VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

PŘÍSTUPOVÝ SYSTÉM PRO APLIKACI CARSHARING

ACCESS CONTROL SYSTEM FOR CARSHARING APPLICATION

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Filip Texl

AUTHOR

VEDOUCÍ PRÁCE — Ing. Petr Dzurenda, Ph.D.

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Filip Texl ***ID:*** 230688

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Přístupový systém pro aplikaci carsharing

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku přístupových systémů v rámci carsharing aplikací. Zaměřte se zejména na použité kryptografické protokoly, komunikační rozhraní, klíčový managment. Seznamte se s vývojem Android mobilních aplikací a komunikační technologií Bluetooth Low Energy (BLE). Po domluvě s vedoucím navrhněte a implementujte offline přístupový systém pro carsharing umožňující odemčení a uzamčení vozidla. Systém bude využívat Android mobilní aplikaci a technologii BLE.

DOPORUČENÁ LITERATURA:

[1] MENEZES, Alfred, Paul C. VAN OORSCHOT a Scott A. VANSTONE. Handbook of applied cryptography. Boca Raton: CRC Press, c1997. Discrete mathematics and its applications. ISBN 0-8493-8523-7.

[2] Android Developers [online]. Google [cit. 2022-09-01]. Dostupné z: https://developer.android.com/

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 26.5.2023

Vedoucí práce: Ing. Petr Dzurenda, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce se věnuje přístupovým systémům s důrazem na carsharingové aplikace, a to jak z teoretického hlediska, kdy analyzuje jejich výhody, nevýhody a použitá kryptografické primitiva, tak i z hlediska praktického, kdy v praktické části je návrh přístupového systému a jeho následná realizace. V bakalářské práci se též využívá technologie Bluetooth Low Energy a Near Field Communication, které jsou opět popsány z teoretického hlediska, zejména pak klíčové pojmy těchto technologií, tak i jejich praktické využití pro přístupové systémy.

KLÍČOVÁ SLOVA

Advanced Encryption Standard, autentizace, Bluetooth, Bluetooth Low Energy, Java, Kotlin, Near field communication, NFC čtečka, přístupové systémy, sokety, symetrická kryptografie, TCP komunikace

ABSTRACT

The bachelor thesis is devoted to access systems with an emphasis on carsharing applications, both from a theoretical point of view, where it analyzes their advantages and disadvantages and uses cryptographic primitives, and also from a practical point of view, where the practical part includes the design of an access system and its subsequent implementation. Bluetooth Low Energy technology and Near field communication are also used in the bachelor thesis, which is again described from a theoretical point of view, especially the key concepts of these technologies, as well as its practical use for access systems.

KEYWORDS

Access control systems, Advanced Encryption Standard, authentication, Bluetooth, Bluetooth Low Energy, Java, Kotlin, Near field communication, NFC reader, sockets, symetric cryptography, TCP communication

TEXL, Filip. *Přístupový systém pro aplikaci carsharing*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 80 s. Bakalářská práce. Vedoucí práce: Ing. Petr Dzurenda, PhD.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Filip Texl
VUT ID autora:	230688
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Přístupový systém pro aplikaci carsharing

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Petru Dzurendovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Úvod **13**

1 Přístupové systémy pro carsharingové aplikace **14**
1.1 Analýza přístupových systémů 14
1.2 Kryptografické protokoly v přístupových systémech 18
1.2.1 Advanced Encryption Standard 19
1.2.2 Secure Hash Algorithm 21

2 Vývoj Android aplikací **23**
2.1 Proces vývoje aplikací 25
2.1.1 Analýza a plánování 25
2.1.2 Návrh 26
2.1.3 Implementace 29
2.1.4 Testování 30
2.1.5 Nasazení 31
2.1.6 Údržba 31
2.2 Kotlin 32
2.3 Základní koncepty vývoje v Androidu 34

3 Technologie Bluetooth **35**
3.1 Rozdíl mezi Bluetooth a BLE 35
3.2 Přehled zásobníku BLE protokolu 36
3.3 Specifikace BLE 37
3.3.1 Profily 37
3.3.2 Služby 38
3.3.3 Generic Attribute Profile 38
3.3.4 Attribute Profile 39

4 Technologie NFC **41**
4.1 Technické specifikace NFC 41
4.2 Principy komunikace 42
4.3 Bezpečnost NFC 43

5 Návrh a implementace přístupového systému **45**
5.1 Identity Provider 46
5.2 Řídící jednotka OBU 49
5.2.1 BLE peripheral 49
5.2.2 NFC Server 51

5.2.3 TCP Server 53
5.3 Uživatelská mobilní aplikace 54
5.4 Demonstrace funkčnosti a konfigurace 62
5.5 Experimentální měření 65

Závěr **71**

Literatura **72**

Seznam symbolů a zkratek **77**

A Obsah elektronické přílohy **79**

Seznam obrázků

1.1 Princip fungování SC^2 Share systému 15
1.2 Schéma protokolu SePCAR 17
1.3 Schéma šifry AES 20
1.4 Princip fungování provozního režimu CBC-MAC 21
2.1 Diagram procesu vývoje aplikací 25
2.2 Ukázka diagramu pracovního postupu pro e-shop 27
3.1 Zásobník BLE protokolu 37
3.2 Možné využití modelu klient – server 38
3.3 Zapouzdření GATT 39
3.4 Rozdíl mezi indikací a notifikací 40
4.1 Struktura NDEF zprávy 42
4.2 Používané módy NFC v závislosti na typu zařízení 43
4.3 Relay útok na mobilní zařízení se zapnutým NFC 44
5.1 Abstraktní schéma přístupového systému 45
5.2 Návrh přístupového systému první verze protokolu 46
5.3 Návrh přístupového systému druhé verze protokolu 47
5.4 Autentizační token uživatele 48
5.5 Komunikace přes Java Socket Programming 48
5.6 Návrh OBU peripheral 50
5.7 Diagram tříd OBU 52
5.8 Vývojový diagram OBU 55
5.9 Vývojový diagram mobilní aplikace 56
5.10 Diagram tříd mobilní aplikace 57
5.11 Vývojový diagram NFC sekce mobilní aplikace 59
5.12 Struktura upravených APDU zpráv a odpovědí 61
5.13 Vývojový diagram NFC sekce mobilní aplikace 62
5.14 Vývojový diagram TCP sekce mobilní aplikace 63
5.15 Generování parametrů aplikací Identity Provider 63
5.16 Úspěšné spojení a předání parametrů aplikacím 64
5.17 Čtení a zápisy do BLE charakteristik 64
5.18 Výsledek autentizace uživatele 64
5.19 Zaregistrovaná NFC čtečka NFC serveru 65
5.20 Komunikace prostřednictvím NFC 65
5.21 Komunikace prostřednictvím TCP 65
5.22 Uživatelské rozhraní mobilní aplikace 66
5.23 Okno s úspěšnou autentizací 67
5.24 Průměrný čas podle protokolu 69

5.25 Průměrný čas podle délky klíče 70
5.26 Průměrný čas podle verze protokolu 70

Seznam tabulek

1.1 Přehled zmíněných přístupových systémů 19
2.1 Vybrané verze Androidu 24
3.1 Srovnání klasického Bluetooh a BLE 36
5.1 Měření BLE peripheral první verze protokolu 67
5.2 Parametry notebooku použitého při měření 67
5.3 Měření BLE peripheral druhé verze protokolu 68
5.4 Měření TCP serveru 68
5.5 Měření NFC serveru po výměně parametrů 68
5.6 Měření NFC serveru celé komunikace 68

Seznam výpisů

2.1 Skupina přepínačů v xml souboru 28
2.2 Ukázka kódu v Kotlinu 33
5.1 Zřízení serveru přes Java Socket 47
5.2 Posílání objektu klientovi 49
5.3 Rozebrání bajtového pole a proměnná counter 52
5.4 Ošetření výjimky při použití třídy CryptoCore 53
5.5 Shared preferences 56
5.6 Skenování BLE peripheral skrze UUID služby 57
5.7 Implementace zpětných volání 58
5.8 Deklarace nové aktivity v AndroidManifest.xml 59
5.9 Volání služby přes Intent 60

Úvod

Rozličné aplikace carsharingu pro sdílení jednoho automobilu více uživateli v průběhu let zaznamenaly značný technologický rozvoj i zájem uživatelů. Pro jejich přístupové systémy lze použít různé bezdrátové technologie, jako například Wi-Fi, NFC (Near Field Communication), Bluetooth nebo také BLE (Bluetooth Low Energy). Z hlediska praktičnosti se jeví jako nejlepší technologie Bluetooth Low Energy, a to jak z hlediska nízké spotřeby energie, tak i ostatních výhod, které má oproti ostatním bezdrátovým technologiím v případu užití této práce.

S ohledem na bezpečnost je v dnešní době nezbytné myslet na přístupové systémy. Cílem této práce je navrhnout a implementovat přístupový systém, který využívá technologii BLE a zabezpečenou, šifrovanou komunikaci. Technologie BLE je porovnána s technologií Bluetooth, a také je provedena její analýza. Dále je také provedena analýza současných systémů pro carsharing z hlediska bezpečnosti.

Při vývoji mobilních aplikací existují jisté principy a zásady, které usnadňují tento vývoj. Tyto zásady a principy jsou v práci názorně vysvětleny spolu s navrženými nástroji pro jednotlivé fáze vývoje mobilní aplikace. Nedílnou součástí vývoje mobilních aplikací pro platformu Android je programovací jazyk Kotlin, který má v práci vlastní podkapitolu. Kompletní obraz vývoje na platformě Android je pak doplněn o základní koncepty vývoje v prostředí Android.

Kromě BLE je v práci také použita technologie NFC. Tato technologie je také analyzována a jsou představeny její základní pojmy a koncepty. Dále jsou analyzovány útoky na tuhle technologii společně s potřebnými opatřeními, které zabraňují uvedeným útokům.

V úvodní kapitole 1 jsou analyzovány přístupové systémy pro carsharing. V kapitole 2 jsou popsány aspekty vývoje aplikací pro platformu Android a programovací jazyk Kotlin. Kapitola 3 se zabývá technologií BLE. V kapitole 4 jsou pak popsány technologie NFC a v kapitole 5 je popsána samotná implementace přístupového systému, jeho návrh a také experimentální měření implementovaných serverů.

1 Přístupové systémy pro carsharingové aplikace

Dle průzkumu [1] carsharingové služby zaznamenaly v poslední dekádě značný vzestup. V roce 2017, kdy byl průzkum prováděn, bylo dokonce zaznamenáno, že přes 90 % uživatelů používajících carsharing se k této službě přidalo v roce 2016 a později. Z toho jasně vyplývá, že poptávka po carsharingových službách má vzestupný trend a v budoucnu bude stále běžnější. Kromě praktických důvodů, proč carsharingové aplikace používat, jako např. odpadnutí povinnosti nosit s sebou klíče od auta, nebo také možnosti vyzkoušet si vícero aut, mohou být také důvody enviromentální nebo finanční. Dle studie [2] carsharingové služby přispívají ke snížení vlastnictví automobilů. Důsledkem je tedy potom snížení uhlíkové stopy. Z hlediska finančního se poté jeví využívání carsharingových služeb jako finančně výhodnější pro uživatele, kteří řídí pouze příležitostně, než klasické půjčování aut ve fyzických půjčovnách [3]. V dnešní době řeší mnoho měst problém s příliš velkým počtem automobilů. Carsharing může být opět jedním z možných řešení tohoto problému, jelikož průměrné časové využití automobilů ve městech je 5 %, což implikuje, že z časového hlediska je 95 % času auto nevyužito [4]. V neposlední řadě je zde pak další zásadní bod, a tím je důvěra uživatelů v technologie a bezpečnost [5]. Uživatelé musí vědět, že systémy pro carsharingové aplikace jsou bezpečné a spolehlivé, což jsou zásadní předpoklady pro správné fungování těchto služeb.

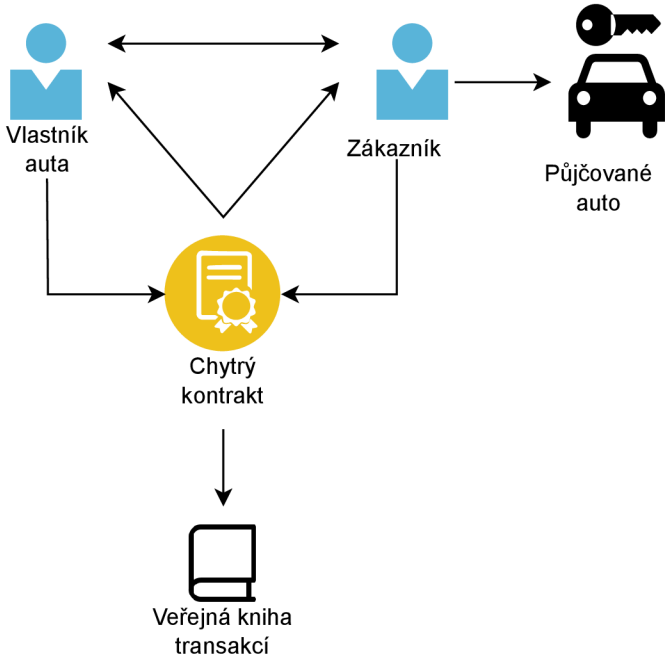
1.1 Analýza přístupových systémů

V současné době již existuje mnoho rozličných systémů nebo prací věnujících se carsharingu. Tato kapitola se věnuje konkrétním systémům a jejich koncepcemi v rámci bezpečnosti.

SC² *Share* [6] navrhuje systém rezervací a plateb, který využívá další, již existující protokoly jako např. SePCAR. Autoři *SC²* *Share* zmiňují jako výhody tohoto systému oproti ostatním:

- **Férovost**, kvůli neexistenci centralizované ceny.
- **Odolnost vůči únikům dat**, díky tomu, že v systému není jeden centrální bod náchylný na selhání. Tahle skutečnost je zapříčiněna návrhem systému, který je postaven na blockchainu a peer to peer principu.
- **Kompletní funkcionality**, kde autoři zmiňují, že kombinace jejich funkcionalit jako např. schopnosti řešit konflikty, jsou v carsharingových systémech jedinečné.
- **Nákladově efektivní** kvůli absenci provize.

Tento systém využívá tzv. chytrých kontraktů a technologie blockchain. Jelikož je tento systém zaměřen hlavně na rezervaci a platby, kryptografické algoritmy a podpisy jsou převzaty právě ze SePCAR. Fungování *SC*^{2} *Share* je znázorněno na obrázku 1.1.



Obr. 1.1: Princip fungování SC^2 Share systému

Systém *PRESTvO* [7] není přímo zaměřen na carsharingové aplikace, nicméně v souladu s tím, jak je navržen, je možné uvažovat nad jeho nasazením do implementace carsharingové služby i proto, že je v této službě implementována komunikace s palubní jednotkou automobilu OBU (On Board Unit) skrz chytrý telefon. Hlavním motivem autorů byla slabá bezpečnost radio-frekvenčních klíčů, které dle autorů postrádaly bezpečnostní nároky na dnešní dobu. Proto se *PRESTvO* zaměřuje na použití silných kryptografických protokolů a primitiv. V systému autoři implementují role jako vlastník automobilu, řidič, technik, dětský pasažér, obsluha a pasažér. Autoři si stanovili následující cíle jejich systému:

- **Bezpečné řízení přístupu**, což je hlavní cíl tohoto systému.
- **Flexibilní politika řízení přístupu**, určena kombinací dvěma různými přístupy k řízení přístupu.
- **Delegování práv**.
- **Soukromí uživatelů**, kdy identita uživatelů je anonymní.
- **Vyhledatelnost uživatelů** v případě sporů.
- **Flexibilní použitelnost bezdrátových technologií** jako např. WiFi, Bluetooth nebo NFC.

- **Komplexní výkonnostní testy**.

HERMES [8] je systém zaměřený přímo na sdílení vozidel a přístupu k nim. Tento systém je postaven na SePCAR s tím, že rozšiřuje efektivnost a škálovatelnost. Z kryptografického hlediska je zde obsaženo šifrování. Všechna data přenášená mezi zařízením uživatele a servery HERMES jsou šifrována pomocí standardních průmyslových protokolů, jako je SSL (Secure Sockets Layer) / TLS (Transport Layer Security). I servery samotné, kde jsou uloženy uživatelské informace a finanční transakce jsou chráněny pokročilými bezpečnostními opatřeními, jako jsou firewally a systémy detekce narušení. Z praktického hlediska je systém časově velmi efektivní, kdy operace přístupových tokenů na OBU činí pouhých 62 ms.

HERMES také implementuje vícefaktorovou autentizaci pro zajištění bezpečnosti platformy. Uživatelé musí poskytnout kombinaci hesla a jedinečného kódu vygenerovaného mobilní aplikací HERMES, což pomáhá zabránit neoprávněnému přístupu k platformě a zajistit, aby do sítě sdílení aut měli přístup pouze oprávnění uživatelé.

Pro zajištění bezpečného a transparentního záznamu všech transakcí používá *HERMES* technologii blockchain. Použití této technologie pomáhá zajistit, že transakce nemohou být pozměněny nebo zmanipulovány, a poskytuje uživatelům a správcům záznam o všech transakcích odolný proti neoprávněné manipulaci, což činí další bezpečnostní nástavbu platformy.

Autoři zmiňují, že hlavním přínosem *HERMES* a nástavbou oproti jeho základu SePCAR jsou následující faktory:

- **Propracovaný design systému pro lepší zabezpečení a soukromí**, který potlačuje bezpečnostní rizika spojená s neznámými poskytovateli služeb. Tuhle problematiku řeší pomocí vícestranného výpočtu MPC (Multiparty Computation).
- **Podpora efektivity a škálovatelnosti**, kdy se používají určitá kryptografická primitiva a MPC protokoly. Je zde např. použito AES-CBC-MAC (Advanced Encryption Standard Cipher Block Chaining Message Authentication Code) nebo HtMAC mód.
- **Vylepšená implementace a benchmarking včetně prototypu OBU.**

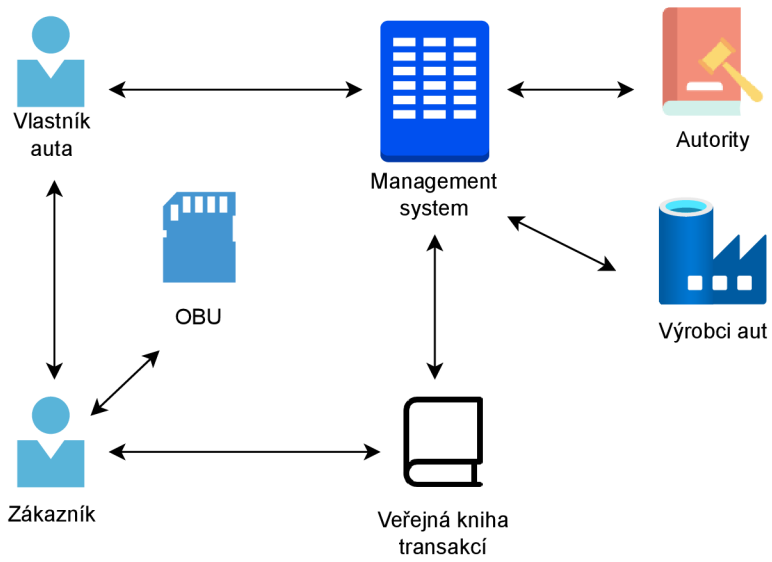
iShare [9] je koncept sdílení auta datovaný již v roce 2013 a implementovaný společností IDIADA Automotive technology [10]. Jelikož se jedná o projekt již starší, autoři projektu si kladli obecné cíle a výhody carsharingu jako např. uhlíkově neutrální města, kterým má pomoci právě carsharing. Kvůli uhlíkově neutrálním městům se v projektu *iShare* počítá s plně elektrickými auty, a to dokonce i s jasně danými specifikacemi auta.

Koncept počítá se stejnými nástroji, jako ostatní systémy čili chytré zařízení, OBU, server pro ukládání informací a samozřejmě automobil, v iShare ale konkrétní

automobil. V čem se ale v návrhu *iShare* odlišuje od ostatních je používání čárového kódu, a také čtečka čárového kódu jako základní komponent v architektuře. V rezervaci auta jsou potom zohledňovány uživatelské údaje, PIN a čárový kód. Koncept počítá i s tehdy budoucím využitím technologií jako NFC nebo Mirrorlink.

Zmínku si v této kapitole zaslouží určitě již zmíněný *SePCAR* [11]. *SePCAR* je plně decentralizovaný protkol pro přístup k automobilům, který umožňuje uživatelům sdílení automobilů bez obětování jejich soukromí a bezpečnosti. Pro přístup k automobilu trvá protokolu 1,55 sekund.

SePCAR nabízí propracovaný model, na kterém protokol staví viz obrázek 1.2. Hlavním bodem tohoto modelu je bezklíčový management systém, který komunikuje s výrobci automobilů, vlastníkem auta a má také přístup k veřejné knize transakcí a funguje jako MPC, kdy kompletně spravuje zacházení s klíči, a to jak u jejich generování, distribuci, aktualizací a nebo při jejich zneplatnění. Předpokládá se, že vlastník auta a zákazník budou mezi sebou komunikovat skrze chytré mobilní zařízení. OBU je vybaveno bezdrátovým rozhraním ke komunikaci jako např. NFC, Bluetooth nebo LTE (Long Term Evolution). Jednotlivé digitální klíče jsou očekávány od výrobce. Přístupové tokeny jsou ukládány ve veřejné knize transakcí.



Obr. 1.2: Schéma protokolu SePCAR

Z hlediska kryptografie se autoři rozhodli použít v protokolu *SePCAR* následující kryptografická primitiva:

- Pro šifrování a dešifrování veřejného klíče je použit algoritmus RSA (Rivest Shamir Adleman).
- Pro ověřování a podpis veřejného klíče je též použit algoritmus RSA.

- Pro šifrování a dešifrování symetrického klíče je použit algoritmus AES v módu CTR (Counter Mode).
- Pro MAC funkci symetrického klíče je použito CBC-MAC společně s AES.
- Jako hashovací funkce byly použity funkce SHA-2 (Secure Hash Algorithm) nebo SHA-3.

Celkový přehled zmíněných přístupových systémů lze vidět v tabulce 1.1. Zde jsou popsány případy užití přístupových systému, jestli tyhle systémy pracují v online nebo offline prostředí, a jaká kryptografická primitiva nebo jiné technologie zmíněné přístupové systémy využívají. Jako první je zmíněn systém SC^{2}Share. Tento systém přebírá kryptografii od SePCAR. Přínos z hlediska bezpečnosti tkví v chytrých kontraktech a technologii blockchain. Označení online režimu SC^{2}Share dostává díky chytrým kontraktům, které pracují právě v online režimu.

Systém PRESTvO používá pro symetrický klíč algoritmus AES a jako hashovací funkci SHA-2. Dále také jako primitiva pro veřejné klíče navrhuje použití RSA, ale zmiňuje jako výhodnější použít Diffie-Hellman na bázi eliptických křivek. Kromě toho také PRESTvO navrhuje pokročilejší techniky jako skupinové podpisy a podpisy založené na identitě. Jako použité komunikační technologie navrhuje NFC, Bluetooth nebo Wi-Fi. V závislosti na použité technologii je tedy tento systém v online či offline režimu.

Stejně jak SC^{2}Share, tak i systém HERMES je postaven na SePCAR. Další podobností je také použití technologie blockchain. Dále také používá pro přenos dat SSL/TLS. Z pokročilejších kryptografických primitiv je zde možno si všimnout vícestranných výpočtů. I HERMES používá AES, konkrétněji AES v módu CBC. Z hashovacích funkcí je zde vybrána funkce SHA-3.

