vezmime si napríklad štítok typu MIFARE DESFire EV1, obsahujúci textové dáta vo forme NDEF. Po naskenovaní štítku Android zariadením, sa zapuzdrí tento text do Intentu, ktorého `Intent.action` bude označený ako `ACTION_NDEF_DISCOVERED`.

V prípade, že rovnaký štítok nie je naformátovaný pre NDEF dáta a obsahuje napríklad len natívne podporované dáta, špecifické pre štítky typu MIFARE DESFire EV1, sa zapuzdrenie do `ACTION_NDEF_DISCOVERED` nepodarí. Tag dispatch system sa teda pokúsi o možnosť na druhom mieste – `ACTION_TECH_DISCOVERED`. Zapuzdrenie do Intentu s týmto typom `Intent.action` už bude úspešné, pretože štítok MIFARE DESFire EV1 je kompatibilný s niekoľkými NFC technológiami, ako napríklad s `IsoDep`, viz 2.1.

Aplikácie, ktoré majú záujem o nejaký z týchto Intentov, musia mať vytvorený príslušný Intent filter.

2.1.5.2.5 Intent filter pre ACTION_NDEF_DISCOVERED: Tento Intent filter umožňuje špecifikovať typ požadovaných dát čo najpresnejšie. Napríklad, ak sa má aplikácia spustiť po priložení NFC štítku, ktorý obsahuje NDEF dáta typu text vo formáte MIME, definuje nasledujúci Intent filter:

Výpis kódu 2.4 Definícia Intent filtra pre NDEF dáta typu text vo formáte MIME

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
    <data android:mimeType="text/plain" />
    <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
```

Táto definícia je zapísaná v konfiguračnom súbore aplikácie `AndroidManifest.xml`. [30]

2.1.5.2.6 Intent filter pre ACTION_TECH_DISCOVERED: Tento Intent filter umožňuje špecifikovať typ technológie, ktorou štítok musí disponovať, aby s ním aplikácia bola schopná či ochotná komunikovať. [30] V rámci tejto bakalárskej práce sa komunikuje so štítkom MIFARE DESFire EV1 pomocou technológie `IsoDep`. V prípade, že sa má Android aplikácia spustiť po priložení štítku podporujúceho `IsoDep`, musí to definovať nasledovne:

Výpis kódu 2.5 Definícia Intent filtra pre štítok podporujúci IsoDep

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <tech-list>
        <tech>android.nfc.tech.IsoDep</tech>
    </tech-list>
</resources>
```

[8] jedná sa o typy akcie Intent.action

Táto definícia sa neukladá priamo do `AndroidManifest.xml` ale do samostatného súboru, väčšinou zvaného `nfc_tech_filter.xml`. [30] A ten sa do `AndroidManifest.xml` importuje nasledovne:

Výpis kódu 2.6 Import tech filter do AndroidManifest.xml

```
<intent-filter>
    <action android:name="android.nfc.action.TECH_DISCOVERED"/>
</intent-filter>
<meta-data
    android:name="android.nfc.action.TECH_DISCOVERED"
    android:resource="@xml/nfc_tech_filter"/>
```

2.1.5.2.7 Intent filter pre ACTION_TAG_DISCOVERED: Do tohoto Intentu sa zaobalí NFC štítok, o ktorom má Android minimum informácií. Poskytuje prístup k dôležitému objektu – `Tag`. Objekt `Tag` je prístupný aj z predošlých NFC Intentov. Reprezentuje stav NFC štítku v momente priloženia k zariadeniu. Taktiež obsahuje už spomínanú[9] metódu `getTechList()`. Jedná sa o najmenej odporúčaný Intent filter. [30, 34] Dá sa definovať v `AndroidManifest.xml` následovne:

Výpis kódu 2.7 Definícia Intent filtra pre ACTION TAG DISCOVERED

```
<intent-filter>
    <action android:name="android.nfc.action.TAG_DISCOVERED"/>
</intent-filter>
```

2.1.5.2.8 Android foreground dispatch system: V predošlých sekciách sme si popisovali automatické spustenie vhodnej aplikácie na základe Intentu, vytvoreného po priložení NFC štítku. V prípade, že užívateľ spustil nejakú aplikáciu ešte pred priložením NFC štítku, má táto aplikácia, ak je na popredí, možnosť vyžadovať si prioritné práva spracovať `Intent` zo štítku. Túto funkcionalitu ponúka Android foreground dispatch system. Aplikácia musí definovať špeciálny Intent filter v zdrojovom kóde. V ňom špecifikuje, o aké typy Intentu má prioritný záujem, keď je spustená a na popredí. [31] Napríklad, ak má aplikácia prioritne spracovať NFC štítok, ktorý obsahuje NDEF dáta typu text vo formáte MIME,[10] definuje nasledujúci Intent filter:

Výpis kódu 2.8 Definícia Intent filtra pre NDEF dáta typu text vo formáte MIME pre foreground dispatch

```
val filter = IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED).apply {
                addDataType("text/plain")
            }
```

Ak má spustená aplikácia prioritne spracovať NFC štítok podporujúci `IsoDep`,[11] definuje nasledujúci filter:

[9] Spomínaná v sekcii 2.1.5.1.1 – zistí všetky technológie, ktoré štítok podporuje.
[10] Obdobný príklad ku 2.4.
[11] Obdobný príklad ku 2.5.

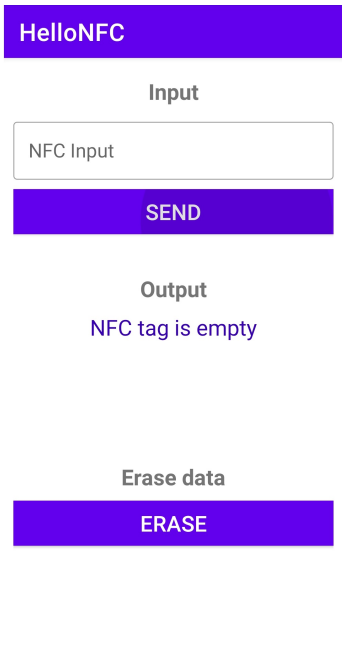
Výpis kódu 2.9 Definícia Intent filtra pre štítok podporujúci IsoDep pre foreground dispatch

```
var nfcTechLists: Array<Array<String>> = arrayOf()

val isoDepTechList = arrayOf(IsoDep::class.java.name)
nfcTechLists = nfcTechLists.plus(isoDepTechList)
```

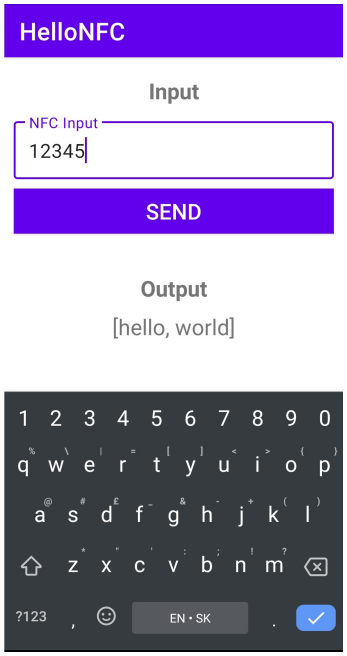
Tieto Intent filtre sa v aplikácii registrujú pomocou metódy `enableForegroundDispatch(...)`, ktorá sa spravidla spúšťa vždy po tom, čo sa aplikácia dostane na popredie. Metódu poskytuje trieda `NfcAdapter`. Týmto sa zabezpečí, že spustená aplikácia obdrží daný `Intent` prioritne. [31]

2.1.5.3 Android NFC skúšobná aplikácia

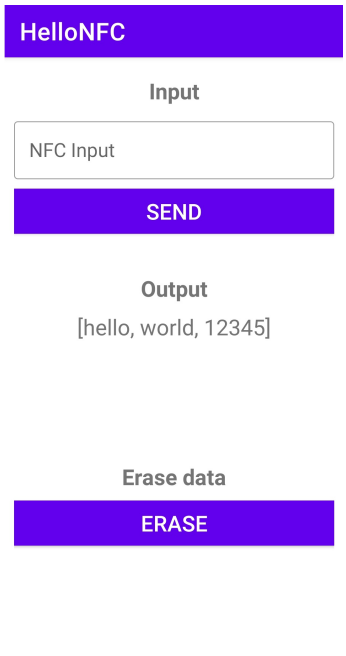
V rámci procesu analýzy tvorby Android aplikácií využívajúcich NFC, bola vytvorená jednoduchá, skúšobná aplikácia na komunikáciu so štítkom MIFARE DESFire EV1.[12] Táto aplikácia umožňuje zápis krátkych textových reťazcov do štítku vo formáte NDEF. Textový obsah štítku vie taktiež zobraziť či úplne odstrániť. S využitím teórie zo sekcie 2.1.5.2.1, bol zvolený konkrétny typ NDEF dát ako `TNF_WELL_KNOWN` so špecifikáciou na typ `RTD_TEXT` – `text/plain` podľa MIME. Každý osobitný textový reťazec je uložený, spolu s informáciou o type NDEF dát, v práve jednom zázname `NdefRecord`. To limituje jeho maximálnu dĺžku, pretože dlhšie reťazce môžu byť zapísané len s použitím viacerých záznamov `NdefRecord`. Všetky záznamy `NdefRecord` sú napokon uložené v jedinej správe `NdefMessage`. Aplikácia využíva klasický Intent filter na reagovanie po priložení štítku k zariadeniu. Taktiež definuje foreground dispatch na prioritné spracovanie štítku v prípade, že je aplikácia spustená a na popredí.



Obr. 2.7 Rozhranie skúšobnej NFC aplikácie. Obsah naskenovaného štítku je prázdny, viz „Output“.



Obr. 2.8 Naskenovaný štítok obsahuje reťazce „hello“ a „world“. Do políčka „Input“ sa zadáva ďalší reťazec „12345“.



Obr. 2.9 Naskenovaný štítok obsahuje trojicu zapísaných reťazcov „hello“, „world“ a „12345“.

[12] Odkaz: `https://gitlab.fit.cvut.cz/popovma1/hellonfc/tree/ndef`

Táto aplikácia bola počiatočne vytvorená za účelom hlbšieho pochopenia Android NFC dokumentácie. Po čase vznikla jej upravená verzia, ktorá je schopná komunikovať s NFC štítkami pomocou výmeny bajtového poľa, s využitím technológie `IsoDep`. Táto verzia bola napokon využívaná na analýzu natívnych príkazov pre MIFARE DESFire EV1, a na testovanie knižnice, vytvorenej v tejto práci.

2.2 Rešerš existujúcich riešení

Nasledujúce podkapitoly sa venujú existujúcim riešeniam. Cieľom bolo vyhľadať a analyzovať existujúce knižnice, ktoré implementujú NFC komunikáciu s kartou MIFARE DESFire EV1 na zariadení Android. Jedná sa teda hlavne o riešenia v jazyku Java. Ďalej sú z dôvodu nízkeho počtu konkurenčných riešení zahrnuté aj tie, ktoré nie sú na Android zariadenie určené.

2.2.1 TapLinx SDK

TapLinx SDK od spoločnosti MIFARE je populárna knižnica, určená pre vytváranie aplikácií, využívajúcich komunikáciu s NFC kartami od výrobcu MIFARE. Tento produkt nahradil SDK MIFARE, ktorý bol v minulosti distribuovaný vývojárom za poplatok. Rozhranie TapLinx SDK je momentálne verejne dostupnou a vylepšenou verziou. To umožňuje vývojárom rýchlejšie dizajnovanie a vytváranie aplikácii, nielen pre platformu Android. [36]

Obr. 2.10 TapLinx Developer Community banner [36]

2.2.1.1 Výhody

Plná funkčnosť v operačnom systéme Android. SDK je vytvorené v jazyku Java a obsahuje funkcie špecifické pre prácu v Androide.

Rýchly a jednoduchý vývoj aplikácií. Komplexné nízko-úrovňové príkazy sa dajú jednoducho nájsť a nahradiť príkazmi v TapLinx Javadoc dokumentácii. [36]

Špecializované fórum pre komunitu developerov. Na fóre sa nachádzajú video tutoriály či príklady zdrojových kódov. V rámci tejto komunity môžu vývojári pokladať otázky a získať tým pomoc od TapLinx podpory alebo od ostatných vývojárov.

Podpora viacerých produktov od NXP. Jeden SDK podporuje všetky typy MIFARE kariet. Zároveň umožňuje aj vývoj pre ďalšie NFC technológie od NXP, ako NTAG a ICODE. [36]

2.2.1.2 Nevýhody

Časovo náročná a komplikovaná inštalácia knižnice. Celý postup na inštaláciu, opísaný v manuáli, má 9 normostrán. Pozostáva z niekoľkých krokov: [26]

1. Registrácia emailovej adresy na webstránke `www.mifare.net/devcenter`.
2. Výber a registrácia unikátneho názvu balíčku aplikácie, a zároveň názvu našej aplikácie. Dané informácie sa po uložení nedajú zmeniť.
3. Vygenerovanie unikátneho „online“ kľúča.

4. Vytvorenie špeciálnej registračnej metódy v našej aplikácii, ktorá slúži na online overenie licencie. Licencia overená pomocou vygenerovaného kľúča umožní aplikácii knižnicu používať.
5. Pridanie povolení, ktoré aplikácia potrebuje od Android zariadenia na plnú funkcionalitu. Obsahuje napríklad povolenie používať internet.[13]
6. Zaregistrovanie knižnice do závislostí projektu pomocou technológie Maven.[14]
7. Alternatívne k predošlému kroku, je možné stiahnuť si knižnicu vo forme AAR[15] súboru a manuálne ju začleniť do vývojového prostredia projektu.

