VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Kristián Klasovitý

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

BEZPEČNOSTNÍ SYSTÉM VYUŽÍVAJÍCÍ CHYTRÁ ZAŘÍZENÍ
SECURITY SYSTEM USING SMART DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Kristián Klasovitý

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

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Kristián Klasovitý ***ID:*** 196068
Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Bezpečnostní systém využívající chytrá zařízení

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s vývojem webových aplikací, Android aplikací a s komunikačními technologiemi NFC a BLE. Na tomto základě implementujte bezpečnostní systém využívající chytrá zařízení, jako jsou chytré telefony, hodinky či čipové karty. Systém bude založený na využití moderních komunikačních technologií NFC, BLE či WIFI. Přístupový systém bude dále poskytovat autentizaci komunikujících stran, utajení, autentičnost a integritu přenášených dat.

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. 2020-09-14]. Dostupné z: https://developer.android.com/

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

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

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

ABSTRAKT

Práce se věnuje implementaci autentizačního systému, kdy se uživatelé přihlašují pomocí Android mobilního telefonu společně s chytrými hodinkami, které používají operační systém WearOS. Systém využívá protokolu s nulovou znalostí, který je založen na Schnorrovu protokolu a umožňuje autentizovat server i několik zařízení uživatele. Systém využívá komunikační rozhraní NFC, Bluetooth, technologii HCE a je implementován v jazyce Java. Pro výpočty na mobilních zařízeních je zde využito nativních funkcí jazyka C.

KLÍČOVÁ SLOVA

Android, autentizace, Schnorrův protokol, WearOS, nativní funkce, NFC, Bluetooth, HCE, Java, důkaz s nulovou znalostí, eliptické křivky

ABSTRACT

This thesis deals with the implementation of authentication system, where users can use theirs Android mobile phone and WearOS smart watch to log in. This system uses a zero-knowledge protocol based on Schnorr protocol and it provides authentication of a server and of multiple devices of the user. The system uses technologies NFC, Bluetooth, HCE and it is implemented in Java. For some of the computation on mobile devices the system uses native functions written in C.

KEYWORDS

Android, authentication, Schnorr protocol, WearOS, native functions, NFC, Bluetooth, HCE, Java, zero-knowledge proof, elliptic curves

KLASOVITÝ, Kristián. *Bezpečnostní systém využívající chytrá zařízení*. Brno, 2021, 69 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Petr Dzurenda, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Bezpečnostní systém využívající chytrá zařízení“ jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské 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

PODĚKOVÁNÍ

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

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Dnešní společnost si velice rychle zvykla používat řadu elektronických zařízení k mnoha různým účelům. Ať už se jedná o mobilní telefony, počítače, tablety, chytré hodinky nebo čipové karty, to vše se stalo běžnou součástí našich životů. Možnosti využití těchto přístrojů jsou téměř neomezené. Dříve bylo kupříkladu možné platit pouze fyzickými penězi, pak přišly na řadu platební karty a dnes je možné zaplatit svou útratu přiložením mobilního telefonu či chytrých hodinek k platebnímu terminálu. Úkolem těchto elektronických zařízení je mimo jiné usnadnit nám život a ušetřit čas.

Všichni ví, jak zdlouhavé může být přihlašování do různých služeb na internetu. A při vědomí toho, co by mělo splňovat silné heslo a že by mělo být u každé služby zadáno heslo jiné, se nabízí otázka, jak zadávání hesel zrychlit a zjednodušit tak, aby bylo zároveň bezpečné. Proč se v dnešní době zdržovat složitým vypisováním hesel a dalších přihlašovacích údajů, když by k tomuto účelu mohlo stačit přiložení mobilního telefonu?

Takováto autentizace, pokud bude zpracována kvalitně, například dvoufázově, by mohla být rychlá a bezpečná. Nezřídka slýcháme, že lidé, kteří pracují na důležitých postech v bance či elektrárně, používají heslo 12345. S dvoufázovým způsobem autentizace by mohly problémy se slabými hesly zcela vymizet. S řešením této problematiky by mohla pomoci technologie *Near Field Communication* (NFC), která je v současné době využívána zejména k placení v obchodech nebo pro přístup do budov pomocí čipových karet.

Cílem této bakalářské práce proto bude vyvinout systém, který by rychlou a bezpečnou autentizaci (např. do různých aplikací a webů) pomocí chytrých zařízení umožňoval. Uživatel by se mohl autentizovat přiložením mobilního telefonu se systémem Android, propojeného s chytrými hodinkami, ke čtečce karet. Hodinky budou vybaveny operačním systémem WearOS (Android pro chytré hodinky).

V bakalářské práci bude implementována komunikace přes NFC mezi počítačem a mobilním telefonem a také komunikace přes Bluetooth mezi telefonem a chytrými hodinkami. Za pomoci těchto komunikačních rozhraní bude implementován protokol zajišťující autentizaci uživatele pomocí důkazu s nulovou znalostí.

1 Kryptografické funkce a protokoly

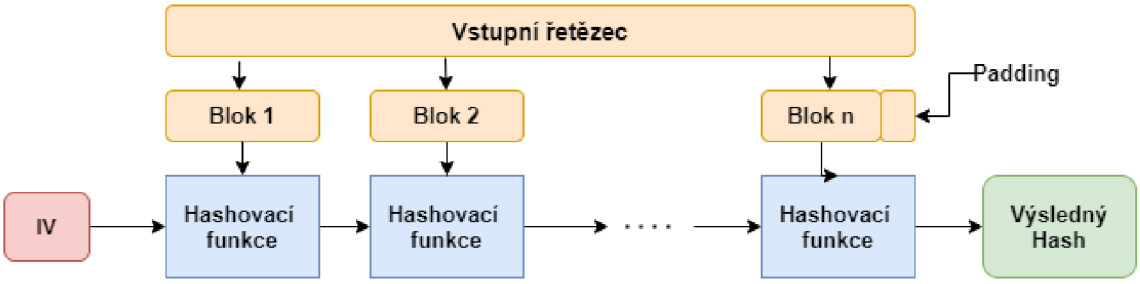
Tato kapitola se věnuje přiblížení některých kryptografických funkcí a protokolů, které se v systému bezpečného přístupu používají.

1.1 Hashovací funkce

Hashovací funkce jsou jedním ze základních nástrojů kryptografie. Hashovací funkce je matematická funkce, která dokáže převést libovolně dlouhou vstupní zprávu na řetězec znaků o fixní délce, která je zpravidla dána typem hashovací funkce. Výstupní řetězec znaků se nazývá hash (otisk), a je většinou mnohonásobně kratší, než řetězec vstupní. Hashovací funkce funguje takovým způsobem, že při změně byť jednoho znaku vstupní zprávy bude výsledný otisk naprosto jiný. Vypočítání otisku zprávy je velmi rychlá operace, proto se velice často používá v kryptografických systémech [1].

První důležitou vlastností hashovací funkce je jednosměrnost. Jednosměrnost znamená, že nelze najít inverzní algoritmus k hashovací funkci, tedy pokud známe otisk zprávy, nemůžeme z něj vypočítat původní zprávu. Další vlastností každé bezpečné hashovací funkce je bezkoliznost. To znamená, že je výpočetně náročné, a v reálném čase nemožné, najít ke zprávě x s otiskem h(x) druhou zprávu y tak, aby obě zprávy měly stejný hash, tedy h(x)=h(y). Stejně tak by mělo být výpočetně náročné najít dvě libovolné zprávy se se stejným otiskem.

Prakticky hashovací algoritmus funguje tak, že vstupní zpráva se rozdělí na bloky určité délky a každý takový blok postupně vstupuje do nějaké matematické hashovací funkce. Tato hashovací funkce má dva vstupy, kde první je hash předchozího bloku (nebo v případě prvního bloku inicializační vektor) a druhým vstupem je aktuální blok zprávy. Poslední hash v takovém algoritmu je potom hash celé zprávy [2]. Schéma hashovacího algoritmu je zobrazeno na obrázku 1.1.



Obr. 1.1: Schéma hashovacího algoritmu.

Mezi nejznámější hashovací algoritmy patří rodina algoritmů *Secure Hash Algorithm* (SHA). Celkem existují 4 verze tohoto algoritmu, SHA-0, SHA-1, SHA-2, SHA-3. První dva algoritmy vytváří otisk s délkou 160 bitů. Oba tyto algoritmy se dnes nepovažují za bezpečné, jelikož byl nalezen útok, při kterém je i u novější verze SHA-1 možné v reálném čase najít kolize. Algoritmus SHA-2 existuje v několika verzích a umí tvořit hashe o délkách 224, 256, 384, nebo 512 bitů. SHA-2 se považuje za bezpečný a v současné době je to jeden z nejpoužívanějších hashovacích algoritmů. Nejnovější z těchno algoritmů je SHA-3. SHA-3 funguje na jiném principu než předchozí hashovací funkce, a to na principu "houby"(sponge function). Verze SHA-3 zatím není tolik rozšířená jako její předchůdce, ačkoli byla zvolena jako doporučený hashovací algoritmus již v roce 2015. Hlavními důvody je, že SHA-2 se stále považuje za bezpečnou a také to, že v době přechodu ze SHA-1 většina hardwaru a softwaru SHA-3 nepodporovala, a tak většina systémů přešla na SHA-2 [3].

V praxi se hashování používá například při elektronickém podpisu, kdy podepisovat celý soubor by bylo časově náročné, proto se podepisuje pouze otisk souboru, nebo například u ukládání hesel, kde by bylo nebezpečné ukládat hesla v databázi v otevřené podobě, a tak se hesla ukládají v podobě hashe.

1.2 Symetrické šifry

Symetrická šifra je algoritmus, kde se k šifrování i dešifrování používá stejný klíč. Tento klíč je potřeba znát na obou stranách šifrované komunikace, proto je před zahájením šifrované komunikace nutné si tento klíč nějakým způsobem vyměnit. Jelikož je posílání klíče otevřeným kanálem nebezpečné a doručení klíče na přenosném médiu nepraktické, používají se v praxi protokoly pro ustálení společného klíče, jako je Diffieho–Hellmanův protokol distribuce klíčů. Pomocí těchto protokolů je možné na nezabezpečeném kanále vytvořit společný klíč dvou stran bez jeho vyzrazení.

Existují dva základní druhy symetrických šifer, a to blokové šifry a proudové šifry. Proudové šifry zpracovávají otevřený text po jednotlivých bitech, kdy tento proud je většinou kombinován s pseudonáhodným proudem bitů funkcí XOR, tento pseudonáhodný proud většinou vzniká kombinací klíče a šifrovacího algoritmu. Blokové šifry rozdělí otevřený text na bloky stejné délky a šifrují jednotlivé bloky. Základní blokové šifry šifrují každý blok stejně, tudíž nepotřebují paměť k uchovávání minulých stavů. Naopak u proudových šifer se transformace šifrovací funkce může měnit v závislosti na stavu, ve kterém se právě nachází, proto se také někdy nazývají stavové šifry. Pro odstranění problému blokových šifer, kdy je každý blok šifrován stejně, a tedy dva stejné bloky budou zašifrovány na stejný zašifrovaný blok, se u blokových šifer používají různé operační módy. Tyto módy do algoritmu zanesou náhodnost například přidáním inicializačního vektoru a závislosti jednotlivých bloků na sobě.

Díky tomu pak ze dvou stejných bloků nevznikne stejný šifrovaný blok a blokové šifry se poté více podobají šifrám proudovým [4].

Symetrické šifry se používají tam, kde je potřeba šifrovat větší množství dat, například pro šifrování utajené komunikace dvou stran, nebo šifrování databází, kde by šifrování asymetrickými šiframi zabralo příliš dlouhou dobu.

Algoritmus DES

Data Encryption Standard (DES) je symetrická bloková šifra, vytvořená v 70. letech společností IBM. V roce 1976 se DES stal standardem pro symetrické šifrování, tím byl až do roku 2002, kdy přestal být považován za bezpečný. DES používá klíče délky 64 bitů, ovšem pouze 56 bitů je efektivních, 8 bitů je pouze kontrolních, šifruje bloky délky 64 bitů a šifrování bloku se provádí v 16 iteracích. Právě krátký klíč je největší slabinou šifry, další slabinou je také to, že se při šifrování blok rozděluje na dvě 32 bitové části.

Pro zvýšení bezpečnosti šifry byl později vytvořen algoritmus Triple DES (3DES), ten vychází z algoritmu DES, ale používá klíč o délce 168 bitů, což odpovídá třem klíčům standardní šifry DES. Proces šifrování šifrou 3DES začíná rozdělením klíče na 3 klíče o délce 56 bitů, poté je otevřený text zašifrován prvním klíčem, dešifrován druhým klíčem a nakonec zašifrován klíčem třetím. Někdy se také používá varianta s klíčem o délce 112 bitů, kdy první i třetí klíč jsou shodné [5].

Algoritmus AES

Advanced Encryption Standard (AES) je dnes jedna z nejpoužívanějších symetrických blokových šifer. Původní název šifry byl Rijndael, doporučenou šifrou pro symetrické šifrování se stala již v roce 2002 [6]. AES používá klíče délky 128, 192, nebo 256 bitů a pracuje s bloky o 128 bitech. Jednotlivé délky klíčů také určují počet iterací šifrovacího algoritmu. Pro klíč délky 128 bitů je to 10 iterací, pro 192 bitový klíč 12 iterací a pro 256 bitový klíč 14 iterací. Každá iterace algoritmu se skládá ze čtyř kroků, těmi jsou: záměna bajtů, prohození řádků, kombinování sloupců a přidání klíče iterace [7].

Existuje několik provozních módů, které AES používá. Nejjednodušším módem je *Electronic Codebook* (ECB). V tomto módu se každý blok šifruje zvlášť, a tedy se stejným klíčem budou dva stejné bloky šifrované na stejný šifrovaný blok, proto tento mód není v praxi doporučené používat. Druhým módem, který je již v praxi použitelný, je mód *Cipher Block Chaining* (CBC). U tohoto módu se první blok nejdříve xoruje s inicializačním vektorem a až poté se šifruje. Každý následující blok se pak xoruje s předchozím zašifrovaným blokem. Dalšími operačními módy, které AES používá, jsou *Cipher FeedBack* (CFB), *Output FeedBack* (OFB) a *Counter*

Mode (CTR). Speciálním módem je mód *Galois/Counter Mode* (GCM), který kromě šifrování zajišťuje i autentizaci [8].

1.3 Asymetrická kryptografie

Asymetrická kryptografie odstraňuje hlavní problém symetrické kryptografie, kterým je distribuce tajných klíčů. V asymetrické kryptografii se totiž používají dva různé klíče pro šifrování a dešifrování. Ovšem asymetrická kryptografie neobsahuje pouze algoritmy pro šifrování, ale také algoritmy pro distribuci klíčů nebo pro elektronické podpisy.

Základem asymetrické kryptografie je dvojce klíčů, které se nazývají veřejný a soukromý klíč. Veřejný klíč bývá specifickým způsobem spočítán ze soukromého klíče. Při šifrování se používá veřejný klíč a při dešifrování klíč soukromý, proto stačí, když uživatel zveřejní svůj veřejný klíč. Potom mu můžou libovolní uživatelé posílat zašifrované zprávy, ale jen on je může dešifrovat. Naopak při podepisování pomocí asymetrické kryptografie se k podpisu používá soukromý klíč a k ověření veřejný klíč. V asymetrické kryptografii se využívá těžkých matematických problémů, jako je například problém faktorizace velkých čísel a problém diskrétního logaritmu. Problém faktorizace velkých čísel je založen na tom, že pokud máme dvě různá velká prvočísla p, q a jejich součin N, a známe pouze N, není v rozumném čase možné zjistit čísla p, q, ze kterých se N skládá. Problém diskrétního logaritmu je založen na tom, že pokud máme rovnici $a^x = y \,(mod\, n)$ je jednoduché spočítat y, pokud známe a, x, n, ale nelze jednoduše zpětně spočítat x, pokud známe hodnoty a, y, n. Díky použití těchto matematických problému v kryptografii je možné vytvořit dvojici klíčů, kdy z veřejného klíče útočník nedokáže spočítat tajný soukromý klíč.

V asymetrické kryptografii se používají delší klíče než v symetrické kryptografii, tradiční délka klíče například u algoritmu *Rivest–Shamir–Adleman* (RSA) je dnes 2048 bitů. Šifrování pomocí asymetrické kryptografie je daleko pomalejší než šifrování symetrickou šifrou, proto se asymetrické šifrování používá spíše pro podpisová schémata, kde se podepisuje například pouze hash souboru, k ověření identity nebo k domluvě klíče pro symetrické šifrování [9].

Často používaným protokolem pro domluvu klíče je Diffieho–Hellmanův protokol. Ten umožňuje za použití asymetrické kryptografie ustálit společný šifrovací klíč mezi dvěma entitami po otevřeném kanále, který poté může být použit pro symetrické šifrování. Diffie-Hellmanův protokol pro výměnu klíče mezi dvěma stranami Alicí a Bobem funguje v několika jednoduchých krocích [10]:

1. Obě strany se dohodnou na parametrech, kterými jsou prvočíslo p o délce alespoň 1024 bitů, generátor g grupy nebo podgrupy $\mathbb{Z}_p^*$ řádu q, kdy q by mělo být dlouhé alespoň 160 bitů. Tyto parametry jsou veřejné.