Protokol SePCAR, na němž je založeno vícero přístupových systémů, nemá výraznou odchylku od ostatních systémů, co se používání primitiv týče. Používá šifru RSA, tak jak většina systémů, dále pak pro šifrování a dešifrování symetrického klíče algoritmus AES v čítacím módu. Z hašovacích funkcí pak využívá SHA-2 a SHA-3. Autentizace automobilu probíhá v offline režimu.

1.2 Kryptografické protokoly v přístupových systémech

Tahle kapitola se věnuje detailnějšímu popisu často vyskytujícím se kryptografickým protokolům v přístupových systémech. Mezi ně patří například AES-CBC-MAC, AES-GCM (Galois/Counter Mode) a SHA, které jsou použity i v návrhu a implementaci této bakalářské práce. Dále je také popsána šifra RSA, kterou používá většina přístupových systémů pro šifrování dešifrování veřejného klíče.

Tab. 1.1: Přehled zmíněných přístupových systémů

Přístupový systém	Případ využití	Offline/online	Kryptografická primitiva a jiné technologie
SC^2 Share	Rezervace a platby	Online	blockchain, MPC, jinak převzato ze SePCAR
PRESTvO	Koncepce přístupového systému, silná bezpečnost	Online i offline	SHA-2, AES, skupinové podpisy, podpisy založené na identitě, ECDH
HERMES	Sdílení vozidel a přístup k nim	Online i offline	SSL/TLS, blockchain, AES-CBC-MAC, jinak převzato ze SePCAR
iShare	Koncept pro sdílení vozidel		Čtečka čárového kódu
SePCAR	Protokol pro sdílení vozidel	Offline	RSA, AES-CTR, AES-CBC-MAC, SHA-2, SHA-3

1.2.1 Advanced Encryption Standard

AES je standardizovaná symetrická bloková šifra vybraná vládou Spojených států Amerických k zabezpečení informací [12]. AES byl původně znám pod názvem Rijndael, což je název vzniklý přesmyčkou dvou autorů této šifry – Joana Daemena a Vincenta Rijmena. Tahle šifra byla vybrána Americkým úřadem pro standardizaci NIST ve veřejné soutěži. AES je nástupcem šifry DES (Data Encryption Standard), která byla náchylná k útokům hrubou silou.

Klíčů u AES mohou být 128, 192 nebo 256 bitové délky, přičemž bloky jsou vždy 128 bitové, to znamená, že AES pracuje s bloky o 128 bitů na vstupu i výstupu. AES je založen na substitučně-permutační síti, což znamená, že provádí sekvenci propojených operací, které zahrnují nahrazení a promíchání vstupních dat.

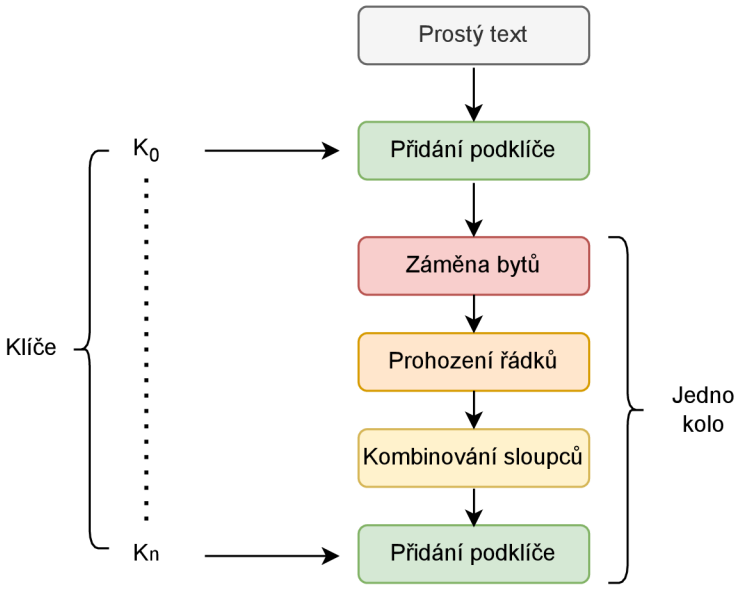
Jako první jsou u AES šifry vypočítány všechny podklíče, které jsou odvozeny z inicializačního klíče algoritmu. Počet podklíčů je závislý od počet kol, které se v AES provádí. Tento počet je určen podle délky klíče, a to:

- 10 kol pro 128 bitové klíče.
- 12 kol pro 192 bitové klíče.
- 14 kol pro 256 bitové klíče.

Po vytvoření podklíčů se opakují kola s následujícími operacemi:

- Přidání podklíče.
- Záměna bajtů.
- Prohození řádků.
- Kombinování sloupců.

Před prvním kolem se nejdříve provede operace přidání podklíče, kdy každý bajt stavu je zkombinován s inicializačním podklíčem za pomoci operace xor. Poslední kolo je bez operace kombinování sloupců a má tedy tři operace. Popsané vztahy jsou uvedené na obrázku 1.3.



Obr. 1.3: Schéma šifry AES

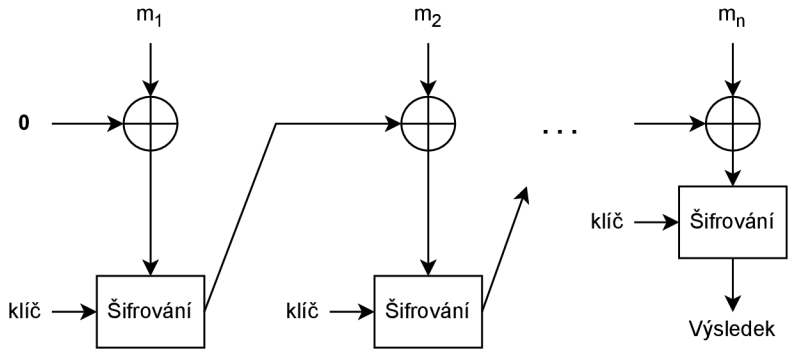
Záměna bajtů je substituční část substitučně-permutační sítě. V tomto kroku je každý bajt nahrazen jiným pomocí tabulky zvané *S-Box*. Bajty jsou zaměňovány takovým způsobem, že žádný bajt není zaměněn stejným bajtem, a také není nahrazen jiným bajtem, který je komplimentem aktuálního bajtu. Výsledkem této operace je stejně velká 16 bajtová matice.

Prohození řádků je jednoduchá operace, kdy v rámci řádku dojde k posunu prvků doleva. První řádek zůstává nezměněn, druhý řádek je jednou posunut doleva, třetí řádek dvakrát doleva a poslední řádek je posunut třikrát doleva.

Kombinování sloupců je v podstatě násobení matic. Každý sloupec je násoben určitou maticí, a jako výsledek je každý sloupec pozměněn.

AES se užívá v různých provozních režimech, což jsou způsoby, jakými jsou blokové šifry užívány v případě, že jsou zprávy delší než velikost bloku. Provozní režimy také umožňují znovupoužití stejného klíče. V přístupových systémech se často vyskytuje provozní režim CBC-MAC. Jak lze vidět na obrázku 1.4, jako **inicializační**

vektor se v tomto provozním režimu obvykle používá nula, což je zároveň hlavní slabina tohoto provozního režimu. Zpráva se rozdělí na bloky stejné délky. Výsledek šifrování klíče se zprávou je vstupem do operace xor, kdy druhým vstupem je následující blok zprávy. Výsledek této operace je vstupem šifrování. Po zpracování všech bloků zprávy se použije poslední blok šifrované zprávy jako MAC. Při ověřování zprávy pomocí CBC-MAC se opakuje stejný proces, ale výsledný MAC se porovná s očekávaným MAC. Pokud se shodují, zpráva je považována za autentickou.



Obr. 1.4: Princip fungování provozního režimu CBC-MAC

Dalším zmíněným provozním režimem je GCM. GCM kombinuje čítačový režim CTR a autentizace prostřednictvím konečného Galoisového tělesa, odkud pramení název tohoto provozního režimu. GCM funguje tak, že otevřený text je postupně podroben operaci xor společně se zašifrovanými hodnotami čítače. Autentizační data se vytváří pomocí násobení v konečném tělese, které jsou také podrobeny operaci xor s úvodní zašifrovanou hodnotou čítače. Pro zajištění bezpečnosti je důležité volit čítač vždy jinak, aby se v různých šifrových textech neodhalil překryv bloků zašifrovaných čítačů. GCM je stejně jako spoustu jiných provozních režimů navržen tak, aby fungoval s různými délkami vstupní zprávy. Obsahuje také **inicializační vektor**.

1.2.2 Secure Hash Algorithm

SHA je rodina hašovacích algoritmů [13]. Hašovací funkce mají tu vlastnost, že libovolně dlouhý vstup převedou do výstupu pevně dané délky. Jen nepatřičná změna na vstupu kompletně změní výstup. Inverzní funkce k hašu by neměla být proveditelná. Aktuální verze SHA jsou SHA-1, SHA-2, a také od roku 2012 SHA-3, která byla v soutěži odvozena od algoritmu *Keccak* [14].

SHA-1 je hašovací funkce, jenž byla považována za bezpečnou do roku 2005, kdy se ukázaly možnosti kolizí. Kolize v kontextu hašovacích funkcí znamenají, že algoritmus různým vstupům vygeneruje stejný výstup. Jelikož už se funkce nepovažuje

za bezpečnou, nedoporučuje se její nasazení do nových aplikací. Útok na tuhle funkci je však teoretický a jeho provedení ještě dosud nebylo realizováno [15]. Zpráva se rozdělí na bloky o délce 512 bitů. Každý blok se před svým zpracováním ještě rozdělí na 16 slov o délce 32 bitů. Celkem má funkce 80 rund. Funkce produkuje výstupy dlouhé 160 bitů (20 bajtů).

Kvůli výše zmíněným důvodům se dnes již preferuje použití funkce SHA-2, která má dvě verze známé pod jmény SHA-256 a SHA-512. Tyhle verze používají 32 a 64 bitová slova. Existují také zkrácené verze těchto hašovacích funkcí, známé jako SHA-224 a SHA-384, které lze použít pro kteroukoli část algoritmu.

Útoky hrubou silou nejsou na SHA-2 tak účinné jako na SHA-1. Provézt útok hrubou silou pro zprávu o délce L by totiž vyžadovalo 2^L operací, díky čemuž je SHA-2 mnohem bezpečnější proti těmto druhům útoků.

2 Vývoj Android aplikací

Android je otevřený operační systém postaven na Linuxu a určen primárně pro mobilní zařízení [16]. Android nabízí vývojářům sjednocený přístup k aplikacím využívající operační systém Android, což znamená, že vývojáři využívající nástroje k vývoji aplikací využívajících Android se nemusí starat o implementaci na konkrétní zařízení, jestliže bude využívat operační systém Android.

První beta verze Android SDK (Software Development Kitu) byla vydána v roce 2007. První komerční verze Android, která nesla název Android 1.0, byla vydána v září roku 2008. Zdrojový kód je dostupný jako otevřený a svobodný software. Google poskytuje zdrojový kód k operačnímu systému Android, nyní nazvaný jako AOSP (Android Open Source Project), pod licencí Apache License version 2.0 a zbytek včetně změn v jádru Linuxu pod licencí GNU General Public License version 2 [17].

Operační systém Android nabízí nepřeberné množství funkcí a vlastností. Důraz na rozličnost a kvalitu použitých technologií je veliký, jelikož Android je přímým konkurentem operačního systému iOS od Applu. Použité vlastnosti a funkce jsou následující:

- **Uživatelské rozhraní.** Základní rozhraní Android OS nabízí uživatelsky přívětivé rozhraní, což je výhoda i pro vývojáře, kdy už v základu mají k dispozici rozhraní s kvalitním designem.
- **Zprávy.** Pro zprávy se používají standardní SMS a MMS.
- **Webový prohlížeč.** Založen na otevřeném softwaru Webkit layout engine spolu s JavaScript engine od Google Chrome podporující HTML5 a CSS3.
- **Bezdrátové technologie.** Využívá se technologií Bluetooth, BLE, GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Wi-Fi, LTE, NFC a WiMAX.
- **Wi-Fi Direct.** Technologie, která umožňuje aplikacím přímé párování nad připojením s velkou šířkou pásma.
- Pro **úložiště** se využívá odlehčené relační databáze a SQLite.
- **Android Beam,** populární technologie založena na NFC, která umožňuje uživatelům okamžité sdílení pouhým dotknutím dvou telefonů zároveň, pokud mají zapnuté NFC.
- **Google Cloud Messaging**, služba, která umožňuje vývojářům posílat krátké zprávy uživatelům jejich mobilní aplikace.
- **Multi-touch.** Android má nativní podporu vícedotykového ovládání displeje.
- **Multi-tasking.** Uživatel může přepínat mezi různými aplikacemi, které běží zároveň.
- **Mnohojazyčnost.** Android podporuje jednosměrný a obousměrný text.
- **Podpora médií.** Android podporuje média H.263, H.264, MPEG-4 SP, AMR,

AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF a BMP.

- **Widgety.** Widgetům jde měnit velikost. Uživatel si může widget zobrazit větší s vícero detaily a také zmenšit, kdy widget obsahuje pouze jednoduchou informaci nebo úkon.

API (Application Programming Interface) level označuje číselné vyjádření verze Android API [18], která poskytuje soubor funkcí pro vývojáře, které jsou dostupné pro aplikace Android. API level také můžeme chápat jako sadu rozhraní pro programování aplikací, které vývojáři mohou použít ke komunikaci s operačním systémem Android a k přístupu k funkcím zařízení, jako jsou například senzory, kamera nebo internetové připojení. Každá verze Android má také svůj vlastní API level, který obsahuje oproti předchozí verzi nové funkce a vylepšení, které jsou přidané do operačního systému Android.

Jak již bylo zmíněno, API level se označuje číslicemi, ale má také zastoupení jmenné. Vývojáři aplikací mohou použít toto číslo k určení, zda jsou nové funkce a vylepšení dostupné pro určitou verzi Android a zda mohou používat nové funkce nebo musí použít alternativní řešení, aby se zajistilo, že aplikace bude fungovat na všech zařízeních. Vybrané verze operačního systému Android a jejich API level jsou uvedeny v tabulce 2.1.

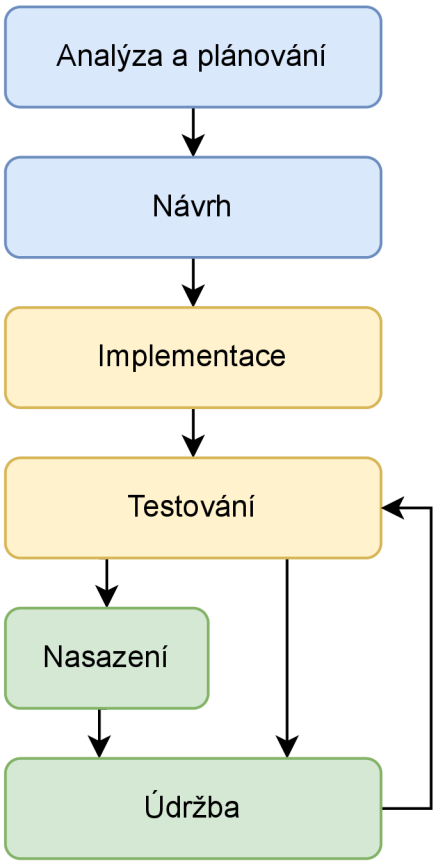
Pro vývojáře je důležité znát API level, aby aplikace fungovala správně na všech zařízení a aby mohla využívat všech funkcí. Mimo jiné je také vhodné znát API level kvůli zpětné kompatibilitě. Zpětnou kompatibilitu zajišťuje u Androidu ABI (Application Binnary Interface) [19]. ABI je soubor pravidel, která určují, jak se aplikace propojují s operačním systémem a s jinými aplikacemi. Tyto pravidla jsou zdokumentována a mění se jen zřídkakdy, aby se zajistila kompatibilita se staršími aplikacemi.

Tab. 2.1: Vybrané verze Androidu

Jmenné označení	**Verze**	**API level**	**Rok vydání**
Android13	13	API level 33	2022
Android12	12	API level 31	2021
Android11	11	API level 30	2020
Pie	9	API level 28	2018
Oreo	8.0.0	API level 26	2017
Lollipop	5.0	API level 21	2014
Honeycomb	3.0	API level 12	2011
Eclair	2.0	API level 5	2009
Base Android	1.0	API level 1	2008

2.1 Proces vývoje aplikací

Proces vývoje aplikací můžeme rozdělit do šesti fází [20], což je znázorněno na obrázku 2.1. Každá fáze má svá specifika, a také různé nástroje, které se mohou používat pro ulehčení procesu vývoje aplikace, přičemž každá fáze je v této kapitole blíž popsána společně s návrhem různých nástrojů pro ulehčení práce s konkrétní fází návrhu.



Obr. 2.1: Diagram procesu vývoje aplikací

2.1.1 Analýza a plánování

První fáze vývoje aplikace, a to ať už na mobilní zařízení, nebo jakékoliv jiné, je vytyčení základní strategie vývoje, naplánování si celého procesu vývoje a zodpovězení na klíčové otázky. Jestliže se aplikace vyvíjí pro zákazníka, tak potom shrnutí požadavků zákazníka a jejich následné zpracování. Základních otázek, které mohou zaznít při prvotním plánování je mnoho například:

- Jaký bude účel aplikace?
- Jaký problém se bude snažit řešit?

- Kdo jsou koncoví zákazníci aplikace?
- Na koho aplikace cílí?
- Jakých výsledků se bude aplikace snažit docílit?

Mezi další věci, které je dobré zanalyzovat, je konkurence. Je dobré zanalyzovat, zda není trh s aplikacemi již přesycen aplikacemi toho druhu, který je v plánu vyvíjet. Pokud je míra důvěry v projekt vysoká i přes velký počet konkurenčních aplikací na trhu, je potřeba si vytyčit, co bude přidaná hodnota aplikace nad ostatními, v čem bude aplikace vynikat.

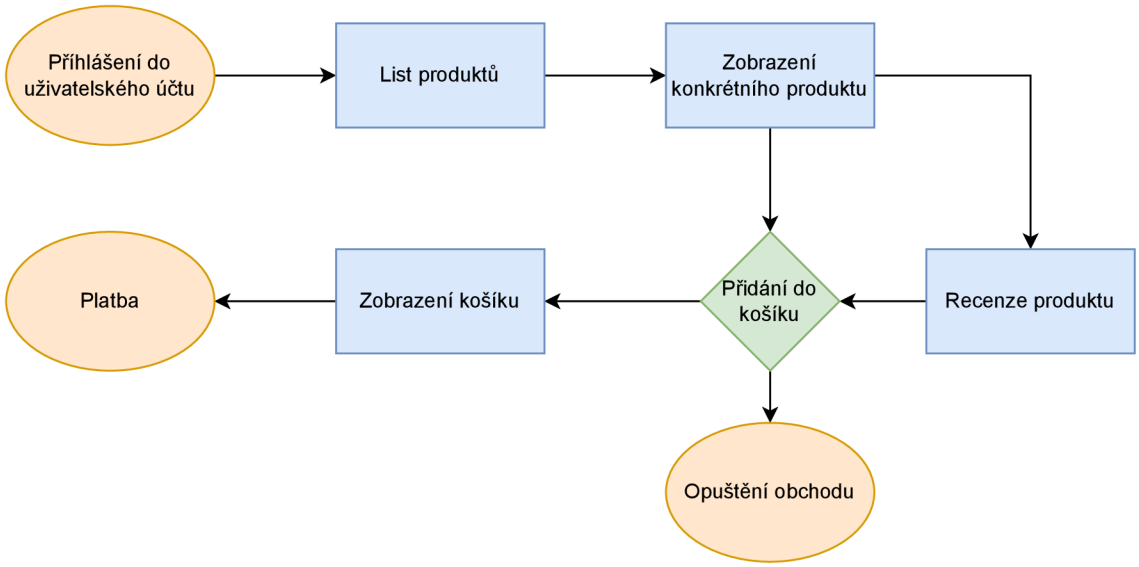
Při diskuzi o finančních záležitostech jsou dvě klíčová témata, a to monetizace aplikace a cena vývoje. Na vývoj aplikace se podle zdroje [22] vynaloží 150 000 $ až 200 000 $ a jeho délka se odhaduje zhruba na 4 až 6 měsíců podle konkrétních případů.

Mezi poslední záležitost k zvážení je výběr platformy a technologií k vývoji. V případě mobilních aplikací je tu zřejmý výběr mezi dvěma nejpopulárnějšími operačními systémy na mobilní telefony, a to Android a iOS, kdy Android podle [23] byl ve čtvrtém kvartálu roku 2022 na 71,8 % mobilních zařízení a iOS na 27,6 %. V případě volby operačního systému Android se jeví jako nejvýhodnější vyvíjet aplikaci v nástrojích přímo od tvůrců Androidu, a to ve vývojovém prostředí Android Studio. O programovacím jazyku Kotlim pojednává kapitola 2.2. V případě operačního systému iOS potom jeho autoři (Apple) doporučují používat SwiftUI a UIKit [24]. Jako programovací jazyk se zde nabízí Swift a nebo také C#. V případě, že se plánuje vyvíjet na Androidu a iOS zároveň, je možnost využít rozhraní Flutter.

2.1.2 Návrh

Návrhem se zde myslí celkový design aplikace, její návrh a hlavně **uživatelské rozhraní** UI (User Interface). Uživatelské rozhraní je klíčovým prvkem každé aplikace, protože ovlivňuje způsob, jakým uživatelé interagují s aplikací a jak ji vnímají. U uživatelského rozhraní můžeme vnímat technický přesah až do uměleckého spektra, neboť uživatelské rozhraní je kromě přívětivosti používání aplikace hlavně o celkovém vzhledu.

Jako první krok návrhu by měl být stanovení si základních pravidel informační architektury systému čili jaké data bude aplikace shromažďovat, jaké data uvidí uživatel a jak se budou uživateli zobrazovat. Dále jsou zde role při zobrazování dat, kdy se berou v potaz uvažované skupiny uživatelů, kdy každá skupina má různá práva pro zobrazování různých dat. Užitečným nástrojem pro **uživatelský zážitek** UX (User Experience) jsou diagramy pracovních postupů. Nástroj pro vytváření těchto diagramů je například *Miro*. Ukázka diagramu pracovního postupu je na obrázku 2.2 [25].



Obr. 2.2: Ukázka diagramu pracovního postupu pro e-shop

Na obrázku je znázorněn diagram pracovního postupu pro smyšlenou aplikaci e-shopu, kdy je zachycen proces nákupu produktu, který aplikace zákazníkovi nabízí. Obecná pravidla pro tvorbu diagramů pracovního postupu jsou používání geometrických tvarů při různých situací. Ty mohou být následující:

- **Ovál** symbolizuje začátek nebo konec programu.
- **Obdélník** symbolizuje proces nebo akci. Jedná se o nejčastěji používaný symbol v diagramu.
- **Paralelogram** symbolizuje vstup nebo výstup externích dat.
- **Kosočtverec** symbolizuje rozhodnutí, kdy se proces aplikace rozdělí na více cest.
- **Šipka** symbolizuje směr procesu aplikace. Společně s obdélníkem je to nejčastější symbol diagramů pracovního postupu.

Jakmile je hotový diagram pracovního postupu, dalším krokem jsou takzvané *wireframy*, což jsou zjednodušené náčrty vzhledu uživatelského rozhraní aplikace, které slouží k vizualizaci a testování návrhu aplikace před samotným vývojem. Wireframy poskytují základní strukturu a rozložení prvků na obrazovce při různých scénářích aplikace, ale obvykle neobsahují grafický design, barvy nebo detaily. Cílem wireframů je pomoci vytvořit logickou a uživatelsky přívětivou strukturu aplikace, která zohledňuje potřeby uživatelů a zjednodušuje navigaci a interakci s aplikací. Wireframy mohou být vytvořeny ručně na papíře nebo pomocí různých nástrojů určené právě pro Wireframy. Jeden z těchto nástrojů je například *Lucidchart*. Klade se zde také důraz hlavně na uživatelský zážitek.