V niektorých krokoch inštalácie nie je zrejmé ako postupovať. Konkrétny priebeh na sprevádzkovanie je v príručke opísaný pre starú verziu vývojového prostredia Android Studio. Kroky, ktoré sa líšia, si musí nahradiť developer sám, čo ho v niektorých prípadoch veľmi spomaľuje.

Obr. 2.11 How To Get TapLinx [36]

Knižnica potrebuje využívať internet na overovanie licencie. Táto knižnica nie je najvhodnejšia pre podmienky, kde vlastníci Android zariadení nemajú zabezpečený prístup k internetu. Alternatíva k online riešeniu je stiahnutie a začlenenie spomínanej AAR knižnice do projektu, ktorá funguje v offline podmienkach. Ďalej je potrebné si na webstránke vygenerovať prídavný „offline“ kľúč. Používanie tejto alternatívy si ale naďalej vyžaduje prítomnosť „online“ kľúča v zdrojovom kóde. Taktiež sa naďalej vyžaduje začleniť povolenie aplikácie pristupovať k internetu. Cieľom tejto offline alternatívy sa teda javí byť len prídavným overením platnosti licencie pri dočasnom nepripojení k internetu. [26]

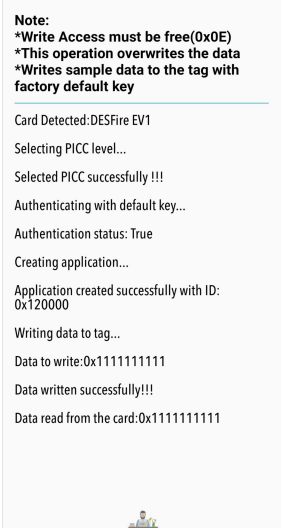
Výpis kódu 2.10 Definovanie povolení, ktoré TapLinx vyžaduje od Android zariadenia

```
<uses-permission android:name="NFC" />
<uses-permission android:name="WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="INTERNET" />
<uses-permission android:name="ACCESS_NETWORK_STATE" />
```

2.2.1.3 Aplikácie využívajúce knižnicu

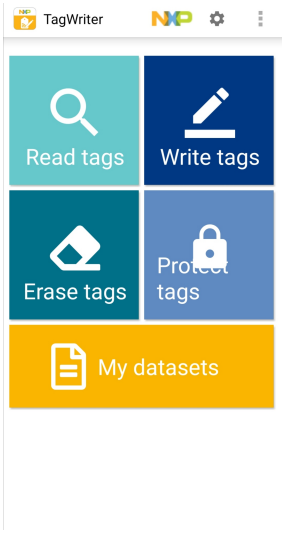
2.2.1.3.1 TapLinx SDK Sample App: Ukážková aplikácia, ktorej zdrojový kód je pre vývojárov prístupný na webstránke MIFARE. Aplikácia po spustení čaká na NFC tag. Po priložení tagu je možné tlačidlom na zápis previesť preddefinovanú sériu príkazov. Príkazy vytvoria aplikáciu, zapíšu do nej dáta a napokon dáta prečítajú.[16]

[13] Tieto povolenia sa definujú vždy v súbore AndroidManifest.xml danej aplikácie.
[14] Vo veľkom množstve knižníc je toto jediný potrebný krok na ich sprevádzkovanie
[15] AAR – android archive library
[16] Odkaz: `https://mifare.net/wp-content/uploads/2019/08/srcsamplenxpnfclib-1.7.zip`

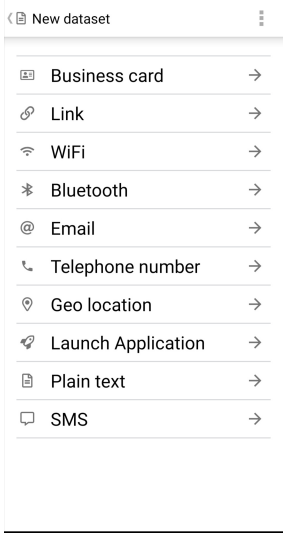


Obr. 2.12 TapLinx SDK Sample App vykoná preddefinovanú sériu príkazov

2.2.1.3.2 NFC TagWriter by NXP: Krátke údaje ako email, telefónne číslo či odkaz na webstránku sa dajú pomocou tejto aplikácie pohodlne uložiť do NFC tagu. Údaje na karte sa dajú prečítať, zašifrovať či zmazať. Dáta sú ukladané vo formáte NDEF. Ako vývojárovi mi táto aplikácia pomohla tým, že dokáže zmazať informácie na karte a formátovať ju do výrobného nastavenia. Na Google Play má nad jeden milión stiahnutí.[17]



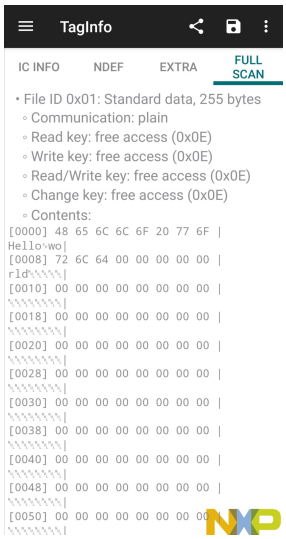
Obr. 2.13 Menu aplikácie TagWriter ponúka rôzne možnosti



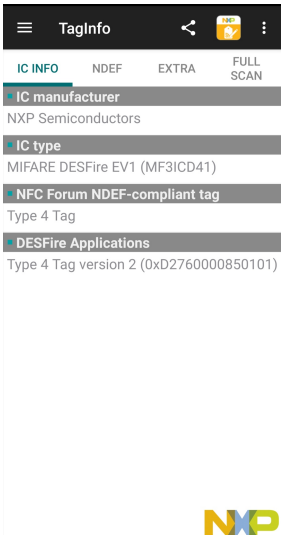
Obr. 2.14 TagWriter pracuje s typmi údajov ako email, telefónne číslo či odkaz na webstránku

2.2.1.3.3 NFC TagInfo by NXP: Aplikácia[18] vhodná pre všetkých, ktorých zaujíma, aký typ karty majú v rukách či aký je obsah jej pamäti v bajtoch. Aplikáciu som využíval počas testovania, na overenie správneho zápisu dát. Nainštalovalo si ju viac ako 500-tisíc užívateľov.[37]

[17] Odkaz: `https://play.google.com/store/apps/details?id=com.nxp.nfc.tagwriter`
[18] Odkaz: `https://play.google.com/store/apps/details?id=com.nxp.taginfolite`



Obr. 2.15 TagInfo: Na testovacej karte je zapísaný reťazec „Hello world“



Obr. 2.16 TagInfo: Školský ISIC je typu MIFARE DESFire EV1

2.2.1.4 Zhrnutie

TapLinx SDK sa javí ako veľmi dobré riešenie pre Android vývojárov. Pokrýva mnoho technológií od MIFARE a je pravidelne aktualizované. Na druhú stranu, prvotná inštalácia knižnice je problematická. Nevýhodou je aj potrebné pripojenie zariadenia k internetu.

2.2.2 nfcjlib

Nfcjlib je knižnica od Github užívateľa[19] Daniel Andrade. *„Umožňuje klientovi interagovať s čipovou kartou MIFARE DESFire EV1“* [38]

2.2.2.1 Výhody

Knižnica je naprogramovaná v jazyku Java. To znamená jednoduchšie použitie pri vývoji v Androide.

Pokrýva veľké množstvo šifrovacích algoritmov. Obsahuje šifry typu AES, CMAC, CRC či DES.

Spĺňa podmienky pre fungovanie v offline režime. Používanie si nevyžaduje žiadnu registráciu licencie.

2.2.2.2 Nevýhody

Knižnica je nepriehľadná. Všetky príkazy na NFC komunikáciu sa nachádzajú v jedinej triede `DESFireEV1.java`, ktorá presahuje 2500 riadkov.

Knižnica je neúplná *„Nie je implementovaných všetkých 35 príkazov.“* [38] Užívateľ by si v prípade potreby musel rozšíriť knižnicu sám.

[19] `https://github.com/andrade`

Potreba podrobnejšej znalosti MIFARE DESFire EV1. Užívateľ knižnice nie je odbremenený od potreby hlbšej znalosti komunikácie s touto kartou. Musí napríklad vedieť, že na vkladanie niektorých číselných argumentov sa používa formát Little-endian.[25] Taktiež musí všetky argumenty príkazov vkladať v podobe sekvencie bajtov a strážiť si pritom ich poradie.

2.2.2.3 Porovnanie kódu

Porovnanie čitateľnosti kódu knižnice nfcjlib a knižnice vzniknutej v rámci tejto práce. Oba kódy vytvoria rovnakú sériu príkazov, ktoré pošlú zo zariadenia do NFC tagu. Cieľom je vytvoriť aplikáciu, zvoliť ju a autentizovať sa kľúčom. Kód je pre účel príkladu zjednodušený a predpokladá sa, že jednotlivé príkazy sa neukončia s chybou.

Výpis kódu 2.11 Ukážka kódu nfcjlib pri porovnaní s riešením práce

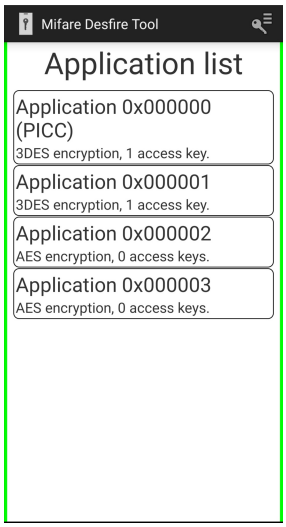
```
/* ID = 0x0F0000 hex = 15 dec */
final byte[] APPLICATION_ID = new byte[] {0x0F, 0x00, 0x00};

/* create application
* (0xEF means without initial protection)
* (0x02 means Key2TDEA cipher and two application keys) */
desfire.createApplication(APPLICATION_ID, (byte) 0xEF, (byte) 0x02);

/* select application */
desfire.selectApplication(APPLICATION_ID);
/* authenticate inside application with key 0x00 and cipher 2K3DES */
desfire.authenticate(new byte[8], (byte) 0x00, KeyType.DES);
```

Výpis kódu 2.12 Ukážka kódu z riešenia práce pri porovnaní s nfcJlib

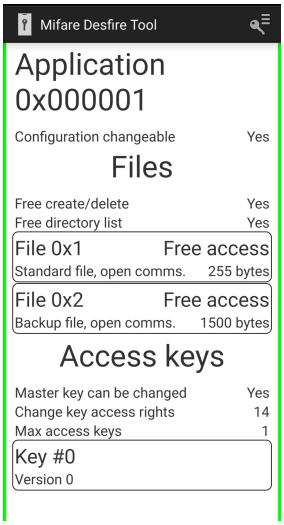
```
val appID = 15

/* create application
* (without initial protection)
* (Key2TDEA cipher and two application keys) */
CreateApplication(
    appID = appID,
    keySett1 = AppKeySettings(
        changeKeyAccessRight = APP_KEY_AUTH,
        appKeySettingsChangeable = true,
        createFileUnprotected = true,
        directoryUnprotected = true,
        appMasterKeyChangeable = true
    ),
    keySett2 = KeySett2(
        appKeyType = KeyType.Key2TDEA,
        numberOfAppKeys = 2
    )
).transceive(isoDep)

/* select application */
SelectApplication(appID).transceive(isoDep)
/* authenticate inside application with key 0x00 and cipher 2K3DES */
Authenticate(keyNo = 0).transceive(isoDep)
```

2.2.2.4 Aplikácie využívajúce knižnicu

2.2.2.4.1 MIFARE DESFire Tool: Aplikácia[20] zobrazí informácie o štruktúre karty – aplikácie, súbory a ich kľúče. Nezobrazí daný obsah súborov ako NFC TagInfo by NXP, no užívateľské rozhranie je podľa mňa navrhnuté krajšie a je príjemnejšie na použitie. Aplikácia má vyše 50-tisíc stiahnutí. Využíva nfcjlib ako jeden zo zdrojov. [39]



Obr. 2.17 MIFARE DESFire Tool: Na karte sa nachádzajú 4 aplikácie



Obr. 2.18 MIFARE DESFire Tool: V aplikácii číslo 1 sa nachádzajú 2 súbory

2.2.2.5 Zhrnutie

Knižnica nfcjlib je funkčné riešenie vhodné pre offline použitie. Jedná sa ale len o malý projekt, ktorý nebol aktualizovaný od roku 2013. [38]

2.2.3 Ďalšie riešenia

Z analýzy vyplýva, že neexistuje mnoho riešení, ktoré by boli vhodné na vývoj v Androide.