2. Alice si zvolí náhodné tajné číslo a z grupy $\mathbb{Z}_q$, Bob si stejným způsobem zvolí tajné číslo b.
3. Alice spočítá hodnotu $A = g^a \ mod \ p$ a pošle ji Bobovi, Bob spočítá $B = g^b \ mod \ p$ a pošle ji Alici.
4. Alice spočítá tajný klíč $K = B^a \ mod \ p$, Bob stejný spočítá stejný klíč $K = A^b \ mod \ p$.

Jelikož a i b jsou tajné hodnoty a díky diskrétnímu logaritmu je nezle zpětně vypočítat z hodnot A a B, nemůže útočník vypočítat hodnotu tajného klíče K.

1.4 Kryptografie nad eliptickými křivkami

Kryptografie nad eliptickými křivkami se používá k vytvoření kryptografických systémů veřejného klíče [11]. Základem bezpečnosti je problém diskrétního logaritmu eliptické křivky, tedy že nelze vypočítat diskrétní logaritmus náhodného prvku na eliptické křivce, pokud známe pouze generující bod křivky. Složitost tohoto problému je daná velikostí křivky.

Každý algoritmus založený na problému diskrétního logaritmu v multiplikativní grupě je možné transformovat do struktury aditivní grupy. Výhodou eliptických křivek oproti modulární aritmetice je vyšší bezpečnost pro stejnou délku klíčů a také vyšší rychlost. Proto se eliptické křivky hojně využívají na zařízeních s omezenou výpočetní silou.

V kryptografii se často používají křivky, které se skládají z bodů vyhovujících zjednodušené Weierstrassově rovnici 1.1 a bodem v nekonečnu.

$$y^2 = x^3 + ax + b \ (mod \ p) \tag{1.1}$$

Pro používání eliptické křivky v praxi je potřeba na obou stranách komunikace znát doménové parametry křivky [12]. Tyto parametry jsou: prvočíslo p definující těleso F_p, konstanty a, b použity v definující rovnici, generující bod G, řád n generujícího bodu a kofaktor h. Počet bodů křivky $E(F_p)$ je dán součinem řádu n a kofaktoru h, kdy h musí být číslo menší nebo rovné 4 a nejlépe 1.

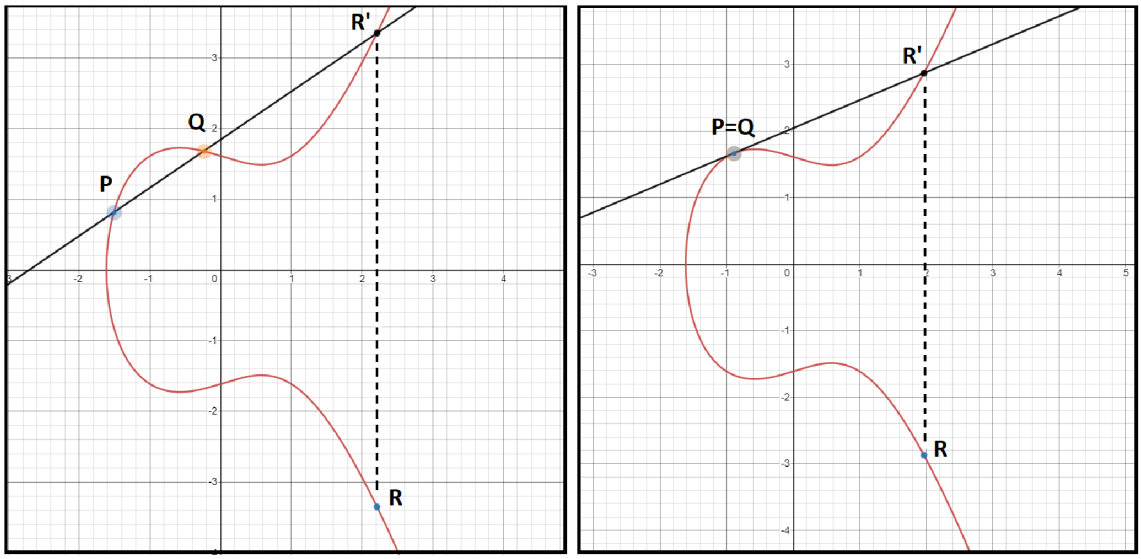
Na eliptických křivkách se dají provádět dvě základní operace s body, a to sčítání bodů a násobní bodu skalárem.

Sčítání bodů

Při sčítání dvou bodů [13] P a Q na eliptické křivce může vzniknout několik případů. Pokud se body P a Q nerovnají je vedena přímka body P, Q. V místě průniku přímky a eliptické křivky vzniká bod R'. Výsledný bod R tak, že P+Q=R, dostaneme osovou souměrností přes osu X. Sčítání dvou různých bodů je graficky znázorněno

na obrázku 1.2 vlevo. Pokud mají body P a Q stejnou souřadnici x, potom přímka skrz ně již eliptickou křivku neprotne a výsledkem je bod v nekonečnu.

Druhou situací při sčítání bodů, která může nastat je ta, že bod P a Q je ten samý bod. V takovém případě mluvíme o zdvojení bodu. V takovém případě je bodem P vedena tečna a v místě průniku tečny a eliptické křivky vzniká bod R', výsledný bod R nalezneme znovu pomocí osové souměrnosti. Zdvojení bodů se většinou zapisuje jako P+P=2P. Zdvojení je znázorněno na obrázku 1.2 vpravo. Pokud by souřadnice y tohoto bodu byla rovna 0, byl by výsledkem bod v nekonečnu, jelikož tečna eliptickou křivku neprotne (protne ji v nekonečnu).

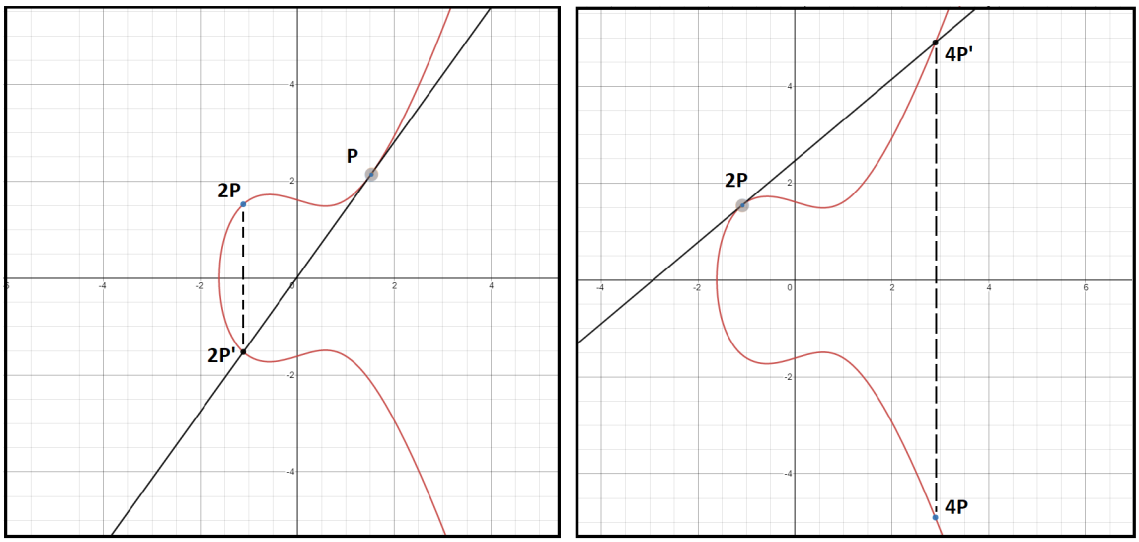


Obr. 1.2: Sčítání bodů na eliptické křivce.

Násobení bodu skalárem

Násobení bodu skalárem je na eliptické křivce definováno jako několikanásobné sečtení bodu. Tedy například v případě, kdy chceme P vynásobit 4, můžeme napsat 4P=P+P+P+P. Takovýto způsob by byl v praxi, kde se počítá s velkými čísly, nepraktický a dá se zjednodušit. Například pokud bychom hledali výsledek násobení 16P, můžeme ho spočítat jako 2P=P+P, 4P=2P+2P, 8P=4P+4P, 16P= 8P+8P, tím jsme násobení zkrátili z 15 sčítání na 4. Grafický příklad násobení pro 4P je znázorněn na obrázku 1.3. Zde se v prvním kroku spočítá bod 2P jako P+P. Tedy je vedena tečna bodem P a v místě průniku tečny a eliptické křivky vzniká bod 2P', bod 2P se poté dostane pomocí osové souměrnosti. V druhém kroku je spočítána hodnota 4P jako 2P+2P, nejdříve je vedena tečna bodem 2P a v místě průniku tečny

a eliptické křivky vzniká bod 4P', osovou souměrností poté vznikne bod 4P. Stejně by se dalo postupovat pro další mocniny dvou.



Obr. 1.3: Násobení bodu skalárem pro 4P

V kryptografii se využívá toho, že pokud bod vynásobíme velkým číslem, nemůže útočník v reálném čase zjistit, jakým skalárem jsme bod vynásobili, pokud zná pouze počáteční a konečný bod, tedy z rovnice $A{=}a \cdot G$ nedokáže spočítat a, pokud zná A, G. Tento fakt se využívá například při tvorbě dvojice veřejného a soukromého klíče, kdy si uživatel vygeneruje náhodné číslo a vynásobí jím bod G. Toto náhodné číslo je poté soukromý klíč a výsledný bod je klíč veřejný.

1.5 Protokoly s nulovou znalostí

Protokol s nulovou znalostí je takový protokol, kde se jedna strana (prover/uživatel) snaží prokázat znalost nějaké tajné informace X druhé straně (ověřovateli) bez předání samotné hodnoty X nebo jakékoliv informace o ní. Výsledkem protokolu je pouze informace o tom, zda je tvrzení uživatele pravdivé nebo ne. Každý protokol s nulovou znalostí musí mít tři základní vlastnosti:

1. **Úplnost**: Každé pravdivé tvrzení provera bude vždy přijato ověřovatelem. Pod pravdivým tvrzením si můžeme představit tvrzení, že prover zná tajnou hodnotu X.
2. **Spolehlivost**: Každé nepravdivé tvrzení provera bude skoro vždy odmítnuto ověřovatelem.
3. **Nulová znalost**: Pokud je tvrzení provera pravdivé, ověřovatel se nedozví nic než to že je pravdivé.

Hlavní výhodou nulové znalosti je to, že pokud komunikace probíhá po veřejné síti v otevřené podobě, útočník, který bude na síti poslouchat, se nemůže dozvědět uživatelovu tajnou hodnotu X. Při jednom opakování protokolu je pravděpodobnost 50%, že prover oklame ověřovatele. Proto je nutné tento protokol opakovat několikrát, kdy pravděpodobnost oklamání ověřovatele je 2^{-n} pro n opakování, pro velké n je tato pravděpodobnost zanedbatelná. Takovýto protokol by byl ve skutečnosti neefektivní, jelikož by muselo být mezi oběma stranami vyměněno velké množství zpráv [14].

Sigma protokoly

V praxi se spíše než přímo protokoly s nulovou znalostí využívají takzvané Sigma protokoly [15] (také psané Σ protokoly), které vychází z protokolů s nulovou znalostí, ale stačí si mezi proverem a ověřovatelem vyměnit tři zprávy k dokázání znalosti tajné informace X, která v tomto případě bývá soukromý klíč. Nejčastěji jsou Σ protokoly založené na problému diskrétního logaritmu.

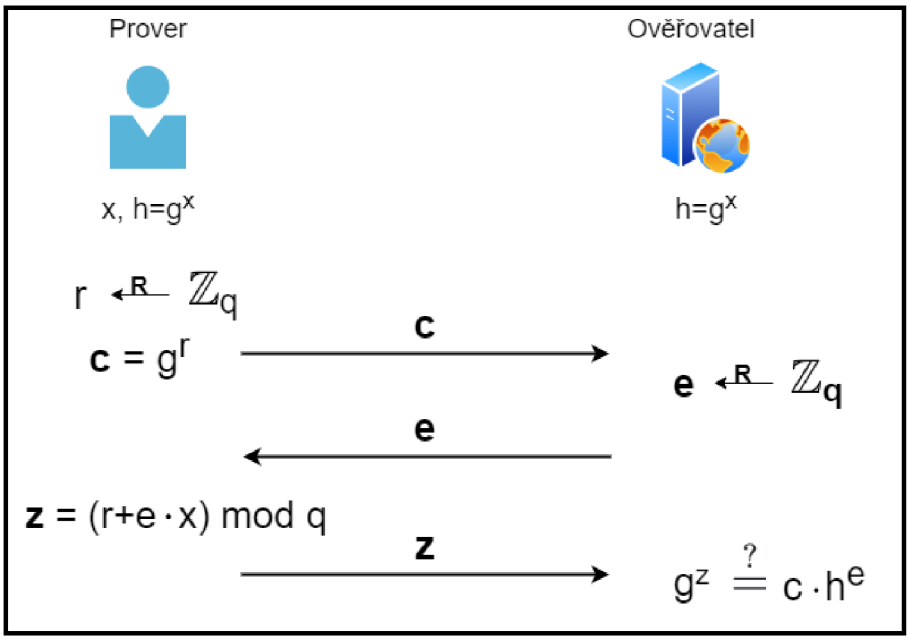
Rozlišujeme dvě základní formy Σ protokolů, a to interaktivní a neinteraktivní protokoly. V interaktivních protokolech generuje výzvu e ověřovatel, tyto protokoly se nejčastěji používají v identifikačních schématech. V neinteraktivních protokolech si výzvu generuje sám prover, což je na jednu stranu bezpečnější, jelikož prover nemůže dostat od zlomyslného ověřovatele upravenou výzvu, která by mohla odhalit jeho tajemství, ale na druhou stranu je protokol v tomto případě otevřen jiným útokům, například v některých systémech by si útočník mohl zaznamenat komunikaci a později ji použít k přístupu. Neinteraktivní Σ protokoly se tedy spíše používají u podpisových schémat.

Schnorrův protokol

Jedním z nejznámějších a i dnes stále používaných Σ protokolů je Schnorrův protokol [16]. Prover v protokolu dokazuje znalost diskrétního logaritmu x nějakého prvku h grupy G tak, že: $h = g^x \in G$, kde G je grupa řádu q, h si v praxi můžeme představit jako veřejný klíč a x jako klíč soukromý, grupa G a generátor grupy g jsou veřejné parametry. Průběh protokolu je znázorněn na obrázku 1.4. Díky tomu, že je protokol založen na problému diskrétního logaritmu, je možné ho implementovat i nad eliptickými křivkami.

Důkaz, že se jedná o sigma protokol se může provést tak, že dokážeme, že splňuje všechny vlastnosti sigma protokolů [17]. Rovnicí 1.2 můžeme dokázat úplnost.

$$g^z \equiv g^{ex+r} \equiv g^{ex} g^r \equiv h^e \cdot c \tag{1.2}$$



Obr. 1.4: Průběh Schnorrova protokolu

Takto jsme dokázali, že pokud budou prover i ověřovatel postupovat podle protokolu, bude prover přijat. Dále také můžeme dokázat, že Schnorrův protokol splňuje vlastnost speciální spolehlivosti. Pro dokázání této vlastnosti se použije důkaz sporem. Předpokládejme, že máme záznam dvou vyhovujících konverzací *(c, e, z)* a *(c, e', z')* mezi proverem a ověřovatelem, kdy prover nezná tajnou hodnotu x, ale je schopen ověřovatele přesvědčit o opaku. Jelikož c je v obou konverzacích stejné, lze použít extraktor pro získání tajné hodnoty x rovnicemi 1.3:

$$\begin{aligned} g^z = h^e c, g^{z'} = h^{e'} c \\ c = \frac{g^z}{h^e} = \frac{g^{z'}}{h^{e'}} \\ g^{z-z'} = h^{e-e'} \\ h = g^{\frac{z-z'}{e-e'}} \\ x = \frac{z-z'}{e-e'} \end{aligned} \tag{1.3}$$

Tím, že jsme schopni extrahovat x, jsme vyvrátili původní tvrzení, že prover nezná x, tudíž jsme dokázali vlastnost speciální spolehlivosti, tedy že prover bude přijat pouze, pokud zná tajnou hodnotu x. Vlastnost třícestného vzoru dokazovat nemusíme, jelikož je zřejmá z obrázku 1.4. Vlastnost speciální nulové znalosti pro poctivého ověřovatele můžeme dokázat sestrojením simulátoru, jehož výstup bude nerozpoznatelný od výstupu reálného protokolu. Máme-li výstup reálného protokolu *(c, e, z)*, simulátor bude postupovat tímto způsobem:

1. Simulátor zvolí náhodnou odpověď $z \in \mathbb{Z}_q$.
2. Simulátor spočítá závazek $c' = g^z h^{-e}$, kde e je vstupem simulátoru bráno z výstupu reálného protokolu.
3. Výstupem simulátoru bude *(c',e, z')*, jelikož z je náhodné číslo, je i c náhodné číslo, tudíž nejsme schopni rozlišit výstup reálného protokolu a simulátoru.

Sestrojením simulátoru, který dokázal vytvořit záznam protokolu nerozpoznatelný od záznamu reálného protokolu i bez znalosti tajné hodnoty x, jsme dokázali i poslední vlastnost Σ protokolů [17].