Jedním z důležitých prvků návrhu jsou *stylové balíčky*. Stylové balíčky jsou uži-

tečné nejen pro firmy, které mají distinktivní barevné schéma, ale i pro každého vývojáře, kdy udržují vzhledovou konzistenci uživatelského rozhraní. Tahle součást návrhu většinou obnáší zvolení správné barevné kombinace tak, aby byla pro uživatele příjemná, a aby barvy společně ladily. Pro nezkušené vývojáře je možnost využít základní styly od Googlu a Applu pro Android respektive iOS aplikace. Celkově do stylového balíčků můžeme zahrnout:

- Barevné schéma.
- Písmo (rodinu písma).
- Vzhled tlačítek.
- Návrh widgetů.

Návaznost na wireframy mají *makety*. Makety jsou vykreslením vzhledu aplikace. V podstatě se jedná o výsledek spojení wireframů se stylovým balíčkem. Do maket by se měla také již promítnout práce na diagramech pracovního postupu. Pro makety existuje řada nástrojů, jimiž jsou například *Wix.com*, *UIzard*, *Bubble* nebo *Moqups*.

Pro interaktivní možnost otestovat uživatelské rozhraní i uživatelský zážitek uživatele jsou zde *prototypy*, které se od maket liší tedy právě interaktivností. Pro časovou náročnost jejich tvorby někteří vývojáři tento krok přeskakují, avšak vyplatí se vyhradit si rozpočet pro tento krok, a to ať už finanční, tak i časový. Prototypy umožňují vývojářům testovat různé koncepty a provedení aplikace a mohou být také použity k prezentaci a získání zpětné vazby od potenciálních investorů, zákazníků nebo uživatelů. Mezi užitečné nástroje pro tvorbu prototypů můžeme řadit nástroje *Figma*, *Proto.io* a nebo také nástroje zmíněné v maketách, a to například *UIzard*.

Uživatelské rozhraní umožňuje navrhnout a implementovat také Android Studio, nebo nástroj od Androidu, *Jetpack Compose*. V případě Android Studia se editují xml soubory, které určují vzhled aplikace a uživatelského rozhraní. Tyhle xml soubory lze upravovat buď klasickým způsobem, kdy se píšou značky komponentů jako například na výpisu 2.1, kde je ukázka přepínačů (anglicky Radio Button), které jsou umístěné ve skupině přepínačů, nebo také interaktivním a uživatelským přívětivým způsobem, který Android Studio umožňuje, a to přetahováním požadovaných komponentů na obrazovku, kterou uvidí uživatel. Další možností je mít obrazovku rozdělenou na xml kód a interaktivní rozhraní. V případě interaktivního návrhu ale nestačí komponenty umístit na obrazovku, musí se jím nastavit patřičné parametry jako názvy komponent nebo jejich umístění a omezení v návrhu, aby mohl být návrh převeden na xml soubor. V případě Jetpack Compose se uživatelské rozhraní píše kódem a rozdílnou, leč podobnou syntaxí Kotlinu.

Výpis 2.1: Skupina přepínačů v xml souboru

```
1 <RadioGroup
2         android:id="@+id/radioGroup"
```

```
3           ...
4           app:layout_constraintTop_toBottomOf=
               "@+id/authenticatedLabel">
5
6           <RadioButton
7               android:id="@+id/radioBLE"
8               android:layout_width="0dp"
9               android:layout_height="match_parent"
10              android:layout_weight="1"
11              android:text="BLE" />
12
13          <RadioButton
14              ... />
15
16          <RadioButton
17              ... />
18
19      </RadioGroup>
```

2.1.3 Implementace

Klíčovými aspekty implementace je vývoj backendu a frontendu. Není to však jen naprogramování kódu, ale měla by se brát v potaz celá technologická struktura. Co se backendu týče, ten je zásadní pro škálovatelnost a udržitelnost celé mobilní aplikace. Hraje zde roli výběr programovacího jazyka (viz kapitola 2.2), ale také je zde otázka serverů a databází. Je potřeba si naplánovat, jestli jsou pro aplikace potřebné servery, s jakou kapacitou, nebo jestli se nevyplatí mít serverovou otázku vyřešenou kompletně nebo částečně skrz cloudy. V případě robustnější aplikace ,a právě komunikace aplikace se servery a databázi je vhodné myslet na aplikační rozhraní. Jako nástroj pro implementaci backendu aplikace se v případě vývoje na Android nejvíce nabízí Android Studio, ve kterém byla implementována i aplikace této bakalářské práce.

Frontend především implementuje vzhled návrhu zmíněného v předešlé kapitole, kdy bere v potaz především zmíněné makety a prototypy. Implementace frontendu je závislá od toho, na jaké platformě aplikace bude. Od toho také existují přístupy k vývoji vzhledem k platformám.

Aplikace může být vyvíjena pouze na jednu platformu. V případě, že je vyvíjena pouze na Android a časem by například investoři nebo management chtěl aplikaci

spouštět i na iOS, je potřeba aplikaci implementovat celou i na iOS. Tento přístup může být nákladný, ale oproti tomu rychlý.

Aplikace také může být vyvíjena napříč platformami. Zde se již od začátku myslí na vývoj v nástrojích, který umožňuje vývoj na vícero platformách. Zde je vývoj pomalejší, ale za to má větší cílovou skupinu zákazníků.

U hybridního přístupu k platformám se vyvíjí aplikace primárně jako webová aplikace. V pozdější fázi je pak aplikace zpřístupněna i jako mobilní verze. Tohle řešení je vhodné v případech, kdy není tolik času vyvíjet aplikaci přístupem vývoje napříč platformami, ale přesto záměr mít aplikaci na vícero platformách existuje.

2.1.4 Testování

Po dokončení implementace aplikace následuje v procesu vývoje neméně důležitá část před jejím nasazením, a tou je testování. Při robustnějších projektech se pro účely testování zřizují oddělení pracovní pozice zaměřené pouze na testování. Testování lze také rozdělit do více podsekcí:

- Funkčnost.
- Výkon.
- Bezpečnost.
- Testování uživatelského rozhraní.
- Testování platformy.
- Testování zařízení.

V případě **funkčnosti** se testuje celková funkčnost programu, tedy všechny případy konfigurace jednotlivých vstupů od uživatele a zda obsahuje program nějaké chyby. Přihlédnutí na varování od vývojového prostřední nebo dokonce chyby. Dále se testuje, zda při stejném vstupu od uživatelů vznikne vždy stejný výstup, pokud samozřejmě není záměr aplikace jiný. Užitečnou pomocí při testování funkčnosti je zpřístupnit testování aplikace uživatelům pod takzvaným dřívějším přístupem.

Výkon aplikace můžeme měřit pomocí rozličných kvantitativních parametrů, zejména pak rychlost aplikace při různých událostech. Dalšími kritérii jsou pak jak aplikace zvládá vzrůstající počet uživatelů, jak dlouho se aplikace načítá, jakou má aplikace velikost, nebo jak spotřebovává procesor a operační paměť.

Nezanedbatelným aspektem a zvlášť v dnešní době je **bezpečnost** aplikace. Obzvlášť pokud je aplikace zaměřena na bankovnictví nebo zdravotnictví je bezpečnost aplikace na prvním místě. I u dalších druhů aplikace je ale na místě se bezpečností zaobírat. Pokud se pracuje s některými uživatelskými daty, je nežádoucí, aby si je mohl přečíst kdokoliv jiný. Navíc, pokud se shromažďují osobní údaje, tak při jejich případném úniku mají organizace povinnost oznámit tento únik příslušným úřadům [27].

Navržené a implementované **uživatelské rozhraní** by se také mělo otestovat. Zde se testuje, jestli je UI uživatelsky přívětivé a intuitivní, a taky zda implementace navazuje na různé aktivity tak, jak to bylo zamýšlené v návrhu.

Testování platformy a zařízení se provádí v případech, kdy je známa problematická platforma nebo zařízení, na kterém bude aplikace běžet. U testování zařízení je také možné porovnávat výsledky výkonových testů různých zařízení.

Testování je možné dělat čistě manuálně, nebo také automaticky či poloautomaticky pomocí různých nástrojů jako je například *Browser Stack* [28].

2.1.5 Nasazení

Jakmile je aplikace naprogramovaná, implementovaná a otestovaná, tak už nic nebrání samotnému nasazení aplikace mezi uživatele. Nejvíce pravděpodobný a zároveň nejčastější způsob nasazení aplikace je její zveřejnění na App Store nebo Google Play, kde je nejdříve potřeba vytvořit si zde vývojářský účet. Nasazení aplikace na App Store může být komplikovanější, protože App Store více hlídá kvalitu aplikací a obecně je více striktnější vzhledem ke kritériím co musí aplikace splňovat [29], než je tomu u Google Play, i proto je menší počet aplikací na App Store, než na Google Play. V procesu zveřejňování aplikace je potřeba vyplnit několik formulářů, a taky vyplnit metadata aplikace. Mezi údaji, které je potřeba vyplnit, mohou být následující údaje:

- Název aplikace.
- Kategorie.
- Popis.
- Ikona aplikace.
- Klíčová slova.
- Abstrakt.
- Screenshoty aplikace.

Před nasazením musí být aplikace ověřena a certifikována, aby se zajistilo, že je bezpečná a spolehlivá. Dalším krokem je tedy podat žádost o certifikaci, která může trvat několik dní až týdnů [30]. Jestliže aplikace vyžaduje přihlášení uživatelů, je potřeba pro účely certifikace přiložit testovací uživatelský účet. Kromě dvou výše zmíněných obchodů se lze ale ještě vydat alternativní cestou nasazení aplikace. Existuje také možnost soukromého nasazení aplikace, které umožňují obě platformy [31, 32].

2.1.6 Údržba

Vydáním a nasazením aplikace mezi veřejnost proces vývoje ještě nekončí. Aby byla aplikace úspěšná na trhu, mělo by se pracovat se zpětnou vazbou uživatelů. Nejspíš

žádná aplikace se neobejde bez menších či větších nepředvídatelných chyb či událostí. Proto by se mělo po nasazení dbát na zpětnou vazbu uživatelů, a také analyzovat výstupy aplikace a různé metriky pro neustálé zlepšování aplikace. Mezi měřitelné metriky a parametry, které jsou vhodné analyzovat, patří:

- Počet stáhnutí aplikace.
- Počet aktivních uživatelů.
- Průměrná doba používaní aplikace.
- Statistiky konkurenčních aplikací.
- Hodnocení a recenze aplikace.

Po zpracování zpětné vazby a měření výše zmíněných metrik je možné pracovat na nových verzích aplikace, které potom znovu absolvují fázi testování.

Závěrem lze konstatovat, že byly popsány všechny základní fáze vývoje mobilní aplikace. Samozřejmě, že k popsanému obecnému konceptu lze přidat další fáze podle potřeb vývojářů nebo investorů, každopádně při dodržení uvedených fází by měl být výsledkem vývoje aplikace se slibným potenciálem.

2.2 Kotlin

Kotlin je moderní staticky typovaný programovací jazyk, který používá více než 60 % profesionálních vývojářů Android [33] a pomáhá zvyšovat produktivitu, spokojenost vývojářů a bezpečnost kódu. V Kotlinu je též napsána mobilní aplikace této bakalářské práce. Oproti Javě nabízí Kotlin jednodušší syntaxi pro začátečníky, není zde nutnost ukončovat každý řádek středníkem, je plně kompatibilní s Javou a má prvky, které pomůžou programátorům vyhnout se častým chybám. Kotlin může být také popsán jako odlehčená a zmodernizovaná verze Javy.

Jako hlavní výhody Kotlinu jeho autoři zmiňují:

- **Expresivní a stručný.** Možnost sdělovat své myšlenky výstižně a jednoduše při omezení použití zbytečného kódu. Podle průzkumu 67 % profesionálních vývojářů, kteří používají Kotlin, zaznamenali zvýšení své produktivity.
- **Bezpečnost kódu.** Kotlin nabízí rozmanité jazykové funkce, které umožňují vyhnout se častým programátorským chybám, jako jsou například výjimky nulového ukazatele. Díky tomu jsou aplikace pro Android, které používají Kotlin, o 20 % méně náchylné k selhání.
- **Interoperabilita.** Je možné volat Java kód z Kotlinu nebo Kotlin kód z Javy. Kotlin je plně interoperabilní s programovacím jazykem Java, což znamená, že v rámci svého projektu lze používat jak Kotlin, tak Javu podle preferencí a potřeb programátora.
- **Strukturovaná souběžnost.** S pomocí korutin v Kotlinu lze pracovat s asynchronním kódem stejně snadno jako s kódem blokovým. Korutiny výrazně

usnadňují správu úloh na pozadí, a to jak u síťových volání, tak při přístupu k místním datům.

Na výpisu 2.2 jsou ukázány hlavní vlastnosti kódu psané v Kotlinu.

Výpis 2.2: Ukázka kódu v Kotlinu

```
1 fun main() {
2     val numbers = listOf(1, 2, 3, 4, 5)
3     val evenNumbers = numbers.filter { it % 2 == 0 }
4     println("Even numbers: $evenNumbers")
5
6     val name: String? = null
7     val length: Int = name?.length ?: -1
8     println("Length of name: $length")
9 }
```

První část kódu vytváří seznam čísel od 1 od 5, filtruje pouze sudá čísla, a poté vypisuje výsledek na konzoli. Druhá část kódu definuje proměnnou „name“ jako nullable typ „String?“, který může buď obsahovat řetězec nebo null. Následně se délka řetězce uloží do proměnné „length“. Pokud je „name“ null, proměnná „length“ bude mít hodnotu -1. Vlastnosti Kotlinu použité na výpisu 2.2 jsou:

- **Bezpečné nullable typy**. Nullable typ „String?“ na řádku 6 znamená, že hodnota může být buď řetězec nebo null. Kód zde používá bezpečnou operaci „?.“, aby se zabránilo chybám nulového ukazatele.
- **Operátor „?:“**, nebo také nazývaný operátor Elvis se používá k tomu, aby se vrátila hodnota, pokud není null, nebo výchozí hodnota, pokud je hodnota null. V této ukázce je použit na řádku 7 a používá se k tomu, aby se vrátila hodnota -1, pokud je proměnná „name“ null.
- **Interpolace textového řetězce**. Interpolace textového řetězce je funkce, která umožňuje vkládat proměnné do řetězců pomocí značky „$“. Ve výpisu kódu na řádku 8 se používá k tomu, aby se vypsala délka řetězce v konzoli a na řádku 4 k tomu, aby se vypasala sudá čísla ze seznamu.
- **Lambda výrazy**. Kotlin podporuje lambda výrazy, což jsou malé kódové bloky, které mohou být předány do jiných funkcí jako argumenty. Tyto výrazy umožňují psát kód, který je snadno čitelný a psaný s použitím méně řádků. Lambda výraz je použit na řádku 3, a to konkrétně výraz `{ it % 2 == 0}`, který filtruje sudá čísla.
- **Funkcionální programování**. Kotlin je silně funkcionální programovací jazyk, což znamená, že podporuje řadu vlastností, jako jsou například lambdy, funkcionální rozhraní a podobně. Tyto vlastnosti usnadňují psaní kódu, který je krátký, čitelný a modulární.

- **Rozšíření funkcí**. Kotlin umožňuje rozšířit existující třídy o nové funkce pomocí tzv. rozšíření funkcí. Tyto funkce mohou být volány jako běžné metody třídy, což umožňuje psát kód, který je snadno čitelný a psaný s použitím méně řádků.

2.3 Základní koncepty vývoje v Androidu

Vývoj aplikací pro operační systém Android má své unikátní koncepty, které jsou popsány v této kapitole. Tahle kapitola se věnuje zejména populárním konceptům, jako `Context`, `Activity` nebo `Intent` [34].

`Intent` představuje objekt zprávy, který je posílán z jedné aktivity (komponenty uživatelského rozhraní) do druhé aktivity, aby se spustila nebo předala data. `Intent` může být použit k spuštění jiné aktivity, jako například zobrazení nové obrazovky s jiným obsahem nebo k předání dat mezi aktivitami. Pokud je `Intent` použit k přenosu dat, může být nastaven tak, aby předával textová data, obrázky, zvuky nebo jiné soubory. `Intent` také umožňuje spouštět aktivity v jiných aplikacích, což umožňuje integraci různých funkcí aplikací do jednoho uživatelského rozhraní nebo UX.

`Intent` je definován pomocí názvu akce (action), typu dat (data type) a dalších informací, jako jsou kategorie (category), komponenta a extra parametry. Tyto informace jsou použity pro identifikaci a spuštění požadované aktivity. Použití objektu `Intent` vývojářům pomáhá k vytváření interaktivních a dynamických aplikací na platformě Android.

`Activity` jsou třídy spojené s uživatelským rozhraním aplikace. Tyhle třídy jsou zpravidla svázány s xml soboury, které definují vzhled uživatelského rozhraní. Z hlediska interaktivity se v těchto třídách definuje chování uživatelského rozhraní, například jaká akce se vykoná po stisknutí tlačítko a podobně. V aplikace je vždy minimálně jedna aktivita, která je hlavní (`MainActivity`). Ta se zpravidla zobrazuje jako první po spuštění aplikace. Aktivita také může spouštět další aktivitu.

`Context` je objekt, který poskytuje přístup k jednotlivým komponentům aplikace a informacím s nimi spojenými. Skrz `Context` je například možný přístup k systémovým službám službám nebo k databázi. Dále se dá `Context` získat z komponent jako `Activity`, `Service` nebo `Application`.

3 Technologie Bluetooth

BLE je bezdrátová technologie pro komunikaci na krátkou vzdálenost vyvinuta týmem Bluetooth SIG (Special Interest Group) a je hlavní funkcí Bluetooth 4.0 specifikace [35]. Hlavní výhodou BLE je jeho nízká spotřeba baterie, která má dle výrobce výdrž mezi 2 dny a 14 lety, a proto je hojně využíván v řídících a monitorovacích aplikacích [35].

Co se týče komunikačních topologií, BLE podporuje klasickou point-to-point topologii, tak jako většina komunikačních technologií, ale také využívá všesměrový režim [36], který umožňuje přenášet data do nepřeberného množství koncových zařízení zároveň. Poslední topologií je tzv. *Mesh*, který umožňuje pomocí Bluetooth vytvářet rozsáhlé, spolehlivé sítě [36].

V rámci BLE existuje spousta klíčových pojmů, které je potřeba znát, aby se člověk orientoval v problematice této technologie. Těmto pojmům je věnováno několik dalších kapitol práce.

3.1 Rozdíl mezi Bluetooth a BLE

Z abstraktního hlediska je BLE technologií Bluetooth, tedy můžeme Bluetooth chápat jako obecnější pojem. I když je ale BLE podmnožinou Bluetooth, v implementaci jsou rozdíly mezi těmito pojmy zásadní, proto je potřeba si hlavní rozdíly uvézt.

Asi největším rozdílem je spotřeba baterie. Jak můžeme vidět v tabulce 3.1, zatímco spotřeba baterie klasického Bluetooth se dá označit jako vysoká (cca 1 W), spotřeba u BLE je, jak už název napovídá, nízká a to cca 0,01 W – 0,05 W, zatímco u klasického Bluetooth je spotřeba, jak už bylo zmíněno, výrazně vyšší. U rychlosti přenosu dat záleží na konkrétní specifikaci a modulaci, klasické Bluetooth tu má ale většinou převahu. BLE ovšem v posledních letech dokáže této rychlosti konkurovat, a to třeba s konfigurací *LE 2M PHY (Low Energy 2 Mb/s Physical)*, která má rychlost až 2 Mb/s, kdy u klasického Bluetooth můžeme dosahovat s konfigurací *EDR (Enhanced Data Rate) PHY* při modulaci 8DPSK (Differential Phase-Shift Keying) rychlostí až 3 Mb/s.

Atributy, ve kterých se klasické Bluetooth a BLE naopak shodují, jsou např. šířka pásma, kdy obě technologie používají šířku pásma 2,4 GHz. Dále také komunikační rozsah, který činí 10–30 m. GFSK (Gaussian Frequency-Shift Keying) modulaci využívají obě technologie, klasické Bluetooth pak také $\pi/4$ DQPSK (Differential Quadrature Phase-Shift Keying) a 8DPSK. Tohle porovnání je též shrnuto v tabulce 3.1.

Dalším důležitým aspektem je proces párování. Zatímco u klasického Bluetooth je párování nutný proces [37], BLE párování nevyžaduje respektive není povinné.

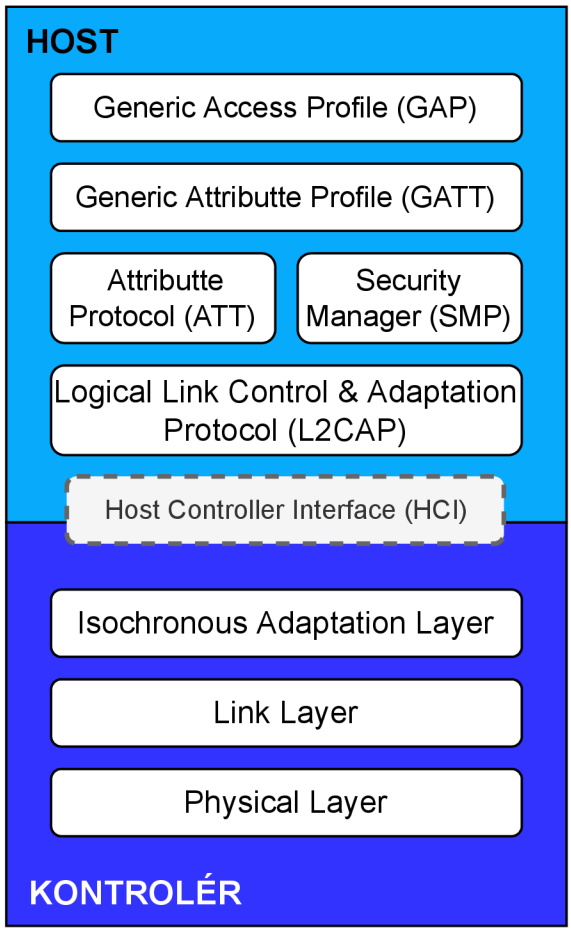
Tab. 3.1: Srovnání klasického Bluetooh a BLE

	Klasické Bluetooth	Bluetooth LE
Spotřeba baterie Šířka pásma Počet kanálů	cca 1 W 2,4 GHz ISM Band 40 s rozestupem 2 MHz	cca 0,01 W – 0,05 W 2,4 GHz ISM Band 79 s rozestupem 1 MHz
Modulace	GFSK	$\pi/4$ DQPSK, 8DPSK
Rychlost přenosu dat	LE 2M PHY: 2 Mb/s LE 1M PHY: 1 Mb/s LE Coded PHY (S=2): 500 Kb/s LE Coded PHY (S=8): 125 Kb/s	EDR PHY (8DPSK): 3 Mb/s EDR PHY ($\pi/4$ DPQSK): 2 Mb/s BR PHY (GFSK): 1 Mb/s
Komunikační topologie	Point-to-Point Broadcast Mesh	Point-to-Point

Situace je ale trochu komplexnější, než u klasického Bluetooth. BLE má kromě párování i proces spojování (anglicky *bonding*) [38]. U BLE párováním rozumíme proces výměny informací nutných ke vzájemné komunikaci dvou zařízení a poskytnutí zabezpečené komunikace. Tento proces zahrnuje také vzájemnou výměnu klíčů. Spojování (bonding) je proces, kdy informace získané z procesu párování, jsou uchovány na fyzickém zařízení, aby se při každém vzájemném spojení dvou zařízení nemusel tento proces činit znovu [38]. V případě carsharingových aplikací se může jevit proces spojování u BLE jako výhodný, neboť pokud si uživatel bude chtít např. propůjčit auto, které už si dříve skrz aplikaci půjčoval, zařízení už nemusí absolvovat proces párování a aplikace bude rychlejší.