Spomeniem ešte knižnicu s názvom libfreefare, ktorá sa zdá byť dobrou knižnicou pre riešenia v jazyku C.[21]

Ďalšia knižnica v jazyku C, ktorá má za cieľ implementovať časť šifrovanej komunikácie je DESFireAES.[22] Táto knižnica má zaujala tým, že obsahuje PDF dokument[23] odkazujúci na mnoho zdrojov, ktoré popisujú komunikáciu s MIFARE DESFire EV1. Tieto zdroje sa hľadajú na internete veľmi zložito, pretože na hlavnú dokumentáciu MIFARE DESFire EV1 sa vzťahuje zmluva o mlčanlivosti.[40]

[20] Odkaz: `https://play.google.com/store/apps/details?id=com.skjolberg.mifare.desfiretool`
[21] `https://github.com/nfc-tools/libfreefare`
[22] `https://github.com/revk/DESFireAES`
[23] `https://github.com/revk/DESFireAES/blob/master/DESFire.pdf`

2.3 Sada príkazov MIFARE DESFire EV1

V tejto časti si predstavíme sadu príkazov vytvorených pre natívnu komunikáciu s NFC kartou MIFARE DESFire EV1. Ako už bolo čiastočne popísané v 2.1.4.2, tieto príkazy dokážu upraviť či šifrovať obsah karty. Príkazy budú predstavené v nadväzujúcom poradí tak, aby odrážali možný postup tvorby obsahu na karte. To znamená:

1. Vytvorenie aplikácie na karte
2. Vytvorenie súboru v aplikácii
3. Zápis do súboru
4. Čítanie zo súboru
5. Deaktivácia súboru
6. Deaktivácia aplikácie
7. Formátovanie karty
8. Autentizácia užívateľa

2.3.0.0.1 Príkazy: Príkazy sa posielajú zo zariadenia do karty vo forme bajtového poľa. Každý príkaz má definovanú svoju presnú štruktúru. Jediný povinný atribút, ktorým začínajú všetky existujúce príkazy, je „command code“ – jedno-bajtová, hexadecimálna hodnota reprezentujúca daný typ príkazu. Za ním môžu nasledovať bajty, ktoré špecifikujú parametre príkazu. Napríklad ID a veľkosť súboru, ktorý chceme vytvoriť. Taktiež môžu nasledovať bajty v podobe užívateľských dát, ako napríklad dáta, ktoré do súboru chceme zapísať. [41, 42]

2.3.0.0.2 Odpovede: Karta prijatý príkaz spracuje a reaguje na neho odoslaním odpovede. Na základe tejto odpovede môže zariadenie zistiť, či bol príkaz úspešne spracovaný, prípadne môže obdržať extra dáta. Odpoveď karty má definovanú podobnú štruktúru ako príkaz. Obsahuje povinný bajt „return code“,[24] ktorý značí úspešnosť spracovania príkazu. Za týmto bajtom môžu, v prípade úspechu, nasledovať ďalšie parametre či dáta. V prípade neúspechu sa vždy jedná o jedno-bajtovú odpoveď s hodnotou return code, ktorá popisuje typ chyby. Prehľad niektorých dôležitých hodnôt return code je možné vidieť v tabuľke 2.1. [25]

Hodnota	Return code	Popis
0x00	`OPERATION_OK`	Úspech
0x1C	`ILLEGAL_COMMAND_CODE`	Nepodporovaný príkaz
0x9E	`PARAMETER_ERROR`	Neplatná hodnota parametru v príkaze
0x7E	`LENGTH_ERROR`	Neplatná dĺžka príkazu
0x9D	`PERMISSION_DENIED`	Nepovolené použitie príkazu
0xAE	`AUTHENTICATION_ERROR`	Neoprávnené použitie príkazu
0xF0	`FILE_NOT_FOUND`	Požadovaný súbor neexistuje
0xAF	`ADDITIONAL_FRAME`	Úspech, karta očakáva doplňujúci príkaz

Tabuľka 2.1 Prehľad niektorých dôležitých hodnôt return code [25]

[24] Nazývaný taktiež ako response code

2.3.0.0.3 Formát príkazov: Ako už bolo mnohokrát spomenuté, komunikácia medzi kartou a zariadením spočíva vo výmene bajtových polí. V počítačovej vede je zvykom zapisovať bajt pomocou hexadecimálnej číselnej sústavy. V hexadecimálnej sústave existuje 16 znakov, ktoré odpovedajú číselným hodnotám 0–15. Tieto znaky sú 0x0, 0x1, 0x2, ..., 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF.[25] Jeden tento znak sa dá reprezentovať 4 bitmi. Pretože bajt má 8 bitov, môžeme ho skonštruovať spojením dvoch 4-bitových hexadecimálnych znakov. Rozsah hodnôt jedného bajtu je tým pádom 0x00 až 0xFF, respektíve 0 až 255. Zlučovaním týchto bajtov sa utvárajú bajtové polia, pomocou ktorých sa dajú efektívne prenášať informácie. [43]

2.3.0.0.4 Reprezentácia číselných hodnôt v príkazoch: Keďže bajt dokáže uložiť rozsah číselných hodnôt len od 0 do 255,[26] je potrebné na uloženie väčšieho číselného údaju použiť viac bajtov. Existujú dva spôsoby ukladania:

Big-endian: Do bajtového poľa sa na miesto s najnižšou adresou uloží najviac významný bajt. Za ním nasledujú menej významné bajty.

Little-endian: Do bajtového poľa sa na miesto s najnižšou adresou uloží najmenej významný bajt. Za ním nasledujú významnejšie bajty. [44]

Hexadecimálna hodnota	4-bajtový Big-endian	4-bajtový Little-endian
0x01	0x00 00 00 01	0x01 00 00 00
0x4E 20	0x00 00 4E 20	0x20 4E 00 00
0xFF FF FF	0x00 FF FF FF	0xFF FF FF 00
0x80 00 00 01	0x80 00 00 01	0x01 00 00 80

Tabuľka 2.2 Rozdiel medzi zápisom pomocou Big-endian a Little-endian

MIFARE DESFire EV1 používa na viac-bajtové číselné údaje v parametroch príkazov zápis Little-endian. Často sa jedná o 3-bajtové hodnoty na reprezentáciu prirodzených čísel – rozsah hodnôt je 0 až 16 777 215.[27] Všetky hodnoty sú nezáporné, takže v zápise nie je potrebné rozlišovať medzi znamienkami + a −, viz tabuľka 2.3. [25, 45]

Číselná hodnota	3-bajtová hexadecimálna hodnota bez znamienka	3-bajtový Little-endian
1	0x00 00 01	0x01 00 00
255	0x00 00 FF	0xFF 00 00
20 000	0x00 4E 20	0x20 4E 00
16 777 215	0xFF FF FF	0xFF FF FF

Tabuľka 2.3 Zápis 3-bajtovej hexadecimálnej hodnoty bez znamienka do 3-bajtového Little-endian

V mnohých príkazoch sú taktiež využívané 4-bajtové hodnoty, ktoré reprezentujú klasický 32-bitový Integer. Rozsah hodnôt je −2 147 483 648 až 2 147 483 647.[28] Hodnoty môžu byť záporné aj nezáporné, takže je potrebné rozlišovať medzi znamienkami + a −. Na kódovanie týchto čísel sa využíva doplnkový kód – rozlišovanie znamienka zabezpečuje bit s najnižšou adresou. Ak je jeho hodnota 0, jedná sa o kladné číslo. Ak je jeho hodnota 1, ide o záporné číslo, ktoré je potrebné náležite zakódovať, viz tabuľky 2.4 a 2.5. [25]

[25] Hexadecimálne hodnoty sa značia prefixom „0x“
[26] V rámci prirodzených čísel.
[27] $16777215 = (2^{24} - 1) = (2^{3 \cdot 8} - 1)$
[28] $-2147483648 = -2^{31}$, $2147483647 = (2^{31} - 1)$

Číselná hodnota	4-bajtová hexadecimálna hodnota so znamienkom	Operácia
255	0x00 00 00 FF	
		- 0x00 00 00 01
	0x00 00 00 FE	
		bitová negácia NOT
−255	0xFF FF FF 01	

Tabuľka 2.4 Prevod čísla 255 na -255 v doplnkovom kóde. V zápise -255 je bajt s najnižšou adresou 0xFF, čo je binárne 1111 1111. Bit na najnižšej adrese je 1, čo značí že číslo je záporné.

Číselná hodnota	4-bajtová hexadecimálna hodnota so znamienkom	4-bajtový Little-endian
1	0x00 00 00 01	0x01 00 00 00
−2 147 483 647	0x80 00 00 01	0x01 00 00 80
255	0x00 00 00 FF	0xFF 00 00 00
−255	0xFF FF FF 01	0x01 FF FF FF

Tabuľka 2.5 Zápis 4-bajtovej hexadecimálnej hodnoty so znamienkom do 4-bajtového Little-endian

Na tomto mieste je vhodné spomenúť fakt, ktorý zatiaľ nie je relevantný, ale v implementačnej časti bude využívaný. Keďže jedným z cieľov práce je uľahčiť užívateľovi zadávanie číselných parametrov, knižnica automaticky vykonáva konverziu medzi hexadecimálnymi hodnotami a dátovým typom `Int`. Rozsah hodnôt typu `Int` je −2 147 483 648 až 2 147 483 647. To sa presne zhoduje so 4-bajtovou hexadecimálnou hodnotou so znamienkom. Avšak, na prácu s 3-bajtovou hexadecimálnou hodnotou bez znamienka – v rozsahu 0 až 16 777 215 – by sa mohlo javiť ako výhodnejšie použiť dátový typ `UInt`. Ten je definovaný pre prirodzené čísla od 0 do 4 294 967 295.[29] Keďže sa na rozdiel od typu `Int` jedná o bez-znamienkový dátový typ, nie je potrebné sa zaoberať konverziou do doplnkového kódu. V tomto prípade ale knižnica taktiež využíva `Int`, pretože hodnoty od 0 do 16 777 215 sú z hľadiska zápisu v doplnkovom kóde rovnaké, viz tabuľka 2.6. Tento fakt uľahčuje užívateľovi knižnice pracovať s číslami. Nemusí kombinovať `UInt` a `Int`, pretože celá knižnica bezpečne využíva `Int`.

Číselná hodnota	4-bajtová hexadecimálna hodnota so znamienkom	3-bajtová hexadecimálna hodnota bez znamienka
1	0x00 00 00 01	0x00 00 01
255	0x00 00 00 FF	0x00 00 FF
20 000	0x00 00 4E 20	0x00 4E 20
16 777 215	0x00 FF FF FF	0xFF FF FF
16 777 216	0x01 00 00 00	-
−1	0xFF FF FF FF	-

Tabuľka 2.6 Hodnoty od 0 do 16 777 215 sú z hľadiska zápisu rovnaké pre 4-bajtovú hexadecimálnu hodnotu so znamienkom ako aj pre 3-bajtovú hexadecimálnu hodnotu bez znamienka

[29] $4294967295 = (2^{32} - 1)$

2.3.1 CreateApplication

Príkaz, ktorý vytvorí na karte novú aplikáciu.

2.3.1.1 Štruktúra príkazu

1	3	1	1
Command code	Application ID	App Key Settings	Key Settings 2

Štruktúra príkazu: CreateApplication.
Spodný riadok popisuje potrebné parametre a ich poradie,
horný riadok popisuje ich veľkosť v bajtoch.

Command code: `0xCA`

Application ID: Unikátne číslo na identifikáciu aplikácie. Jedná sa o prirodzené, hexadecimálne číslo vo formáte Little-endian.

App Key Settings: Definuje bezpečnostné nastavenia aplikácie pomocou jednotlivých bitov.

- Bity 7–4 definujú prístup ku zmene kľúča.
- Bit 3 definuje možnosť budúcej zmeny konfigurácie.
- Bit 2 definuje prístup ku tvorbe a mazaniu súborov v aplikácii.
- Bit 1 definuje prístup ku informáciám o súboroch.
- Bit 0 definuje možnosť budúcej zmeny hlavného kľúča aplikácie.

Key Settings 2: Definuje prídavné bezpečnostné nastavenia.

- Bity 7–6 definujú typ šifrovania: $'00' \rightarrow$ `2K3DES`, $'01' \rightarrow$ `3K3DES`, $'10' \rightarrow$ `AES`.
- Bity 5–4 sú nedefinované.
- Bity 3–0 definujú počet kľúčov, s ktorými môže aplikácia pracovať. Maximálny počet kľúčov je 14, minimálny je 0.

2.3.1.2 Štruktúra odpovede

1
Return code

Štruktúra odpovede:
CreateApplication.

Return code: Značí úspešnosť spracovania príkazu.

2.3.1.3 Príklad

```
CA 01 00 00 EF 81   ⟶
                    ⟵   00
```

Príklad príkazu: CreateApplication.
Na ľavej strane je príkaz odoslaný zo zariadenia,
na pravej strane je odpoveď odoslaná z karty.

V tomto príklade je odoslaný zo zariadenia do karty príkaz na vytvorenie aplikácie s identifikačným číslom 1. `App Key Settings` sa skladá z hodnoty `E`, ktorou povoľuje úpravu kľúča po autentifikácii, a taktiež z hodnoty `F`, ktorou povoľuje všetky budúce zmeny v aplikácii a jej súboroch. Hodnota `1` v `Key Settings 2` definuje prítomnosť jedného kľúča a hodnota `8` definuje šifrovanie pomocou `AES`.

Odpoveď z karty je `00` – `OPERATION_OK`, čo značí, že aplikácia bola úspešne vytvorená. [25, 41, 42, 46]

2.3.2 SelectApplication

Tento príkaz zvolí vybranú aplikáciu. Ak je aplikácia zvolená, je možné pristupovať k jej dátam ďalšími príkazmi. Po zvolení sa môžeme napríklad pokúsiť čítať z jej súborov. Na karte MIFARE DESFire EV1 môže byť v jeden čas zvolená práve jedna aplikácia.

2.3.2.1 Štruktúra príkazu

1	3
Command code	Application ID

Štruktúra príkazu: SelectApplication

Command code: `0x5A`

Application ID: Unikátne číslo aplikácie, ktorú chceme zvoliť.

2.3.2.2 Štruktúra odpovede

1
Return code

Štruktúra odpovede: SelectApplication

Return code: Značí úspešnosť spracovania príkazu.