1.6 Pojmy autorizace, autentizace a řízení přístupu

Pro pochopení systému řízení přístupu je vhodné nejdříve definovat některé pojmy, se kterými se lze u takového systému setkat.

Autentizace je proces ověření identity uživatele nebo zařízení. V praxi si lze autentizaci představit jako zadání uživatelského jména a hesla, tím ověřovatel ví, kdo žádá o přístup, a zda se opravdu jedná o uvedeného uživatele. Dnes se hodně používá také takzvaná dvoufaktorová autentizace. Při takovém ověření identity v procesu ověření figuruje nějaký další faktor, tím může být například zařízení jako mobilní telefon, na který uživatel obdrží jednorázový kód, který zadá spolu s heslem [18].

Autorizace je proces přidělování práv uživateli, například práva někam přistupovat, nebo s něčím manipulovat. Tato práva si lze představit jako povolení přístupu do aplikace a povolení manipulace se soubory ve veřejných složkách. Proces autorizace většinou v systému následuje po autentizaci, kdy je identita uživatele ověřena [19].

Systém řízení přístupu je takový systém, který kontroluje, kdo může přistupovat do určitého systému, prostředí nebo například do budovy. Tento systém umožňuje více uživatelům přístup do systému, monitoruje, kdo k systému přistupuje a také provádí procesy autentizace a autorizace, které jsou hlavními nástroji takového systému [20].

2 Možnosti vývoje aplikace pro operační systém Android

2.1 Java a Kotlin

Při vývoji aplikací pro Android zařízení jsou používány dva základní jazyky, Java a Kotlin [21]. Java byl první jazyk používaný pro vývoj aplikací, ale z důvodu žaloby ze strany Oracle - vývojářů Javy, se v roce 2019 stal preferovaným jazykem pro vývoj na Android Kotlin. Přesto dnes stále velká část programátorů preferuje jazyk Java, a to z důvodu lepší znalosti jazyka, nebo lepší dostupnosti kódů psaném v tomto jazyce.

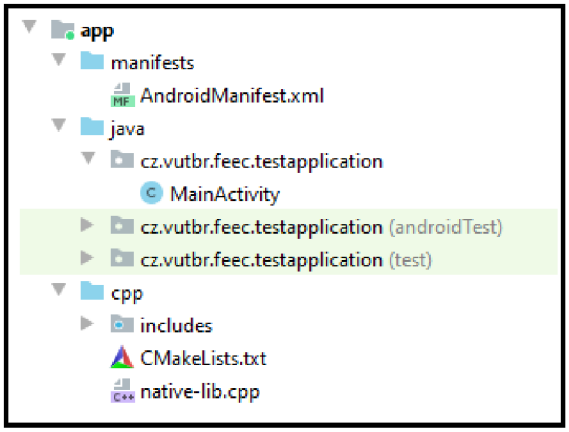
Kotlin byl vyvinut programátory z IDE Jet Brains přímo pro Android. Poprvé byl představen v roce 2011 a je používán od roku 2016. Hlavními výhodami oproti Javě by měla být jeho jednoduchost. V Kotlinu není možné přiřadit proměnné hodnotu Null, a tudíž nemůže dojít k výjimce NullPointerException. V případě snahy o přiřazení nebo vrácení hodnoty Null, kód selže už během kompilace. Další výhodou je, že se v Kotlinu nezachytávají výjimky, což může ulehčit práci a zkrátit kód, ale na druhou stranu vést k pádu programu v případě chyby. V Kotlinu také není nutné explicitně přiřazovat datový typ proměnným. V případě, kdy je potřeba vytvořit třídu, která má uchovávat data, je možné vytvořit Data class, a kompilátor se sám postará o vytvoření getterů, setterů a podobných funkcí. Kotlin vychází z Javy, díky čemuž je s Javou kompatibilní. V jednom projektu je tak možné používat třídy napsané v Javě i v Kotlinu.

2.2 Android NDK a jazyk C

Android *Native Development Kit* (NDK) je nástroj, s jehož pomocí je možné použití kódu napsaného v jazyce C nebo C++ na Androidu. NDK je vhodné použít tam, kde je potřeba dosáhnout malé odezvy a rychlého vykonání výpočtů, nebo při použití již vytvořených C nebo C++ knihoven.

Nástroj je možné používat ve verzích Android Studia 2.2 nebo vyšších pro kompilaci kódu v jazyce C do tzv. nativní knihovny za pomoci Gradlu. Pro vytvoření projektu s podporou jazyka C stačí v Android Studiu nainstalovat NDK a při vytváření projektu z aktivit vybrat Native C++, tím Android Studio vytvoří všechny potřebné složky a soubory [22]. Na obrázku 2.1 lze vidět souborovou strukturu nově vytvořeného projektu, kde Android studio vygenerovalo složku cpp pro soubory jazyka C a také hlavní soubor native-lib.cpp. Dále na obrázku2.2 je zobrazena část automaticky vygenerovaného souboru build.gradle, která definuje použití nativních

funkcí. Do vytvořeného souboru native-lib.cpp potom programátor píše svoje funkce v jazyce C/C++, které je potom možné volat z Javy za pomoci *Java Native Interface* (JNI). Názvy funkcí v souboru native-lib.cpp musí mít specifický formát, obsahující název balíčku a třídy, ve které budou používány. Příklad definice funkce v souboru native-lib.cpp je ve výpise 2.1, na řádku 3 lze vidět příklad, jak musí vypadat název takové funkce, na řádku 8 je poté demonstrován způsob vracení proměnné typu string zpět do Javy. Výpis 2.2 dále ukazuje, jak lze tuto funkci volat ze třídy Javy. Na začátku výpisu je zobrazeno, jak se ve třídě Javy načítá knihovna native-lib, na řádku 6 je zde uveden příklad volání samotné nativní funkce a na posledním řádku je ukázáno, jakým způsobem je nutné nativní funkci definovat uvnitř třídy v Javě. Zobrazená funkce je automaticky vygenerovanou funkcí po vytvoření projektu, která uživateli demonstruje, jak se pracuje s nativními funkcemi. Při práci s proměnnými v C je také vhodné používat tzv. nativní datové typy, jako je např. jint nebo jbyteArray. Tímto způsobem se lze vyhnout některým problémům při vracení proměnných zpět do Javy. V rámci těchto funkcí, které jsou volány z Javy, je možné používat další importované C knihovny pro provedení interních operací.



Obr. 2.1: Ukázka automaticky vytvořených složek projektu

```
externalNativeBuild {
    cmake {
        path "src/main/cpp/CMakeLists.txt"
        version "3.10.2"
    }
}
```

Obr. 2.2: Ukázka části build.gradle souboru vygenerovaného projektu.

Výpis 2.1: Příklad definice nativní funkce v souboru native-lib.cpp

```
extern "C" JNIEXPORT jstring JNICALL
Java_cz_vutbr_feec_testapplication_MainActivity_stringFrom
    JNI(JNIEnv* env, jobject /* this */) {
    std::string hello = "Hello from C++";
    //returning variable trough JNIEnv
    return env->NewStringUTF(hello.c_str());
}
```

Výpis 2.2: Příklad volání nativní funkce z Javy

```
static {
//loading of the native-lib library
    System.loadLibrary("native-lib");
}
//example of a call of native function
String helloFromC=stringFromJNI();
...//definition of native function inside the Java class
public native String stringFromJNI();
```

Teoreticky je možné psát aplikace pro Android čistě v C nebo C++, ale není to výhodné. Mnoho funkcí a služeb, které jsou pro Android implementované v Javě nebo Kotlinu by bylo velmi složité programovat v Jazyce C, a například při práci s *User Interface* (UI) by bylo nutné brát v úvahu architektury procesorů a grafických čipů, proto je nejvýhodnější volbou použití Javy nebo Kotlinu a volání C funkcí pro vykonávání náročných výpočtů.

V rámci této práce byla pro vývoj aplikace zvolena Java s použitím NDK pro volání funkcí jazyka C tam, kde je možné tímto způsobem ušetřit značné množství času. Jelikož aplikace na straně počítače bude naprogramována v Javě, z důvodu kompatibility použitých knihoven pro práci s eliptickými křivkami a větší znalosti jazyka autorem, bude zvolena Java namísto Kotlinu na straně Android zařízení.

2.3 Kryptografické knihovny

Tato podkapitola popisuje vybrané knihovny jazyka C a Javy umožňující operace nad eliptickými křivkami, které jsou použity ve vyvinutých aplikacích.

2.3.1 Java - Spongy Castle

Java knihovna Spongy Castle [23] byla vybrána z důvodu dobré kompatibility s Android zařízeními. První zvažovanou Java knihovnou byla Bouncy Castle, ale s její kompatibilitou na Androidu může být problém. Spongy Castle je víceméně identická, jelikož většina funkcí je převzata právě ze zmíněné knihovny Bouncy Castle. Spongy Castle je v projektu použit jako poskytovatel (provider) funkcí Java Security. Knihovna obsahuje velké množství kryptografických funkcí, z knihovny budou v praktické části testovány například funkce pro dohodnutí tajného klíče pomocí algoritmu *Elliptic-Curve Diffie–Hellman* (ECDH), nebo funkce pro operace s body na křivce. Knihovna používá pro reprezentaci bodů na křivce datový typ `ECPoint`, pro násobení bodu konstantou se pro konstantu používá datový typ `BigInteger`. Body je také možné převést na pole bajtů buď ve zkrácené formě (compressed), která obsahuje pouze souřadnici X a prefix, nebo v celé formě, která obsahuje souřadnice X a Y spolu s prefixem.

2.3.2 Jazyk C - micro-ecc

Micro-ecc [24] je knihovna pro jazyk C. Jejími hlavními funkcemi jsou algoritmy *Elliptic Curve Digital Signature Algorithm* (ECDSA) a ECDH, tedy algoritmy nad eliptickými křivkami. Knihovna je většinou používána v zařízeních s omezenou výpočetní silou nebo tam, kde je důležité rychlé provedení kryptografických algoritmů. Knihovna podporuje 5 základních eliptických křivek, a to: secp160r1, secp192r1, secp224r1, secp256r1, a secp256k1. V rámci práce je využita i tato knihovna upravená panem Tadeášem Cvrčkem [25], která je rozšířená o operace s body na eliptické křivce. Knihovna by měla být odolná vůči většině útoků pomocí postranních kanálů a fungovat na většině architektur procesorů. Knihovna body na křivce reprezentuje jako pole `uECC_word_t`, což je pole typu `uitn32_t`, nebo jiné bitové délky v závislosti na architektuře procesoru.

2.3.3 Jazyk C - tiny-AES, Crypto

Knihovna tiny-AES [26] je malá knihovna jazyka C, která umožňuje šifrování pomocí šifry AES, a to v módech ECB, CTR a CBC. V základě knihovna používá 128 bitové klíče, ale lze zde definovat i klíče délky 192 a 256 bitů. Hlavní výhodou knihovny je její malá velikost.

Knihovna Crypto [27] je knihovna vyvinutá primárně pro Arduino, ale lze použít i na jiných zařízeních. Knihovna obsahuje například funkce pro hashování, jako je SHA256 nebo SHA3, funkce pro generaci náhodných čísel a několik různých šifrovacích algoritmů.

3 Komunikační technologie NFC a Bluetooth

Tato kapitola se věnuje popisu komunikačních technologií, které budou v rámci praktické části použity.

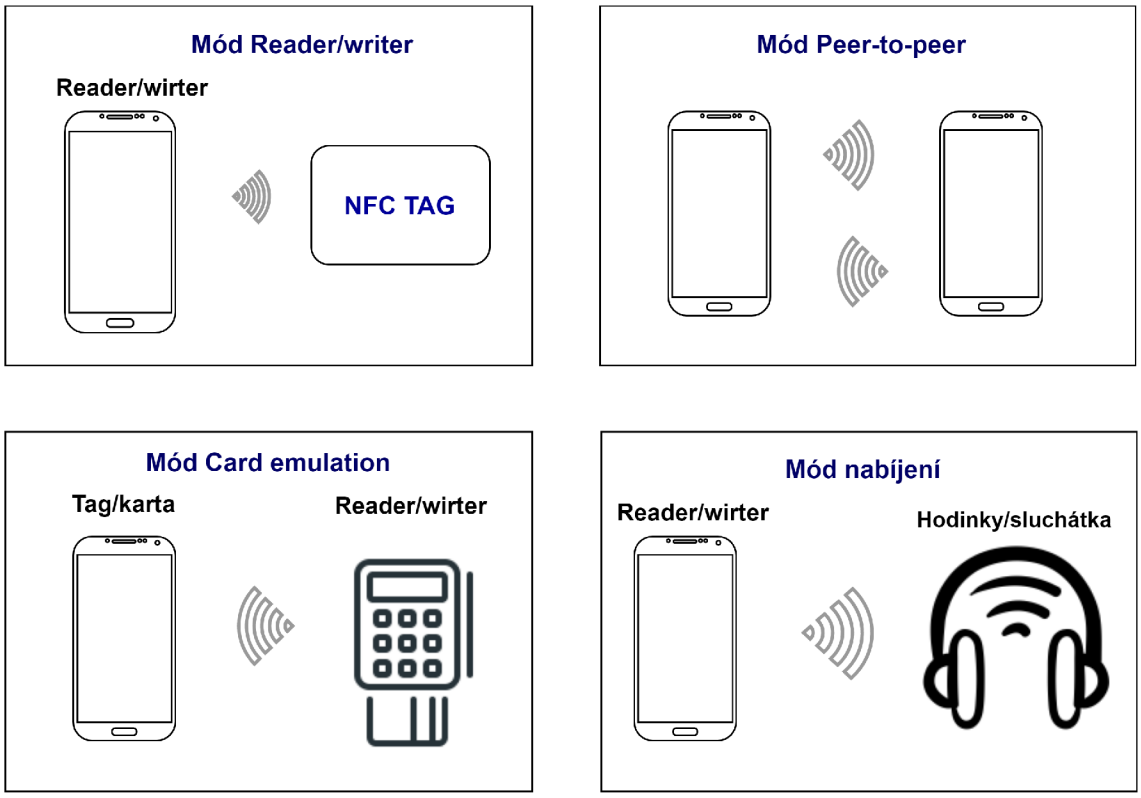
3.1 Technologie NFC

NFC [28] je technologie bezdrátové komunikace mezi dvěma elektronickými zařízeními na vzdálenost menší než 4 centimetry. Obecně se během komunikace jedno zařízení chová jako terminál a druhé jako "tag"(karta). Terminál je aktivní zařízení, které vysílá rádiové signály (elektromagnetické pole) a komunikuje tímto způsobem s tagem. Tag je v tomto případě pasivní zařízení a komunikuje s terminálem modulováním elektromagnetického pole. Pokud je tag elektrické zařízení, jako například mobilní telefon, může s terminálem komunikovat aktivně střídavým přenosem signálů. Standardy technologie jsou založené na standardech *Radio Frequency Identification* (RFID) (ISO/IEC 14443 a FeliCa), ISO/IEC 18092 a na těch definovaných organizací NFC Forum [29]. Technologie se nejčastěji používá k placení pomocí platebních karet, nebo v přístupových systémech, kde jsou čipové karty používány jako identifikátor osoby. Jednou z nevýhod je, že NFC samo o sobě nezabezpečuje komunikaci, ani neobsahuje ochranu proti odposlechu a změně přenášených dat. Proto je pro zajištění bezpečného přenosu žádoucí využití protokolů vyšších vrstev.

Existují čtyři operační módy NFC zařízení [30], tyto módy jsou zobrazeny na obrázku 3.1. Prvním je mód reader/writer, v tomto módu může NFC zařízení (mobilní telefon) číst data uložená v NFC tagu a použít je k dalším operacím, bývají to například webové stránky nebo telefonní čísla. Další je mód Peer-to-Peer, v tomto módu si můžou dvě NFC zařízení vyměňovat data, většinou malé velikosti, jako například výměna vizitek, nebo parametry potřebné k sestavení Bluetooth spojení. Asi nejpoužívanější je mód Card Emulation, který umožňuje NFC zařízení, aby se tvářilo jako bezkontaktní karta, díky tomu je možné platit pomocí mobilního telefonu beze změny zavedené infrastruktury. Posledním módem je bezdrátové nabíjení, kdy je možné nabíjet přes NFC malá zařízení, jako jsou například Bluetooth sluchátka nebo hodinky.