3.2 Přehled zásobníku BLE protokolu

Zásobník BLE protokolu se sestává z několika vrstev, z nichž některé jsou povinné a některé jsou volitelné. Tyhle vrstvy se pak řadí do dvou hlavních logických bloků, a to *kontroléru* a *hosta* [40]. *Kontrolér* obsahuje fyzickou a linkovou vrstvu. *Host* potom zastupuje funkcionality vyšších vrstev, jmenovitě L2CAP (Logical Link Control and Adaptation Protocol), ATT (Attribute Protocol), GATT (Generic Attribute Profile), SMP (Security Manager Protocol) a GAP (Generic Access Profile). Komunikaci mezi *hostem* a *kontrolérem* zařizuje HCI (Host Controller Interface). Různé implementace v rámci aplikační vrstvy mohou být zahrnuty v zásobníku nad *hostem*. Uvedené vztahy jsou znázorněny na obrázku 3.1.



Obr. 3.1: Zásobník BLE protokolu

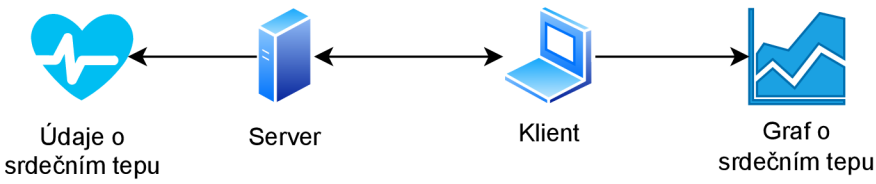
3.3 Specifikace BLE

Specifikace jsou důležité k vzájemné kompatibilitě komunikujících stran. K lepšímu užívání Bluetooth jsou již tyto specifikace zahrnuty ve většině hardwarových a softwarových zařízení [39]. Nejobecnější a zároveň základní specifikací je *Bluetooth Core Specification* [39]. Zajišťuje všechny procesy a zároveň definuje, jak Bluetooth technologie bude fungovat, a jaké části jsou uvolněny k implementaci. Verze *Bluetooth Core Specification* jsou zpětně kompatibilní čili novější verze jsou schopné komunikovat s verzemi staršími. Dalšími důležitými specifikacemi jsou *profily* a *služby*.

3.3.1 Profily

Komunikace dvou BLE zařízení ve většině případů funguje přes model klient – server [40]. Server obsahuje nějaká data a klient je určitým způsobem využívá a nakládá s nimi. Můžeme si vzít modelovou situaci, kdy monitor srdečních funkcí funguje jako server, který má v sobě údaje o srdečním tepu a klient je program, který uchovává

v sobě informace o tepu a dělá z nich např. graf. Popsaná situace je naznačena na obrázku 3.2.



Obr. 3.2: Možné využití modelu klient – server

Profily tedy určují:

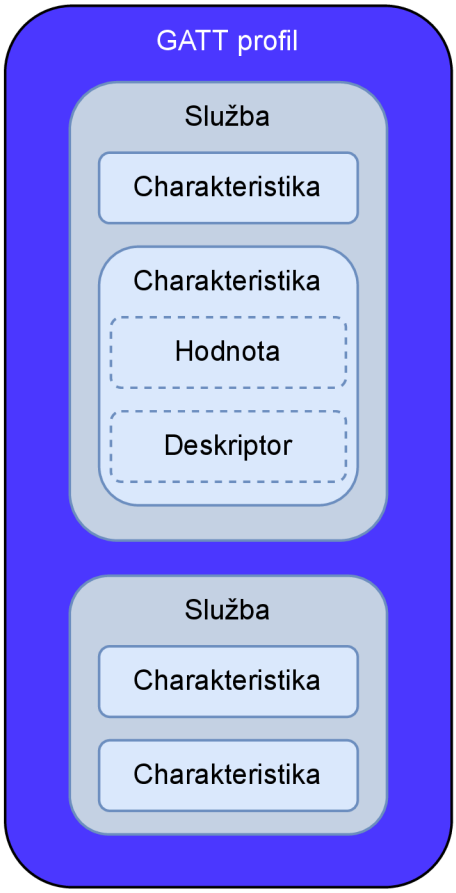
- Jak se bude k datům přistupovat.
- Nakládání dat ze strany klienta s ohledem na server, ke kterému klient přistupuje.
- Definují role v komunikaci (klient – server).

3.3.2 Služby

Stavová data na serverech se nacházejí ve formálně definovaných datových položkách známých jako charakteristiky a deskriptory. Charakteristiky a deskriptory jsou seskupeny uvnitř konstrukcí známých jako služby [40]. Obecně služby zastřešují data, které jsou si tématicky shodná nebo aspoň podobná. Službám lze také určit jisté chování společně s jejich obsahem. V praxi to může znamenat např. službu, která obsahuje varovná čísla pro srdeční tep. Této službě by se mohlo nastavit chování, kdy server by upozornil klienta v momentě, kdyby se srdeční tep přiblížil hodnotám nastavených v jmenované službě.

3.3.3 Generic Attribute Profile

GATT definuje rámec, který používá ATT pro objevování služeb a výměnu charakteristik z jednoho zařízení do druhého [41]. O charakteristikách bylo pojednáno v podkapitole 3.3.2. Data související se službami a charakteristikami jsou uložena v atributech. Obsah GATT a jeho zapouzdření je znázorněn na obrázku 3.3. Zde můžeme vidět, že GATT se sestává z jedné či více služeb, které v sobě obsahují opět jednu až více charakteristik. Ty potom v sobě uchovávají hodnotu a maximálně jeden deskriptor.



Obr. 3.3: Zapouzdření GATT

3.3.4 Attribute Profile

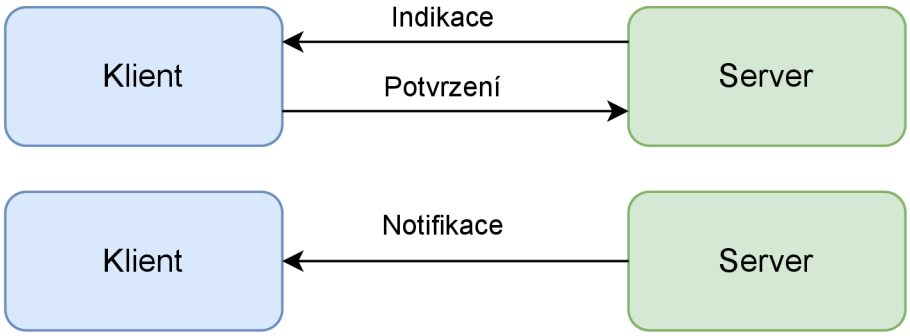
ATT definuje komunikaci mezi dvěma zařízeními, a to ve vztahu klient – server, nad vyhrazeným kanálem L2CAP [41], přičemž server definuje strukturu a organizaci dat pomocí atributů a služeb. Každé BLE zařízení může poskytovat sadu služeb, které obsahují různé atributy. Atribut je datová struktura, která uchovává informace spravované GATT protokolem, který jak můžeme vidět na obrázku 3.1 funguje nad ATT. Role klienta nebo serveru je určena GATT a je nezávislá na roli slave nebo master.

Klient může k těmto atributům přistupovat na základě požadavků serveru, kdy mu oznamuje, že chce k těmto datům přistupovat. Server může klientovi posílat zprávy, které se dělí na:

- **Notifikace**, kdy klient neposílá serveru potvrzení.
- **Indikace**, kdy klient pošle serveru také zprávu, zda bylo vyžádání atributu úspěšné.

Rozdíl mezi indikací a notifikací je znázorněn na obrázku 3.4. Zde lze vidět, že v případě notifikace je komunikace jednosměrná (na obrázku případ ze serveru ke

klientovi), zatímco u indikace je komunikace obousměrná (na obrázku případ, kdy klient posíla serveru potvrzení o přijetí indikace).



Obr. 3.4: Rozdíl mezi indikací a notifikací

4 Technologie NFC

NFC [42] je bezdrátová technologie používaná ke komunikaci mezi zařízeními jako jsou chytrý telefon, NFC čtečka nebo NFC značka, a která umožňuje rychlý přenos dat na krátkou vzdálenost. Využití této technologie tkví například v bezkontaktních platebních transakcí nebo v bezdotykovém přenosu mezi dvěma zařízeními.

Technologie NFC je založena na RFID (Radio-Frequency Identification), ale oproti RFID má kratší dosah, dále NFC umožňuje čtečce, aktivnímu zařízení, nebo v síťovém pojetí serveru, vytvářet radio-frekvenční proud, který komunikuje s jiným zařízením s NFC. Pasivní zařízení, jako je NFC značka, komunikují s aktivními zařízeními (např. s čtečkou) a uchovávají informace, aktivně ale nečtou další zařízení. Mobilní telefon může vystupovat jako aktivní i pasivní zařízení. V případě komunikace dvou aktivních zařízení mezi sebou jsou pak obě entity schopné odesílat a přijímat informace.

4.1 Technické specifikace NFC

NFC LLCP (Logical Link Control Protocol) specifikace stanovuje peer-to-peer oboustrannou komunikaci mezi zařízeními s podporou NFC na druhé vrstvě OSI (Open Systems Interconnection) [43]. Specifikace definuje dva typy služeb, a to *bez připojení* a *orientovanou na připojení*. NFC LLCP je kompaktní protokol, který poskytuje pevné základy na peer-to-peer komunikaci mezi zařízeními.

Digital Protocol Technical Specification specifikuje běžné vlastnosti NFC, které jsou běžně využívány bez nutnosti modifikace v hlavních oblastech, pro které je NFC využíváno, jako např. finanční služby nebo veřejná doprava. Tahle specifikace pokrývá, jak už název napovídá, digitální rozhraní NFC služeb, a také poloduplexní komunikaci NFC zařízení, a to v známých módech jako čtení / zápis nebo emulace karet. Specifikace je potom vázána k NFC LLCP specifikaci.

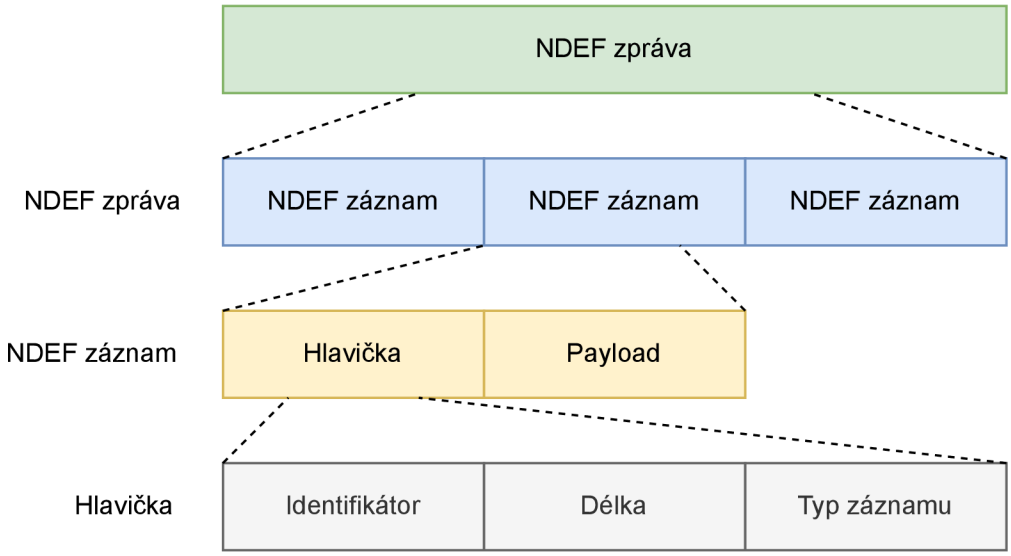
Activity Technical Specification vysvětluje, jak Digitální protokol může být využívám v komunikacích napříč NFC zařízeními. Používá k tomu stavební bloky, zvané Aktivity, které pomáhají připravovat komunikační protokol. Tyto aktivity mohou být použity tak, jak je navrženo v této konkretní specifikaci, nebo mohou být upraveny a modifikovány dle potřeb. Společně s profily pak tvoří několik verzí této specifikace, kdy dřívější verze obsahují profily a aktivity společně, v novějších verzích pak mají profily své vlastní specifikace.

Simple NDEF Exchange Protocol Technical Specification je důležitá specifikace z hlediska standardizace posílaných zpráv napříč NFC zařízeními, kdy se posílají data v takzvaném NDEF formátu (NFC Data Exchange Format). Tento

protokol využívá NFC LLCP typ služby orientovanou na připojení v transportním módu ke spolehlivému přenosu dat v NDEF formátu.

NDEF formát je formát zprávy respektující návrh NFC fóra [44]. Struktura této zprávy je následující: NDEF zpráva se sestává s jednoho nebo více NDEF záznamů. NDEF záznam má pak na základní úrovni dvě položky, a to typ záznamu a samotný payload. Znázornění NDEF zprávy je na obrázku 4.1. NDEF záznamů existuje celá řada, kdy jednotlivé typy se používají dle případu užití. Mezi typy NDEF záznamů lze jmenovat:

- URI NDEF záznam.
- Absolutní NDEF záznam.
- Známý NDEF záznam.
- Neznámý NDEF záznam.
- Prázdný NDEF záznam.
- Textový NDEF záznam.
- Mime typ NDEF záznam.

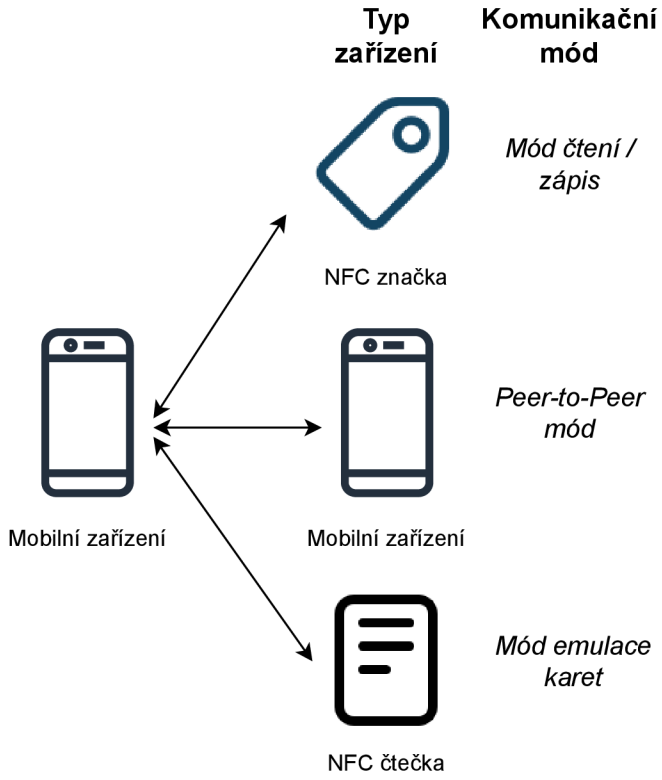


Obr. 4.1: Struktura NDEF zprávy

4.2 Principy komunikace

Při komunikaci přes NFC jsou tři nejčastější entity, a to NFC značka, mobilní zařízení a NFC čtečka, přičemž, jak bylo zmíněno v předchozí kapitole, tyhle entity mohou být buďto aktivní nebo pasivní zařízení. Možné komunikační módy jsou znázorněné na obrázku 4.2. Tyhle módy mohou být v interakci s mobilním zařízením peer-to-peer, čtení/ zápis nebo emulace karet [45].

Každý komunikační mód má svá specifika a standardy, dle kterých se řídí, a to např. ISO/IEC (International Organization for Standardization) 18092 NFCIP-1, ISO/IEC 21481 NFCIP-2, JIS X 6319-4/Felica a ISO/IEC 14443. Standardy používané pro bezkontaktní karty nebo pro emulaci karet mají označení NFC-A, NFC-B, respektive NFC-F.



Obr. 4.2: Používané módy NFC v závislosti na typu zařízení

4.3 Bezpečnost NFC

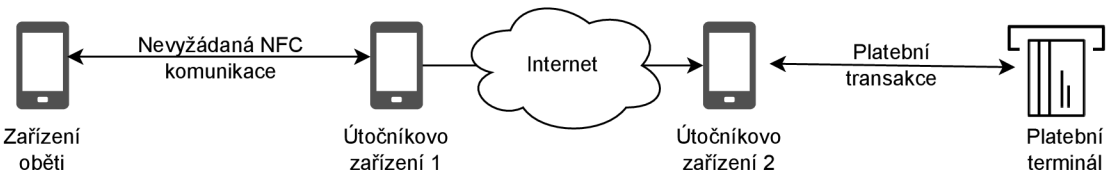
Ačkoliv svoji povahou, kdy přenos přes NFC probíhá na krátké vzdálenosti (cca do 10 cm), se může jevit NFC jakožto bezpečná technologie, na bázi její nativní podpory nemusí být bezpečnost dostatečná [46]. Existuje řada útoků, které je možné provést právě na NFC zařízení. Jako první lze jmenovat **odposlechy**. Při odposlechu je útočníkova největší překážka malá vzdálenost, při které spolu entity přes NFC komunikují, i přes to je odposlech možný. Dále při odposlechu hraje roli mnoho dalších faktorů. Velkým faktorem je zde útočníkovo vybavení. Je potřeba mít specifické nástroje pro uskutečnění odposlechu. Tímto nástrojem může být např. *Proxmark* [47], nástroj vyvinut pro švýcarskou armádu, který umí zachytávat RFID nebo NFC komunikaci. Mezi další faktory lze řadit lokaci, kde hraje roli např. šum. Opatřením

proti odposlechu je šifrování komunikace mezi NFC zařízeními. V momentě, kdy se útočníkovi podaří zachytit komunikaci, která je ale šifrovaná, bez znalosti klíčů má pro něj zachycená komunikace malou informační hodnotu.

Bezdrátové technologie jsou citlivé na útok **odepření služby** DOS (Denial of Service) čili *DOS attack* a není tomu jinak u NFC. Cílem útoku odepření služeb je zahlcení NFC zařízení v takové míře, že NFC zařízení není dostupné a nedokáže provést požadované operace. Tohohle útoku se dosahuje např. pomocí RFID nebo NFC *rušičky*. Proti útokům, kdy je používaná rušička není mnoho účinných opatření. Jedno z nich je případ, kdy NFC zařízení aktivně vyhledává frekvence rušičky, a v případě shody přestane přenášet data.

Útok mužem uprostřed je útok, kdy účastníci komunikace věří, že komunikují přímo mezi sebou, ale namísto toho jsou ustanovena dvě spojení, kdy útočník komunikuje s každou stranou komunikace zvlášť. V teoretickém scénáři, kdy Alice i Bob spolu komunikují v aktivním režimu, může Eva odposlechnout komunikaci a v momentě, kdy Alice chce přenášet zprávu Bobovi, Eva tuhle komunikaci přeruší, pošle Bobovi zprávu, kde vystupuje jako Alice. Alice bude čekat na odpověď, kterou dostane od Evy v domnění, že je Eva Bob. Prakticky je ale téměř nemožné v NFC komunikaci tento klasický útok mužem uprostřed zrealizovat.

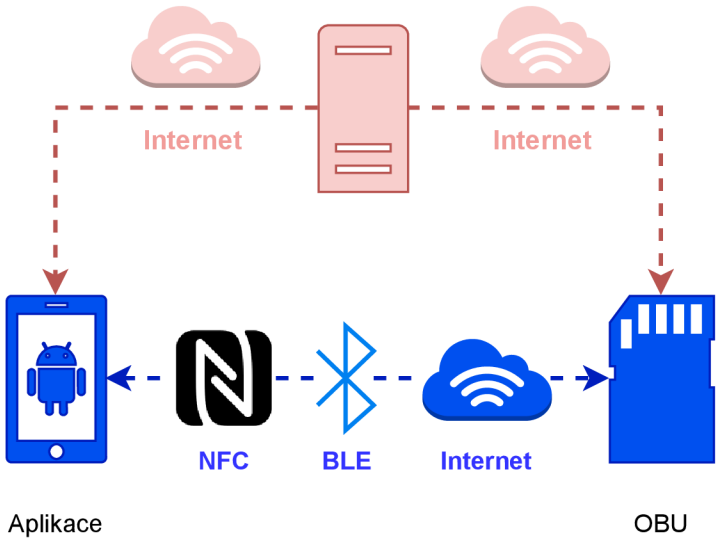
Zvláštním typem útoku mužem uprostřed je takzvaný **relay** útok. V případě NFC se tento útok děje v případě, že zařízení oběti je telefon v komunikačním módu emulace karet. Útočník musí mít v tomto případě dvě zařízení, nebo musí být dva útočníci. Útočník se přiblíží k oběti s prvním zařízením a bez jeho vědomí proběhne NFC komunikace s jeho zařízením, které je přeneseno na druhé zařízení útočníka, které může udělat např. platební transakci. V praxi je možno setkat se s tímto útokem v hromadné dopravě, kde útočník může zkoušet, kdo má zapnuté NFC, a bez zpozorování učinit velké množství útoků. Tahle situace je znázorněna na obrázku 4.3. Ochrana proti tomuto útoku je neponechávat zapnuté NFC ve svém mobilním zařízení celou dobu. Další případy relay útoků se dějí u bez-klíčových automobilů.



Obr. 4.3: Relay útok na mobilní zařízení se zapnutým NFC

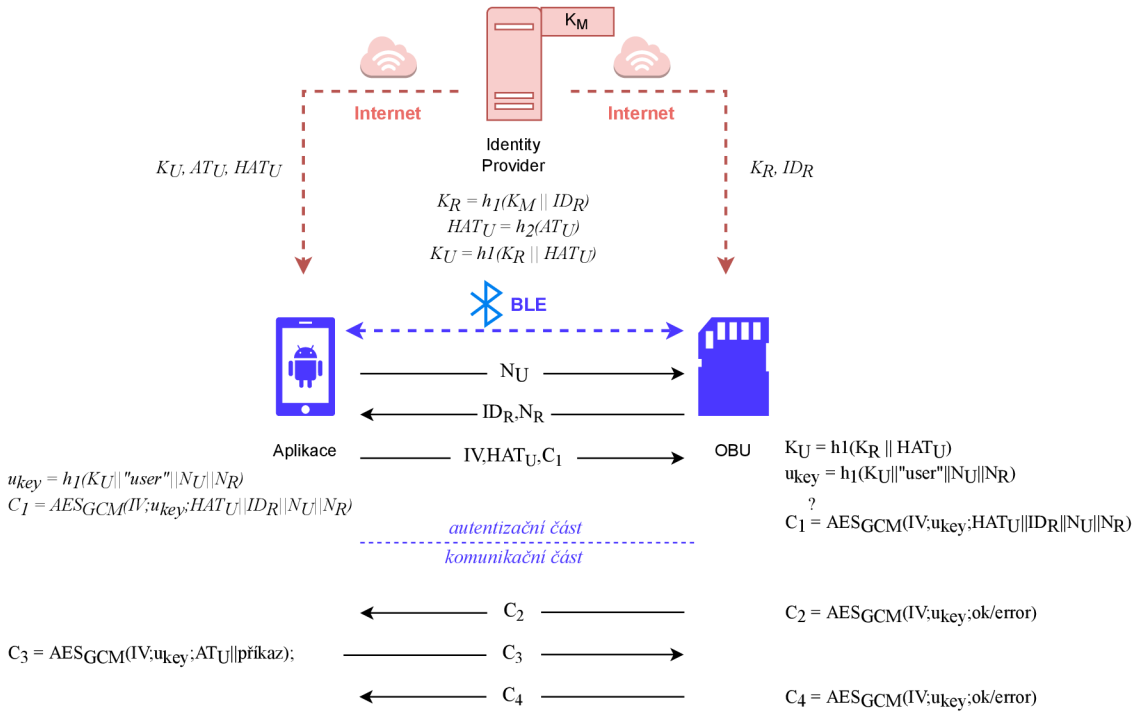
5 Návrh a implementace přístupového systému

Návrh a implementace přístupového systému je inspirován článkem [49]. Z abstraktního hlediska je návrh naznačen na obrázku 5.1. Na obrázku můžeme vidět tři entity, které jsou stěžejním bodem této bakalářské práce, a které tvoří přístupový systém pro carsharingovou aplikaci. Identity Provider 5.1 generuje parametry jako kryptografické klíče, identifikátory, autentizační token (obrázek 5.4), a také haše těchto parametrů. Tyhle parametry pak předá mobilní aplikaci 5.3 a řídící jednotce OBU 5.2 přes internet. Mobilní aplikace a OBU spolu pak komunikují skrze BLE, NFC nebo Wi-Fi, a to ve třech bezpečnostních úrovních.