2.3.2.3 Príklad

```
5A 01 00 00   ⟶
              ⟵   00
```

Príklad príkazu: SelectApplication

V príklade je odoslaný príkaz na zvolenie aplikácie s identifikačným číslom 1. Odpoveď z karty je `00` – `OPERATION_OK`, čo značí, že aplikácia bola zvolená a ďalšími príkazmi môžeme pristupovať k jej obsahu. [25, 41, 42]

2.3.3 CreateStdDataFile

Vytvorí dátový súbor v práve zvolenej aplikácii. Dátový typ súboru umožňuje užívateľovi zapísať do súboru ľubovoľné dáta vo forme sekvencie bajtov, viz 2.1.4.2.1.

2.3.3.1 Štruktúra príkazu

1	1	1	2	3
Command code	File number	Communication mode	Access rights	File size

Štruktúra príkazu: CreateStdDataFile

Command code: `0xCD`

File number: Unikátne číslo na identifikáciu súboru. Jedná sa o prirodzené, hexadecimálne číslo v rozsahu 0–31.

Communication mode: Reprezentuje úroveň komunikačnej bezpečnosti. Je možné komunikovať jednoduchým textom bez šifrovania, jednoduchým textom s autentifikáciou pomocou funkcie `MAC`[30] či na plne šifrovanej úrovni. Možné hodnoty: `0x00` → `PLAIN`, `0x01` → `MAC`, `0x03` → `FULL`.

Access rights: Definujú prístupové práva k operáciám so súbormi. Pre každý typ operácie sú vyhradené 4 bity. Typ operácie je voľne prístupný → `0xE`, prístupný po autentifikácii kľúčom aplíkácie → `0x0…0xD`, alebo úplne neprístupný → `0xF`.

- Bity 15–12 definujú práva pre operácie typu `read`.
- Bity 11–8 definujú práva pre operácie typu `write`.
- Bity 7–4 definujú práva pre operácie typu `readwrite`.
- Bity 3–0 definujú práva pre operácie typu `change`.

File size: Veľkosť súboru. Jedná sa o prirodzené, hexadecimálne číslo vo formáte Little-endian. Maximálna veľkosť súboru závisí od veľkosti karty.

2.3.3.2 Štruktúra odpovede

1
Return code

Štruktúra odpovede: CreateStdDataFile.

Return code: Značí úspešnosť spracovania príkazu.

2.3.3.3 Príklad

```
CD 01 00 EEEE FF0000   ⟶
                       ⟵   00
```

Príklad príkazu: CreateStdDataFile.

Správa odoslaná zo zariadenia obsahuje príkaz na vytvorenie dátového súboru s číslom 1. Komunikácia je nešifrovaná, respektíve `PLAIN`. Štvorica bajtov `EE EE` definuje voľný prístup k súborovým operáciám. Veľkosť súboru je `0xFF 00 00`, respektíve 255.

Odpoveď z karty je `00` – `OPERATION_OK`, čo značí, že súbor bol úspešne vytvorený. [25, 41, 42, 46, 47]

[30] MAC – Message Authentication Code – slúži na zaistenie integrity prenášanej správy.

2.3.4 CreateBackupDataFile

Vytvorí záložný dátový súbor v práve zvolenej aplikácii. Záložný dátový typ súboru umožňuje užívateľovi zapísať do súboru ľubovoľné dáta vo forme sekvencie bajtov. Rozdiel medzi štandardným dátovým súborom je, že ich zápis si vyžaduje extra potvrdzovací príkaz, viz 2.1.4.2.1. Štruktúra príkazu je zhodná s vytváraním klasického dátového súboru z 2.3.3, jediný rozdiel je v `Command code`, ktorý má hodnotu `0xCB`. [25, 41, 42, 46, 47]

2.3.5 WriteData

Zapíše dáta vo forme bajtového poľa do dátového súboru alebo do záložného dátového súboru. V prípade zápisu väčšieho množstva dát, je potrebné príkaz rozdeliť do častí. V prípade záložného dátového súboru je po zápise potrebné odoslať extra príkaz `CommitTransaction` na potvrdenie zápisu.

2.3.5.1 Štruktúra príkazu

1	1	3	3	1-52
Command code	File number	Offset	Length	Data

Štruktúra príkazu: WriteData

Command code: `0x3D`

File number: Číslo súboru, do ktorého chceme zapisovať.

Offset: Bajtový posun od začiatku súboru. Od tohoto posunu sa začne zápis dát. Jedná sa o prirodzené, hexadecimálne číslo vo formáte Little-endian.

Length: Dĺžka zapisovaných dát v počte bajtov. Jedná sa o prirodzené, hexadecimálne číslo vo formáte Little-endian.

Data: Dáta, ktoré chceme zapísať do súboru, vo formáte bajtového poľa. V prípade, že veľkosť dát presahuje 52 bajtov, je zvyšok dát odoslaných v nadväzujúcom príkaze `Additional frame`.

1	1-59
Additional frame	Data

Štruktúra príkazu: WriteData - Additional frame

Additional frame: `0xAF`

Data: Nadväzujúce dáta, ktoré chceme zapísať do súboru. Maximálna veľkosť je 59 bajtov. V prípade potreby môže byť znovu použitý v nadväzujúci príkaz `Additional frame`.

2.3.5.2 Štruktúra odpovede

1
Return code

Štruktúra odpovede: WriteData

Return code: Značí úspešnosť spracovania príkazu.

2.3.5.3 Príklad

3D 05 00 00 00 3C 00 00		
4C 6F 72 65 6D 20 69 70 73		
75 6D 20 64 6F 6C 6F 72 20		
73 69 74 20 61 6D 65 74 2C		
20 63 6F 6E 73 65 63 74 65		
74 75 72 20 61 64 69 70 69		
73 63 69 6E 67 20 65	⟶	
	⟵	AF
AF 6C 69 74 20 73 65 64 2E	⟶	
	⟵	00

Príklad príkazu: WriteData

V príklade je do štandardného dátového súboru na karte zapísaný 60-bajtový reťazec „Lorem ipsum dolor sit amet, consectetur adipiscing elit sed.“. Prvý odoslaný príkaz definuje zápis do súboru s číslom 5, offset 0 a veľkosť dát `0x3C 00 00`, respektíve 60 bajtov. Následne je, počínajúc bajtom `0x4C`, reprezentujúci znak `L`, odoslaných prvých 52 bajtov. Karta pomocou odpovede `AF` – `ADDITIONAL_FRAME` potvrdzuje úspešné prijatie dát, a zároveň vyjadruje, že očakáva zvyšných 8 bajtov v nasledujúcej správe. Nasledujúca správa odoslaná do karty začína taktiež bajtom `ADDITIONAL_FRAME`, za ktorým nasledujú zvyšné dáta. Finálna odpoveď z karty je `00` – `OPERATION_OK`, čo značí, že dáta boli úspešne zapísané. [25, 41, 42]

2.3.6 CommitTransaction

Tento príkaz sa používa na potvrdenie zápisu do záložného dátového súboru alebo hodnotového súboru. Nasleduje vždy po príkaze na zápis. V prípade, že karta tento príkaz neobdrží, je zápis neplatný a obsah súboru ostáva nezmenený.

2.3.6.1 Štruktúra príkazu

1
Command code

Štruktúra príkazu: CommitTransaction

Command code: `0xC7`

2.3.6.2 Štruktúra odpovede

1
Return code

Štruktúra odpovede: CommitTransaction

Return code: Značí úspešnosť spracovania príkazu.

2.3.6.3 Príklad

```
3D  05  00 00 00  3C 00 00
4C 6F 72 65 6D 20 69 70 73
75 6D 20 64 6F 6C 6F 72 20
73 69 74 20 61 6D 65 74 2C
20 63 6F 6E 73 65 63 74 65
74 75 72 20 61 64 69 70 69
   73 63 69 6E 67 20 65      ⟶
                             ⟵   AF
AF  6C 69 74 20 73 65 64 2E  ⟶
                             ⟵   00
                          C7 ⟶
                             ⟵   00
```

Príklad príkazu: CommitTransaction

Tento príklad je rozšírením príkladu `WriteData`, viz 2.3.5.3. Jedná sa o zápis do záložného hodnotového súboru s potvrdením zápisu pomocou príkazu `CommitTransaction`.

2.3.7 ReadData

Umožňuje prečítať dáta vo forme bajtového poľa z dátového súboru alebo zo záložného dátového súboru. V prípade čítania väčšieho množstva dát je potrebné odpoveď z karty rozdeliť do častí.

2.3.7.1 Štruktúra príkazu

1	1	3	3
Command code	File number	Offset	Length

Štruktúra príkazu: ReadData

Command code: `0xBD`

File number: Číslo súboru, z ktorého chceme čítať.

Offset: Bajtový posun od začiatku súboru. Od tohoto posunu sa začne čítanie dát. Jedná sa o prirodzené, hexadecimálne číslo vo formáte Little-endian.

Length: Dĺžka dát, ktoré chceme prečítať, v počte bajtov. Hodnota 0 značí prečítanie celého obsahu súboru. Jedná sa o prirodzené, hexadecimálne číslo vo formáte Little-endian.

2.3.7.2 Štruktúra odpovede

1	1-59
Additional frame	Data

Štruktúra príkazu: ReadData - Additional frame

Additional frame: `0xAF`

Data: Nadväzujúce dáta, ktoré chceme prečítať. Maximálna veľkosť je 59 bajtov. V prípade potreby môže byť znovu použitý response `Additional frame`. V prípade, že sa jedná o poslednú odpoveď – s dátami do 59 bajtov – bude return code `OPERATION_OK`.

1	1-59
Return code	Data

Štruktúra príkazu: ReadData

Return code: Značí úspešnosť spracovania príkazu.

Data: Dáta, ktoré chceme prečítať. Maximálna veľkosť je 59 bajtov.

2.3.7.3 Príklad

```
BD 05 00 00 00 3C 00 00   ⟶
                                AF 4C 6F 72 65 6D 20 69 70
                                73 75 6D 20 64 6F 6C 6F 72
                                20 73 69 74 20 61 6D 65 74
                                2C 20 63 6F 6E 73 65 63 74
                                65 74 75 72 20 61 64 69 70
                                69 73 63 69 6E 67 20 65 6C
                          ⟵        69 74 20 73 65 64
                   AF     ⟶
                          ⟵   00 2E
```

Príklad príkazu: ReadData

V príklade je z dátového súboru na karte prečítaný 60-bajtový reťazec „Lorem ipsum dolor sit amet, consectetur adipiscing elit sed.“. Prvý odoslaný príkaz spúšťa čítanie zo súboru s číslom 5, offsetom 0 a čítaním `0x3C 00 00`, respektíve 60 bajtov. Dáta sú rozdelené do dvoch odpovedí. Prvá odpoveď obsahuje prvých 59 bajtov. Jej return code je `ADDITIONAL_FRAME`, čo naznačuje, že odpoveď je neúplná a ďalšie dáta budú odoslané v ďalších odpovediach. Zariadenie si dáta následne vyžiada príkazom `0xAF` – `ADDITIONAL_FRAME`. Return code druhej odpovede je `OPERATION_OK`, čo znamená, že obsahuje poslednú časť dát. V tomto prípade sa jedná o jediný bajt `0x2E`, reprezentujúci znak ’.’. [25, 41, 42]

2.3.8 DeleteFile

Natrvalo deaktivuje vybraný súbor v aplikácii. Deaktivácia zabezpečí to, že súbor už nebude užívateľovi prístupný a jeho identifikačné číslo môže byť znovupoužité v inom súbore. Avšak, alokovaná pamäť pre tento súbor ostáva neuvoľnená. Deaktivácia súboru si môže vyžadovať prvotnú autentifikáciu pomocou príkazu `Authenticate`.

2.3.8.1 Štruktúra príkazu

1	1
Command code	File number

Štruktúra príkazu: DeleteFile

Command code: `0xDF`

File number: Číslo súboru, ktorý chceme deaktivovať.

2.3.8.2 Štruktúra odpovede

1
Return code

Štruktúra odpovede: DeleteFile

Return code: Značí úspešnosť spracovania príkazu.

2.3.8.3 Príklad

```
DF 05   ⟶
        ⟵   00
```

Príklad príkazu: DeleteFile

V príklade je odoslaný príkaz na deaktivovanie súboru s identifikačným číslom 5. Odpoveď z karty je `00` – `OPERATION_OK`, čo značí, súbor bol úspešne deaktivovaný. [25, 41]

2.3.9 DeleteApplication

Natrvalo deaktivuje aplikáciu na karte. Deaktivácia zabezpečí to, že aplikácia nebude užívateľovi prístupná a jej identifikačné číslo môže byť znovupoužité v inej aplikácii. Avšak, alokovaná pamäť pre túto aplikáciu ostáva neuvoľnená. Deaktivácia aplikácie si vyžaduje prvotnú autentifikáciu pomocou príkazu `Authenticate`.

2.3.9.1 Štruktúra príkazu

1	3
Command code	Application ID

Štruktúra príkazu: DeleteApplication

Command code: `0xDA`

Application ID: Unikátne číslo aplikácie, ktorú chceme deaktivovať.

2.3.9.2 Štruktúra odpovede

1
Return code

Štruktúra odpovede: DeleteApplication

Return code: Značí úspešnosť spracovania príkazu.

2.3.9.3 Príklad

```
DA 01 00 00   ⟶
              ⟵   00
```

Príklad príkazu: DeleteApplication

V príklade je odoslaný príkaz na deaktivovanie aplikácie s identifikačným číslom 1. Odpoveď z karty je `00` – `OPERATION_OK`, čo značí, aplikácia bola úspešne deaktivovaná. [25, 41]

2.3.10 Format

Formátuje kartu do výrobného nastavenia. Celá užívateľská pamäť je uvoľnená. Formátovaniu karty musí predchádzať autentifikácia pomocou príkazu `Authenticate`.