3.1.1 Technologie HCE

Host Card Emulation (HCE) [31] je technologie, která umožňuje emulaci čipových karet pouze za pomoci softwaru. Technologie se používá v mobilních zařízeních, kde

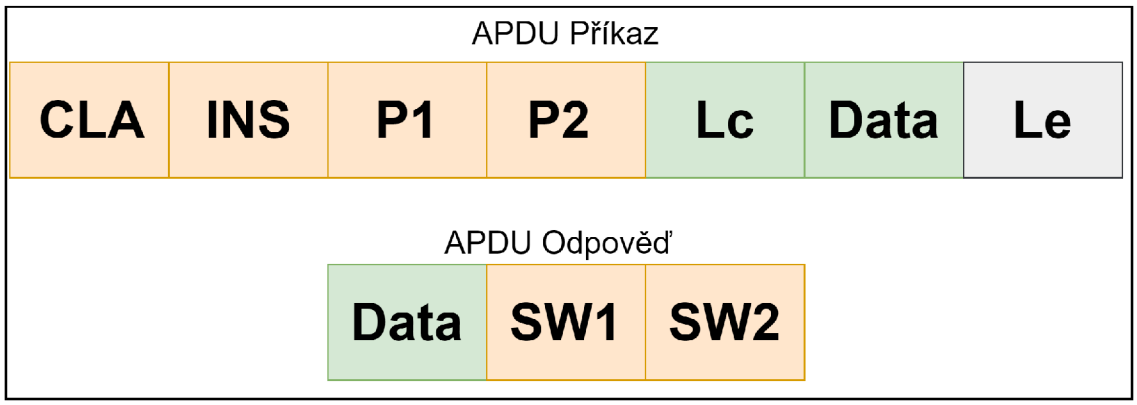


Obr. 3.1: Operační módy NFC zařízení.

je možno ji používat od verze Android 4.4. Karta může být v mobilním zařízení emulována buď použitím bezpečnostního prvku nebo pomocí hostitelské emulace. V případě hostitelské emulace jsou data ze čtečky NFC kontrolerem posílána přímo procesoru, na kterém běží Android aplikace. Při použití bezpečnostního prvku kontroler posílá data danému bezpečnostnímu prvku, a tudíž aplikace se na transakci nijak nepodílí.

HCE dává vývojáři možnost přijímat zprávy od čtečky karet a definovat na ně odpovědi. Těmto zprávám se říká *Application Protocol Data Unit* (APDU) příkazy. APDU je komunikační protokol mezi čtečkou karet a čipovou kartou [32]. Komunikace funguje posíláním APDU příkazů kartě, nebo v případě využití technologie HCE telefonu, který kartu emuluje. Struktura těchto příkazů je definována standardem ISO/IEC 7816-4. Příkazy se zasílají ve formě bajtů. Struktura APDU příkazu je zobrazena na obrázku 3.2, první bajt (CLA) značí třídu instrukce, druhý bajt (INS) kód instrukce, třetí a čtvrtý (P1, P2) jsou parametry instrukce, pátý bajt (Lc) říká, kolik bajtů mají přenášená data, za ním následují bajty dat a poslední bajt (Le) značí, kolik bajtů dat je očekáváno nazpět. Každý APDU příkaz musí obsahovat první čtyři zmíněné bajty, které jsou v obrázku vyznačené oranžovou bar-

vou, říká se jim APDU hlavička. Příkaz musí obsahovat bajt Lc pouze v případě, že přenáší nějaká data. Bajt Le bude obsahovat jenom takový příkaz, který očekává data nazpět.



Obr. 3.2: Struktura APDU příkazu a odpovědi.

První příkaz, který musí čtečka odeslat, je příkaz pro výběr *Application Identifier* (AID). Tím čtečka říká, s jakou aplikací chce komunikovat. Tento příkaz vždy obsahuje stejnou čtyřbajtovou hlavičku, a to 00A40400. Bajt Lc zde říká, kolik bajtů má AID. Následuje AID aplikace, se kterou chce čtečka komunikovat. Poslední bajt značící, kolik dat se očekává nazpět, je většinou 0x00.

Další proprietární příkazy si může programátor navrhnout podle vlastního uvážení. První bajt proprietárního příkazu značící třídu příkazu, je vždy 0x80. Kód instrukce i parametry si už může vývojář zvolit libovolně. Struktura odpovědi na tyto příkazy je zobrazena ve spodní části obrázku 3.2. První část odpovědi obsahuje bajty dat a na konci odpovědi jsou dva bajty SW1 a SW2, ty značí stav zpracování příkazu. Většinou by tyto bajty měly být 0x90 a 0x00, což značí úspěšné provedení příkazu.

3.2 Technologie Bluetooth

Bluetooth [33] je bezdrátová technologie umožňující propojení dvou nebo více elektronických zařízení a komunikaci mezi nimi pomocí rádiových vln ultra vysoké frekvence. Standard byl vytvořen v roce 1994 firmou Ericsson a je definován v IEEE 802.15.1. Bluetooth operuje v rozsahu frekvence 2400-2483,5 MHz, na této frekvenci používá technologii *Frequency Hopping Spread Spectrum* (FHSS). Principem této technologie je přeskakování mezi více frekvencemi během přenosu. Tím je pásmo Bluetooth rozděleno na 79 kanálů, každý kanál má šířku pásma 1 Mhz. Bluetooth

během komunikace skáče mezi těmito kanály až 1600krát za sekundu. Důvodem této technologie je vyhnutí se kolizím s jinými komunikačními technologiemi [34].

Existuje mnoho verzí Bluetooth, většina z nich je zpětně kompatibilní. V dnešních zařízeních se nejčastěji setkáme s verzí Bluetooth 5.0, která oproti minulým verzím nabízí větší přenosovou rychlost a přenos na delší vzdálenost. Je ale zpětně kompatibilní pouze s verzí 4.0 a novějšími. S vydáním verze Bluetooth 4.0 byla také přijata tzv. Bluetooth Core Specification version 4.0, která obsahuje tři základní protokoly, Classic Bluetooth, Bluetooth high speed a *Bluetooth Low Energy* (BLE). High speed verze technologie poskytuje nejvyšší rychlost díky propojení technologií Bluetooth a Wi-Fi, kde Bluetooth se používá pro navázání spojení a Wi-Fi pro přenos dat.

3.2.1 Bluetooth Low Energy

BLE [35] je technologie bezdrátové osobní sítě umožňující komunikaci mezi zařízeními. BLE vychází z technologie Bluetooth, ale není s ní kompatibilní. Hlavní výhodou oproti klasické verzi Bluetooth je nízká spotřeba energie, naopak nevýhodou je pomalejší rychlost přenosu, proto se BLE používá v zařízeních, kde je důležitá dlouhá výdrž baterie, například v senzorech, ve zdravotnictví nebo v bezpečnostních zařízeních.

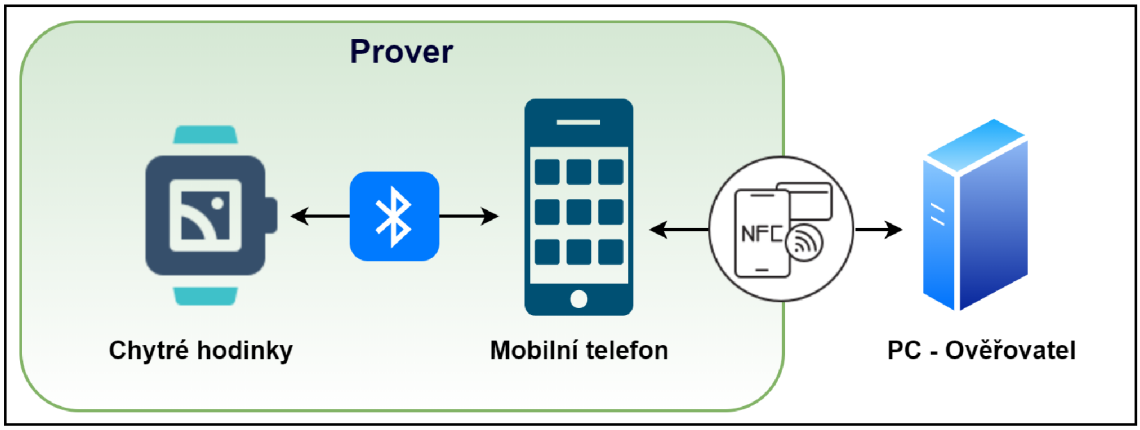
BLE se hodí pro aplikace, které nepotřebují odesílat velké množství dat ani neudržují stálé spojení. Stačí jim například pouze přijmout informace o čase a teplotě jednou za určitou časovou periodu. Na zařízení s operačním systémem Android je možné ji využívat od verze Android 4.3. Jedná se tedy hlavně o průmyslovou technologii používanou v automatizovaných systémech. U Android zařízení je využívaná pro komunikaci s fitness zařízeními, monitory tepové frekvence a dalšími snímači.

Při použití BLE se zařízení zúčastňující se komunikace dělí na dva typy, centrální (klient) a periferní (server). Roli periferního zařízení plní například bezdrátová sluchátka, monitory srdečního tepu nebo snímače teploty. Tato zařízení vysílají informaci o své dostupnosti a poskytují rozhraní pro komunikaci. Centrálním zařízením je v praxi obvykle chytrý telefon nebo počítač. Tato zařízení před zahájením komunikace skenují dostupná periferní zařízení v okolí. Způsob, kterým se data pomocí BLE přenáší, se nazývá *Generic Attribute* (GATT) profil. V tomto profilu může být periferní zařízení v jeden okamžik připojeno pouze k jednomu centrálnímu zařízení. Ve chvíli, kdy se periferní zařízení spojí s centrálním zařízením, nebude již vysílat informace potřebné k připojení. K centrálnímu zařízení naopak lze připojit více periferních zařízení najednou [36].

4 Implementace autentizačního systému pro více zařízení

Tato kapitola se věnuje implementaci autentizačního systému. Ve výsledném systému figurují 3 zařízení. Prvním zařízením je PC, které plní roli ověřovatele v systému, druhým zařízením je Android mobilní telefon, který funguje jako primární zařízení provera, který dokazuje znalost soukromého klíče. Posledním zařízením jsou chytré hodinky, které plní roli sekundárního zařízení provera, tedy fungují jako druhý faktor při autentizaci.

Celkově systém obsahuje 3 aplikace, jednu pro každé zařízení. Tyto aplikace mezi sebou komunikují přes různá komunikační rozhraní. PC aplikace komunikuje s aplikací mobilního telefonu přes rozhraní NFC, mobilní aplikace navíc komunikuje s aplikací v chytrých hodinkách přes Bluetooth.



Obr. 4.1: Schéma použitých zařízení a komunikačních rozhraní v systému.

4.1 Výkonové testy kryptografických knihoven

Prvním krokem při implementaci systému bylo porovnání rychlosti kryptografických funkcí jazyka C a Javy na mobilním telefonu a na chytrých hodinkách. Hlavním cílem výkonových testů bylo zjistit, jak velký je rychlostní rozdíl různých kryptografických funkcí implementovaných v Javě a v jazyce C, aby bylo možné se rozhodnout, zda je výhodné na Android zařízení či chytrých hodinkách používat nativní funkce a počítat některé operace v jazyce C.

V Javě byla pro testování použita knihovna Spongy Castle [23] a pro jazyk C se používala knihovna micro-ecc [24], tiny-AES [26] a knihovna Crypto [27]. Testování

i výkonové testy probíhaly na mobilním telefonu Xiaomi Redmi Note 8 Pro (6GB RAM, procesor Mediatek Helio G90T - 8 jader, 2,05 GHz) s verzí Android 10, a na chytrých hodinkách Huawei Watch 2 4G LEO-DLXX (768 MB RAM, Qualcomm Snapdragon 2100 - 4 jádra, 1,2 GHz) s operačním systémem Wear OS verze 2.19.

Během měření se zaznamenávaly časy pro: Podpis pomocí ECDSA s použitím křivky secp256k1 [37] (v tabulce sloupek ECDSA S), ověření podpisu ECDSA (ECDSA V), šifrování jednoho bloku pomocí AES-256-CBC (AES E), dešifrování pomocí AES-256-CBC (AES D), hashování pomocí SHA-256 (SHA), sčítání dvou bodů na eliptické křivce secp256k1 (EC+), násobení bodu na eliptické křivce skalárem (EC*) a provedení protokolu ECDH, kde se do výsledného času počítalo i vytvoření jednotlivých klíčů (ECDH).

	ECDSA S	ECDSA V	AES E	AES D	SHA	EC+	EC*	ECDH
Java 1	5,3	4,4	5,1	4,9	0,094	0,056	9,785	117
Java loop	1,9	2,2	0,45	0,35	0,092	0,051	1,59	6,5
C	3,09	3,1	0,071	0,063	0,017	0,081	2,82	3,25

Tab. 4.1: Srovnání některých algoritmů v jazyce C a v Javě na Android mobilním telefonu (časy v ms).

	ECDSA S	ECDSA V	AES E	AES D	SHA	EC+	EC*	ECDH
Java 1	77,49	56,28	13,1	12,8	3,4	1,07	171	1453
Java loop	26,5	27,3	6,2	6,6	1,19	0,157	15,9	86
C	18,01	18,2	0,272	0,263	0,123	0,53	15,4	46

Tab. 4.2: Srovnání některých algoritmů v jazyce C a v Javě na chytrých hodinkách (časy v ms).

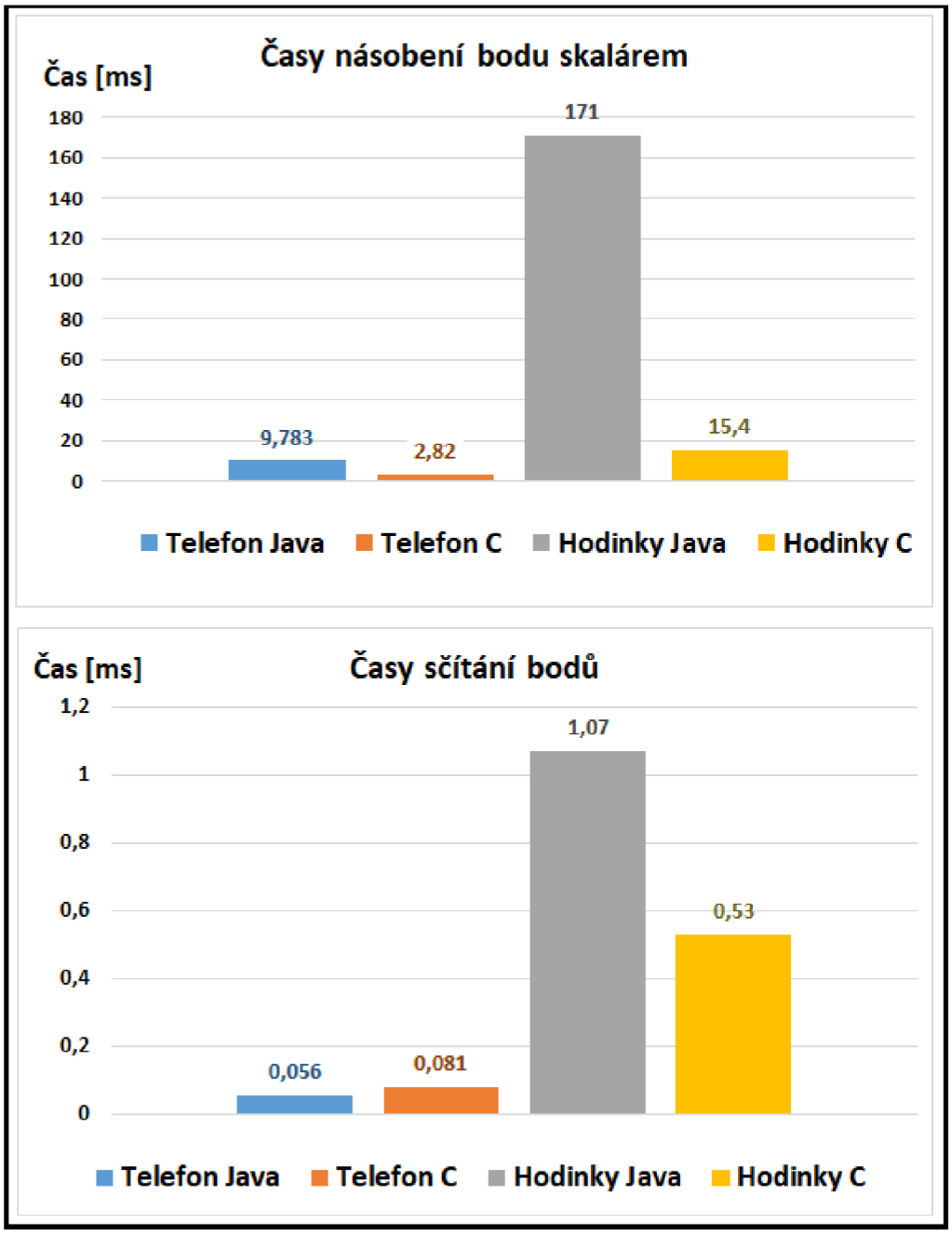
Při měření došlo k zajímavému zjištění ohledně funkcí implementovaných v Javě. Pokud byly některé funkce zavolány několikrát za sebou, například ve `for` smyčce, bylo poté jejich vykonání rychlejší než jejich první zavolání po zapnutí programu. Důvodem toho je nejspíše u několika funkcí (AES, ECDH) volání funkce `GetInstance()`, které při prvním zavolání trvá velmi dlouhou dobu. V případě testování rychlosti blokové šifry AES-256 se čas dešifrování liší podle toho, zda bylo před tím provedeno šifrování a zavolána funkce `GetInstance()` pro požadované parametry. V případě testu, kdy se šifrování i dešifrování provádí v rámci běhu jednoho programu, bude čas dešifrování odpovídat času v řádku Java loop. V případě zavolání dešifrování

před šifrováním bude čas odpovídat hodnotě v řádku Java 1 a bude velmi podobný času šifrování. U násobení bodu je to pravděpodobně způsobené tvorbou objektu, který toto násobení na dané křivce provádí. V protokolu implementovaném v rámci autentizační aplikace nás ale více zajímají časy prvního volání. Časy prvního volání nám ukazuje řádek Java 1 v tabulce 4.1, řádek Java loop ukazuje průměrný čas jednoho provedení algoritmu při vykonávání ve smyčce (při deseti opakováních).