Obr. 5.1: Abstraktní schéma přístupového systému

Přístupový systém je navržen do tří logických celků. Prvním je generování klíčů a parametrů Identity providerem a jejich následné předání skrz internet zbylým dvou entitám, jenž jsou mobilní aplikace a OBU. Druhá část je již vykonávána skrz bezdrátové technologie, a to mezi mobilní aplikací a OBU. V této části se uživatel používající aplikaci autentizuje. Je-li uživatel autentizován, proběhne třetí část, a to část komunikační. Zde již mezi sebou komunikují zmíněné entity pomocí zašifrovaných zpráv. Popsané vztahy a logické celky lze vidět na obrázku 5.2. Druhá verze protokolu, jenž je implementována i s technologiemi NFC a Wi-Fi, je znázorněna na obrázku 5.3. Komunikace je zkrácena z 6 na 4 kroky. Uživatel rovnou posílá `ATU` (autentizační token uživatele) společně s příkazem. OBU pak ověří, zda se hašované `ATU` rovná poslanému `ATU` a pošle nešifrovanou odpověď autentizace, čímž se šetří výpočetní výkon zařízení.

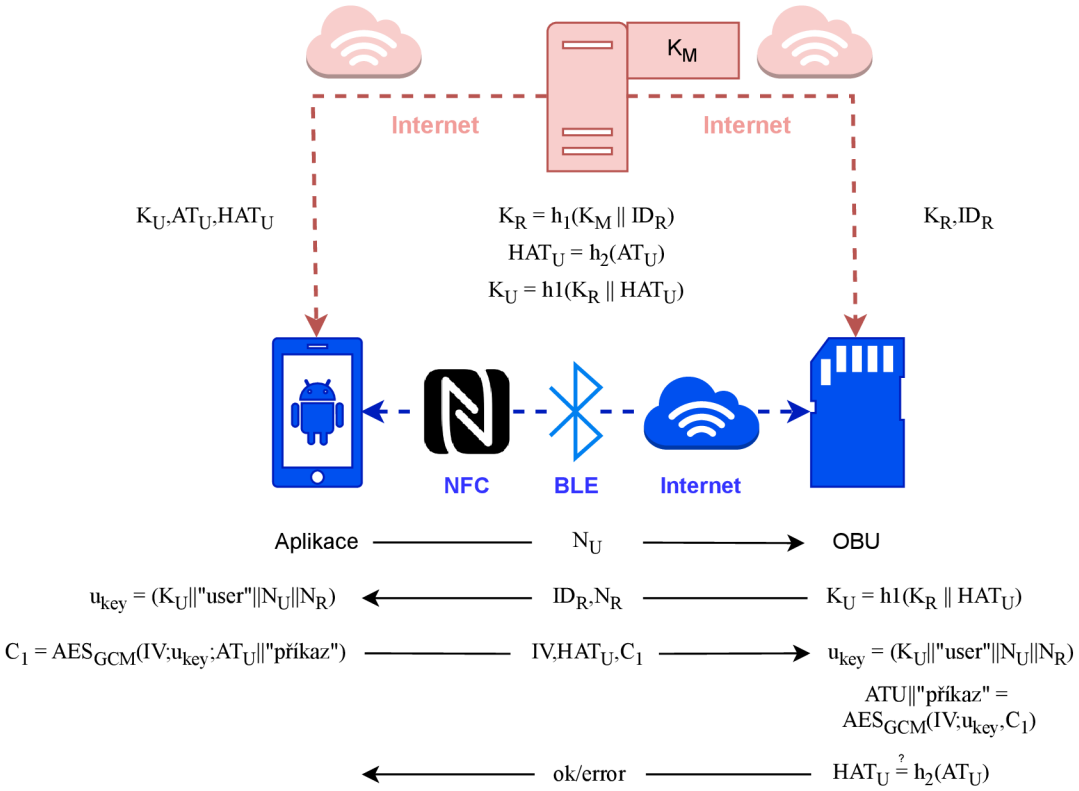


Obr. 5.2: Návrh přístupového systému první verze protokolu

5.1 Identity Provider

Pro implementaci Identity Providera byl zvolen programovací jazyk Java. Jak již bylo zmíněno, Identity Provider se stará o generování kryptografických klíčů a identifikátorů. Klíče mohou mít délku 128, 192 nebo 256 bitů. V těchto bezpečnostních úrovních pak komunikují spolu mobilní aplikace a OBU. Klíče jsou uchovávány v hexadecimálním zápisu. Jako první se vygeneruje `master key` K_M, z něhož jsou odvozené všechny další klíče v relaci. Dále se vygeneruje celočíselný identifikátor OBU zařízení ID_R. Tento identifikátor by měl být pro každé OBU jedinečný. V této bakalářské práci se pracuje pouze s jedním serverem (v BLE terminologii *peripheral*), proto je identifikátor ID_R v implementaci stejný. V praxi je nutno zavést systém inkrementace, aby byla zajištěna jedinečnost. Pomocí hašovací funkce h_1, což podle bezpečnostní úrovně je funkce SHA-1, SHA-224 nebo SHA-256, je zahašován zřetězený vstup K_M a ID_R. Výstupem je pak `reader key` K_R.

Důležitým prvkem generovaných parametrů od Identity Provideru je `ATU`. `ATU` je bajtové pole znázorněno na obrázku 5.4. Prvních 8 bajtů je unikátní identifikační číslo uživatele (UID), identifikační typ (IT) definuje typ autentizačního tokenu a stanovuje roli uživatele, „S“ značí servisního inženýra, „K“ pak klienta. Bajty na indexech 9–15 jsou vyhrazeny pro identifikační číslo vozidla (VID), následují bajty definující začátek platnosti autentizačního tokenu, přičemž formát je BCD (Binary

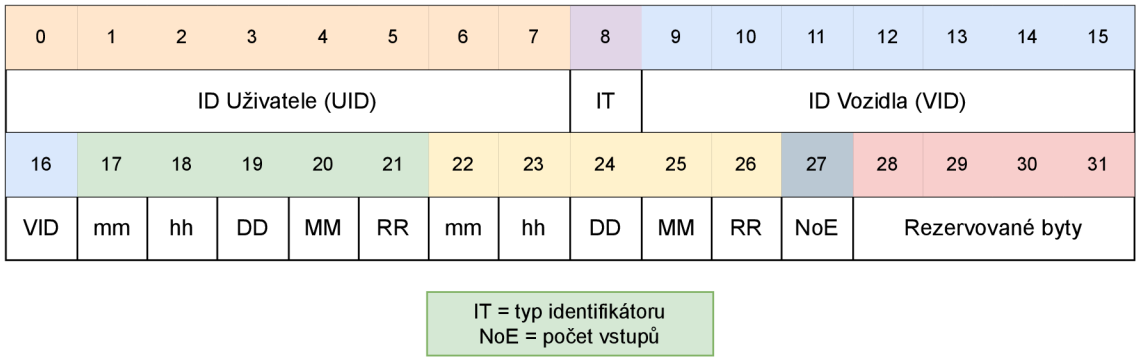


Obr. 5.3: Návrh přístupového systému druhé verze protokolu

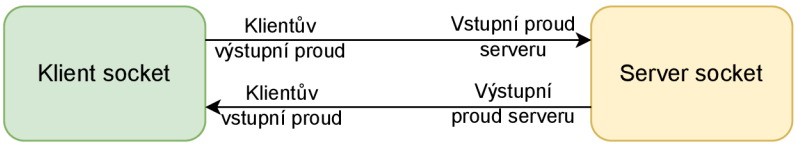
Coded Decimal) časový kód. Konkrétněji RR:MM:DD:hh:mm, kdy „R“ značí rok, „M“ je měsíc, „D“ rozumíme den, „h“ značí hodiny a „m“ minuty. Bajty definující konec platnosti autentizačního tokenu jsou také v BCD formátu. Počet vstupů NoE (Number of Entries) značí kolikrát se může k autentizačnímu tokenu přistupovat v jeho platné době. Poslední 4 bajty jsou pak rezervované k individuálním záležitostem. Tyhle parametry aplikace čte z textového souboru. V případě, že nejsou dodány, jsou nastavené výchozí hodnoty.

Pomocí hašovací funkce h_2, což podle bezpečnostní úrovně je funkce SHA-256, SHA-384 nebo SHA-512, je zahašováno `ATU`. Výstupem je potom `HATU`. Výsledkem funkce h_1 pro zřetězený vstup K_R a `HATU` je `user key` K_U. V momentě, kdy jsou všechny výše zmíněné parametry vygenerovány, je Identity Provider připraven přenášet tyhle parametry aplikaci a OBU přes internet.

Pro přenášení parametrů bylo využito *Java Socket programmingu*. *Java Socket programming* je snadné přenášení objektů nebo zpráv přes internet znázorněné na obrázku 5.5. Server otevře socket na zvoleném portu, a potom může přijímat nebo odesílat zprávy nebo objekty přes vstupní a výstupní proud. Příprava serveru přes Java Socket Programming lze vidět na výpisu 5.1.



Obr. 5.4: Autentizační token uživatele



Obr. 5.5: Komunikace přes Java Socket Programming

Výpis 5.1: Zřízení serveru přes Java Socket

```
1  public static final int PORT = 10001;
2  ServerSocket serverSocket = new ServerSocket(PORT);
3  System.out.println("Server is up and running on ip " +
4  serverSocket.getInetAddress().getLocalHost().
       getHostAddress()
5  + " port: " + PORT);
6  Socket socket = serverSocket.accept();
7
8  ObjectOutputStream objectOutputStream = new
9  ObjectOutputStream(socket.getOutputStream());
10 ObjectInputStream objectInputStream = new
11 ObjectInputStream(socket.getInputStream());
```

Při posílání objektu jsou zásadní dvě podmínky. Jelikož je objekt potřeba pro proud serializovat, třída přenášeného objektu musí dědit z třídy `Serializable`. Další podmínkou je, že přenášený objekt musí být na obou stranách stejného typu. Stejný objekt v Javě znamená stejný název včetně balíčku. V případě této práce to znamená objekty typu `com.example.doprava_bp.ObuParameters` a `com.example.doprava_bp.AppParameters`.

Implementace Identity Provideru je navržena tak, že ve for cyklu nejdříve otevře server pro OBU. Aplikace metodou `readObject()` přijme vstupním proudem objekt

typu `com.example.doprava_bp.ObuParameters`. Pokud se zpráva od klienta rovná *Hello from OBU!*, server metodou `writeObject()` pošle objekt naplněný vygenerovanými parametry klientovi. Popsaný sled událostí je na výpisu 5.2. Jakmile je klientem OBU objekt obdržen, server zavře socket na uvedeném portu. Ve druhé iteraci cyklu je potom zvolen stejný postup, jen na vstupním a výstupním proudu se tentokrát posílají objekty typu `com.example.doprava_bp.AppParameters`.

Výpis 5.2: Posílání objektu klientovi

```
1 ObuParameters recObuParams = (ObuParameters)
     objectInputStream.readObject();
2         System.out.println(recObuParams.message);
3
4         if (recObuParams.message.equals("Hello from
             OBU!")) {
5             obuParameters.message = "Hi! - from the
                 server!";
6             objectOutputStream.writeObject(obuParameters);
7         }
8         serverSocket.close();
```

5.2 Řídící jednotka OBU

Palubní jednotka OBU vystupuje v návrhu přístupového systému jako NFC, TCP (Transmission Control Protocol) nebo BLE server, v terminologii BLE lépe řečeno jako BLE *peripheral*. Pro první verzi protokolu byl implementován BLE server. Pro druhou verzi protokolu je pak možnost komunikovat i s NFC a TCP serverem. U všech aplikací lze komunikovat ve třech bezpečnostních úrovní (128, 192, 256 bitové klíče). O kryptografii se stará ve všech verzích komunikační technologie třída `CryptoCore`. Metody této třídy mají na vstupu šifrové texty a vrací výsledek autentizace. Tato třída je tedy nezávislá na typu použité technologie. Ve všech aplikacích také kromě `CryptoCore` figuruje i třída `Cryptogram`, která uchovává parametry potřebné k šifrování, dešifrování nebo ke stanovení klíče.

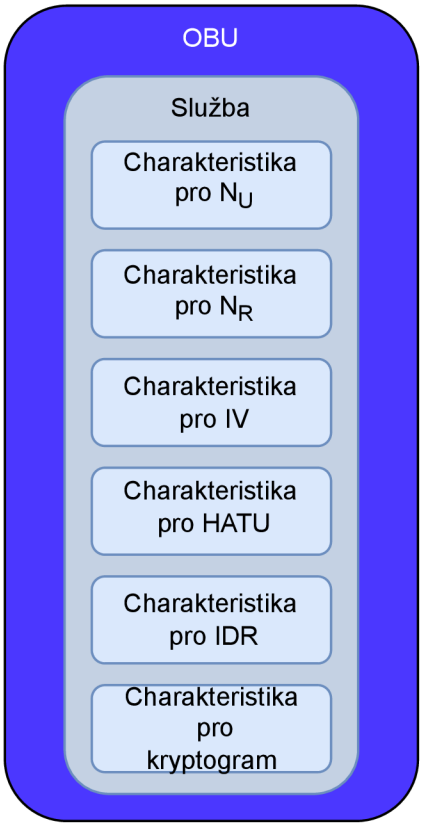
5.2.1 BLE peripheral

Pro implementaci BLE peripheral byla zvolena knihovna v Javě [50], která je postavena na Linux knihovně BlueZ, proto je k implementaci využit i virtuální stroj s operačním systémem Linux v distribuci Ubuntu, verzi 22.04 LTS. Pro správné fungování této aplikace je potřeba mít také Bluetooth adaptér.

Ve výchozím stavu poskytuje knihovna jednu službu a dvě charakteristiky. V implementaci serveru byl počet charakteristik rozšířen na šest, kdy každá charakteristika je určena zvlášť pro určitý přenášený parametr. Opačný přístup by byl mít pouze dvě charakteristiky, kdy jedna slouží k zapisování a druhá ke čtení, v této bakalářské práci byl však použit první zmíněný přístup k lepší názornosti, a také kvůli lepší práci s charakteristikami. Každá charakteristika i služba musí mít svoje unikátní UUID (Universally Unique Identifier), přičemž konkrétní UUID jsou následující:

- Pro službu `18b41747-01df-44d1-bc25-187082eb76bf`.
- Pro nonce uživatele `bc3ba145-f588-4f86-8bc4-fb925a23dc31`.
- Pro nonce ověřovatele `c8ba0ef6-5c27-11ed-9b6a-0242ac120002`.
- Pro IV `27938afb-f6f0-4e19-a4b2-2545da6bad40`.
- Pro HATU `b828f7a3-157a-4a4e-9c9b-3feb19b8e90d`.
- Pro IDR `01e6ee2d-420a-4380-8e14-2a83844d4ae8`.
- Pro kryptogram `23cab78f-28d9-4ecb-bfd1-1bba7b486fa3`.

Návrh OBU peripheral je znázorněn na obrázku 5.6.



Obr. 5.6: Návrh OBU peripheral

Jakmile OBU přijme parametry od Identity Provideru, nastaví přijaté hodnoty u charakteristiky pro $\mathtt{ID}_R$. U charakteristiky pro nonce ověřovatele poté vygeneruje náhodné číslo a přiřadí jej do hodnoty charakteristiky. Jakmile se připojí BLE

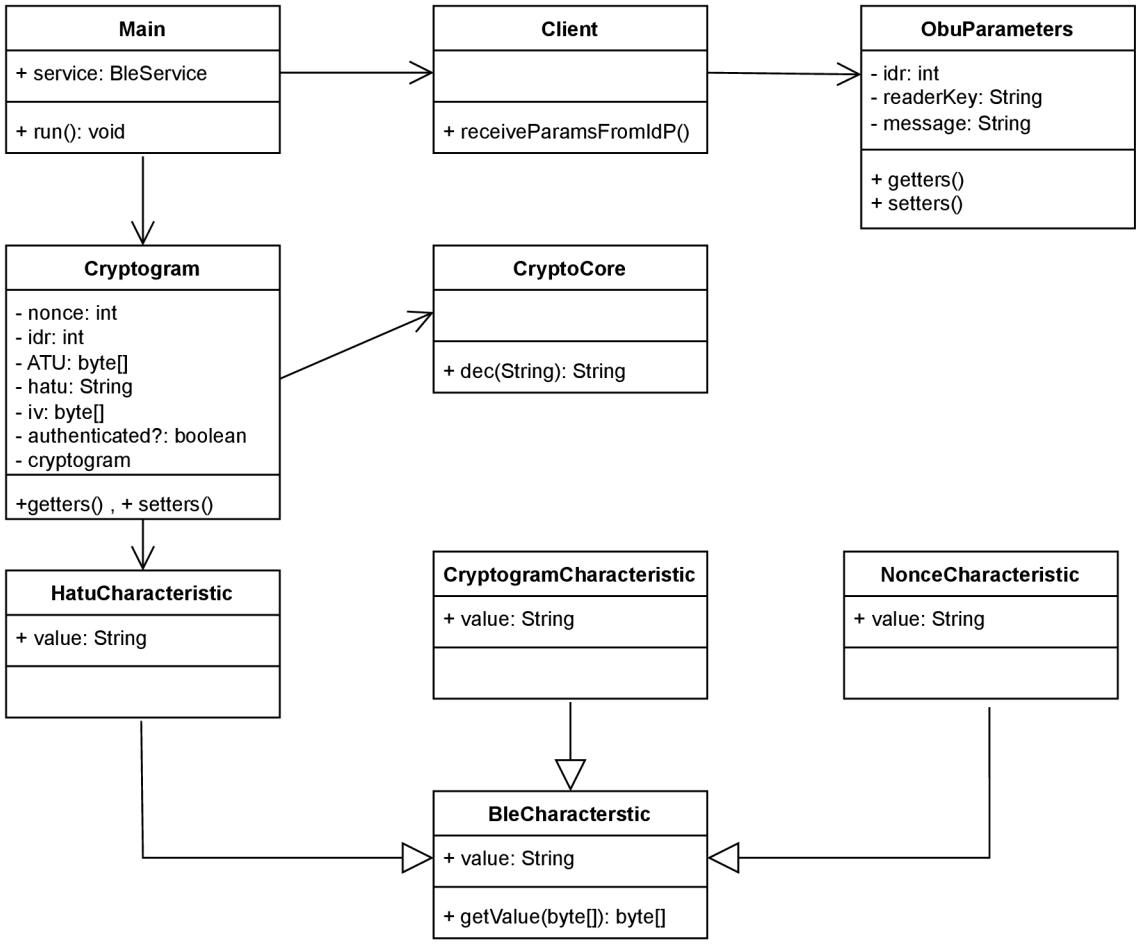
central, v našem případě mobilní aplikace, provedou se operace čtení a zápisu. Po operacích se očekává, že všechny charakteristiky budou mít hodnotu. Aplikace si vypočítá `ukey`, který bude sloužit jako klíč k šifrovaným textům. K tomuto klíči si ale nejdříve musí vypočítat K_U, který vypočítá funkcí h_1, kdy se zřetězí K_R přijaté od Identity Providera a `HATU` přijaté od mobilní aplikace. Poté si aplikace dle obrázku 5.2 zašifruje zřetězený vstup. Jelikož OBU používá stejný klíč i inicializační vektor, výsledek šifrovaného textu by měl být stejný jako hodnota v charakteristice kryptogramu, kde hodnotu zapsala aplikace. Pokud je hodnota skutečně stejná, OBU zapíše do charakteristiky „true“, čímž je uživatel autentizován. Pokud tomu tak není, OBU zapíše do charakteristiky „false“, a tím končí komunikace mezi uživatelem a ověřovatelem. V dalším kroku si OBU vezme opět hodnotu z kryptogramu, kam zapsala mobilní aplikace, odtud si hodnotu rozdělí na autentizační token uživatele a příkaz. `ATU` aplikace zahašuje funkcí h_2 a porovná s hodnotou `HATU` z hodnoty charakteristiky, kam zapsala mobilní aplikace. Za splnění podmínek, že dle `ATU` má uživatel práva k vykonání příkazu, a že zahašované `ATU` se rovná přijatému `HATU`, OBU pošle „true“, v opačném případě pak „false“. Tento sled událostí je zobrazen na vývojovém diagramu na obrázku 5.8. V druhé verzi protokolu je sloučen první a druhý krok dohromady.

Struktura aplikace je znázorněna na obrázku 5.7 v diagramu tříd. V diagramu lze vidět, že OBU si případně parametry převezme od Identity Providera prostřednictvím třídy `Client`, který přenáší instanci třídy `ObuParameters`. Třída `Cryptogram` uchovává potřebné parametry k dešifrování přijatého šifrového textu od mobilní aplikace. Tyhle parametry si bere z hlavní třídy `Main` nebo ze tříd charakteristik, které dědí z třídy `BleCharacteristic`.

5.2.2 NFC Server

NFC Server je postaven na aplikaci od Ing. Petra Dzurendy, PhD. Aplikace je naprogramovaná v Javě a je zde využito knihovny `javax.smardcardio`. Stejně jako u BLE peripheral, i zde aplikace přebírá parametry od Identity Provideru. Aplikace si klíče a další parametry uchovává v textových souborech dle bezpečnostní úrovně. O používané bezpečnostní úrovni rozhoduje proměnná `KEY_LENGTH`, která poté určuje, z jakého souboru se čtou klíče a další parametry. Komunikace skrz technologii NFC probíhá prostřednictvím NFC čtečky. Program určí čtečce co přenášet pomocí metody `sendAPDU()`, jenž má jako vstupní parametr bajtové pole dat, které chceme skrz NFC čtečku přenášet.

Jako první bajtové pole, které aplikace přenáší, je takzvaný *Select AID* (Application Identifier) příkaz, díky němuž mobil rozpozná, že NFC příkaz je zamýšlen pro konkrétní mobilní aplikaci. Jakmile mobilní aplikace zpracuje příkaz a odpoví



Obr. 5.7: Diagram tříd OBU

(blíže popsáno v 5.3) návratovým kódem `90 00`, znamená to, že se spojení s aplikací podařilo a komunikace probíhá dál. V opačném případě spojení nebylo zdařilé a komunikace již dál neprobíhá. Podle počtu přijatých kladných odpovědí je ovlivňována proměnná `counter`. Ta potom určuje, v jaké části protokolu se aplikace a komunikace právě nachází, a jaké parametry je potřeba přenášet přes čtečku. V případě návratového kódu `90 00` jsou pak obsahem odpovědi také parametry od mobilní aplikace, které jsou v případě potřeby převedeny z bajtového pole na příslušný datový typ (textové řetězce nebo celočíselné typy). Tyhle vztahy jsou znázorněné na výpisu 5.3. Na konci komunikace je z přijatých parametrů zavolána třída `CryptoCore`, která ověří autentizaci a autentizační token uživatele a výsledek autentizace je potom poslán mobilní aplikaci.