2.3.10.1 Štruktúra príkazu

1
Command code

Štruktúra príkazu: Format

Command code: `0xFC`

2.3.10.2 Štruktúra odpovede

1
Return code

Štruktúra odpovede: Format

Return code: Značí úspešnosť spracovania príkazu.

2.3.10.3 Príklad

`FC` ⟶

⟵ `00`

Príklad príkazu: Format

V príklade je odoslaný príkaz na formátovanie karty. Odpoveď z karty je `00` – `OPERATION_OK`, čo značí, karta bola úspešne formátovaná. [25, 41]

2.3.11 Authenticate

Zariadenie a karta si v sérii príkazov šifrovaným spôsobom navzájom potvrdia, že vlastnia rovnaký kľúč, a teda že si môžu dôverovať. Autentifikácia je uskutočňovaná pomocou kľúčov v aplikáciách, s využitím algoritmu `2K3DES`. Po úspešnej výmene kľúča sa nachádza karta v rámci danej aplikácie v autentifikovanom stave. Umožňuje napríklad zmenu obsahu aplikácie, zmenu prístupových práv či prístup k zabezpečeným dátam. Po autentifikácii hlavným kľúčom v hlavnej aplikácii karty – s číslom `0x00 00 00`, viz obrázok 2.6 – je možné napríklad deaktivovať aplikácie, zmeniť hodnotu hlavného kľúča či naformátovať obsah karty. Po zvolení inej aplikácie pomocou `SelectApplication` je autentifikovaný stav ukončený. Autentifikácia pozostáva z niekoľkých krokov:

1. Zvolenie kľúča, podľa ktorého sa chceme autentifikovať.
2. Dešifrovanie 8 bajtov `RandomChallengeB`, obdržaných z karty, zašifrovaných zvoleným kľúčom.
3. Vytvorenie 8-bajtovej `RandomChallengeA`.
4. Odoslanie `RandomChallengeA` spolu s `RandomChallengeB`, zašifrované zvoleným kľúčom, do karty.

5. Dešifrovanie 8 bajtov `RandomChallengeA`, obdržaných z karty, a ich overenie.

6. V prípade, že sa odoslaný `RandomChallengeA` zhoduje s prijatým `RandomChallengeA`, prebehla autentifikácia úspešne.

2.3.11.1 Štruktúra príkazu

1	1
Command code	Key number

Štruktúra príkazu: Authenticate part 1

Command code: `0x0A`

Key number: Číslo kľúča v aplikácii, podľa ktorého sa chceme autentifikovať.

1	16
Additional frame	Encrypted challenges

Štruktúra príkazu: Authenticate part 2

Additional frame: `0xAF`

Encrypted challenges: Šifrované bajtové pole, ktoré obsahuje informácie o hodnote kľúča, hodnote `RandomChallengeA` a hodnote `RandomChallengeB`. Na tvorbu týchto 16 bajtov je potrebné použiť niekoľko operácií:

- `decrypt(bytes)` – zašifruje[31] pole bajtov zvoleným kľúčom pomocou algoritmu `2K3DES`.
- `rotateLeft(bytes)` – každý bajt v poli sa posunie o jedno miesto doľava, bajt úplne vľavo sa presunie úplne napravo.[32]
- `bytes1 XOR bytes2` – bitová operácia – exkluzívny OR.
- `bytes1 + bytes2` – zreťazenie bajtových polí `bytes1` a `bytes2`.

Výsledné odoslané bajtové pole vznikne pomocou nasledujúceho vzorca:

```
decrypt(RndA) + (decrypt(decrypt(RndA) XOR rotateLeft(RndB)))
```

2.3.11.2 Štruktúra odpovede

1	8
Additional frame	RandomChallengeB

Štruktúra príkazu: Authenticate part 1

Additional frame: `0xAF`

RandomChallengeB: Náhodne vygenerované pole bajtov `RandomChallengeB`, zašifrované pomocou zvoleného kľúča. Konkrétne dáta teda prídu v podobe: `encrypt(RandomChallengeB)`.

[31] Šifrovanie je v tomto prípade uskutočnené prostredníctvom operácie na dešifrovanie - decrypt(bytes). Karta pozná jedine operáciu šifrovania - encrypt(bytes), pomocou ktorej, vďaka vlastnostiam algoritmu DES, môže zároveň aj dešifrovať.

[32] rotateLeft(FF 11 22 33) = 11 22 33 FF

1	8
Return code	RandomChallengeA

Štruktúra príkazu: Authenticate part 2

Return code: Značí úspešnosť spracovania príkazu.

RandomChallengeA: Pole bajtov vygenerované zariadením a odoslané v predošlom príkaze do karty – slúži zariadeniu na dodatočné overenie správnosti predošlých operácií. Konkrétne dáta prídu v podobe: `encrypt(rotateLeft(RandomChallengeA))`.

2.3.11.3 Príklad

```
                                0A 00   ⟶
                                        ⟵   AF 1D E2 BF F7 32 00 2E
AF 47 D3 23 DB 92 8E 12 B0
A7 44 84 47 EC 28 A4 71         ⟶
                                        ⟵   00 F7 DC 47 16 66 AB 30 EB
```

Príklad príkazu: Authenticate

Prvý príkaz začne zvolením kľúča `0x00` na autentifikáciu.[33] Tento kľúč má momentálne nastavenú hodnotu na `0x00 00 00 00 00 00 00 00`. V rámci prvej odpovede odošle karta zašifrovaný `RandomChallengeB`. Dešifrovaná hodnota `RandomChallengeB` je `0x64 57 15 50 2F D0 B1 E2`. Zariadenie vygeneruje `RandomChallengeA`: `0x5E 08 D2 EC 10 34 BD F6`. Ďalší postup generovania `Encrypted challenges`:

- `decrypt(RandomChallengeA)` – `0x47 D3 23 DB 92 8E 12 B0`
- `rotateLeft(RandomChallengeB)` – `0x57 15 50 2F D0 B1 E2 64`
- `decrypt(RndA) XOR rotateLeft(RndB)` – `0x10 C6 73 F4 42 3F F0 D4`
- `decrypt(decrypt(RndA) XOR rotateLeft(RndB))` – `0xA7 44 84 47 EC 28 A4 71`
- `Encrypted challenges` – `0x47 D3 23 DB 92 8E 12 B0 + 0xA7 44 84 47 EC 28 A4 71`

Toto bajtové pole je zo zariadenia odoslané v rámci príkazu `ADDITIONAL_FRAME`. Druhá odpoveď obsahuje spolu s return code `OPERATION_OK` bajtové pole, ktoré má po dešifrovaní hodnotu `0x08 D2 EC 10 34 BD F6 5E`. Jedná sa o očakávanú hodnotu – `RandomChallengeA` zarotovanú o jeden bajt doľava. Týmto je autentifikácia úspešne dokončená a karta sa nachádza v rámci danej aplikácie v autentifikovanom stave. [25, 41, 48, 49]

2.3.12 GetApplicationIDs

Vráti zoznam všetkých vytvorených aplikácií na karte. Nezahrňuje hlavnú aplikáciu karty s číslom `0x00 00 00`, pretože tá sa na karte nachádza vždy.

2.3.12.1 Štruktúra príkazu

1
Command code

Štruktúra príkazu: GetApplicationIDs

Command code: `0x6A`

[33] Jedná sa o autentifikáciu pomocou hlavného kľúča, pretože v aplikáciách je číslo hlavného kľúča vždy 0x00.

2.3.12.2 Štruktúra odpovede

1	3-57
Return code	Application IDs

Štruktúra príkazu: GetApplicationIDs

Return code: Značí úspešnosť spracovania príkazu.

Application IDs: Pole 3-bajtových čísel reprezentujúcich identifikačné čísla aplikácií. V rámci jednej odpovede je odoslaných 1–19 čísel aplikácií, čo je 3–57 bajtov. Ďalšie identifikačné čísla môžu byť zaslané pomocou `ADDITIONAL_FRAME`.

2.3.12.3 Príklad

```
6A   ⟶
                AF 01 00 00
          02 00 00  03 00 00  04 00 00
          05 00 00  06 00 00  07 00 00
          08 00 00  09 00 00  0A 00 00
          0B 00 00  0C 00 00  0D 00 00
          0E 00 00  0F 00 00  10 00 00
     ⟵    11 00 00  12 00 00  13 00 00
AF   ⟶
                   00
          14 00 00  15 00 00  16 00 00
          17 00 00  18 00 00  19 00 00
     ⟵    1A 00 00  1B 00 00  1C 00 00
```

Príklad príkazu: GetApplicationIDs

Karta odošle pomocou dvoch nadväzujúcich správ všetky identifikačné čísla svojich aplikácií. Na karte je momentálne 28 aplikácií očíslovaných od 1 do 28. Jedná sa o maximálny možný počet vytvorených aplikácií na karte.

2.3.13 Ďalšie príkazy

MIFARE DESFire EV1 poskytuje mnoho príkazov na prácu s jej obsahom. Nasledujúci zoznam poskytuje ďalšie, ktoré boli v rámci tejto baklárskej práce analyzované alebo implementované:

- `GetFileIDs` – Vráti zoznam vytvorených súborov vo zvolenej aplikácii.
- `GetFileSettings` – Vráti nastavenia zvoleného súboru. Jedná sa o rovnaké parametre ako tie, definované príkazom na tvorbu daného typu súboru.
- `FreeMem` – Vráti veľkosť doposiaľ nevyužitej užívateľskej pamäte v bajtoch.
- `AbortTransaction` – Opak `CommitTransaction`, zneplatní všetky vytvorené zmeny.
- `CreateValueFile` – Vytvorí hodnotový súbor, ktorý má v sebe uložené jediné číslo vo formáte 4-bajtového Litte-endian so znamienkom – klasický 32-bitový Integer.
- `GetKeySettings` – Vráti nastavenia kľúčov vo zvolenej aplikácii. Jedná sa o dva bajty, ktoré boli špecifikované v `CreateApplication` – `App Key Settings` a `Key Settings 2`. [25, 41]

2.3.14 Prerušenie spojenia

Rovnako ako všetky bezkontaktné čipové karty, MIFARE DESFire EV1 nemá vlastný zdroj energie. Energiu získava z priloženého zariadenia, ktoré s ním momentálne komunikuje. Môže sa stať, že užívateľ, používajúci kartu počas jej komunikácie s čítačkou, preruší spojenie oddialením oboch zariadení od seba. Táto situácia sa nazýva „card tearing“.

2.3.14.0.1 Prerušenie spojenia počas odosielania príkazu: Problém napríklad nastane, ak užívateľ preruší spojenie počas toho, čo karta zapisuje údaje do svojej pamäti. Karta príde o energiu, zápis nedokončí a dáta v pamäti ostanú neúplné. Tento problém sa dá riešiť pomocou „zrkadlenia“ obsahu, kedy je na obsah súboru vyhradené dvojnásobné miesto. Zápis dát sa uskutočňuje len do jednej časti súboru a je považovaný za platný až v prípade, že karta dostane explicitnú informáciu od zariadenia o ukončení zasielania dát. Tento mechanizmus je v MIFARE DESFire EV1 využívaný v záložnom dátovom súbore, s využitím príkazov `WriteData` a `CommitTransaction`. Pri zápise dát do štandardného dátového súboru, ktorý `CommitTransaction` nevyužíva, nie je garantovaná úplnosť prijatých dát a akákoľvek správa môže byť spracovaná len čiastočne. [50]

Tento fakt je možné odpozorovať testovaním prerušenia spojenia počas zápisu veľkého množstva dát do štandardného dátového súboru: Po niekoľkých opakovaniach testu bol vždy výsledok taký, že zariadenie odoslalo pomocou `WriteData` 60-bajtový `ADDITIONAL_FRAME`,[34] no po prerušení bolo skutočne zapísaných napríklad len 38 bajtov.

2.3.14.0.2 Prerušenie spojenia počas prijímania odpovede: V opačnom prípade môže byť spojenie prerušené počas toho, čo karta odosiela svoje dáta do zariadenia. Dáta môžu prísť v neúplnom stave, čo je zariadenie schopné detekovať, no pomocou vylepšenia karty batériou,[35] je možné tento problém zmenšiť. Pred tým, než karta odošle svoju odpoveď do zariadenia, sa s jeho pomocou dočasne nabije tak, aby bola schopná odoslať plnú odpoveď. MIFARE DESFire EV1 týmto mechanizmom vždy zaručuje plné odoslanie aj maximálnej, 60-bajtovej odpovede. V prípade prerušenia teda vždy odošle buď plnú odpoveď, alebo neodošle žiadnu. Vďaka tomu má zariadenie istotu, že ak nejakú odpoveď obdrží, bude určite v platnom stave. [50]

Tento fakt je možné odpozorovať testovaním prerušenia spojenia počas čítania veľkého množstva dát zo súboru. Po niekoľkých opakovaniach testu bol vždy výsledok rovnaký: Karta odoslala pomocou response na `ReadData` úplný, 60-bajtový `ADDITIONAL_FRAME`. Po prerušení spojenia sa už ďalší `ADDITIONAL_FRAME` neodoslal, pretože na jeho úplný, 60-bajtový prenos, nemala karta dostatok energie.

[34] na ktorý dokonca stihla karta po prerušení spojenia reagovať odpoveďou 0xAF
[35] kondenzátorom

2.4 Požiadavky na knižnicu

Ďalej sa pozrieme na funkčné a nefunkčné požiadavky, na ktoré bol kladený dôraz počas návrhu a tvorby knižnice.