Při porovnávání výsledků je cíleno na časy algoritmů při prvním vykonání v Javě srovnané s časy vykonání knihovnou jazyka C, dále na porovnání rychlosti mobilního telefonu a chytrých hodinek. Rozdílné časy vykonání funkce v rámci jednoho cyklu jsou s ohledem na tuto práci spíše zajímavostí, ale mohly by být důležité například v aplikaci, kdy je ověřován velký počet podpisů v rámci cyklu.

Z tabulky 4.1 je zřejmé, že většina funkcí implementovaných v jazyce C je rychlejších než v Javě. Největší rozdíl je u algoritmu ECDH, kde první vykonání trvá v Javě více než třicetkrát déle. Důvodem jsou objekty, které musí Java vytvořit pro vykonání algoritmu. Pro systém vytvořený v rámci bakalářské práce jsou nejdůležitější operace s body na eliptické křivce a hashování. Hashovací funkce SHA-256 je sice v jazyce C asi pětkrát rychlejší, ovšem celkový rozdíl mezi časy není ani desetina milisekundy. Při testování byl vstup hashovací funkce dlouhý 64 bajtů. Ve vytvořeném systému se budou hashovat pouze vcelku krátké pole bajtů, předávání hodnot do jazyka C a hashování C knihovnou Crypto by se tedy nemuselo vyplatit a neušetřilo by podstatnou část celkového času protokolu. Zajímavý je čas násobení bodu skalárem. Zde se násobil bod G náhodným číslem z rozsahu hodnoty n eliptické křivky. Zde je knihovna micro-ecc více než třikrát rychlejší. Jedná se zde o rozdíl 7 ms na mobilním telefonu, v tomto případě už je výhodné předání hodnot do jazyka C a využití knihovny micro-ecc k násobení bodu skalárem.

U naměřených časů pro chytré hodinky zapsané v tabulce 4.2, si můžeme všimnout toho, že zde jsou v některých případech rozdíly mezi Javou a knihovnami jazyka C ještě větší. Hodnoty časů pro počítání s body na křivce byly vyneseny do grafů na obrázku 4.2. V prvním grafu, který ukazuje časy násobení bodů skalárem, je dobře vidět obrovský rozdíl mezi implementací v Javě a v C na chytrých hodinkách. Z důvodu nižšího výkonu hodinek ve srovnání s telefonem je inicializace objektů potřebných k vykonání algoritmů v Javě ještě časově náročnější. Je zde také dobře vidět rozdíl mezi časem pro výpočet na mobilním telefonu a na hodinkách. Z druhého grafu lze také vypozorovat, že zatímco na mobilním telefonu je sčítání bodů rychlejší v Javě, na hodinkách tomu tak již není. V případě chytrých hodinek je pro nás nejdůležitější operace násobení bodu skalárem na eliptické křivce. Tato operace je na hodinkách více než desetkrát rychlejší v jazyce C, a jedná se zde o časový rozdíl 155 ms, což je za běhu protokolu, kde se tato operace používá, opravdu hodně. Proto i na hodinkách je vhodné použít knihovnu micro-ecc jazyka C pro násobení



Obr. 4.2: Grafy časů pro sčítání bodů a násobení bodu skalárem.

bodů. V porovnání s mobilním telefonem jsou časy algoritmů na chytrých hodinkách v Jazyce C pomalejší přibližně šestkrát a v Javě je tento rozdíl ještě větší, a to mezi 10-36 násobkem časů na mobilním telefonu pro různé algoritmy.

4.2 Použitý autentizační protokol

Protokol implementovaný v autentizačním systému je založen na Schnorrovu protokolu, kdy se ověřuje identita provera i ověřovatele. Protokol je implementován nad eliptickými křivkami. Schéma protokolu je znázorněno na obrázku 4.3. Na obrázku a v implementaci figurují v protokolu tři zařízení, bylo by však možné tento protokol implementovat i pro větší počet zařízení, kdy by každé počítalo svůj důkaz. Na obrázku je také možné vidět, přes jaká komunikační rozhraní se data mezi zařízeními přenášejí. Systém také umožňuje autentizaci pouze pomocí primárního zařízení. V tomto případě autentizace musí být uživatel navíc ověřen heslem a protokol se zjednoduší odstraněním části výpočtů souvisejících s hodinkami. Výpočet důkazu se pak provede způsobem uvedeným v rovnici 4.1.

$$\begin{aligned} r &\in {}_{\mathrm{R}}\mathbb{Z}_{\mathrm{q}} \\ t &= g.r \\ t_{\mathrm{k}} &= t'_{\mathrm{v}}.r \\ e &= H(Y, t, t_{\mathrm{k}}) \\ s &= r - e.sk_0 \ mod \ q \end{aligned} \tag{4.1}$$

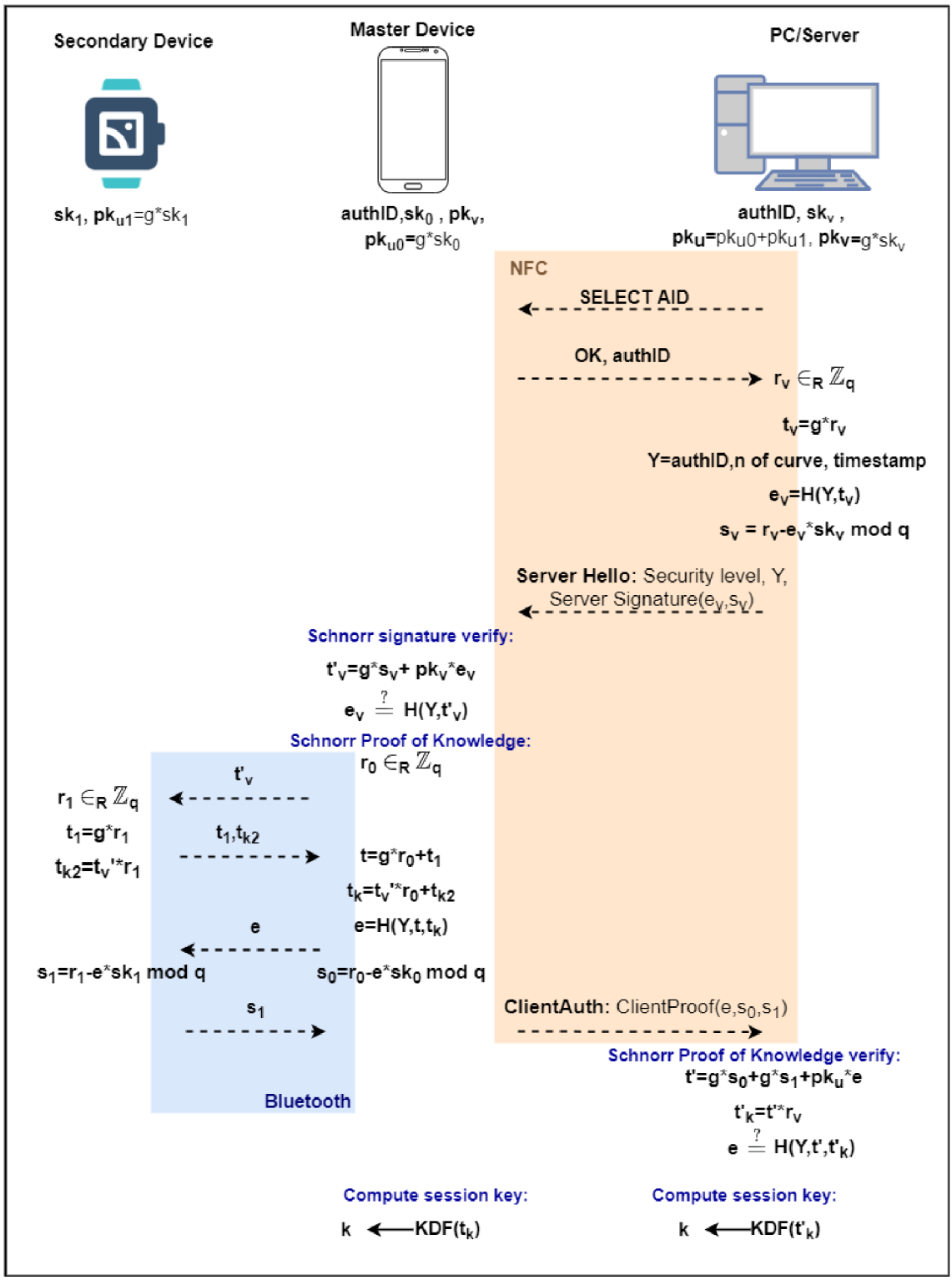
V případě použití jednoho zařízení se mírně liší i výpočet pro ověření, ten je uveden v rovnici 4.2.

$$\begin{aligned} t' &= g.s + pk_{\mathrm{u0}}.e \\ t'_{\mathrm{k}} &= t'.r_{\mathrm{v}} \\ e &\stackrel{?}{=} H(Y, t', t'_{\mathrm{k}}) \end{aligned} \tag{4.2}$$

4.3 Kryptografické parametry systému

Systém ve finální podobě podporuje tři bezpečnostní úrovně, ty se zde nazývají Security Level 0-2. Důvodem k vytvoření několika úrovní zabezpečení bylo především srovnání rychlosti pro různé eliptické křivky.

Hlavním bezpečnostním stupněm je stupeň 2, který zajišťuje nejlepší zabezpečení. Ve stupni 2 je použita 256bitová eliptická křivka secp256k1, hashovací funkce SHA-256 a bloková šifra AES-GCM s klíčem délky 128 bitů. U stupně bezpečnosti 1 je použita křivka secp224r1, hashovací funkce SHA-224 a stejná bloková šifra jako u stupně 2. Poslední stupeň bezpečnosti 0 používá křivku secp160r1, hashovací funkci SHA-1 a blokovou šifru TripleDES-CBC. Bezpečnostní úroveň volí před zahájením komunikace PC aplikace.



Obr. 4.3: Schéma použitého protokolu

4.4 Aplikace pro Android mobilní telefon

Aplikace pro Android mobilní telefon byla vyvíjena v Android Studiu a je napsána v jazyce Java. K některým výpočtům za běhu protokolu se zde používá jazyk C a knihovna micro-ecc. Hlavní funkcí aplikace pro mobilní telefon je v protokolu ověřování podpisu serveru a počítání důkazu znalosti soukromého klíče.

4.4.1 Struktura aplikace pro telefon

V této části jsou stručně popsané jednotlivé třídy, do kterých je aplikace rozdělena. Způsob implementace funkcionalit jednotlivých tříd bude podrobněji vysvětlen v samostatných podsekcích. Účelem této sekce je dát čtenáři představu o tom, kde se jednotlivé funkcionality v programu nachází. Schéma tříd a jejich funkcí je zobrazeno v diagramu na obrázku 4.4. V diagramu je také znázorněno, jaké třídy spolu komunikují.

Třída ECOperations

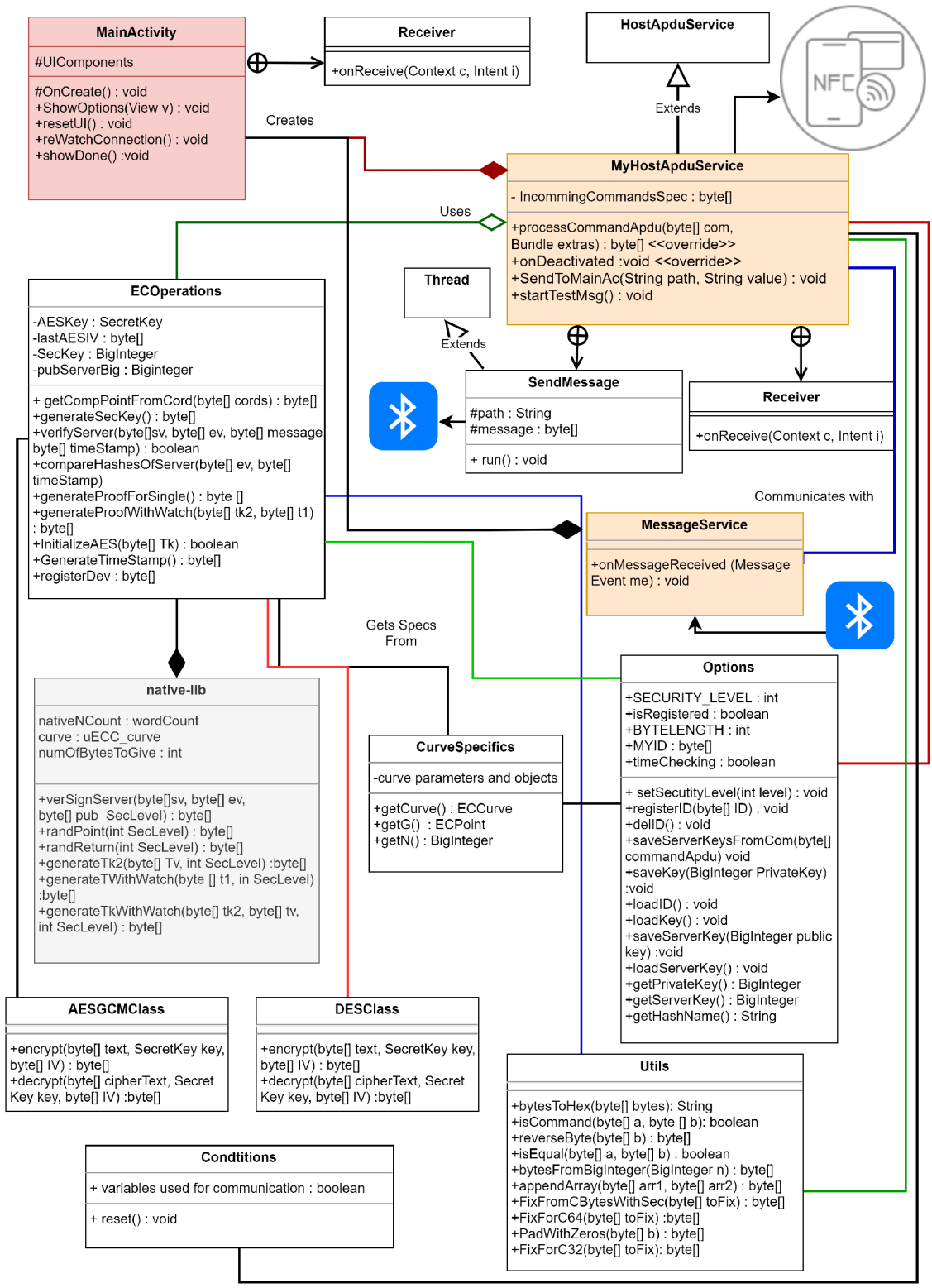
Třída obsahuje funkce nutné pro výpočet důkazu znalosti. Tato třída k některým výpočtům používá nativní funkce jazyka C. Hlavními funkcemi zde jsou funkce `verifyServer(...)` pro ověření podpisu serveru, `GenerateProofWithWatch(...)` pro výpočet důkazu znalosti pro dvě zařízení a `generateProofForSingle(...)`, která zajišťuje výpočet důkazu znalosti pro jedno zařízení.

Třída MyHostAduService

MyHostAduService je hlavní třídou, co se týče výpočtu důkazu během protokolu. V této třídě jsou definované reakce na různé APDU zprávy od PC aplikace. Třída si volá funkce pro výpočty ze třídy EcOperations. Dále se v této třídě přijímají zprávy vysílané v rámci aplikace, které byly přijaty od hodinek a odesílají se z této třídy zprávy hodinkám. Také se z této třídy posílají vysílání do hlavní aktivity v případě, kdy je potřeba aktualizovat grafické rozhraní.

Třída MainActivity

MainActivity je hlavní třídou celé aplikace. V této třídě je metoda `onCreate()`, tato metoda je vždy zavolána po spuštění aplikace. V těle této metody jsou vytvořeny instance služeb, které jsou potřeba, a jsou zde definované reakce na stisky různých tlačítek v aplikaci. Třída také přijímá zprávy od třídy MyHostAduService, které většinou obsahují údaje o výsledku autentizace nebo registrace. Z této třídy je také posílána zpráva hodinkám pro restartování komunikace.



Obr. 4.4: Diagram vytvořených tříd v aplikace pro mobilní telefon.

Třída Conditions

Tato třída obsahuje statické proměnné typu boolean. Hlavním důvodem použití této třídy byl problém, kdy někdy mohlo vysílání od hodinek, nebo vysílání skrz aplikaci přijít víckrát. Poté mohlo dojít ke znehodnocení výsledků, nebo celkově k chybě v aplikaci. Tyto proměnné tedy slouží k tomu, aby program reagoval na přijatou zprávu s daty bodů od hodinek pouze jednou. V praxi to funguje tak, že přijímač se například zeptá na hodnotu proměnné `gotFirstLBM`, v případě vrácení odpovědi true, ji změní na false a pokračuje. Pokud dostane odpověď false, tak se reakce na zprávu nevykoná. Při testování byly nejdříve tyto proměnné vytvořené lokálně v jednotlivých třídách, pak se ale často stávalo, že proměnné nezamezily opětovnému přijetí zprávy. Proto byly tyto proměnné přesunuty do samostatné třídy, kde už splňují svoji funkci.

Třídy AESClass a DESClass

Tyto třídy obsahují funkce pro šifrování a dešifrování pomocí blokových šifer AES a DES.