Výpis 5.3: Rozebrání bajtového pole a proměnná counter

```
1 if(counter == 2){
2     start2 = Instant.now();
3     byte[] hatuArray = new
```

```
        byte[(obuParameters.getKeyLengths()/4)];
4       byte[] nonceArray = new byte[4];
5       System.arraycopy(baResponceAPDU,0,hatuArray,0,(...));
6       userCryptogram.setHatu(new String(hatuArray));
7       System.out.println("Parsovane hatu: " +
            userCryptogram.getHatu());
8       System.arraycopy(baResponceAPDU,(..,),
            nonceArray,0,4);
9       userCryptogram.setNonce(ByteBuffer.wrap(nonceArray).
            getInt());
10      System.out.println("Parsovane nu: " +
            userCryptogram.getNonce());
11      sendAPDU(new byte[]{(byte) 0x00, (byte) 0x01, (byte)
            0x02, (byte) 0x04});
12  }
```

5.2.3 TCP Server

TCP server je stejně jako Identity Provider založen na *Java socket programmingu*. Stejně jako na výpisu 5.2 server otevře socket na určitém portu, který si potom převezme proměnná socketu metodou `accept()`. Následně se vytvoří instance vstupního proudu `objectInputStream` a instance výstupního proudu `objectOutputStream`, tak jak je popsáno na obrázku 5.5. V průběhu ladění bylo zjištěno, že pokud je při celé komunikaci využívána stejná instance `Socket` a `ServerSocket` tříd, může to zapříčinit pády aplikace a vyvolávat výjimky jako například *Connection Refused*. Z tohoto důvodu jsou pro každou část aplikace vytvořeny nové instance těchto tříd.

Při dešifrování přijatého šifrového textu a autentizace může při neúspěchu vyvolat třída `CryptoCore` výjimku typu `AEADBadTagException` nejčastěji s dovětkem *Tag mismatch!*. Tato výjimka není nikterak neobvyklá, jelikož se vyvolá v případě, že je použit špatný klíč. V případě, že by se snažil autentizovat uživatel, který autentizovat být nemá, bude tedy tato výjimka vyvolána pokaždé, a proto je potřeba ji ošetřit v try-catch bloku. Toto ošetření při autentizaci uživatele je znázorněno na výpisu 5.4. Stejným způsobem se potom ošetřuje tato výjimka i v aplikaci NFC serveru.

Výpis 5.4: Ošetření výjimky při použití třídy CryptoCore

```
1   CryptoCore cryptoCore = new
        CryptoCore(obuParameters,userCryptogram,
        receiverCryptogram);
```

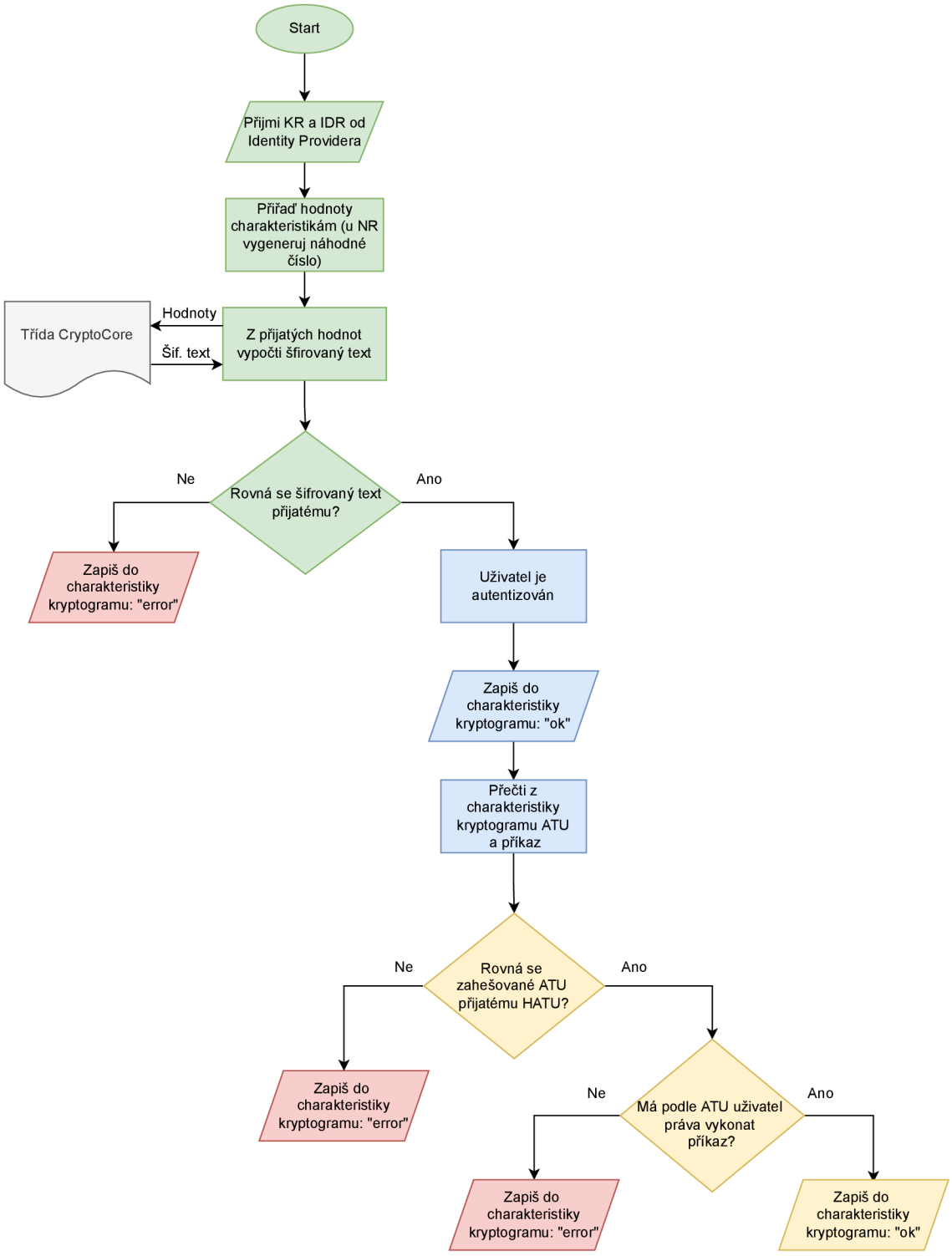
```
2       try {
3           receiverCryptogram.setAuthenticated
                (cryptoCore.dec(userCryptogram.cryptograms
                .get(0)));
4       }
5       catch (AEADBadTagException ex) {
6           receiverCryptogram.setAuthenticated(false);
7       }
```

5.3 Uživatelská mobilní aplikace

Pro mobilní aplikaci bylo zvoleno Android Studio. Výchozím programovacím jazykem je zde Kotlin, některé třídy byly však programované v Javě. Android Studio však poskytuje dobrou kompatibilitu mezi těmito dvěma jazyky. Pro práci s BLE byla využita knihovna [51], která je zaměřená přímo na práci s BLE v Android Studiu. Pro práci s NFC bylo využito knihovny `android.nfc.cardemulation.HostApduService` a pro komunikac s TCP serverem *Java socket programmingu*. Aplikace je navržena dle diagramu tříd, který lze vidět na obrázku 5.10, přičemž třída `MyHCEService` vystupuje jako služba, která je zavolána přes Intent, proto v obrázku nevystupuje. Na diagramu tříd lze vidět, že mobilní aplikace si v případě potřeby vyžádá klíče a paremetry od Identity Providera prostřednictvím třídy `Client`, který přenáší instanci třídy `AppParameters`.

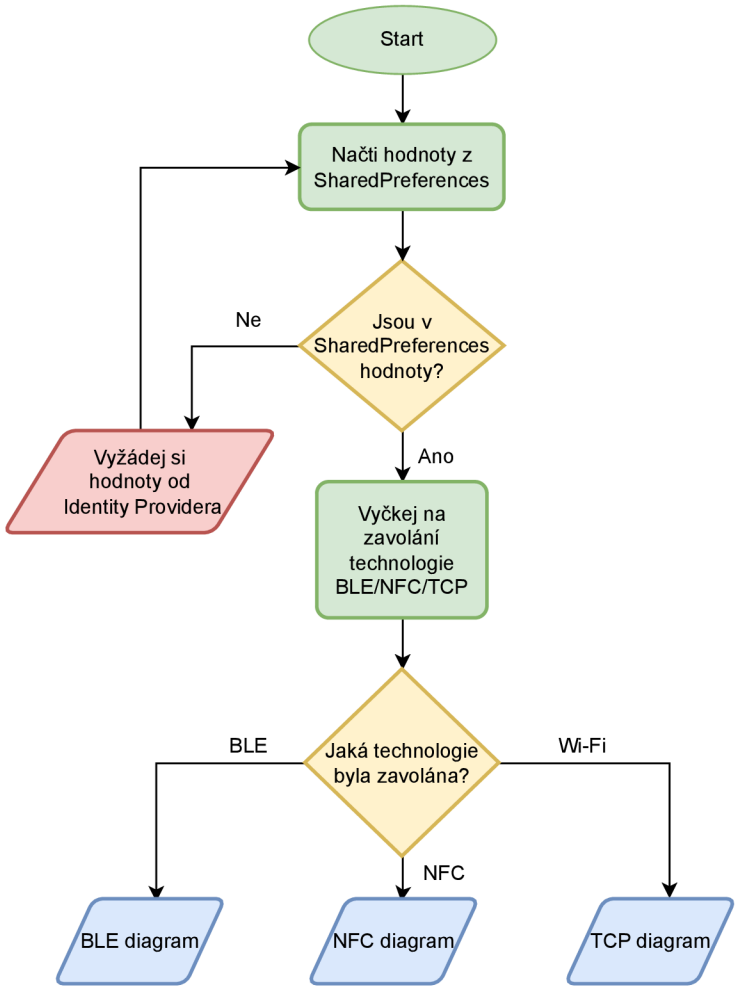
Hlavní logika aplikace tkví v práci s tlačítky, které jsou definované v hlavní třídě `MainActivity`. Po stisknutí tlačítka uživatelem se zavolá metoda `setOnClickListener()`. Poté, co uživatel stiskne tlačítko „vyžádat parametry“, předá Identity Provider aplikaci parametry. Zde se využívá třídy `Client`, která se stará o Java Socket programming 5.1, a kde se přenáší objekt typu `com.example.doprava_bp.AppParameters`. Jakmile uživatel klikne na tlačítko „připojit se k BLE“, využije se třídy `Bluetooth Handler`, která implementuje výše zmíněnou knihovnu. Jakmile uživatel klikne na tlačítko „Wi-Fi“, zahájí komunikaci s OBU prostřednictvím TCP spojení. TCP spojení je obsluhováno přímo v hlavní třídě. Ideální situace je mít i na TCP spojení vlastní třídu, ale vzhledem k malému množství potřebného kódu k této implementaci není pro TCP spojení vytvořena vlastní třída. Pro komunikaci skrz NFC není v aplikace určeno žádné tláčítko, neboť komunikace prostřednictvím NFC je inicializována přiložením mobilního telefonu na NFC čtečku.

Zatímco v aplikacích zastupujících server jsou klíče uchovány v textovém souboru, v mobilní aplikaci jsou k tomuto účelu použity takzvané *shared preferences*. Shared preferences mají syntaxi datového typu mapy a jejich hodnoty se uchovávají



Obr. 5.8: Vývojový diagram OBU

v interním souboru deklarovaném v metodě `getSharedPreferences()`. Po vypnutí aplikace jsou tyto parametry tedy zachovány.Pro všechny tři bitové délky klíčů jsou vytvořeny tyhle proměnné zvlášť. Pro editaci hodnot uložených v shared preferences



Obr. 5.9: Vývojový diagram mobilní aplikace

je potřeba takzvaný editor, který se spojí s shared preferences metodou `edit()`. Po provedení tohohle příkazu se do proměnné shared preferences může zapsat například metodou `putString()`. Sled popisovaných událostí je popsán na výpise 5.5, a taky na vývojovém diagramu na obrázku 5.9, kde je znázorněno, že aplikace nejdřív zjišťuje, zda má uchované klíče a parametry. V případě, že ne, musí si aplikace tyhle parametry vyžádat od Identity Providera. V případě, že parametry uchované má, aplikace čeká, která technologie bude zvolena pro komunikaci s OBU.

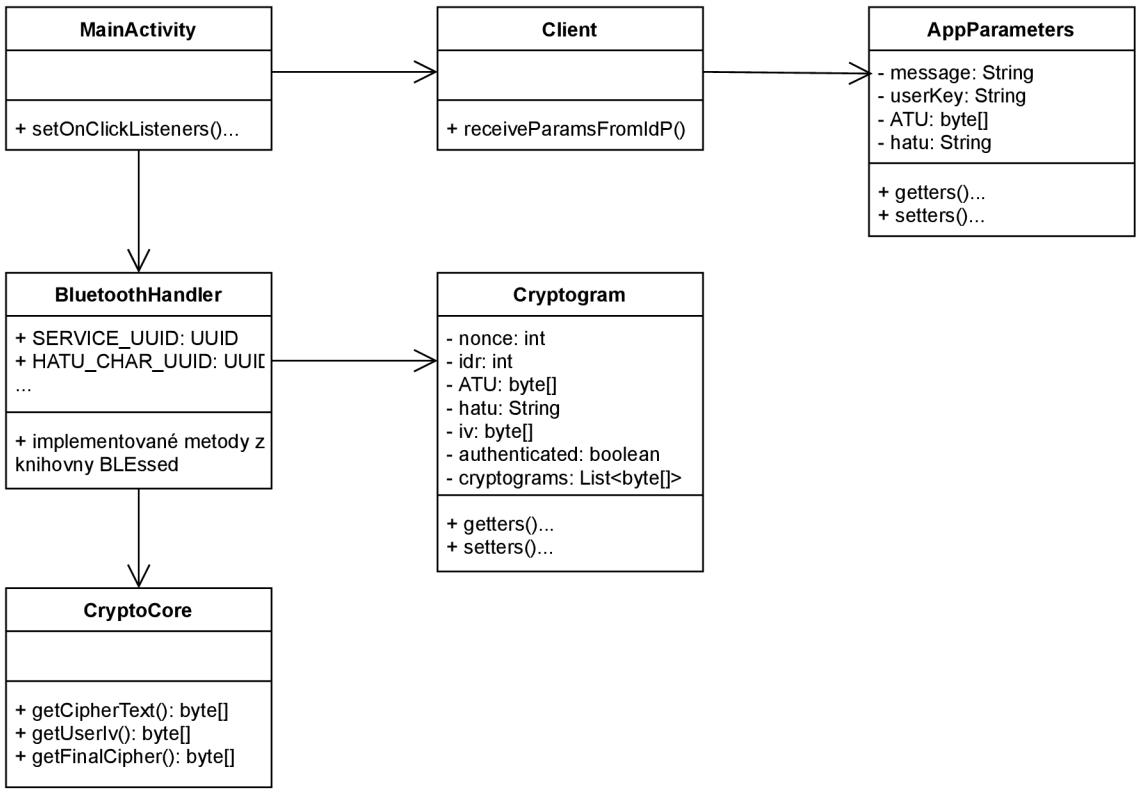
Výpis 5.5: Shared preferences

```
1 val sharedPreferences128 =
    getSharedPreferences("AppParameters128",
    Context.MODE_PRIVATE)
2 val editor128 = sharedPreferences128.edit()
3 editor128.putString("userKey",
    client.appParameters.userKey)
```

Prvním krokem je skenování blízkých BLE *peripherals*. Jelikož UUID služby OBU je nám známá, využívá se pro skenování následujícího příkazu.

Výpis 5.6: Skenování BLE peripheral skrze UUID služby

```
1 val SERVICE_UUID =
    UUID.fromString("18b41747-01df-44d1-bc25-187082eb76bf")
2 bluetoothHandler =
    BluetoothHandler(applicationContext,client.
    appParameters)
3 bluetoothHandler.central.
    scanForPeripheralsWithServices(arrayOf(SERVICE_UUID));
```



Obr. 5.10: Diagram tříd mobilní aplikace

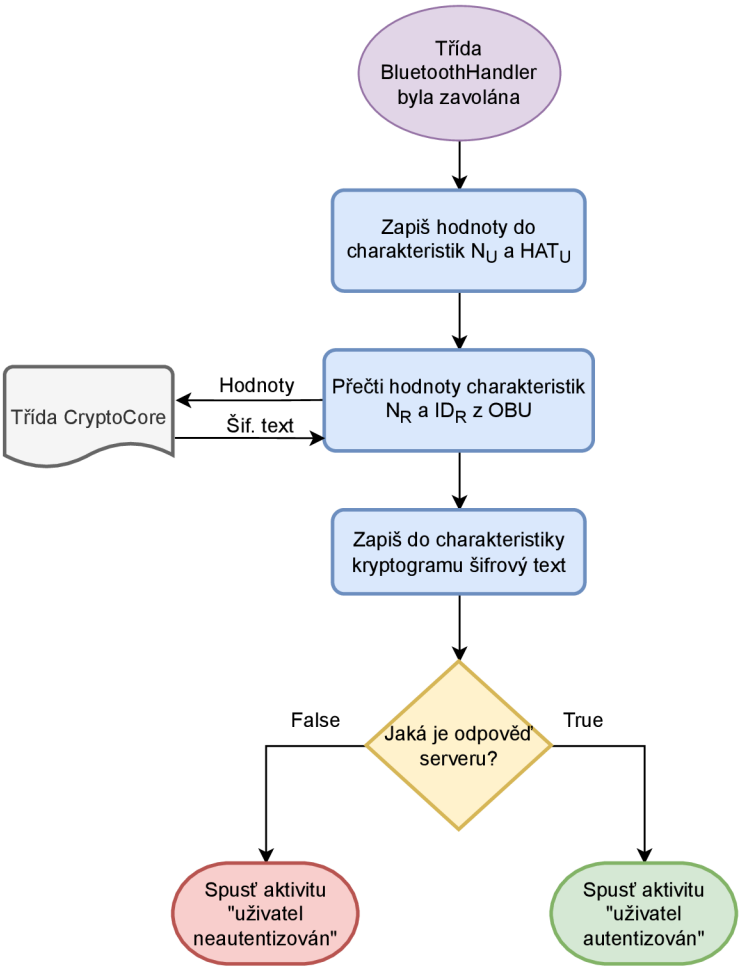
Při implementaci knihovny se pracuje s takzvaným zpětným voláním (anglicky callback). Metody zpětného volání se zavolají poté, co se vykoná nějaká akce, např. při objevení požadované služby se zavolá metoda `onServiceDiscovered()`. V těchto metodách je potom implementováno samotné chování aplikace s ohledem na BLE, proto se tyto metody přetěžují. Uvedená logika je zobrazena na výpisu 5.7, kde mj. můžeme vidět, že zápisy a čtení charakteristik se děje právě v metodě `onService`

`Discovered()`. Při zápisu a čtení charakteristik potom aplikace nastaví notifikace u charakteristik na pravdu.

Výpis 5.7: Implementace zpětných volání

```
1  val peripheralCallback : BluetoothPeripheralCallback =
2          object : BluetoothPeripheralCallback() {
3              override fun
                   onServicesDiscovered(peripheral:
                   BluetoothPeripheral) {
4                   peripheral.readCharacteristic(...)
5                   peripheral.writeCharacteristic(...)
6              }
7              override fun onCharacteristicUpdate(...){...}
8              override fun onCharacteristicWrite(...){...}
9              override fun
                   onNotificationStateUpdate(...){...}
10         }
11 private val bluetoothCentralManagerCallback:
       BluetoothCentralManagerCallback =
12         object : BluetoothCentralManagerCallback() {
13             override fun onDiscoveredPeripheral(...){...}
14             override fun onConnectionFailed(...){...}
15             override fun onConnectedPeripheral(...){...}
16         }
```

V `BluetoothHanndler` se také zapisuje do charakteristiky kryptogramu. Ta je určena k přenášení šifrovaného textu. K tomuto zápisu využívá třídu `CryptoCore`, která jak už bylo zmíněno, se stará o všechnu kryptografii v aplikaci. Zašifruje tedy požadovaný vstup pomocí parametrů ze třídy `Cryptogram`, které jí předá třída `BluetoothHandler`. V případě druhé verze protokolu se použije třída `BluetoothHandlerV2`. V těchto třídách se tedy, jak už bylo zmíněno, zapíše šifrový text, v podobě hexadecimálního textového řetězce, do charakteristiky kryptogramu pomocí metody `writeCharacteristic()`. Metoda `writeCharacteristic()` má ale jako vstupní parametr bajtové pole, a proto, aby bylo možné přenést hexadecimální textový řetězec, se využije instance třídy `BluetoothBytesParser`, která je součástí knihovny `BLESSED`. Uvedený sled událostí pro komunikaci prostřednictvím BLE je znázorněn na obrázku 5.11 Pro šifrování se používá AES (Advanced Encryption Standard) v provozním režimu GCM. Velikost inicializačního vektoru je 12 bytů. V hlavní třídě `MainActivity` je pak možnost volit délku klíčů.



Obr. 5.11: Vývojový diagram NFC sekce mobilní aplikace

Výsledek autentizace, a to ať už prostřednictvím jakékoliv technologie, se zobrazuje v novém okně (v terminologii Android v nové Aktivitě) viz obrázek 5.23. K tomu, aby aplikace spustila nové okno je potřeba vytvořit xml soubor nového okna, který určuje vzhled aktivity. V Android Studiu bývají tyhle aktivity typicky umístěny ve složce `res/layout`. Tento soubor se musí svázat s konkrétní Kotlin třídou v souboru `AndroidManifest.xml`. Toto svázání je ukázáno na výpisu 5.8. Zde se deklaruje, že třída s názvem `AuthActivityBle.kt` zastupuje aktivitu. V samotné třídě se potom provázání s aktivitou uskuteční v metodě `onCreate()` příkazem `setContentView(R.layout.authenticated_ble)`. Důležité je také zmínit, že třída která má reprezentovat aktivitu musí dědit z třídy `AppCompatActivity`.

Výpis 5.8: Deklarace nové aktivity v AndroidManifest.xml

```
1 </activity>
2         <activity android:name=".AuthActivityBle">
3
```

```
4 </activity>
```

O komunikaci prostřednictvím NFC se stará třída `MyHCEService`, jenž dědí ze třídy `HostApduService`. Tato třída je koncipována jako služba, proto má svá specifika. K používání třídy jako služby je zapotřebí volat třídu přes Intent. U volání služby je vhodné, aby Intent předával nějaké parametry, se kterými potom služba pracuje. Tento proces volání služby je demonstrován na výpise 5.9. Zde lze vidět vytvoření objektu typu *Intent* a jeho spojení s třídou `MyHCEService`. Dále zmiňované předávání parametrů probíhá v metodě `putExtra()`. Nakonec se služba zavolá metodou `startService()`.