2.4.1 Funkčné požiadavky

Funkčné požiadavky popisujú funkcionalitu systému. Upresňujú to, čo systém má alebo nemá užívateľovi poskytovať.

F1 Zápis dát: Knižnica umožňuje zápis dát v podobe bajtového poľa do karty.

F2 Čítanie dát: Knižnica umožňuje čítanie dát v podobe bajtového poľa z karty.

F3 Odstránenie dát: Knižnica poskytuje funkcie na zneplatnenie dát v podobe deaktivácie súboru či aplikácie, alebo na úplné odstránenie dát pomocou formátovania karty.

F4 Autentifikácia: Užívateľ je schopný autentifikovať sa pomocou algoritmu `2K3DES`, na získanie prístupu k zabezpečeným operáciám.

F5 Informácie o karte: Užívateľ si pomocou príkazov knižnice môže vyžiadať informácie o obsahu karty, ako napríklad zoznam vytvorených aplikácií či konfiguráciu dátových súborov.

F6 Manipulácia obsahu karty: Využitím príkazov v knižnici môže užívateľ manipulovať s obsahom karty – vytvárať a odstraňovať aplikácie a ich súbory.

F7 Zabezpečenie obsahu karty: Knižnica umožňuje zabezpečenie obsahu karty. Užívateľ môže obmedziť prístup k aplikačným a súborovým operáciám na kľúč.

2.4.2 Nefunkčné požiadavky

Nefunkčné požiadavky popisujú obmedzenia, v rámci ktorých systém funguje. Môže sa napríklad jednať o obmedzenia na podporované technológie, dostupnosť systému alebo jeho výkon.

N1 Podpora Android: Knižnica funguje výhradne pre použitie v operačnom systéme Android.

N2 Podpora MIFARE DESFire EV1: Knižnica je schopná komunikovať s bezkontaktnou čipovou kartou typu MIFARE DESFire EV1.

N3 Podpora NFC: Komunikácia s kartou prebieha s využitím technológie NFC.

N4 Plná funkčnosť v offline podmienkach: Použitie knižnice je plne funkčné na zariadeniach, bez potreby pripojenia k internetu.

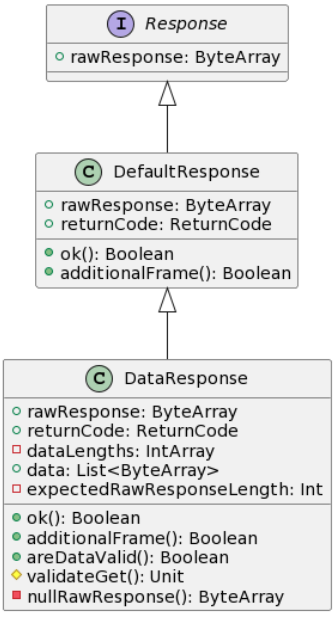
N5 Rozšíriteľnosť: Objektový návrh knižnice umožňuje jednoduché rozšírenie knižnice o ďalšie príkazy.

Kapitola 3
Návrh a realizácia

Na základe získaných znalostí z predošlých kapitol, bude v tejto časti popísaný finálny návrh knižnice a jeho implementácia. V prvých dvoch častiach je popísaný objektový návrh a implementácia pre príkazy zo zariadenia a odpovede z karty. V ďalšej časti sú príklady tried, ktoré tieto implementácie využívajú, od jednoduchších po zložitejšie. Na koniec je popísaná tvorba špeciálnych tried, ktoré buď uľahčujú prácu s vytváraním nových príkazov, alebo spájajú viacero príkazov do jedného. [1]

3.1 Response

Štruktúru „response“, alebo odpovede z karty, viz 2.3.0.0.2, tvorí jeden interface a dve triedy.



Obr. 3.1 Response – diagram tried

[1] Odkaz: `https://gitlab.fit.cvut.cz/popovma1/mifaredesfireev1library`

3.1.1 interface Response

Interface `Response` je základnou štruktúrou, ktorá reprezentuje odoslané dáta z karty do zariadenia. Definuje bajtové pole – `ByteArray rawResponse`. Toto pole obsahuje plnú, nespracovanú odpoveď z karty. Jedná sa o „read-only“ premennú, čo znamená, že hodnota, ktorá bola do `rawResponse` priradená, je nemenná. Read-only premenná sa značí v Kotline pomocou kľúčového slova `val`. Všetky ďalšie premenné tried, ktoré sa vyskytnú v tejto práci, sú taktiež definované pomocou `val`, respektíve ako read-only.[2] [51]

3.1.2 class DefaultResponse

Realizáciou interface `Response` je trieda `DefaultResponse`. Táto trieda slúži na reprezentáciu odpovedí, ktoré sú definované len pomocou jediného bajtu – return code. Väčšina odpovedí na príkazy je definovaná práve týmto spôsobom. Napríklad `CreateApplication` – 2.3.1.2, `CreateStdDataFile` – 2.3.3.2, alebo `Format` – 2.3.10.2. V triede sa nachádzajú dve premenné:

- `rawResponse: ByteArray` – Premenná zdedená z `DefaultResponse`. Jej hodnota je priradená v konštruktore, v momente vytvárania objektu.
- `returnCode: ReturnCode` – Premenná typu `enum ReturnCode`. `returnCode` vznikne extrahovaním prvého[3] bajtu z `rawResponse`. Niektoré hodnoty `enum ReturnCode` je možné vidieť v tabuľke 2.1.

V tejto triede sa ďalej nachádzajú aj dve verejné metódy, ktoré len uľahčujú prístup k premennej `returnCode`:

- `ok(): Boolean` – Vráti `true`, ak hodnota `returnCode` je `OPERATION_OK`.
- `additionalFrame(): Boolean` – Vráti `true`, ak hodnota `returnCode` je `ADDITIONAL_FRAME`.

3.1.3 class DataResponse

Na rozdiel od triedy `DefaultResponse`, ktorá je priamo využívaná na jednoduché, jedno-bajtové odpovede, slúži táto trieda ako návrh pre špecifickejšie typy odpovedí s extra dátami. Je využívaná len ako abstraktná trieda, napriek tomu, že je definovaná voľnejšie – teda ako klasická trieda, z ktorej sa dá dediť. Rozhranie `DataResponse` využívajú napríklad odpovede na `ReadData` – 2.3.7.2, `Authenticate` – 2.3.11.2, alebo `GetKeySettings` – 2.3.13.

`DataResponse` dedí všetky členy od `DefaultResponse`, ku ktorým pridáva nasledujúce premenné a metódy:

- `dataLenghts: IntArray` – Bajtové dĺžky všetkých parametrov, ktoré sú zaslané v rámci odpovede. Definuje, ako sa majú rozdeliť bajty v `rawResponse` na logické časti. Napríklad:
  - Response na `Authenticate Part 1` sa delí na dve časti: 1 bajt pre `Additional frame` a 8 bajtov pre `RandomChallengeB`.
  - Response na `GetKeySettings` sa delí na časti 1, 1 a 1.
  - Response na `ReadData` sa delí na dve časti: 1 bajt pre response code a zvyšné bajty pre obsah správy – tie sú vypočítané dynamicky podľa dĺžky `rawResponse`.
- `data: List<ByteArray>` – Jedná sa o bajty z odpovede `rawResponse`, logicky rozdelené podľa predošlého parametru `dataLenghts`. Obsah `rawResponse` pre `GetKeySettings` by bol v prípade úspechu rozdelený na `data[0]` – bajt reprezentujúci response code, `data[1]` – bajt

[2] Pokiaľ to nebude zmienené inak. Jedná sa len o dve výnimky v triede LongCommand.
[3] a v tomto prípade aj jediného

reprezentujúci `App Key Settings`, `data[2]` – bajt reprezentujúci `Key Settings 2`. V prípade neúspechu by `data[0]` naďalej obsahoval príslušný response code, reprezentujúci neúspech. `data[1]` a `data[2]` by boli vyplnené z dôvodu null-safety na hodnotu `0x00`, bez ďalšieho využitia.

- `expectedRawResponseLength: Int` – Súčet hodnôt z `dataLengths`. Jedná sa o pomocnú súkromnú premennú, reprezentujúcu očakávanú dĺžku dát v prípade, že odpoveď prebehla úspešne.

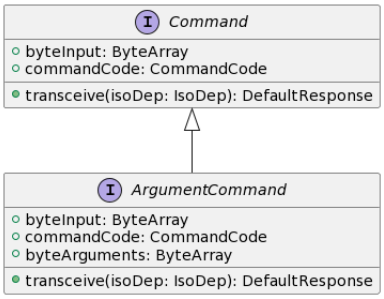
- `areDataValid(): Boolean` – Vráti `true`, ak sú prijaté dáta vo validnom stave, respektíve ak bol prijatý očakávaný response code. Očakávaný response code je buď `OPERATION_OK` alebo `ADDITIONAL_FRAME`, v závislosti od kontextu. Metóda preto využíva už definované metódy `ok()` a `additionalFrame()`.

- `validateGet(): Unit` – Metóda volá `areDataValid()`. V prípade vrátenej hodnoty `false`, program vyhodí výnimku s hláškou „Property unavailable due to error return code: …“. Táto metóda sa volá vždy, keď si chce užívateľ vyžiadať dáta zo získanej response pomocou príslušných `get()` metód. Ak má response code chybovú hodnotu, žiadne prídavné dáta sa v response nenachádzajú, a teda užívateľ si ich nemôže vyžadovať. Užívateľ je preto povinný najprv samostatne overiť platnosť dát pomocou verejne prístupnej metódy `ok()`, a až potom môže bezpečne vyžadovať dáta spracované z odpovede.[4]

- `nullRawResponse(): ByteArray` – Pomocná privátna metóda, na zaistenie null-safety – zapríčiní, že triedna premenná `data` bude bezpečne inicializovaná aj v prípade chyby. Ak má odpoveď chybovú hodnotu, `rawResponse` obsahuje chybový response code, no neobsahuje zvyšné bajty, ktoré by prišli v prípade úspechu. `nullRawResponse()` vytvorí falošný `rawResponse`, ktorý obsahuje pôvodný chybový response code, a taktiež bajty s hodnotou `0x00`, nahradzujúce extra dáta, ktoré boli očakávané. Tento `nullRawResponse` je následne spracovaný namiesto `rawResponse`, čo zapríčiní bezpečnú inicializáciu premennej `data`. Na užívateľa to nemá vplyv, pretože na dosah k týmto falošným dátam mu bráni `validateGet()`. Zároveň má vždy prístup k originálnemu, neupravenému `rawResponse` a `returnCode`.

3.2 Command

Štruktúru „command“, alebo príkazu zo zariadenia do karty, viz 2.3.0.0.1, tvoria dve rozhrania.



Obr. 3.2 Command – diagram tried

[4] Táto funkcionalita je inšpirovaná podľa HTTP knižnice Python Requests. `www.python-requests.org`

3.2.1 interface Command

Interface `Command` je základnou štruktúrou, ktorá reprezentuje odoslaný príkaz zo zariadenia do karty. Definuje najjednoduchší typ príkazu, ktorý obsahuje jediný bajt – command code. Príkazy ako `Format` – 2.3.10.1, `GetApplicationIDs` – 2.3.12.1, či `AdditionalFrame` – 2.3.7.3 si vystačia s touto jedno-bajtovou štruktúrou. Obsahuje dvojicu premenných a jednu metódu:

- `commandCode: CommandCode` – Premenná typu `enum CommandCode`. Jedná sa o unikátnu bajtovú hodnotu, charakterizujúcu daný typ príkazu.
- `byteInput: ByteArray` – Bajtové pole, ktorého obsah bude odoslaný ako príkaz do karty. V tomto prípade obsahuje `byteInput` jediný bajt, ktorým je hodnota `commandCode`.
- `transceive(isoDep: IsoDep): DefaultResponse` – Názov tejto metódy vznikol spojením slov transmit – vysielať, a receive – prijímať. Metóda odošle dáta z premennej `byteInput` do karty. Následne z karty obdrží bajtové pole reprezentujúce odpoveď. Táto odpoveď sa vloží ako `rawResponse` do konštruktora triedy `DefaultResponse`, prípadne do akéhokoľvek potomka tejto triedy, ktorý ju náležite spracuje. V tejto podobe je odpoveď vrátená užívateľovi. Na prenos dát medzi zariadením a kartou je využívaná metóda s rovnakým názvom `transceive(byteInput: ByteArray): ByteArray` od triedy `IsoDep`.

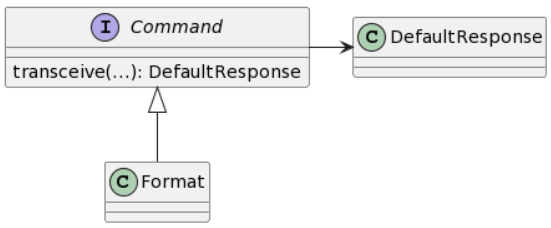
3.2.2 interface ArgumentCommand

Interface `ArgumentCommand` je rozšírením `Command`. Definuje štruktúru pre príkazy, ktoré za command code majú ďalšie bajty, v podobe parametrov príkazu. Toto rozhranie využívajú príkazy ako `CreateApplication` – 2.3.1.1, `WriteData` – 2.3.5.1, či `Authenticate` – 2.3.11.1. Tie v konštruktore príjmu od užívateľa parametre príkazu, spracujú ich do formy bajtového poľa, a uložia do novej premennej `byteArguments: ByteArray`. Premenná `byteInput`, ktorej bajty sa posielajú v rámci `IsoDep.transceive()`, je predefinovaná ako zreťazenie `commandCode` a `byteArguments`.