Třída Utils

Tato třída obsahuje funkce pro převádění datových typů a pro úpravu proměnných pro předávání do jazyka C.

Třída Options

Třída Options uchovává informace o aktuální nastavené bezpečnosti aplikace. Dále tato třída provádí načítání a ukládání klíčů.

Třída CurveSpecifics

Tato třída obsahuje informace o různých eliptických křivkách. Obsahuje objekty `ECCurve` použitých křivek a jejich bod g a hodnotu n. Třída tyto hodnoty vrací na základě nastavení bezpečnosti ve třídě Options.

Třída MessageService

Třída MessageService obstarává službu příjmu zpráv od hodinek a přeposílá je do třídy MyHostAduService.

4.4.2 Implementace protokolu na straně mobilního telefonu

Základní funkce, které se na mobilním telefonu používají pro ověření a výpočet důkazu během protokolu, se nachází ve třídě ECOperations. Tyto funkce používají na počítání některých operací, jako je například násobení bodů na křivce, nativní funkce jazyka C.

Použití nativních funkcí

Jelikož velká část výpočtů na mobilním telefonu (i v chytrých hodinkách) se provádí v jazyce C za pomocí nativních funkcí, je vhodné nejdříve popsat způsob, jakým je toto v sytému implementováno. Všechny funkce, které aplikace používá, jsou definované v nativní knihovně native-lib. Funkce zadefinované v souboru native-lib je poté nutné definovat i ve třídě Javy, která je bude používat. Definice funkce se bude podobat definici funkce v rozhraní. Definice obsahuje jméno funkce, typ návratové hodnoty a výpis argumentů funkce, v poslední řadě musí každá taková definice obsahovat klíčové slovo native. Takováto definice v Javě bude vypadat například takto: `public native byte[] generateTk2(byte[] tv, int SecLevel);`, název i počet a typ argumentů musí odpovídat funci v knihovně native-lib. Příklad definice funkce jazyka C v knihovně native-lib a způsob vracení a načítání proměnných je zobrazen ve výpise 4.1. Na prvních 3 řádcích výpisu vidíme způsob definice C funkce. Na řádku 4 je vidět způsob, jakým se hodnoty z Javy zpracovávají pro počítání v jazyce C. Na řádku 6 je definice nového bodu tak, aby ho bylo možné použít pro počítání a uložení hodnot. Na řádku 7 je provedena operace sčítání bodů pomocí knihovny micro-ecc, všechny proměnné zde musí být předány do funkce jako proměnné typu `UECC_word_t`. První jsou předávány body, které mají být sečteny, jako poslední je předána proměnná, do které se má výsledek uložit. Na řádku 8 je definováno nové pole pomocí operátoru `env`, který umožní toto pole poté předat do Javy. Toto pole je poté naplněno hodnotami z proměnné `point1` a vráceno do Javy.

Výpis 4.1: Příklad nativní funkce v C (vypočítání bodu t)

```
1 extern "C"
2 JNIEXPORT jbyteArray JNICALL
3 Java_cz_vutbr_feec_watchwithmobile_EcOperations_generateT
    WithWatch2(JNIEnv *env, jobject /* this */,
  jbyteArray t1) {
4   jbyteArray ct1= reinterpret_cast<jbyteArray>(env->
  GetByteArrayElements(t1, NULL));
5   jbyte* point1;
6   point1= reinterpret_cast<jbyte *>(new uECC_word_t[
  nativeNCount * 2]());
```

```
7       uECC_point_add(reinterpret_cast<const uECC_word_t *>(
    ct1), reinterpret_cast<const uECC_word_t *>(randPoint2),
        reinterpret_cast<uECC_word_t *>(point1), curve);
8       jbyteArray newArray = env->NewByteArray(
    numOfBytesToGive); //size is numOfBytesToGive, it has
    been changed in signature verification function
9       env->SetByteArrayRegion(newArray, 0,numOfBytesToGive,
    point1);//release the array for Java to use
10      return newArray;
11  }
```

Při implementování protokolu a předávání hodnot do jazka C pomocí JNI se vyskytlo několik problémů. Prvním problémem bylo to, že body předané do jazyka C a hodnoty vracené do C neodpovídaly tomu, jaký by měl být výsledek operací. Po bližším zkoumání bylo zjištěno, že bajty bodů C knihovny micro-ecc se vrací v opačném pořadí, než jak tyto body reprezentuje Java knihovna Spongy Castle. Pro vypořádání s tímto problémem bylo vytvořeno několik funkcí v knihovně utils, které přeskládají bajty tak, aby jim správně rozuměla C knihovna, a poté je přeskládají zpět po vrácení do Javy. Jednotlivé funkce fungují jinak při použití různých křivek. Příklad části takové funkce, která je zavolána při použití 256bitové křivky, je uveden ve výpise 4.2. Zde je bajtové pole rozděleno na dvě půlky (souřadnice X a Y) a každá souřadnice je seřazena pozpátku.

Výpis 4.2: Funkce pro úpravu souřadnic bodů pro jazyk C.

```
1  byte[] newByte = new byte[64];
2  byte[] x = Arrays.copyOfRange(toFix, 0, 32);
3  byte[] y = Arrays.copyOfRange(toFix, 32, toFix.length);
4  for (int i = 0; i < 32; i++) {
5      newByte[i] = x[31 - i];
6  }
7  for (int i = 0; i < 32; i++) {
8      newByte[i + 32] = y[31 - i];
9  }
10 return newByte;
```

Podobná funkce musela být vytvořena i pro předávání čísel, která jsou použita jako skaláry pro násobení bodů. Tyto funkce se liší pro různé křivky, které jsou v systému použity. Výpis 4.2 obsahuje pouze funkci pro 256bitovou křivku, u 224 a 160bitových křivek nastal ještě problém, kdy se body z C vracely se čtyřmi bajty nul místo platných bajtů bodu. Proto je nutné, aby se z C do Javy vracelo např. 64 bajtů

místo 56 u 224bitové křivky, a z takového pole odstranit ještě čtveřice nul. Oproti 256bitové křivce se u menších křivek navíc přidává čtveřice nulových bajtů na konec každé souřadnice před předáním do C. Příklad předávání hodnot do jazyka C z Javy je zobrazen ve výpise 4.3. Zde můžeme na prvním řádku výpisu vidět převod typu `ECpoint` na pole bajtů. Dále je potřeba oříznout první bajt z pole, jelikož se jedná o prefix, který se v C nepoužívá. Na pátém řádku je zavolána C funkce, které se předávají upravené proměnné. Vrácený bod z C je poté na řádku 7 přeskládán a nakonec je zavolána funkce `getCompPointFromCord(byte[] point)`. Tato funkce dokáže ze souřadnic vytvořit objekt typu `ECPoint`, který pak může zase převést do stlačené zakódované podoby, tedy přidá prefix. Takový bajt může být potom použit pro ověření podpisu, jelikož je nyní ve stejné podobě, ve které ho používá PC aplikace při tvorbě hashe.

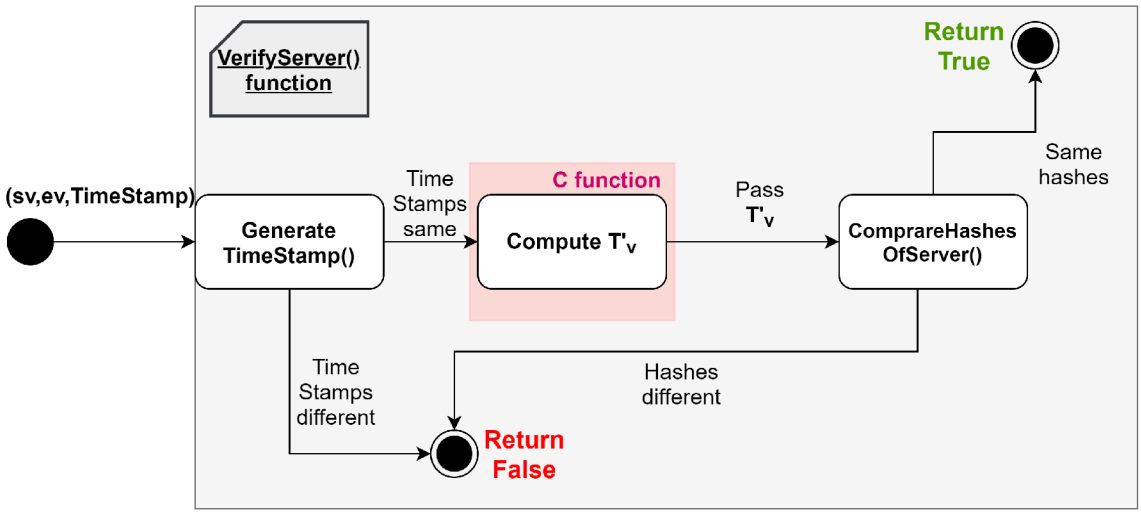
Výpis 4.3: Příklad volání C funkce a předávání proměnných do jazyka C

```
1 //converting uncompressed public key to byte array
2 byte[] pubCByte = PubServerEC.getEncoded(false);
3 pubCByte = Arrays.copyOfRange(pubCByte, 1, pubCByte.
     length);//cutting the prefix
4 //C function call with calls to functions that make Java
     points usable in C
5 tvC= verSignServer2(utils.FixForC32(sv),utils.FixForC64(
     pubCByte),utils.FixForC32(ev),Options.SECURITY_LEVEL);
6 //Fixing the point so it can be used in Java
7 tvC=utils.FixForC64(tvC);
8 //Converting the C point into compressed byte array
     representation of ECPoint
9 byte[] compTvC = getCompPointFromCord(tvC);
```

Ověření podpisu serveru

Prvním krokem aplikace mobilního telefonu během protokolu, před vypočítáním vlastního důkazu, bude vždy ověření podpisu serveru. Po obdržení tohoto podpisu přes rozhraní NFC je zavolána funkce `verifyServer(...)` třídy `ECOperations`, které jsou předána data obdržená od serveru. Diagram na obrázku 4.5 znázorňuje způsob, jakým ověřovací funkce postupuje.

Funkce `verifyServer(...)` nejdříve kontroluje validitu časové známky tím, že si sama vygeneruje časovou známku funkcí `GenerateTimeStamp()`, a zjistí, zda jsou známky stejné, popřípadě zda jsou stejné alespoň v rozmezí minuty. Poté přichází na řadu samotné ověření podpisu, výpočty potřebné k ověření se provádí v jazyce C.



Obr. 4.5: Stavový diagram funkce `verifyServer(...)`.

Z jazyka C se vrací bod t'_v. Poté je zavolána funkce `compareHashesOfServer(...)`, která vytvoří hash z bodu t'_v a z proměnné Y, a tento hash porovná s hodnotou e. Pokud jsou stejné, vrací funkce `verifyServer(...)` true a server je úspěšně ověřen.

Výpočet důkazu znalosti

Důkaz pro dvě zařízení se počítá spolu s chytrými hodinkami. Při výpočtu důkazu znalosti mobilní aplikace postupuje tímto způsobem:

1. Mobilní telefon vygeneruje pomocí nativní funkce náhodné číslo r_0 v rozsahu n použité eliptické křivky a poté pošle hodinkám přes Bluetooth hodnotu t'_v a bajt reprezentující použitou bezpečnostní úroveň.
2. Po přijetí bodů t_1 a t_{k2} od hodinek je zavolána funkce `GenerateProofWithWatch(...)`, která nejdříve spočítá hodnotu t pomocí nativní funkce v jazyce C, poté hodnotu t_k pomocí další nativní funkce a dále spočítá hodnotu e, která je hash zřetězení bajtů Y, t a t_k. Nakonec funkce spočítá hodnotu s_0, ta se počítá v Javě s pomocí typu `BigInteger` v modulu n eliptické křivky. Tento výpočet je velmi rychlý, a proto se hodnoty nepředávají do jazyka C, ale počítá se v Javě. Funkce v posledním kroku vytvoří první část zprávy, která bude předána jako důkaz znalosti, a tvoří ji hodnota e a s_0. Během funkce `GenerateProofWithWatch(...)` je také spočítán klíč, který může být poté použit pro šifrování pomocí blokové šifry AES nebo 3DES. Po dokončení této funkce se odešle hodnota e hodinkám, aby také mohly vypočítat svoji část důkazu.
3. Telefon po přijetí důkazu od hodinek přidá hodnotu s_1 na konec zprávy pro

server. Tuto zprávu, obsahující hodnoty e, s_0 a s_1, pošle serveru přes NFC. Pokud se uživatel ověřuje pouze jedním zařízením, tak mobilní telefon po ověření serveru počítá důkaz bez hodinek, a to voláním funkce `generateProofForSingle(...)` ze třídy ECOperations. Uvnitř této funkce se také používají pro výpočty nativní funkce.

4.4.3 Implementace komunikačních rozhraní na mobilním telefonu

Pro správnou funkčnost aplikace pro mobilní telefon v systému autentizace bylo nutné implementovat dvě komunikační rozhraní. První je komunikace pomocí NFC s PC aplikací, kde mobilní telefon využívá technologii HCE pro odesílání odpovědí na APDU příkazy. Druhým rozhraním je Bluetooth pro komunikaci s chytrými hodinkami při výpočtu společného důkazu.

NFC na Android telefonu

Android telefon během komunikace přes NFC zastupuje roli NFC tokenu, tedy chová se jako čipová karta s pomocí technologie HCE. Aby bylo možné použití HCE na mobilním telefonu, musí telefon disponovat technologií NFC a verzí Androidu 4.4 nebo vyšší. Také je u některých telefonů potřeba v nastavení NFC povolit HCE a dát systému Android povolení k vybrání aplikace pro odpověď, jinak se může stát, že telefon bude stále odpovídat výchozí aplikací. Při psaní programu je nejdříve potřeba upravit soubor AndroidManifest.xml a přidat oprávnění přistupovat k technologii NFC daného zařízení. Dále je nutné definovat AID, neboli identifikátor vyvíjené aplikace. Podrobný návod nastavení aplikace je dostupný na stránkách Android Developers[31].

Pro emulování karty je třeba vytvořit v Javě třídu, která dědí ze třídy `HostApduService`, a přetížit metodu `processCommandApdu()`, která zpracovává příchozí příkazy, a metodu `onDeactivated()`, ta říká co, se stane po ukončení spojení. Roli této třídy v programu zastupuje třída `MyHostAduService`. V metodě `processCommandApdu` bylo potřeba definovat odpovědi na očekávané příkazy ze strany čtecího zařízení. Pro rozlišení různých instrukcí byla vytvořena metoda `isCommand()` ve třídě `Utils`, která srovnává příchozí příkaz s některým z očekávaných příkazů. Metoda tedy rozlišuje různé kódy instrukcí a podle toho aplikace pozná, jak na daný příchozí příkaz odpovědět a jaká data jsou v něm například přiložena. Odpověď se odesílá pomocí příkazu return v těle metody `processCommandApdu` ve formě pole bajtů. Příklad odpovědi přes NFC je uveden ve výpise 4.4. Zde je vidět přetížení metody `processCommandApdu` a reakce na první zprávu, kterou je zpráva Select AID.

Na řádku 6 je zřejmá podmínka, kdy v případě, že zařízení nebylo registrováno, je vráceno pouze pole `A_OKAY`. Pokud je zařízení registrováno, vrací navíc svoje ID.