Výpis 5.9: Volání služby přes Intent

```
1 val intent = Intent(this, MyHCEService::class.java)
2 intent.putExtra("UserKey", appParameters.userKey)
3 intent.putExtra("Hatu", appParameters.hatu)
4 intent.putExtra("KeyLength", appParameters.keyLengths)
5 intent.putExtra("Atu", appParameters.atu)
6 startService(intent)
```

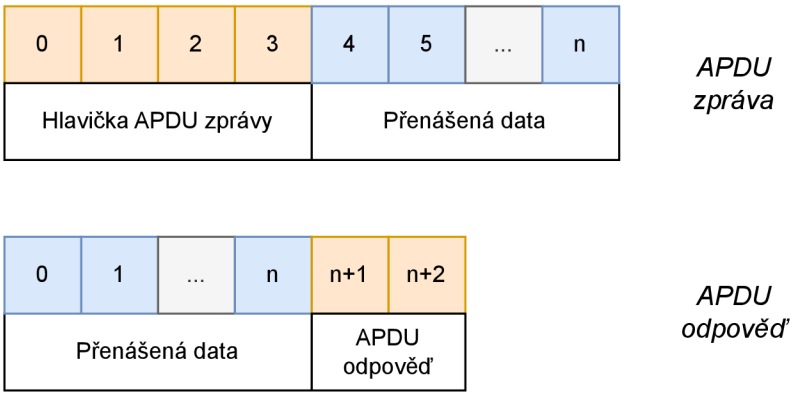
Dále je potřeba k zprovoznění komunikace skrz NFC tuhle službu deklarovat v souboru `AndroidManifest.xml`, a to značkou `<service>`. Ve značce `<service>` je také nutná značka `<meta-data>`, která obsahuje název xml soubor s AID uživatelské aplikace. Případně je také možné ve značce `<service>` deklarovat `<intent-filter>`.

Hlavní část implementace NFC tkví v metodě `proccessCommandApdu()`, která zpracovává bajtové pole přijaté od NFC čtečky. Logika v této metodě je taková, že podle stanovené hlavičky bajtového pole metoda pozná, o jakou část komunikace protokolu se jedná, a po stanovení této části potom probíhá převádění bajtového pole na proměnné a volání kryptografického jádra, obdobně jako je tomu na straně aplikace NFC serveru. Tato metoda vrací bajtové pole, které obsahuje odpověď aplikace, a také přenášené parametry dle protokolu.

APDU (Application Protocol Data Unit) zprávy jsou celkem čtyři. Pro zjednodušení implementace je struktura zpráv upravena dle obrázku 5.12. Ideální stav by byl implementovat APDU zprávy striktně podle daného APDU protokolu. Vzhledem k malému množství přenášených zpráv a jednoduší implementaci byla však strukutra těchto zpráv upravena.

Metoda `proccessCommandApdu()` má celkem 5 větví, které jsou podle přijaté hlavičky následující:

- V případě, že se přijatá APDU zpráva rovná definovanému `SELECT AID`, aplikace pošle zpět odpověď `90 00` o úspěšně navázané komunikaci.

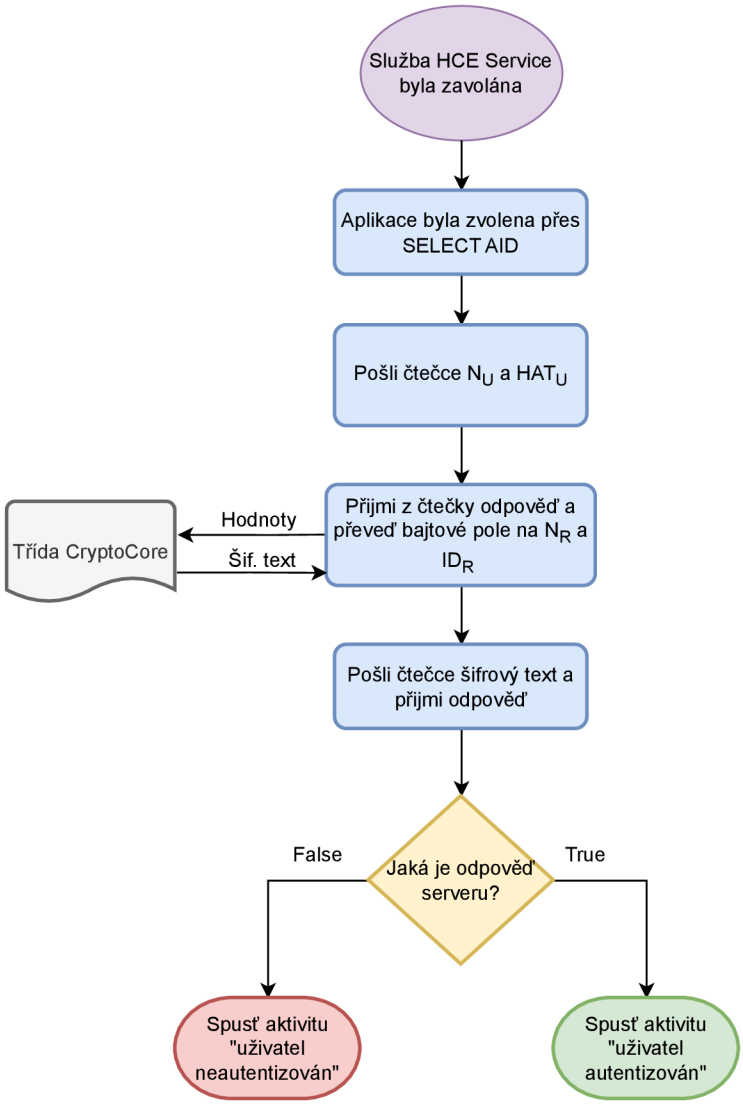


Obr. 5.12: Struktura upravených APDU zpráv a odpovědí

- V případě hlavičky se jedná o APDU zprávu oznamující začátek protokolu přes NFC. Od OBU se přijmou parametry `IDR` a `NR`. Aplikace pak posílá odpověď `90 00` a jako data posílá `NU` a `HATU`.
- V případě hlavičky se jedná o APDU zprávu oznamjucí, že OBU očekává šifrový text. Aplikace jej tedy pomocí třídy `CryptoCore` vypočítá a pošle OBU společně s odpovědí `90 00`.
- V případě hlavičky se jedná o APDU zprávu oznamující, že uživatel byl autentizován. Aplikace poté spouští novou aktivitu s tímto oznámením a pošle OBU odpověď `90 00`.
- V případě hlavičky se jedná o APDU zprávu oznamující, že uživatel nebyl autentizován. Aplikace poté spouští novou aktivitu s tímto oznámením a pošle OBU odpověď `90 00`.
- V případě, že hlavička neodpovídá žádné dříve zmiňované hlavičce, pošle aplikace OBU negativní odpověď `6D 00`.

Popsané vztahy a sled událostí ve třídě `MyHCEService` jsou uvedené na obrázku 5.13.

Část aplikace, která se stará o TCP komunikaci je obsažena ve třídě `Main Activity`. Ideální by bylo mít na tento účel zvlášť třídu, ale vzhledem k malému obsahu této implementace je tato komunikace obsažena v hlavní třídě. Technika implementace je stejná jako u Identity Provideru a TCP serveru. I zde se používá *Java Socket programmingu* a syntaxe, i když je v Kotlinu, je v podstatě identická. Co je zde oproti těmto implementacím navíc, je zobrazování výsledku implementace prostřednictvím nového okna, tak je to popsáno dříve v této kapitole. Aplikace přijímá a odesílá TCP serveru instance třídy `Cryptogram`, což jsou serializované objekty. Vývojový diagram TCP sekce na obrázku 5.14 je podobný jako u BLE a NFC sekce. Rozdíl je v posílaných objektech, kdy u BLE sekce se zapisuje a čte z charakteristik a zprávy mají podobu textového řetězce, v NFC sekci se posílají bajtová pole a v TCP sekci se posílají serializované objekty.

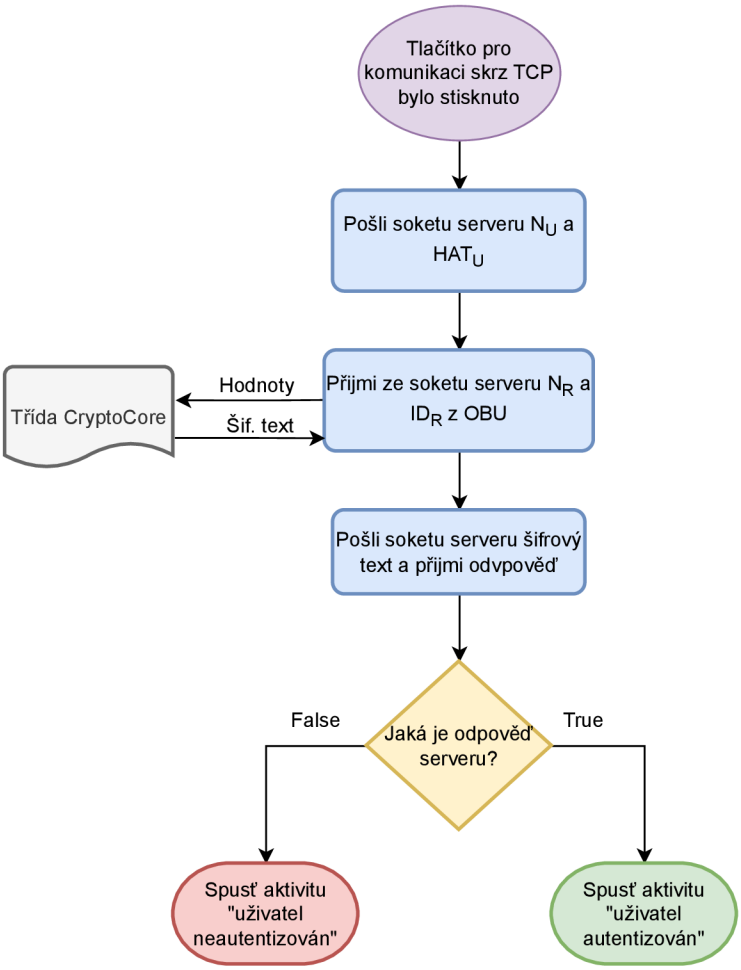


Obr. 5.13: Vývojový diagram NFC sekce mobilní aplikace

5.4 Demonstrace funkčnosti a konfigurace

Tato kapitola se věnuje demonstraci funkčnosti všech aplikací, včetně konfigurace vstupních parametrů, které ovlivňují chod těchto aplikací. Návod na spuštění těchto aplikací je k dispozici v příloze této bakalářské práce.

Jako první byla zmíněna aplikace Identity Providera. Statické proměnné jsou zde `PORT`, která určuje na jakém portu bude soket vytvořen a `keyLengths`, která určuje bitovou délku generovaných klíčů. Od této proměnné se potom odvíjí i další generované parametry. Dále jsou tu proměnné `userID` a `vehicleID` typu textového řetězce, znak **it** a celočíselná hodnota `noe`. Tyhle proměnné jsou obsahem `ATU` a lze je měnit v souboru `masterkey.txt`, z kterého aplikace tyhle parametry čte společně s hlavním klíčem `KM`. V případě, že tyhle parametry nejsou zvolené, jsou v aplikaci



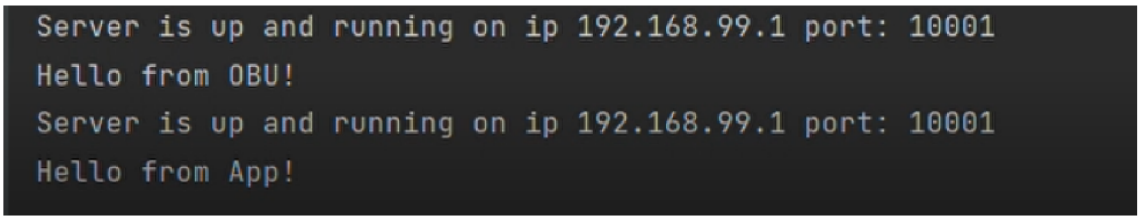
Obr. 5.14: Vývojový diagram TCP sekce mobilní aplikace

nastavené na výchozí hodnoty. Na obrázku 5.15 lze vidět vygenerované parametry aplikací a na obrázku 5.16 pak spojení a úspěšné předání klíčů a parametrů mobilní aplikaci a OBU.

```
Master key KM: 85b729ff53533d3c43cdb4cdbbab632b040cbcadb4bcf782723d091d9cfb41c
IDr: 1
Driver key KR: 667c88d8eee2d04120bb7b9a35be2f0b61844971536191235f1474cada8ab626
ATU: 00000001S000000018000O;0000020000
HATU: 8d6cff92a196329c205dbac4dc442c56ef430e4e813b0c4805a473a7832ec893
User key KU: 53f6ba622e1bf84de75a9f02bf3daec25d4bb9dde470b75dd62fdb07430f6756
```

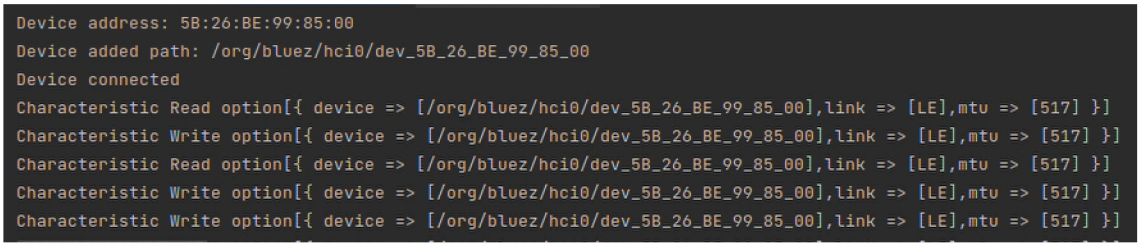
Obr. 5.15: Generování parametrů aplikací Identity Provider

Aplikace BLE peripheral má jako vstupní parametry klíč `KR` a celočíselnou hodnotu `IDR`, které jsou uchovány v textovém souboru `obuparams.txt`. Používanou bezpečnostní úroveň ovlivňuje proměnná `keyLength`. Pro snadnou práci při měření

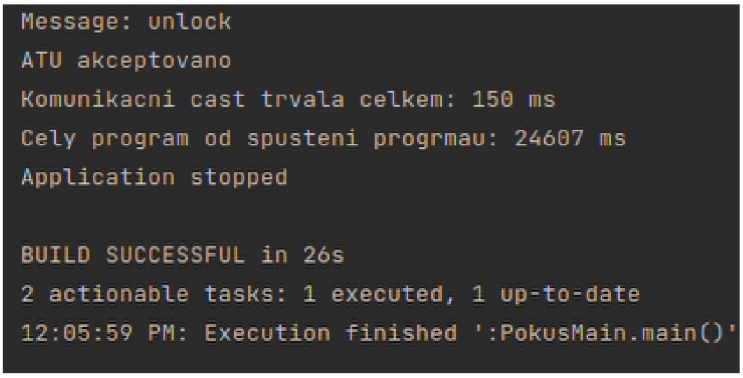


Obr. 5.16: Úspěšné spojení a předání parametrů aplikacím

jsou pro každou bezpečnostní úroveň vytvořeny samostatné textové soubory s příslušnými parametry. Výpis aplikace při úspěšné komunikaci s mobilní aplikaci je na obrázku 5.18 a zápisy a čtení z charakteristik je možné vidět na obrázku 5.17.



Obr. 5.17: Čtení a zápisy do BLE charakteristik

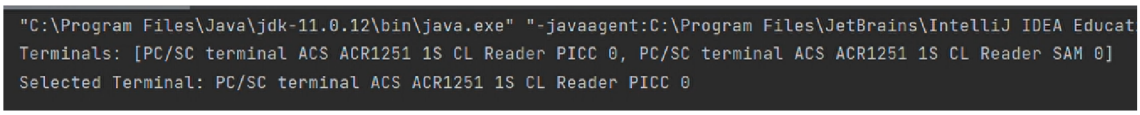


Obr. 5.18: Výsledek autentizace uživatele

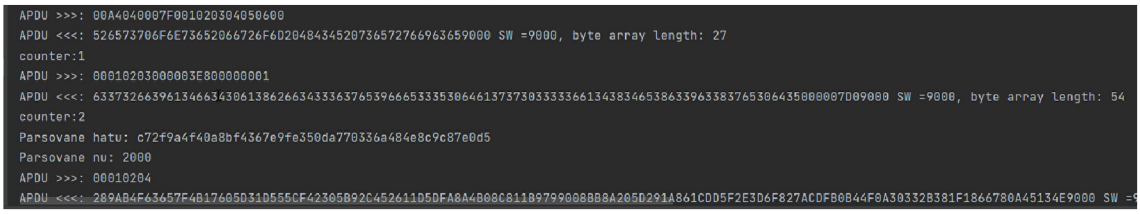
Pro aplikaci NFC serveru platí stejná konfigurace jako u BLE peripheral aplikace. Na obrázku 5.19 lze vidět situaci, kdy aplikace zaregistrovala připojenou NFC čtečku. Na obrázku 5.20 je pak vidět komunikace s mobilní aplikací.

Pro aplikaci TCP serveru také platí stejná konfigurace jako u BLE peripheral aplikace. Na obrázku 5.21 lze vidět komunikace s mobilní aplikací a navazování spojení na úrovni soketů.

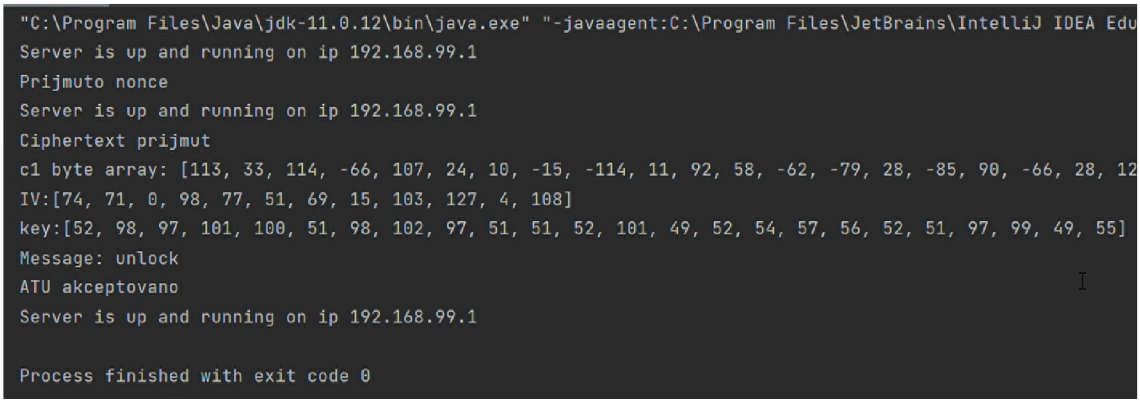
Uživatelské rozhraní aplikace lze vidět na obrázku 5.22. Jak bylo zmíněno v kapitole 5.3, aplikace si uchovává klíče a parametry v struktuře zvané *Shared Pre-*



Obr. 5.19: Zaregistrovaná NFC čtečka NFC serveru



Obr. 5.20: Komunikace prostřednictvím NFC

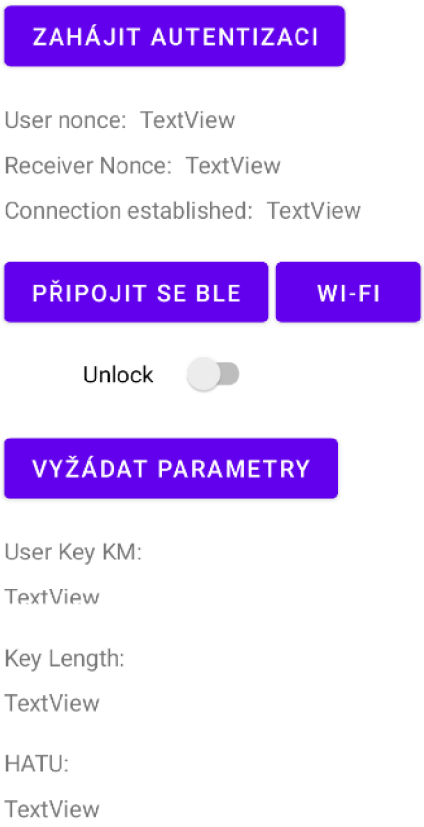


Obr. 5.21: Komunikace prostřednictvím TCP

ferences. Pro přepsání těchto parametrů si musí uživatel vyžádat nové parametry u Identity Providera stiskem na tlačítko „Vyžádat parametry“. Volba délky klíčů není v této bakalářské práci dynamická a tento výběr lze provést změnou hodnoty proměnné `KEY_LENGTH`. Přepínač umístěný uprostřed uživatelského rozhraní ovlivňuje zprávu přenášenou serverům, a to buď „unlock“ nebo „lock“. Nad tímto přepínačem se nachází dvě tlačítka, a to „Připojit se BLE“ a „Wi-Fi“, které zahájí komunikaci s příslušným serverem (BLE peripheral nebo TCP server). Po proběhlé komunikaci se zobrazí nové okno s výsledkem autentizace viz obrázek 5.23. Pro komunikaci s NFC serverem zde není žádné tlačítko, jelikož komunikaci prostřednictvím NFC se inicializuje přiložením mobilního telefonu k NFC čtečce.

5.5 Experimentální měření

Tato kapitola se věnuje experimentálnímu měření doby trvání implementovaných serverů na různých zařízeních. První verze protokolu byla testována pouze na tech-



Obr. 5.22: Uživatelské rozhraní mobilní aplikace

nologii BLE. Zde se měřil čas BLE skenu, komunikační části a autentizační části protokolu. Měření proběhlo s délkou klíčů 128 bitů ve virtuálním stroji Ubuntu. Výsledek tohoto měření ukazuje tabulka 5.5.

Druhá část měření proběhla již na všech implementovaných aplikacích serverů. Měření proběhlo na všech bezpečnostních úrovních a na dvou zařízeních, a to na Rapsberry Pi 3 (v tabulkách uvedené jako zkratka Raps.) a v případě BLE na virtuálním stroji s Ubuntu a v případě NFC a TCP na notebooku s parametry uvedenými v tabulce 5.2 .V tabulce 5.3 je výsledek měření BLE. Zde se měřila komunikační část druhé verze protokolu, tedy až po skenování BLE a po čtení / zápisu do charakteristik, kdy tyhle hodnoty jsou stejné jak v první verzi protokolu, proto nejsou měřeny.

K tomu, aby se mohly časy z NFC a TCP aplikací dát do kontextu s BLE aplikací, jsou tyhle aplikace proměřené také v komunikační části, po výměně parametrů. Tyhle časy jsou uvedené v tabulkách 5.4 a 5.5. V případě NFC je pak proměřena i doba trvání celé komunikace, což je zaznamenáno v tabulce 5.6. V případě TCP trvala celá komunikace ve všech případech přibližně shodně, a to cca 1100 ms, kdy

BLE AUTHENTICATED

Obr. 5.23: Okno s úspěšnou autentizací

Tab. 5.1: Měření BLE peripheral první verze protokolu

Kryptografické jádro + autentizace [ms]	konunikační část [ms]	scan [s]
109	191	5
69	167	5
89	273	2
72	205	5
104	213	5
110	166	5
74	175	5
93	247	4
69	212	3
76	**129**	**5**

Tab. 5.2: Parametry notebooku použitého při měření

Operační systém	Windows 10
Procesor	Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz
RAM	16 GB
Grafická karta	NVIDIA GeForce GTX 1650

nejdelší čas zabralo navázání spojení u posledního soketu.