3.3 Konkrétne príkazy a odpovede

Účelom tejto časti je pozrieť sa na niektoré triedy, ktoré využívajú návrhy command a response na realizáciu konkrétnych príkazov a odpovedí. Začneme s jednoduchšími príkazmi a postupne sa dostaneme k zložitejším.

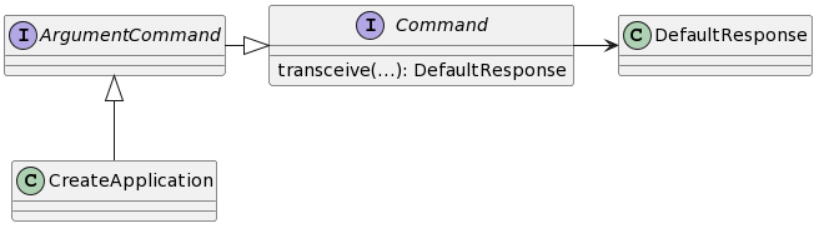
3.3.1 Format



Obr. 3.3 Format – diagram tried

Z pohľadu štruktúry ide o najjednoduchší príkaz na vytvorenie, pretože jeho definícia obsahuje len command code. Definícia odpovede na tento príkaz taktiež obsahuje len response code, viz 2.3.10.1. Na vytvorenie tejto triedy stačí zdediť `interface Command` a vyplniť hodnotu premennej `commandCode`. Príkaz užívateľ zašle do karty volaním: `Format().transceive(isoDep)`.

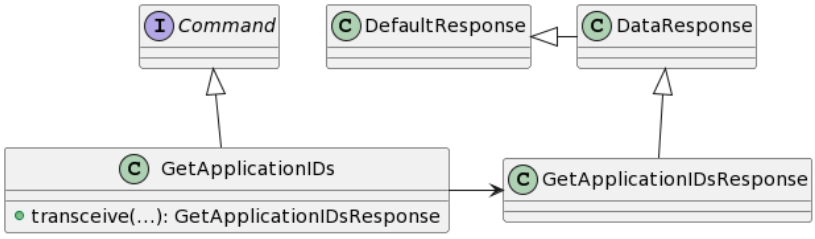
3.3.2 CreateApplication



Obr. 3.4 CreateApplication – diagram tried

`CreateApplication` vo svojom príkaze obsahuje prídavné argumenty, ktoré musia byť pripojené za command code, preto je v tomto prípade využitý `ArgumentCommand`, viz 2.3.1.1. Tieto argumenty vloží užívateľ pomocou konštruktora, a následne budú prevedené do `byteArguments`. Odpoveď na príkaz si vystačí s `DefaultResponse`, pretože obsahuje len response code. Príkaz užívateľ zašle do karty volaním, v ktorom špecifikuje hodnotu parametrov. Povinný parameter je len `Application ID`, nepovinné parametre `App Key Settings` a `Key Settings 2` môže automaticky doplniť knižnica predvolenými hodnotami. Príkaz je možné odoslať volaním: `CreateApplication(appID = 1).transceive(isoDep)`.

3.3.3 GetApplicationIDs



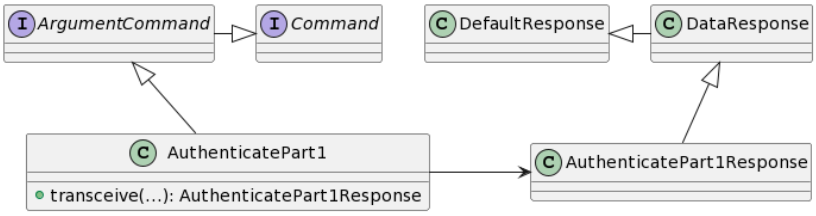
Obr. 3.5 GetApplicationIDs – diagram tried

Príkaz `GetApplicationIDs` pozostáva z jediného bajtu command code. Odpoveď obsahuje prídavné dáta v podobe listu identifikátorov, viz 2.3.12.1. Z tohoto dôvodu je potrebné využiť `DataResponse`. Realizáciou `DataResponse` je trieda `GetApplicationIDsResponse`, ktorá definuje a spracováva štruktúru očakávanej odpovede. Po úspešnom spracovaní odpovede, môže užívateľ využiť `get()` metódu na definovanú premennú `appIDList: List<Int>`, v ktorej sa nachádzajú identifikátory aplikácií.

Výpis kódu 3.1 Príklad získania appIDList z GetApplicationIDs

```
val response = GetApplicationIDs().transceive(isoDep)
if(response.ok()) println(response.appIDList)
```

3.3.4 AuthenticatePart1



Obr. 3.6 AuthenticatePart1 – diagram tried

Príkaz odosiela spolu s command code prídavné dáta, viz 2.3.11.1, čo znamená, že je potrebné využiť `ArgumentCommand`. Odpoveď taktiež obsahuje za response code ďalšie bajty a s využitím `DataResponse` definuje `AuthenticatePart1Response`, viz 2.3.11.2. Po úspešnom spracovaní odpovede sa sprístupní hodnota v premennej `randomChallengeBEncrypted: ByteArray`.

Výpis kódu 3.2 Príklad získania randomChallengeB z AuthenticatePart1

```
val authPt1Response = AuthenticatePart1(keyNo).transceive(isoDep)
if(!authPt1Response.areDataValid()) return authPt1Response

val randomChallengeB = decrypt(authPt1Response.randomChallengeBEncrypted)
```

3.4 Špeciálne pomocné triedy

Cieľom tejto časti je pozrieť sa na niektoré triedy, ktoré uľahčujú prácu s vytváraním nových príkazov, alebo spájajú viacero príkazov do jedného.

3.4.1 LongCommand

Pomocný príkaz – zaobaľuje dlhý príkaz, ktorého veľkosť v bajtoch by sa nezmestila do jediného volania `IsoDep.transceive()`. Táto trieda je využívaná triedou `WriteData` – 2.3.5.3: Príkaz s dátami na zápis, presahujúci 60 bajtov, je automaticky rozdelený do menších častí a odoslaný pomocou `ADDITIONAL_FRAME` v rámci viacerých volaní `IsoDep.transceive()`. Posledný obdržaný response code v `LongCommand` je vrátený ako response code pre `WriteData`. Užívateľ knižnice môže vďaka tomuto odoslať dáta ľubovoľnej dĺžky jediným príkazom a nemusí riešiť rozdeľovanie dát na 60-bajtové časti pomocou `ADDITIONAL_FRAME`.

3.4.2 LongResponse

Pomocný príkaz – odošle `byteInput` a spojí dlhú odpoveď, ktorá by sa nemusela zmestiť do jediného volania `IsoDep.transceive()`. Táto trieda je využívaná triedami `ReadData` – 2.3.7.3 a `GetApplicationIDs` – 2.3.12.3. V prípade, že odpoveď presahuje veľkosť 60 bajtov, je rozdelená na viacero častí, ktoré si je potrebné vyžiadať pomocou príkazu pomocou `ADDITIONAL_FRAME`. `LongResponse` toto vyžadovanie zautomatizuje a užívateľovi vráti jednu ucelenú odpoveď, s príslušným response code a extra dátami.

3.4.3 WriteBackupData

Príkaz `WriteBackupData` zlučuje dvojicu príkazov – `WriteData` a `CommitTransaction` – do jedného. Obe príkazy je potrebné použiť pri zápise do záložného súboru, viz 2.3.6.3. V prípade úspešného zápisu pomocou `WriteData` je automaticky odoslaný `CommitTransaction` na potvrdenie zápisu, čo zjednodušuje prácu s knižnicou.

3.4.4 Authenticate

Príkaz `Authenticate` vždy pozostáva z dvoch častí, ktoré na seba nadväzujú, viz 2.3.11.3. Autentizáciu začne `Authenticate Part 1`, pričom sa špecifikuje kľúč. `Authenticate Part 2` pracuje s náhodne generovanými výzvami. Na konci je ešte potrebné samostatne skontrolovať výzvu vrátenú z `Authenticate Part 2` response. Trieda `Authenticate` zaobaľuje tieto dva príkazy, rovnako ako konečnú kontrolu. Užívateľ je na autentizáciu povinný zadať len číslo kľúča a jeho hodnotu,[5] volaním: `Authenticate(keyNo = 0).transceive(isoDep)`.

[5] Predvolená hodnota kľúča je pole bajtov s hodnotami 0x00.

Kapitola 4
Testovanie

Táto kapitola sa venuje testovaniu vytvorenej knižnice. Cieľom testovania bolo overenie funkčnosti a zaistenie kvality vytvoreného kódu. Príkazy boli otestované pomocou rôznych typov testov, od jednotkových testov po systémové, kvôli ktorým vznikla vlastná testovacia aplikácia, komunikujúca s kartou.

Na účely zjednodušenia testovania kódu boli využité dva nástroje: Kotest[1] a MockK[2]. Kotest je testovací framework pre Kotlin, ktorý umožňuje jednoduchú tvorbu prehľadných testov. Poskytuje taktiež možnosť overovania výsledkov pomocou assert knižnice či pomocou property testov. MockK je nástroj na mockovanie vytvorený pre Kotlin. Mockovanie uľahčuje izolované testovanie objektov, ktoré sú závislé na ďalších objektoch tým, že tieto ďalšie objekty nahradí falošnými testovacími imitáciami s preddefinovaným chovaním.

4.1 Jednotkové testy

Jednotkové testy sú jednoduché testy, ktoré testujú malé časti kódu. Sú automatické a trvajú veľmi krátku dobu. Tento typ testov bol využívaný napríklad na otestovanie funkcií na konverziu medzi dátovými typmi či na operácie pracujúce s bajtovými poliami. Nasledujúci kód testuje funkciu `splitData()`, ktorá rozdelí bajtové pole na definované podčasti. Táto metóda je využívaná pri spracovávaní odpovede z `rawResponse`.

Výpis kódu 4.1 Testovanie funkcie splitData()

```
test("splitData") {
    val arr: ByteArray = "00112233445566778899AA".toByteArray()

    val splitArr = arr.splitData(4, 3, 4)
    assertEqualsSplitData(splitArr[0], "00112233")
    assertEqualsSplitData(splitArr[1], "445566")
    assertEqualsSplitData(splitArr[2], "778899AA")
    ...
    val splitArr3 = arr.splitData(0, 11, 0)
    assertEqualsSplitData(splitArr3[0], "")
    assertEqualsSplitData(splitArr3[1], "00112233445566778899AA")
    assertEqualsSplitData(splitArr3[2], "")
}
```

[1] https://kotest.io/
[2] https://mockk.io/

4.2 Integračné testy

Jedná sa o testy väčšieho rozsahu, ktoré overujú funkčnosť jednotlivých komponent. Sú odohrávané v prostredí, ktoré sa približuje reálnemu využitiu. V tomto prípade to znamená, že v testoch je pomocou mockovania napodobňované chovanie NFC karty. Príkazy odošlú svoje dáta, na ktoré dostanú od simulovanej karty odpoveď. Následne je overená správnosť spracovania tejto odpovede, v prípadoch úspešného alebo neúspešného response code.

Napríklad príkaz `Authenticate` vo svojej prvej časti obdrží po zavolaní `transceive()` namockované, šifrované dáta, na ktoré musí náležite reagovať ich dešifrovaním a podľa nich zostaviť druhú časť príkazu.

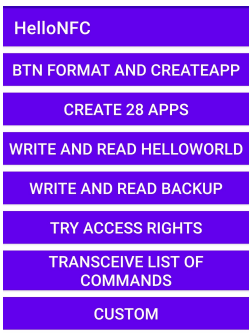
Najkomplikovanejšie testy boli vytvorené pre triedy `LongCommand` a `LongResponse`, u ktorých bol simulovaný a otestovaný prenos 1500 bajtov rozdelených do 26 `transceive()` volaní.

4.3 Systémové testy

Systémové testy vznikli za účelom otestovania programu ako celok. Keďže sa jedná o knižnicu pre Android, ktorá má byť využívaná v konkrétnych aplikáciách a konkrétnych zariadeniach, bolo potrebné vytvoriť Android aplikáciu, ktorá by túto knižnicu využila.[3] Testovacia aplikácia, ktorá vznikla je schopná kombinovaním príkazov z knižnice uskutočňovať jednoduché operácie:

- Formátovanie karty a následne vytvorenie aplikácie na karte.
- Vytvorenie maximálneho možného počtu aplikácií na karte.
- Vytvorenie súboru na karte, zápis reťazca „Hello world“, a následné overovacie prečítanie obsahu súboru.
- Vytvorenie a manipuláciu so záložnými dátovými súbormi na karte.
- Vytváranie šifrovaných súborov s rôznymi oprávneniami.

Tieto operácie boli navrhnuté tak, aby zahrňovali všetky dôležité príkazy a odpovede, implementované v knižnici. Aplikácia tak slúži na overenie funkčnosti knižnice a zároveň ponúka vývojárom zdrojový kód, ktorý slúži ako ukážka použitia príkazov tejto knižnice.



Obr. 4.1 Snímka obrazovky z aplikácie, určenej na testovanie knižnice

[3] Odkaz: `https://gitlab.fit.cvut.cz/popovma1/hellonfc/tree/testApp`

Záver

Cieľom práce bolo vytvoriť open-source knižnicu, ktorá uľahčí komunikáciu s NFC kartami pre platformu Android.

Výsledkom práce je knižnica naprogramovaná v jazyku Kotlin. Knižnica zjednodušuje komunikáciu s kartou MIFARE DESFire EV1. Ponúka príkazy na zápis, čítanie a odstránenie dát, a taktiež umožňuje autentifikáciu pomocou šifry DES. V prípade potreby je jednoducho rozšíriteľná o ďalšie príkazy. Knižnica nevyžaduje na svoje fungovanie licenciu a je plne funkčná aj za „offline“ podmienok. Finálne riešenie sa po dohode s vedúcim práce nezameriava na typ karty MIFARE Plus EV1, a to z dôvodov nízkej potreby tvorby riešenia pre túto kartu.