Výpis 4.4: Přetížení metody processCommandApdu a odpověď na Select AID

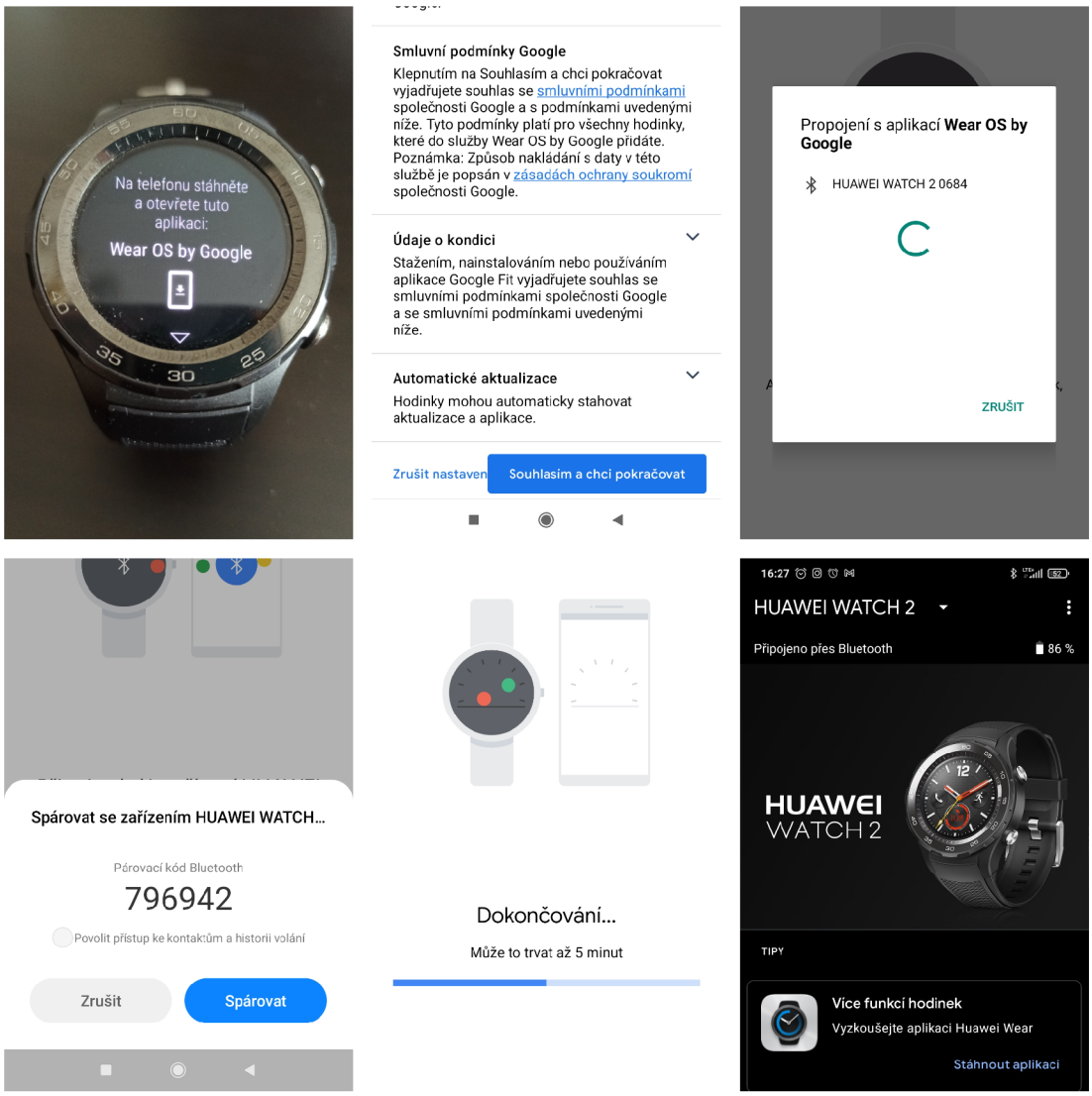
```
1  public byte[] processCommandApdu(byte[] commandApdu,
       Bundle extras) {
2      // First command: select AID
3      if (utils.isEqual(APDU_SELECT, commandApdu)) {
4          Log.i(TAG, "incoming commandApdu: " + utils.
       bytesToHex(commandApdu));
5          byte[] toGive=A_OKAY;
6          if (!Options.isRegistered)//if the phone is not
       registered it only returns OK
7              return A_OKAY;
8          try {
9          //if the phone is registered it will return its
       ID with OK
10             toGive=utils.appendByteArray(Options.MYID,
       A_OKAY);
11         } catch (IOException e) {
12             e.printStackTrace();
13         }
14         return toGive;
15 }
```

Bluetooth komunikace na telefonu

Druhým použitým komunikačním rozhraním je Bluetooth na mobilním telefonu, určené pro komunikaci s chytrými hodinkami s operačním systémem WearOS.

Pro zajištění přenosu dat bylo nejprve nutné vybrat vhodný způsob implementace Bluetooth komunikace pro potřeby aplikace. Byla zvolena komunikace pomocí Data Layer API a Message Client API, které zajišťují komunikaci mezi mobilními zařízeními Android a tzv. wearable zařízeními (chytré hodinky) s využitím Bluetooth a umožňují oboustrannou výměnu dat a snadnou implementaci různých reakcí podle cesty (path) příchozí zprávy. Aby bylo možné použití komunikace přes Data Layer, je nutné nejdřív propojit hodinky s mobilním telefonem pomocí mobilní aplikace WearOS. Toto propojení je potřeba i pro normální fungování hodinek s mobilním telefonem, dá se tedy předpokládat, že uživatel, který tyto hodinky používá spolu se svým telefonem je již bude mít spárované. Postup je velmi jednoduchý, chytré

hodinky, pokud nejsou spárované s jiným telefonem, vyzvou uživatele, aby si na mobilním telefonu stáhl aplikaci WearOS, tu si uživatel musí stáhnou z Google Play obchodu. Postup párování s hodinkami na mobilním telefonu je zobrazen na obrázku 4.6. Hodinky stačí mít zapnuté, případně na nich potvrdit výzvu ke spárování. Po dokončení tohoto párovacího procesu je vytvořen komunikační kanál mezi hodinkami a telefonem, který je možné využít v aplikaci.



Obr. 4.6: Párování WearOS hodinek s telefonem.

Komunikace implementovaná v rámci vyvíjené aplikace vychází z příkladu Jessicy Thronsby [38]. Největší výhodou tohoto způsobu komunikace mezi telefonem a chytrými hodinkami je to, že komunikační kanál je již vytvořen spárováním chytrých hodinek a telefonu v aplikaci WearOS.

Pro implementaci komunikace bylo třeba upravit Manifest soubor aplikace pro telefon, ale i pro chytré hodinky. Hlavní položkou je přidání služby MessageService a bylo třeba přidat všechny položky cest, které budou zprávy používat. Tato úprava Manifest souboru je zobrazena na obrázku 4.7, stejná úprava je provedena i v manifestu aplikace pro chytré hodinky.

```
<service
    android:name=".MessageService"
    android:enabled="true"
    android:exported="true" >
    <intent-filter>
        <action android:name="com.google.android.gms.wearable.MESSAGE_RECEIVED" />
        <data android:scheme="wear" android:host="*" android:pathPrefix="/path1" />
        <data android:scheme="wear" android:host="*" android:pathPrefix="/path2" />
        <data android:scheme="wear" android:host="*" android:pathPrefix="/path3" />
        <data android:scheme="wear" android:host="*" android:pathPrefix="/pathReset" />
        <data android:scheme="wear" android:host="*" android:pathPrefix="/pathRegister" />
    </intent-filter>
</service>
```

Obr. 4.7: Úprava souboru AndroidManifest.xml, pro funkci komunikace přes Bluetooth.

Uvnitř třídy `MyHostAduService` je definována vnořená třída `SendMessage`, která slouží k odesílání zpráv hodinkám přes Data Layer API. Aby nedošlo k blokaci hlavního vlákna, tak se pro odesílání zprávy vytvoří nové vlákno. Tímto vláknem je právě zmíněná třída `SendMessage`, která dědí z třídy `Thread`. Konstruktor třídy má dva parametry, cestu nové zprávy a zprávu ve formátu bajtů. Toto vlákno potom pomocí Wearable API získá dostupné zařízení a odešle zprávu. Pro odeslání je tedy potřeba vytvořit instanci z této třídy a zavolat metodu `run()`. Tato metoda je definovaná uvnitř třídy `SendMessage` a provádí samotný proces odesílání.

Třídou, která přijímá zprávy od hodinek, je třída `MessageService`, ta dědí z třídy WearableListenerService. Aby bylo možné zpracovávat obdržené zprávy, je třeba přetížit metodu `onMessageReceived(...)`. Metoda má jeden argument typu `MessageEvent`, ten obsahuje cestu zprávy a může obsahovat data. V této metodě jsou poté definované reakce na zprávy podle jejich cesty. Jelikož služba pro přijímání zpráv je spuštěna na jiném vláknu, než třída, která s přijatými hodnotami následně pracuje, musí zde, aby bylo možné tyto zprávy předat do třídy pro zpracování, být použit `LocalBroadcastManager`, který vysílá data v rámci dané aplikace. Tato třída neodesílá žádná data mimo aplikaci, ani žádná aplikace nemůže posílat tato vysílání do vytvořené aplikace, tudíž by neměla být bezpečnostní slabinou.

Posílání dat uvnitř aplikace

Výše bylo zmíněno použití třídy `LocalBroadcastManager` pro odesílání přijatých zpráv skrz aplikaci. Jelikož ve výsledné aplikaci je nutné zprávy posílat do třídy `MyHostAduService`, která se zároveň stará o odpovědi na APDU zprávy přijaté přes NFC, musela zde být použita modifikovaná verze třídy `LocalBroadcastManager` [39], která dokáže vytvořit přijímač (Reciver) na novém vlákně. Kdyby byl tento Reciver vytvořen na stejném vlákně jako služba pro přijímání APDU zpráv, nemusela by třída `MyHostAduService` tato vysílání vůbec zachytit.

Pro odeslání zprávy pomocí třídy `LocalBroadcastManager` musí být data nejdříve přidána do instance třídy Intent. Výpis 4.5 ukazuje jak je možné data přijatá přes Wearable API ve třídě `MessageService` předat v rámci aplikace do jiné třídy.

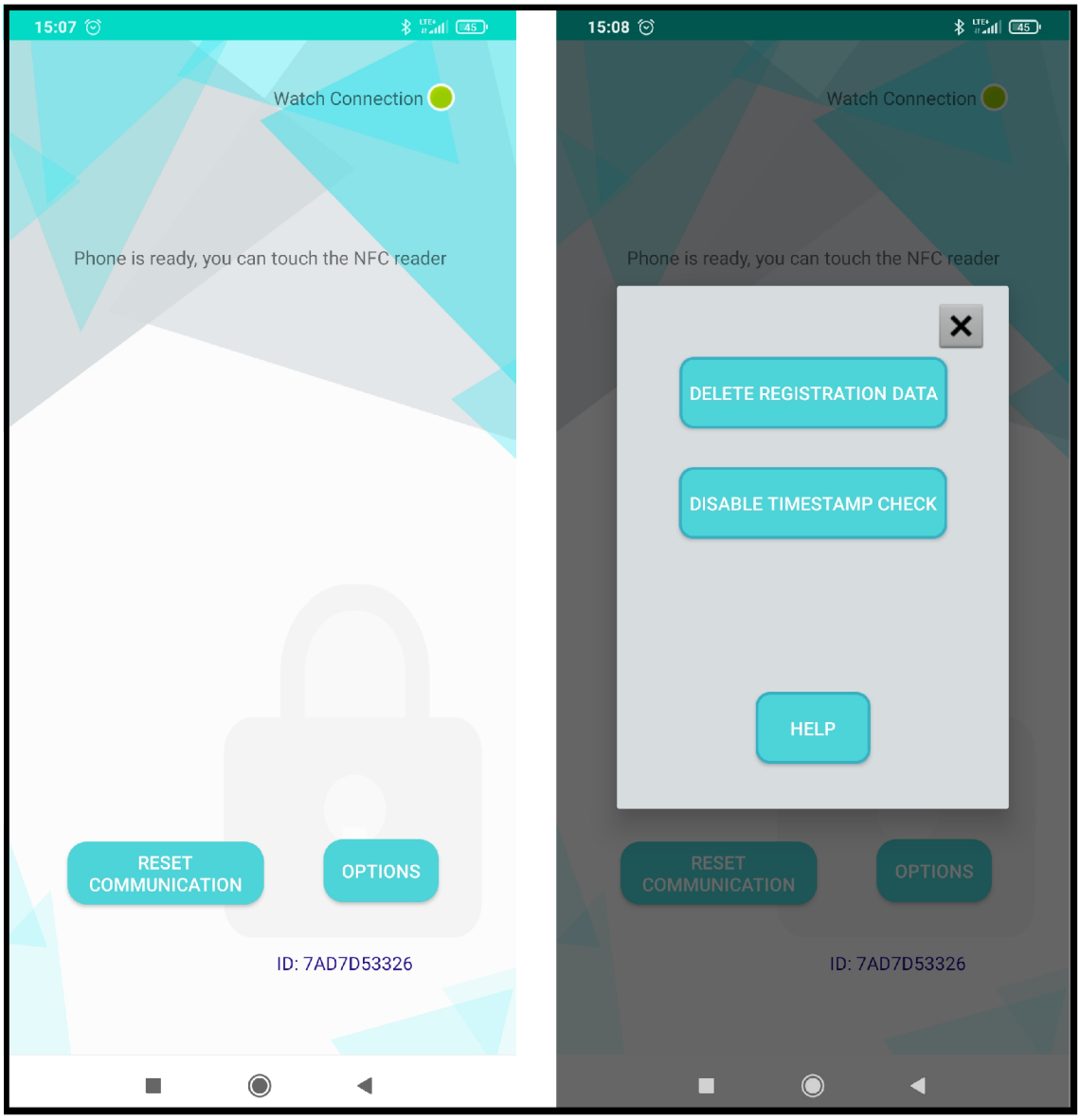
Výpis 4.5: Předání dat ze třídy MessageService skrz aplikaci

```
1  //function that has to be overwritten in messageService
       class
2  public void onMessageReceived(MessageEvent messageEvent)
       {
3      //path1 means we recieved Tk2 and T1 from watch
4      if (messageEvent.getPath().equals("/path1")) {
5          byte[] Tk2T1=messageEvent.getData();
6          Intent messageIntent = new Intent();
7          messageIntent.setAction(Intent.ACTION_SEND);
8          //put data in the intent
9          messageIntent.putExtra("data", Tk2T1);
10         //put path value in the intent, so receiver knows
       what data it is
11         messageIntent.putExtra("path", "1");
12         //send the broadcast
13         LocalBroadcastManager.getInstance(this).
       sendBroadcastSync(messageIntent);
14     }
```

Aby bylo možné tyto zprávy zachytit ve třídě `MyHostApduService`, je uvnitř této třídy nejdříve vytvořena definice vnořené třídy `Reciever`, která dědí ze třídy `BroadcastReceiver`. V této třídě se musí přetížit metoda `OnRecieve(...)`, uvnitř této metody jsou dále definované reakce na jednotlivé zprávy podle hodnoty path předané v Intentu. Když je definice hotova, je nutné tento `Reciever` registrovat při vytvoření instance třídy `MyHostApduService`.

4.4.4 GUI a obsluha aplikace

Obsluha aplikace pro mobilní telefon je vcelku jednoduchá. Vzhled aplikace po spuštění můžete vidět na obrázku 4.8. Aplikace je hned po spuštěné připravena k prove-



Obr. 4.8: Výsledný vzhled aplikace pro Android.

dení autentizace nebo registrace. Indikátor v pravém horním rohu aplikace signalizuje, zda se mobilní telefon dokázal připojit k hodinkám. Pro funkčnost aplikace s hodinkami je potřeba mít zapnuté rozhraní Bluetooth.

Tlačítko **Reset Communication** restartuje komunikaci s hodinkami, přenastavením hodnot ve třídě `Conditions` a odesláním zprávy o restart hodinkám. Tlačítko **Options** otevře dialogové okno s několika možnostmi. Možnost **Delete**

registration data smaže data o registraci v mobilním telefonu a telefonem se bez nové registrace nebude možné přihlásit do systému. Zvolením **Disable Timestamp Check** se vypíná kontrola časové známky v podpisu serveru. Pokud totiž nejsou časy na PC i mobilním telefonu nastaveny stejně, nebyla by časová známka ověřena a po neúspěšném ověření serveru by autentizace neproběhla. Po vypnutí kontroly časové známky je možné například systém testovat na virtuálním stroji s jiným nastavením času. Poslední možnost **Help** ukáže nové dialogové okno s několika radami, jak lze vyřešit některé problémy s nefunkční aplikací. Ve spodní části aplikace je vypsána hodnota ID, ta bude vypsána pouze, pokud bylo zařízení registrováno a bylo mu ID přiděleno. Pokud například registrace neproběhne úspěšně a zařízení má již přiřazenou hodnotu ID, je třeba v nastavení smazat registrační data, aby bylo možné přidělit nové ID.

4.5 Aplikace pro chytré hodinky

Aplikace pro chytré hodinky vychází v mnohém z aplikace pro chytrý telefon. Byla vyvíjena v prostředí Android studia v jazyce Java a s použitím volání nativních funkcí jazyka C. Aplikaci pro chytré hodinky lze jednoduše vytvořit jako modul aplikace WearOS v projektu Android Studia.