V grafu 5.24 lze vidět porovnání časů dle protokolu. Hodnoty sloupců jsou průměrem čísel uvedených v tabulkách. Zde můžeme vidět, že BLE je při odmyšlení dlouho trvajícího BLE skenu časově nejvýhodnější. Nejdéle naopak trvá komunikace přes TCP. Na grafu 5.25 lze pozorovat rozdíly mezi trváním komunikace v závis-

Tab. 5.3: Měření BLE peripheral druhé verze protokolu

Měření	256 Raps.	192 Raps.	128 Raps.	256 VM	192 VM	128 VM
1	548	551	520	105	84	90
2	550	542	542	139	97	96
3	530	555	524	99	84	74
4	549	550	520	89	83	95
5	570	547	529	78	82	74
Průměr	549,4	549	527	102	86	85,8

Tab. 5.4: Měření TCP serveru

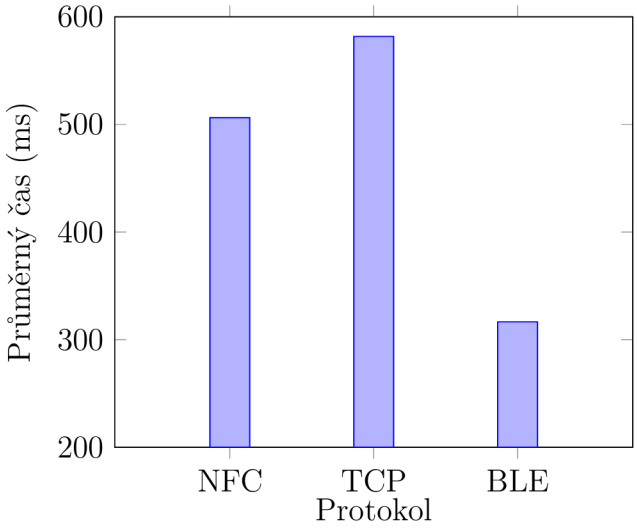
Měření	256 Raps.	192 Raps.	128 Raps.	256 PC	192 PC	128 PC
1	929	949	970	202	206	176
2	996	965	938	143	221	209
3	1062	954	963	214	171	182
4	948	1005	955	195	218	159
5	1004	930	937	173	215	264
Průměr	987,8	960,6	952,6	185,4	206,2	198

Tab. 5.5: Měření NFC serveru po výměně parametrů

Měření	256 Raps.	192 Raps.	128 Raps.	256 PC	192 PC	128 PC
1	835	702	728	338	283	295
2	816	684	664	328	286	276
3	701	703	692	270	287	319
4	686	685	689	315	334	294
5	694	712	711	298	297	267
Průměr	746,4	697,2	696,8	309,8	297,4	290,2

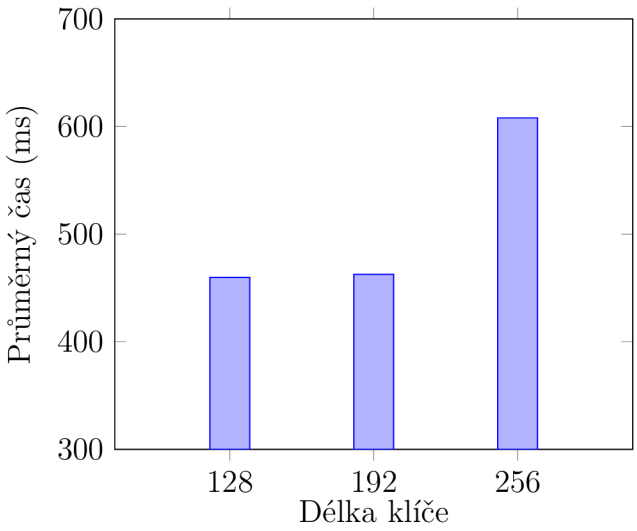
Tab. 5.6: Měření NFC serveru celé komunikace

Měření	256 Raps.	192 Raps.	128 Raps.	256 PC	192 PC	128 PC
1	1113	932	966	797	636	594
2	1068	915	878	800	610	693
3	979	952	952	594	593	753
4	947	915	927	721	627	727
5	943	950	925	603	635	569
Průměr	1010	932,8	929,6	703	620,2	667,2

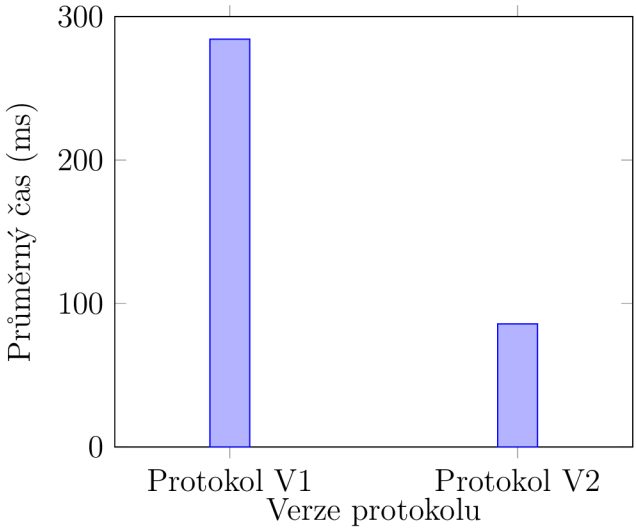


Obr. 5.24: Průměrný čas podle protokolu

losti na délky klíče. Uvedené hodnoty sloupců jsou opět průměrem čísel z tabulek. Z grafu lze vyčíst, že rozdíl mezi 128 a 192 bitovými klíči je v podstatě zanedbatelný, proto se jeví jako lepší volba z hlediska bezpečnosti používat 192 bitové klíče. U 256 bitových klíčů už délka trvání vzroste o cca třetinu. V grafu 5.26 je porovnání první verze protokolu s druhou verzí protokolu. Jelikož meření první verze protokolu proběhlo pouze s 128 bitovými klíči na virtuálním stroji, hodnota sloupce druhé verze protokolu je odvozena také pouze od průměrného času délky trvání komunikačního protokolu s 128 bitovými klíči na virtuálním stroji.



Obr. 5.25: Průměrný čas podle délky klíče



Obr. 5.26: Průměrný čas podle verze protokolu

Závěr

V rámci závěrečné práce byl navrhnut a implementován přístupový systém pro aplikaci Carsharing. Kromě technologie BLE, byla také implementována komunikace využívající technologii NFC a Wi-Fi. Implementovaný protokol je ve dvou verzích, přičemž všechny tři implementované technologie fungují na druhé verzi protokolu a existují pro ně tři bezpečnostní úrovně, a to 128, 192 a 256 bitové délky klíčů. Všechny bezpečnostní úrovně a implementované technologie byly změřeny. V mobilní aplikaci bylo vytvořeno jednoduché uživatelské rozhraní a spouštění nových oken. Při implementaci BLE byla použita knihovna `BLESSED` na straně mobilní aplikace a na straně OBU byla použita knihovna `ble-java`. Knihovna `BLESSED` značně ulehčila práci s BLE a program využívající tuhle knihovnu je přehledně postaven na přetěžování metod zpětného volání. Při implementaci NFC byla použita knihovna `javax.smardcardio`. NFC je zde v režimu emulace karet. Pro TCP server a Identity Providera bylo zvoleno *Java Socket Programmingu*. Aplikace Identity Provideru generuje parametry ve třech zmiňovaných bezpečnostních úrovních, přičemž je možné volit vstupní parametry pro autentizační token uživatele.

V teoretické části byly analyzovány přístupové systémy, kde bylo naznačeno jejich fungování, a také použitá kryptografická primitiva. Dále pak teoretický rozbor technologie BLE a NFC, včetně jejich klíčových pojmů. V neposlední řadě také vývoj mobilních aplikací, kde byl analyzován proces jejich vývoje, a také programovací jazyk Kotlin.

Výsledky experimentálního měření se v rámci kompletní komunikace pohybují do 1 sekundy čili se dají označit za uspokojivé. Ve vzájemném porovnání se zdá být BLE nejrychlejší technologií, ovšem je zde překážka skenování, které trvá v průměru 5 sekund. S přihlédnutím na čas skenování u BLE se jeví použití NFC jako nejlepší varianta, kdy kompletní komunikace (navázání spojení, výměna parametrů, komunikační část protokolu) trvá v průměru 700 milisekund. Použití Wi-Fi se v tomto případě užití jeví jako nejpomalejší varianta.

Pokračování téhle práce by mohlo tkvět v rozšiřování interaktivity aplikací. Na straně mobilní aplikace by optimalizace spočívala v propracovanějším uživatelském rozhraní, kde by mohlo být okno s nastavením, s jakou verzí protokolu je zamýšleno komunikovat, na jaké bezpečnostní úrovni a s jakou komunikační technologií. Na straně Identity Provideru by pak navazující práce mohla spočívat ve vytvoření grafického uživatelského rozhraní, kde by se volitelné parametry mohly volit interaktivně, například délka klíčů nebo vstupní parametry pro autentizační token uživatele.

Literatura

[1] Perboli, G., Caroleo, B. a Musso, S. Car- sharing: Current and potential members behavior analysis after the introduction of the service. *IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*. 2017, (2), 771–776 [cit. 2022-11-23]. Dostupné z: doi:10.1109/ COMPSAC.2017.82

[2] George, C. a Priya Uteng, T. Car sharing. *International Encyclopedia of Transportation* [online]. 142–146 [cit. 2022-11-23]. Dostupné z: doi:doi:10.1016/b978-0-08-102671-7.10632-3

[3] DUNCAN, M. *The cost saving potential of carsharing in a US context10.1007/s11116-010-9304-y* [online]. 2010 [cit. 2022-11-24]. Dostupné z: doi:10.1007/s11116-010-9304-y

[4] SYMEONIDIS, Iraklis, A. MUSTAFA a Bart PRENEEL. Keyless Car Sharing System: A Security and Privacy Analysis. In: *A. KU Leuven, ESAT-COSIC a iMinds* [online]. [cit. 2022-11-24]. Dostupné z: `https://www.esat.kuleuven.be/cosic/publications/article-2671.pdf`

[5] ARM, J., P. DVORSKÝ, P. FIEDLER, C. FALCOU a J. ORLICKÝ. Safety and Security of the Car-Sharing System. *IFAC PapersOnLine* [online]. 2022(55-4), 121–126 [cit. 2022-11-24]. Dostupné z: `www.sciencedirect.com`

[6] MADHUSUDAN, Akash, Iraklis SYMEONIDIS, Mustafa A. MUSTAFA, Ren ZHANG a Bart PRENEEL. *SC2Share: Smart Contract for Secure Car Sharing* [online]. 2019 [cit. 2022-11-25]. Dostupné z: doi:10.5220/0007703601630171

[7] GROZA, B., T. ANDREICA, A. BERDICH, P. -S. MURVAY a E. H. GURBAN. PRESTvO: PRivacy Enabled Smartphone Based Access to Vehicle On-Board Units. IEEE Access [online]. 2020, (8), 119105-119122 [cit. 2022-11-25]. Dostupné z: doi:10.1109/ACCESS.2020.3003574

[8] SYMEONIDIS, Iraklis, Dragos ROTARU, Mustafa A. MUSTAFA, Bart MENNINK, Bart PRENEEL a Panos PAPADIMITRATOS. *HERMES: Scalable, Secure, and Privacy-Enhancing Vehicular Sharing-Access System* [online]. [cit. 2023-02-03]. Dostupné z: doi:10.1109/JIOT.2021.3094930

[9] HERNANDEZ, Jesus, Shahryar DIDARZADEH, Victor PASCUAL a Alvaro ARRUE. *ISHARE — Car sharing concept vehicle* [online]. Listopad 2013 [cit. 2023-02-04]. Dostupné z: doi:10.1109/EVS.2013.6914830

[10] *Applus+ Group IDIADA* [online]. [cit. 2023-02-04]. Dostupné z: `https://www.applusidiada.com/global/en/`

[11] SYMEONIDIS, Iraklis, Abdelrahaman ALY, Mustafa A. MUSTAFA, Bart MENNINK, Siemen DHOOGHE a Bart PREENEL. SePCAR: A Secure and Privacy-Enhancing Protocol for Car Access Provision (Full Version). *European Symposium on Research in Computer Security (ESORICS 2017)* [online]. [cit. 2023-02-04].

[12] HERON, Simon. Advanced Encryption Standard (AES). *Network Security* [online]. 2009, (12), 8-12 [cit. 2023-05-15]. Dostupné z: doi:10.1016/S1353-4858(10)70006-4

[13] Secure Hash Algorithms. *Brilliant.org* [online]. [cit. 2023-05-16]. Dostupné z: `https://brilliant.org/wiki/secure-hashing-algorithms/`

[14] Keccak specifications summary. *Team Keccak* [online]. [cit. 2023-05-16]. Dostupné z: `https://keccak.team/keccak_specs_summary.html`

[15] ZEMAN, Václav. Hašovací funkce, jednorázový podpis. Prezentace prezentována na: [Kurz Aplikovaná kryptografie, Fakulta elektrotechniky a komunikačních technologií VUT v Brně; 2020; Brno, Česká republika.]

[16] *Set up for Android Development* [online]. [cit. 2023-02-17]. Dostupné z: `https://source.android.com/docs/setup/about`

[17] *Android - Overview* [online]. [cit. 2023-02-17]. Dostupné z: `https://www.tutorialspoint.com/android/android_overview.htm`

[18] *Codenames, Tags, and Build Numbers: Android documentation* [online]. [cit. 2023-02-18]. Dostupné z: `https://source.android.com/docs/setup/about/build-numbers`

[19] *Android ABIs: Android documentation* [online]. [cit. 2023-02-18]. Dostupné z: `https://developer.android.com/ndk/guides/abis`

[20] Mobile app development process: Step-by-Step Guide for 2022. *Invonto* [online]. [cit. 2023-02-27]. Dostupné z: `https://www.invonto.com/insights/mobile-app-development-process/`

[21] Mobile App Development Process: Ultimate Guide To Build an App: 7 Key Steps of The Mobile App Development Process. *Velvetech* [online]. [cit. 2023-02-27]. Dostupné z: `https://www.velvetech.com/blog/mobile-app-development-process/`

[22] How much does it cost to make an app. *Invonto* [online]. [cit. 2023-02-27]. Dostupné z: `https://www.invonto.com/insights/how-much-does-it-cost-to-make-an-app/`

[23] Mobile operating systems' market share worldwide from 1st quarter 2009 to 4th quarter 2022. *Statista* [online]. [cit. 2023-02-27]. Dostupné z: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

[24] Develop apps for iOS: Learn the basics of Xcode, SwiftUI, and UIKit to create compelling iOS apps. *IOS App Dev Tutorials* [online]. [cit. 2023-02-27]. Dostupné z: https://developer.apple.com/tutorials/app-dev-training#swiftui-essentials

[25] The ultimate guide to User Flow Diagram. *Bootcamp* [online]. [cit. 2023-03-03]. Dostupné z: https://bootcamp.uxdesign.cc/the-ultimate-guide-to-user-flow-diagram-b108d7de10d

[26] *Build better apps faster with Jetpack Compose* [online]. [cit. 2023-03-27]. Dostupné z: https://developer.android.com/jetpack/compose

[27] Co je to porušení zabezpečení osobních údajů a co by v takovém případě mělo být učiněno?. EvropskÃąkomise [online]. [cit. 2023-03-27]. Dostupné z: https://commission.europa.eu/law/law-topic/data-protection/reform/rules-business-and-organisations/obligations/what-data-breach-and-what-do-we-have-do-case-data-breach_cs

[28] *Browser Stack* [online]. [cit. 2023-03-27]. Dostupné z: https://www.browserstack.com/

[29] *Apple App Store vs Google Play Store (2023 Comparison)* [online]. [cit. 2023-03-28]. Dostupné z: https://cybercrew.uk/software/app-store-vs-play-store/

[30] *App Store Review Guidelines* [online]. [cit. 2023-03-28]. Dostupné z: https://developer.apple.com/app-store/review/guidelines/#safety

[31] *Alternative distribution options* [online]. [cit. 2023-03-28]. Dostupné z: https://developer.android.com/distribute/marketing-tools/alternative-distribution

[32] *Apple Developer Enterprise Program* [online]. [cit. 2023-03-28]. Dostupné z: https://developer.apple.com/programs/enterprise/

[33] Develop Android apps with Kotlin [online]. [cit. 2023-02-20]. Dostupné z: https://developer.android.com/kotlin

[34] *Intents and Intent Filters* [online]. [cit. 2023-03-28]. Dostupné z: https://developer.android.com/guide/components/intents-filters

[35] BLUETOOTH, S. I. G. Specification of the bluetooth system-covered core package version: 4.0. Bluetooth Special Interest Group, 2010.

[36] Bluetooth Technology Overview: Learn About Bluetooth. Bluetooth [online]. [cit. 2022-11-22]. Dostupné z: <https://www.bluetooth.com/learn-about-bluetooth/tech-overview/>

[37] What is pairing?: Help Guide. *© 2013 Sony Corporation* [online]. [cit. 2022-11-22]. Dostupné z: <https://helpguide.sony.net/speaker/srs-bts50/v1/en/contents/02/02/01/01.html>

[38] BLE Pairing and Bonding. Technical Notes [online]. [cit. 2022-11-22]. Dostupné z: <https://technotes.kynetics.com/2018/BLE_Pairing_and_bonding/>

[39] Specifications. *Bluetooth LE Wiki* © [online]. [cit. 2022-11-11]. Dostupné z: <https://bluetoothle.wiki/specifications>

[40] The Bluetooth® Low Energy Primer. *Bluetooth* [online]. [cit. 2022-11-22]. Dostupné z: <https://www.bluetooth.com/wp-content/uploads/2022/05/Bluetooth_LE_Primer_Paper.pdf>

[41] Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. *National Library of Medicine* [online]. [cit. 2022-11-22]. Dostupné z: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3478807/>

[42] *About Near Field Communication* [online]. [cit. 2023-04-16]. Dostupné z: http://nearfieldcommunication.org/about-nfc.html

[43] *NFC Forum Technical Specifications* [online]. NFC Forum [cit. 2023-04-16]. Dostupné z: https://web.archive.org/web/20120804071028/http://www.nfc-forum.org/specs/spec_list/#protts

[44] NFC Data Exchange Format (NDEF). *Go To Tags: Learn* [online]. [cit. 2023-04-17]. Dostupné z: https://learn.gototags.com/nfc/ndef#structure

[45] COSKUN, Vedat, Busra OZDENIZCI a Kerem OK. The Survey on Near Field Communication. *Sensors 2015* [online]. Červen 2015, (15(6), 13348-13405 [cit. 2023-04-16]. Dostupné z: doi:10.3390/s150613348

[46] ALRAWAIS, Arwa. Security Issues in Near Field Communications (NFC). *International Journal of Advanced Computer Science and Applications*, 2020, 11.11.

[47] *Proxmark* [online]. [cit. 2023-04-18]. Dostupné z: `https://proxmark.com/`

[48] ARD RFID BLOCKER (PACK OF 2) - NFC JAMMER. *Lab 401* [online]. [cit. 2023-04-18]. Dostupné z: `https://lab401.com/products/rfid-blocker-nfc-jammer`

[49] DZURENDA, Petr, Lukáš MALINA, Pavel LOUTOCKÝ, František KASL, Pavel KRIŠTOF a Jan HAJNÝ. *Towards to Lightweight and Secure Access Control Systems for Car-Sharing Services* [online]. 2022 [cit. 2022-11-26]. Dostupné z: doi:10.1109/ICUMT57764.2022.9943522

[50] A Java BLE (bluetooth 4.0) library for bluez. *GitHub* [online]. [cit. 2022-12-04]. Dostupné z: `https://github.com/tongo/ble-java`

[51] BLESSED: BLESSED, a Bluetooth Low Energy (BLE) library for Android. *Github* [online]. [cit. 2022-12-04]. Dostupné z: `https://github.com/weliem/blessed-android`

Seznam symbolů a zkratek

Wi-Fi	Wireless Fidelity
NFC	Near Field Communication
BLE	Bluetooth Low Energy
OBU	On Board Unit
SSL	Secure Sockets Layer
TLS	Transport Layer Security
MPC	Multiparty Computation
AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
MAC	Message Authentication Code
PKI	Public key infrastructure
LTE	Long Term Evolution
RSA	Rivest Shamir Adleman
CTR	Counter Mode
SHA	Secure Hash Algorithm
ECDH	Eliptic Curve Diffie-Hellman
GCM	Galois/Counter Mode
DES	Data Encryption Standard
SDK	Software Development Kitu
AOSP	Android Open Source Project
API	Application Programming Interface
ABI	Application Binnary Interface
UI	User Interface
UX	User Experience

XML	Extensible Markup Language
SIG	Special Interest Group
LE PHY	Low Energy Physical
EDR	Enhanced Data Rate
DPSK	Differential Phase-Shift Keying
GFSK	Gaussian Frequency-Shift Keying
DPQSK	Differential Quadrature Phase-Shift Keying
L2CAP	Logical Link Control and Adaptation Protocol
ATT	Attribute Protoco
GATT	Generic Attribute PrOfile
SMP	Security Manager Protocol
GAP	Generic Access Profile
HCI	Host Controller Interface
RFID	Radio-Frequency Identification
NDEF	NFC Data Exchange Format
LLCP	Logical Link Control Protocol
OSI	Open Systems Interconnection
ISO	International Organization for Standardization
DOS	Denial of Service
ATU	Autentizační token uživatele
BCD	Binary Coded Decimal
TCP	Transmission Control Protocol
UUID	Universally Unique Identifier
AID	Application Identifier
APDU	Application Protocol Data Unit

A Obsah elektronické přílohy

V příloze lze najít .zip soubory s kompletními projekty naprogramovaných aplikací. Dále také příloha obsahuje .jar soubory implementovaných serverů.

Spuštění BLE serveru je možné jen na zařízení s linuxovým jádrem. Pro jeho spuštění je potřeba provést instalace viz [50]. Pro všechny soubory .jar platí, že se musí spouštět ve stejném adresáři jako jsou textové soubory `obuparams.txt`. Bez nich aplikace nezná klíče potřebné k šifrované komunikaci. Po vyžádání nových klíčů od mobilní aplikace od Identity Providera je potom nutné manuálně přepsat nové klíče v těchto souborech. Spuštění BLE serveru lze potom spustit následujícím příkazem:

```
1 java -cp BLE<128/192/256>.jar:libmatthew-java-0.8.jar:
    dbus-java-2.7.jar:unix.jar PokusMain
```

Spuštění ostatních dvou serverů je potom možné následujícím příkazem:

```
1 java -jar TCP<128/192/256>.jar
```

nebo

```
1 java -jar NFC<128/256>.jar
```

U TCP serveru je potom nutné zjistit IP adresu, na kterém server běží a podle ní upravit příslušnou proměnnou `val ip` v kódu mobilní aplikace. Pro spuštění NFC serveru je potřeba mít připojenou k zařízení NFC čtečku. V případě spouštění na linuxovém zařízení je potřeba nainstalovat ovladače k příslušné čtečce. V případě volby délky klíčů 192 bitů pro NFC je potřeba vygenerovat nový jar soubor, který nemohl být vzhledem k omezené kapacitě velikosti příloh obsažen.

Pro spuštění mobilní aplikace je potřeba stáhnout přiložený apk soubor na zařízení se systémem Android. Po klepnutí na apk soubor je potřeba povolit instalaci z neznámých zdrojů a dále postupovat podle pokynů na obrazovce pro dokončení instalace. Po dokončení instalace se provede spuštění mobilní aplikace klepnutím na ikonu aplikace.

```
/ ..... kořenový adresář přiloženého archivu
└── Jar soubory ..... jar soubory serverů ke spuštění
    ├── BLE128.jar
    ├── BLE192.jar
    ├── BLE256.jar
    ├── NFC128.jar
    ├── NFC256.jar
    ├── TCP128.jar
    └── TCP192.jar
```

- TCP256.jar
- obuparams.txt
- obuparams128.txt
- obuparams256.txt
- dbus-java-2.7.jar ..... knihovna potřebná pro spuštění BLE serveru
- libmatthew-java-0.8.jar ..... knihovna potřebná pro spuštění BLE serveru
- unix.jar ..... knihovna potřebná pro spuštění BLE serveru

- BleServer_v1.zip ..... zdrojové kódy BLE Serveru pro první verzi protokolu
- BleServer_V2_final.zip zdrojové kódy BLE Serveru pro druhou verzi protokolu
- IdentityProvider.zip ..... zdrojové kódy Identity Provideru
- Mobile_app.zip ..... zdrojové kódy uživatelské mobilní aplikace
- NFCServer.zip ..... zdrojové kódy NFC Serveru
- TCP_terminal_app.zip ..... zdrojové kódy TCP Serveru
- mobile_app.apk ..... apk soubor k instalaci mobilní aplikace