V rámci podúloh spojených s touto prácou, bola podrobne analyzovaná technológia NFC pre platformu Android, a taktiež komunikačný protokol pre prenos dát medzi kartou MIFARE DESFire EV1 a Android zariadením. Ďalej bola vykonaná rešerš niekoľkých knižníc v rôznych programovacích jazykoch, pričom boli zhodnotené ich výhody a nevýhody. Po analýze nasledoval návrh a implementácia knižnice. Finálne riešenie bolo podrobené niekoľkým typom testov.

Bibliografia

1. *Možnosti využití RFID-1. část* [online]. itbiz [cit. 2022-04-13]. Dostupné z : https://www.itbiz.cz/rfid-prvni-dil.
2. *What is RFID and how does it work?* [Online]. Sarah Amsler,Sharon Shea [cit. 2022-04-13]. Dostupné z : https://www.techtarget.com/iotagenda/definition/RFID-radio-frequency-identification.
3. *RFID Sticker* [online]. Signfaith (shenzhen) Technology Co., Ltd. [cit. 2022-04-13]. Dostupné z : https://www.alibaba.com/product-detail/Book-Jewelry-ID-Security-RFID-Sticker_60769189978.html.
4. *RFID Card* [online]. Ubuy Co [cit. 2022-04-14]. Dostupné z : https://www.u-buy.com.ua/en/product/2GAREP4-sumup-plus-card-reader-accept-swipe-chip-and-contactless-payments.
5. ŠTĚPÁN, Bc. Petr. *Technicko-technologické trendy v obchodě*. 2015. Dipl. pr. ZÁPADOČESKÁ UNIVERZITA V PLZNI.
6. *Next-Gen Payment Processing Tech: Contactless RFID Credit Cards - Security News* [online]. Trend Micro Incorporated [cit. 2022-04-13]. Dostupné z : https://www.trendmicro.com/vinfo/us/security/news/security-technology/next-gen-payment-processing-tech-rfid-credit-cards.
7. *RFID - wikipedie* [online]. Wikimedia Foundation [cit. 2022-04-13]. Dostupné z : https://cs.wikipedia.org/wiki/RFID.
8. *Near Field Communication – Wikipédia* [online]. Wikimedia Foundation [cit. 2022-04-14]. Dostupné z : https://sk.wikipedia.org/wiki/Near_Field_Communication.
9. *NFC mobile pay* [online]. Merchant Maverick [cit. 2022-04-13]. Dostupné z : https://www.merchantmaverick.com/wp-content/uploads/2015/09/Accepting-NFC-mobile-payments.jpg.
10. *HOW TO (Bluetooth Series): Pair devices with NFC - YouTube* [online]. Sony Europe [cit. 2022-04-14]. Dostupné z : https://youtu.be/bSJTnv8f-Zs?t=126.
11. *NFC: the Technology Behind Tap-and-Go Communication* [online]. Clearbridge Mobile [cit. 2022-04-14]. Dostupné z : https://clearbridgemobile.com/how-secure-are-nfc-mobile-payments/.
12. *What is NFC in Headphones?* [Online]. Peter Susic [cit. 2022-04-13]. Dostupné z : https://headphonesaddict.com/nfc-in-headphones/.
13. *Bezkontaktné čipové karty | Kodys Slovensko* [online]. KODYS SLOVENSKO [cit. 2022-04-16]. Dostupné z : https://www.kodys.sk/produkty/spotrebny-material/spotrebny-material-pre-tlaciarne-plastovych-kariet/plastove-karty-3.

14. *Contactless smart card - Wikipedia* [online]. Wikimedia Foundation [cit. 2022-04-16]. Dostupné z : https://en.wikipedia.org/wiki/Contactless_smart_card.
15. *WHO urges switch to contactless to slow virus transmission* [online]. Finextra Research 2022 [cit. 2022-04-16]. Dostupné z : https://www.finextra.com/newsarticle/35384/who-urges-switch-to-contactless-to-slow-virus-transmission.
16. *ISO7816 part 4 section 6 with Basic Interindustry Commands (APDU level)* [online]. Jacquinot Consulting, Inc. [cit. 2022-04-17]. Dostupné z : https://cardwerk.com/smart-card-standard-iso7816-4-section-6-basic-interindustry-commands/.
17. *RFID Tags, Contactless Smart Card Technology and Electronic Passports: Frequently Asked Questions* [online]. Secure Technology Alliance [cit. 2022-04-17]. Dostupné z : https://www.securetechalliance.org/resources/pdf/RFID_and_Contactless_Smart_Cards_FAQ_FINAL_042105.pdf.
18. *What is the difference between ISO/IEC 14443 and ISO 7816 smart cards* [online]. dim (https://electronics.stackexchange.com/users/107479/dim) [cit. 2022-04-22]. Dostupné z : https://electronics.stackexchange.com/a/229851.
19. *MIFARE - The Brand of Contactless IC Products* [online]. NXP Semiconductors Austria GmbH Styria [cit. 2022-04-18]. Dostupné z : https://www.mifare.net/en/about-mifare/.
20. *Universal contactless smart card reader symbol* [online]. Wikimedia Foundation [cit. 2022-04-16]. Dostupné z : https://en.wikipedia.org/wiki/Contactless_smart_card#/media/File:Universal_Contactless_Card_Symbol.svg.
21. *MIFARE - Wikipedia* [online]. Wikimedia Foundation [cit. 2022-04-18]. Dostupné z : https://en.wikipedia.org/wiki/MIFARE.
22. *How they hacked it: The MiFare RFID crack explained* [online]. Geeta Dayal [cit. 2022-04-18]. Dostupné z : https://www.computerworld.com/article/2537817/how-they-hacked-it--the-mifare-rfid-crack-explained.html.
23. *Security Statement on Crypto1 Implementations* [online]. Johannes Grüll [cit. 2022-04-18]. Dostupné z : https://www.mifare.net/en/products/chip-card-ics/mifare-classic/security-statement-on-crypto1-implementations/.
24. *MIFARE DESFire EV1 | NXP Semiconductors* [online]. NXP Semiconductors [cit. 2022-04-19]. Dostupné z : https://www.nxp.com/products/rfid-nfc/mifare-hf/mifare-desfire/mifare-desfire-ev1:MIFARE_DESFIRE_EV1_2K_8K.
25. *Mifare® Application Programming Guide for DESFire®* [online]. GIGA-TMS INC. [cit. 2022-04-11]. Dostupné z : https://scancode.ru/upload/iblock/de4/mifare_application_programming_guide_for_desfire_rev.e.pdf.
26. *Starting development with TapLinx SDK - AN11876* [online]. NXP B.V. 2020 [cit. 2022-04-10]. Dostupné z : https://www.mifare.net/wp-content/uploads/2020/09/AN11876.pdf.
27. *MIFARE Plus EV1 | NXP Semiconductors* [online]. NXP Semiconductors [cit. 2022-04-20]. Dostupné z : https://www.nxp.com/products/rfid-nfc/mifare-hf/mifare-plus/mifare-plus-ev1:MIFARE_PLUS_EV1_2K_4K.
28. *MIFARE Plus EV1 - MF1P(H)x1y1* [online]. NXP B.V. 2019 [cit. 2022-04-20]. Dostupné z : https://www.nxp.com/docs/en/data-sheet/MF1P_H_X1Y1_SDS.pdf.
29. *NXP Introduces MIFARE DESFire EV3 IC, Ushers In New Era of Security and Connectivity for Contactless Smart City Services | NXP Semiconductors - Newsroom* [online]. NXP Semiconductors [cit. 2022-04-20]. Dostupné z : https://media.nxp.com/news-releases/news-release-details/nxp-introduces-mifare-desfire-ev3-ic-ushers-new-era-security-and/.

30. *NFC basics | Android Developers* [online]. Android Developers [cit. 2022-04-21]. Dostupné z : https://developer.android.com/guide/topics/connectivity/nfc/nfc.

31. *Advanced NFC overview | Android Developers* [online]. Android Developers [cit. 2022-04-21]. Dostupné z : https://developer.android.com/guide/topics/connectivity/nfc/advanced-nfc.

32. *NFC Data Exchange Format – Wikipedie* [online]. Wikimedia Foundation [cit. 2022-04-21]. Dostupné z : https://cs.wikipedia.org/wiki/NFC_Data_Exchange_Format.

33. *TagTechnology | Android Developers* [online]. Android Developers [cit. 2022-04-22]. Dostupné z : https://developer.android.com/reference/android/nfc/tech/TagTechnology.

34. *Tag | Android Developers* [online]. Android Developers [cit. 2022-04-22]. Dostupné z : https://developer.android.com/reference/android/nfc/Tag.

35. *Vyvíjíme pro Android: Intenty, intent filtry a permissions - Zdroják* [online]. Matěj Konečný [cit. 2022-04-24]. Dostupné z : https://zdrojak.cz/clanky/vyvijime-pro-android-intenty-intent-filtry-a-permissions/.

36. *TapLinx | MIFARE* [online]. NXP Semiconductors Austria GmbH Styria [cit. 2022-04-10]. Dostupné z : https://www.mifare.net/en/products/tools/taplinx/.

37. *NFC TagInfo by NXP - Apps on Google Play* [online]. NXP Semiconductors [cit. 2022-04-11]. Dostupné z : https://play.google.com/store/apps/details?id=com.nxp.taginfolite.

38. *GitHub - andrade/nfcjlib: NFC Java library v0.4* [online]. andrade [cit. 2022-04-11]. Dostupné z : https://github.com/andrade/nfcjlib.

39. *GitHub - skjolber/desfire-tools-for-android: Open source MIFARE DESFire EV1 NFC library for Android* [online]. skjolber [cit. 2022-04-11]. Dostupné z : https://github.com/skjolber/desfire-tools-for-android.

40. *Re: Mifare DESFire EV 1* [online]. TapLinx Support [cit. 2022-04-11]. Dostupné z : https://www.mifare.net/support/forum/topic/mifare-desfire-ev-1/.

41. *MIFARE DESFire EV3 feature and functionality comparison to other MIFARE DESFire products - AN12752.pdf* [online]. NXP B.V. 2021. [cit. 2022-04-28]. Dostupné z : https://www.nxp.com/docs/en/application-note/AN12752.pdf.

42. *libfreefare/mifare-desfire.c at master · nfc-tools/libfreefare · GitHub* [online]. nfc-tools [cit. 2022-04-28]. Dostupné z : https://github.com/nfc-tools/libfreefare/blob/master/libfreefare/mifare_desfire.c.

43. *Hex Editing For Beginners* [online]. Inv Softworks LLC [cit. 2022-04-27]. Dostupné z : http://www.flexhex.com/docs/howtos/hex-editing.phtml.

44. *Endianita – Wikipédia* [online]. Wikimedia Foundation [cit. 2022-04-27]. Dostupné z : https://sk.wikipedia.org/wiki/Endianita.

45. *AN11004 MIFARE DESFire as Type 4 Tag - AN11004.pdf* [online]. NXP B.V. 2013. [cit. 2022-04-28]. Dostupné z : https://www.nxp.com/docs/en/application-note/AN11004.pdf.

46. *Java | MIFARE* [online]. NXP Semiconductors Austria GmbH Styria [cit. 2022-04-29]. Dostupné z : https://www.mifare.net/developer/javadoc/java/.

47. *Message Authentication* [online]. tutorialspoint [cit. 2022-04-29]. Dostupné z : https://www.tutorialspoint.com/cryptography/message_authentication.html.

48. *java - DES Send and Receive Modes for DESFire Authentication - Stack Overflow* [online]. İsmet Alkan - https://stackoverflow.com/users/1275577/ismet-alkann [cit. 2022-04-30]. Dostupné z : https://stackoverflow.com/a/14160507.

49. *DESFire authentication in C Sharp | Chaucery Blog* [online]. James Minchin [cit. 2022-04-30]. Dostupné z : https://blog.chaucery.com/2017/02/desfire-authentication-in-c.html.

50. *The anti-tearing: concept and mechanisms - SpringCard* [online]. Springcard Inc. [cit. 2022-05-02]. Dostupné z : https://www.springcard.com/en/blog/news/the-anti-tearing-concept-and-mechanisms.

51. *Read-Only vs Mutable Variables - Kotlin Crash Course for Programmers* [online]. educative.io [cit. 2022-04-28]. Dostupné z : https://www.educative.io/courses/kotlin-crash-course-for-programmers/B6OPo9G0wqJ.

Obsah priloženého média

- `readme.txt` ..... stručný popis obsahu média
- `links`
  - `HelloNFC - ndef - GitLab` ..... repozitár s Android NFC skúšobnou aplikáciou
  - `HelloNFC - testApp - GitLab` ..... repozitár s testovacou aplikáciou
  - `MifareDesfireEV1Library - GitLab` ..... repozitár knižnice vzniknutej v rámci BP
- `src`
  - `bakalarkatext-master` ..... zdrojové kódy textu práce vo formáte LaTeX
  - `hellonfc-ndef` ..... zdrojové kódy Android NFC skúšobnej aplikácie
  - `hellonfc-testApp` ..... zdrojové kódy testovacej aplikácie
  - `mifaredesfireev1library-master` ..... zdrojové kódy knižnice vzniknutej v rámci BP
- `text`
  - `thesis-popovma1.pdf` ..... text práce vo formáte PDF