4.5.1 Struktura aplikace pro chytré hodinky

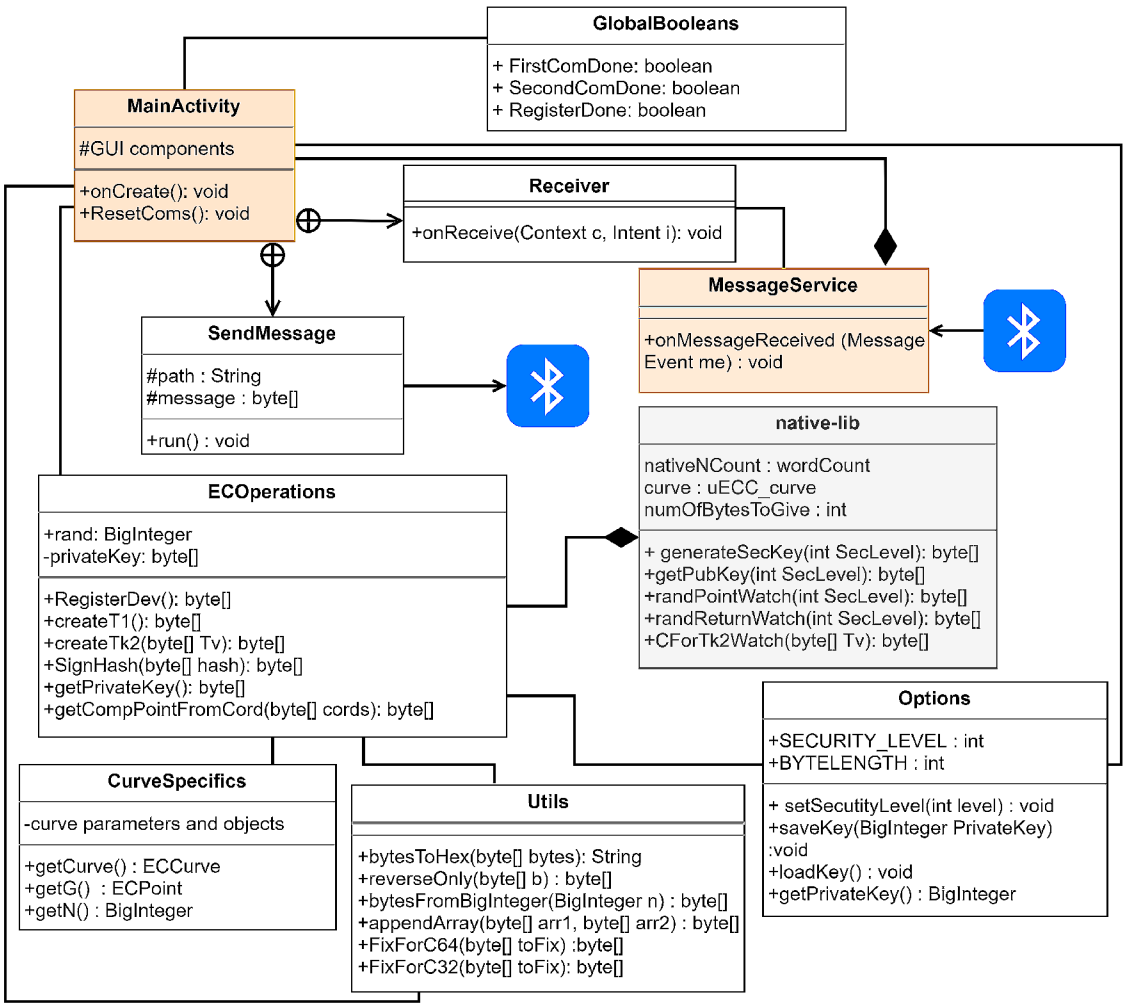
Struktura aplikace pro chytré hodinky se podobá struktuře mobilní aplikace, pouze je zde použit menší počet tříd, jelikož aplikace pro chytré hodinky nepotřebuje umět tolik funkcí jako aplikace pro telefon. Na obrázku 4.9 je zobrazen diagram tříd aplikace. Z obrázku lze vyčíst, jaké třídy spolu navzájem komunikují, a ve které třídě se nacházejí jaké funkce.

Třída MainActivity

Hlavní třída `MainActivity` se stará o obsluhu GUI aplikace a také jsou zde zpracovávány zprávy přijaté ze třídy `MessageService`. Jsou odsud posílány zprávy mobilnímu telefonu a volají se odsud funkce třídy `ECOperations` pro výpočty během protokolu.

Třída ECOperations

Třída, podobně jako na mobilním telefonu, provádí výpočty potřebné při důkazu znalosti. I u chytrých hodinek se odsud volají nativní funkce pro počítání s body na eliptické křivce.



Obr. 4.9: Diagram vytvořených tříd v aplikaci pro chytré hodinky.

Třída MessageService

Třída pro přijímání zpráv přes Bluetooth od mobilního telefonu, zprávy se následně posílají v rámci aplikace do třídy `MainActivity`.

Třída GlobalBooleans

I zde je při komunikaci použita třída, která uchovává statické proměnné typu boolean pro ujištění, že je každá komunikace provedena pouze jednou za běhu protokolu. Touto třídou je třída `GobalBooleans`. Když toto nebylo na hodinkách ošetřené, stalo se například, že byly hodinky vyzvány, aby si vygenerovaly soukromý klíč a odeslaly klíč veřejný. Tato zpráva se do hlavní aktivity dostala vícekrát a hodinky tuto generaci klíčů provedli několikrát. Poté se lišil první klíč, který hodinky odeslaly a poslední klíč, který si uložily a protokol nefungoval.

4.5.2 Implementace protokolu na hodinkách

Hodinky se do protokolu připojují ve chvíli, kdy jsou vyzvány mobilním telefonem k vypočítání hodnot t_1 a t_{k2}. Tato zpráva je zachycena ve třídě `MessageService` a přeposlána do třídy `MainActivity`. Odsud jsou poté volány funkce třídy `ECOperations` pro výpočet hodnot t_1 a t_{k2}, ty jsou počítány za pomoci nativních funkcí jazyka C. Tyto hodnoty jsou následně ze třídy `MainActivity` poslány mobilnímu telefonu přes Bluetooth a hodinky čekají na další zprávu.

Po přijetí další zprávy od telefonu s hodnotou e ve třídě `MessageService` a jejím přeposlání do třídy `MainActivity` je zavolána funkce pro spočítání s_1, která je definována ve třídě `ECOperations`. Tato hodnota je poté odeslána telefonu a tím účast hodinek v protokolu končí.

4.5.3 Komunikační rozhraní na chytrých hodinkách

Chytré hodinky komunikují pouze s mobilním telefonem přes Bluetooth pomocí Data Layer API. Způsob komunikace je zde stejný jako na mobilním telefonu. Třída pro příjem zpráv `MessageService` je prakticky stejná jako na mobilním telefonu, pouze jsou zde definované reakce na jiné hodnoty cest zpráv. Tyto zprávy se zde také přeposílají pomocí třídy `LocalBroadcastManager` do třídy `MainActivity`, v té je poté definice přijímače těchto vysílání a vnořené třídy `SendMessage` pro odesílání zpráv, podobně jako ve třídě `MyHostAPDUService` u mobilního telefonu.

4.5.4 Obsluha a vzhled aplikace pro chytré hodinky

Obr. 4.10: Výsledný vzhled aplikace pro chytré hodinky.

Vzhled aplikace pro chytré hodinky je vcelku minimalistický. Aplikace obsahuje jedno tlačítko pro manuální restart komunikace a poté textové pole, které uživatele

informuje o dokončení komunikace nebo o restartu. Obsluha hodinek je velmi jednoduchá, pro účast v protokolu stačí mít zapnutou vytvořenou autentizační aplikaci a mít hodinky spárované s mobilním telefonem v aplikaci WearOS. Pokud je potřeba resetovat komunikaci po dokončení protokolu, lze tento restart provést i z mobilního telefonu, od kterého hodinky poté obdrží zprávu o restartu.

4.6 PC aplikace

PC aplikace je stěžejní aplikací celého systému. Jedná se o přístupovou aplikaci. Aplikace pro PC byla vyvíjena s pomocí Javy v prostředí IntelliJ IDEA. Výsledná aplikace umožňuje přidávat a odebírat uživatele zastoupené mobilním telefonem, přidávat a odebírat sekundární zařízení pro autentizaci, měnit bezpečnostní úroveň. Hlavní funkcí aplikace je autentizace pomocí jednoho nebo více zařízení. Při použití autentizace jedním zařízením je potřeba uživatele navíc ověřit heslem, zatímco u autentizace více zařízeními to nutné není.

4.6.1 Struktura PC aplikace

Aplikace je rozdělena do několika tříd, zde bude stručně popsáno, k čemu jednotlivé třídy slouží. Na obrázku 4.11 je zobrazen diagram tříd PC aplikace. Jsou zde vidět jednotlivé třídy a metody, také je zde znázorněno, jaké třídy spolu komunikují.

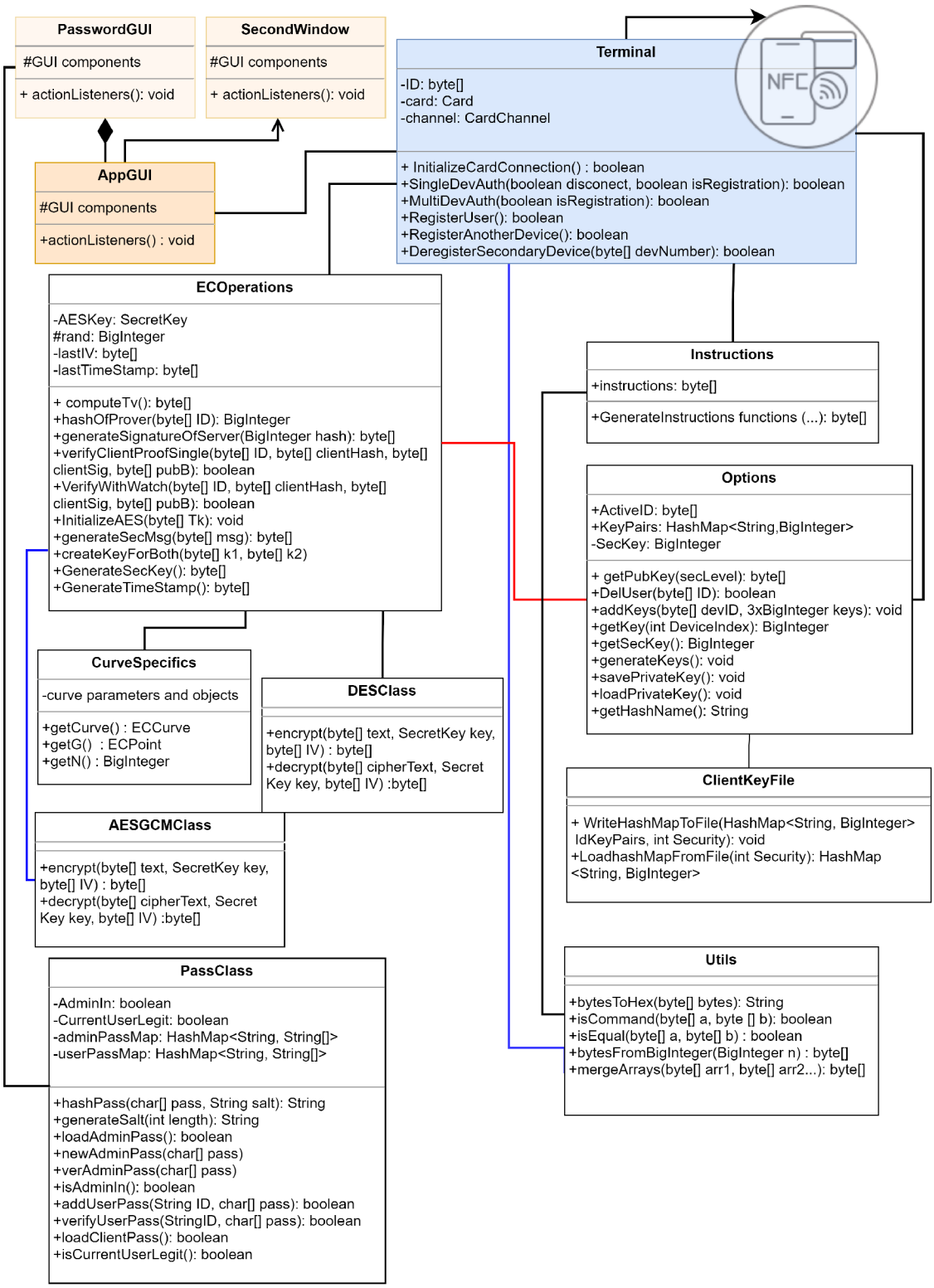
Třída Terminal

Třída `Terminal` je nejdůležitější třídou celé aplikace. Jsou zde definované metody pro ověření uživatele nebo pro registraci nových uživatelů. V této třídě se provádí veškerá komunikace přes NFC s mobilním telefonem. Funkce této třídy jsou volány z třídy, obstarávající GUI. Ze třídy `Terminal` jsou volány funkce třídy `ECOperations` pro provedení výpočtů, které jsou potřeba k ověření uživatele nebo serveru.

Mezi nejdůležitější metody této třídy patří metoda `InitializeCardConnection(...)`, která navazuje spojení s přes NFC, dále metoda `SingleDevAuth(...)`, která zajišťuje autentizaci pomocí jednoho zařízení a metoda `MultiDevAuth(...)`, díky které je možné ověřit uživatele pomocí více zařízení.

Třída ECOperations

Tato třída provádí veškeré výpočty při běhu protokolu, a to za pomoci knihovny Bouncy Castle [40]. Jsou zde definované například metody pro výpočet podpisu serveru nebo pro ověření důkazu znalosti uživatele.



Obr. 4.11: Diagram tříd vytvořené PC aplikace.

Třída Instructions

Třída Instructions uchovává všechny proměnné, které se používají jako APDU příkazy. Dále se zde také nacházejí metody pro stavbu těchto instrukcí. Většinou se vybere instrukce, kterou potřebuje server odeslat a předají se některé metodě v této třídě data, která mají být odeslána v této instrukci. Metoda potom spojí požadovaný příkaz s daty a vrací ho do třídy terminálu.

Třída Options

Tato třída v PC aplikaci v první řadě uchovává informaci o aktuálním bezpečnostním nastavení ve statické proměnné `SECURITY_LEVEL`. Na hodnotu této proměnné se dotazuje program za běhu vždy, když provádí jakoukoli operaci, která se liší vzhledem k nastavené bezpečnosti. Tedy například při načítání klíče se metoda pro načtení nejdříve ptá na hodnotu této proměnné a podle toho potom ví, jaký klíč načíst.

Příklady dalších metod v této třídě jsou například metody pro načítání a ukládání soukromého klíče, metoda pro vytvoření a vrácení veřejného klíče ověřovatele, metody pro mazání uživatelů a jejich klíčů a metoda pro vracení veřejných klíčů uživatelů.

Třída CurveSpecifics

Třída `CurveSpecifics`, podobně jako na mobilním telefonu, uchovává parametry používaných eliptických křivek. Třída tyto parametry vrací podle aktuálního nastavení stupně bezpečnosti ve třídě `Options`.

Třída PassClass

Třída `PassClass` zajišťuje práci s hesly. A to jak ověřování tak ukládání hesel. Hesla jsou ukládána ve formě osolených hashů, spolu s hodnotou soli. Díky tomu by měly být uložené hashe odolné vůči útoku pomocí duhových tabulek (rainbow tables).

Třída Utils

Tato třída obsahuje různé metody, většinou pro převod datových typů. Jsou zde například metody pro převod bajtových polí na `string`, nebo funkce k převádění typu `BigInteger` na bajtové pole.

Třídy pro AES a DES

Posledními třídami, které je potřeba zmínit, před třídami, které tvoří GUI, jsou třídy `AESGCMClass` a `DESClass`. Tyto třídy zajišťují šifrování a dešifrování pomocí

blokových šifer AES a TripleDES. Samotné klíče pro tyto šifry se vytvářejí během protokolu ve třídě `ECOperations` a jsou metodám těchto tříd předávány v argumentech.

Třídy GUI

Pro běh programu jsou klíčové třídy, které tvoří a zajišťují obsluhu GUI. GUI bylo tvořeno v Swing UI Designeru přítomného v prostředí IntelliJ IIEA, pomocí GUI forms. Každé okno se zde skládá ze dvou souborů. První je soubor třídy, tam jsou definované reakce na stisk různých tlačítek a reakce na různé události. Druhým souborem je form, kde je pomocí nástroje designeru vytvořeno samotné grafické uživatelské rozhraní.

V aplikaci jsou celkem definovaná čtyři okna, a to hlavní okno aplikace, okno pro přihlášení, okno pro odstranění uživatele a okno, které se otevře po úspěšném dokončení autentizace. Zde bude popsáno v první řadě fungování hlavního okna, které je nejdůležitější.

Třída hlavního okna se v programu jmenuje `AppGUI`, toto okno je vytvořeno při spuštění v metodě `main`. Je to jediná věc, která se v metodě `main` vykonává, zbytek už je definován ve třídách jednotlivých oken. Ve třídě jsou definovány reakce na zmáčknutí tlačítek, u některých tlačítek se například volají metody třídy `Terminal`, k provedení autentizace. Aby bylo možné tyto metody volat, bez blokace hlavního vlákna, musí být každé volání těchto metod provedeno na novém vlákně. K tomu se zde používají vlákna typu `SwingWorker`, v těch je poté taky možné bezpečně upravovat GUI.

4.6.2 Implementace protokolu na straně ověřovatele

Při autentizaci pomocí více zařízení se volá metoda `MultiDevAuth(...)`, při autentizaci jedním zařízením je to metoda `SingleDevAuth(...)`. Obě metody se nachází ve třídě `Terminal`. Funkce pro výpočty během protokolu tyto metody volají ze třídy `ECOperations`. Program během protokolu postupuje tímto způsobem:

1. Server pošle zprávu přes NFC pro výběr aplikace s kterou chce komunikovat.
2. Server dostane přes NFC odpověď od telefonu s ID uživatele, poté server spočítá svůj podpis, pro dokázání své identity. Při počítání je nejdříve zavolána funkce `hashOfProver`(...) ta spočítá hodnotu e_v postupem, který byl zobrazený na obrázku 4.3. H je zde hashovací funkce, tou je v základním nastavení systému SHA-256. Bod t_v se hashuje jako stlačený (compressed), tudíž je převedený na typ `byte[ ]` z typu `ECPoint`, který se používá při počítání. Po vrácení hodnoty e_v se volá funkce `generateSignatureOfServer(...)`, jejím vstupním parametrem je právě hodnota e_v. Tato funkce spočítá podpis s_v,

ten se již počítá v modulární aritmetice, tudíž jsou zde k výpočtům použity proměnné typu `BigInteger`, funkce vrací hodnotu e_v v podobě typu `byte[ ]`.

3. Po spočítání podpisu je poslaný přes NFC ServerHello, ten se skládá z údajů o aktuální úrovni bezpečnosti, z hodnoty Y a podpisu serveru, kterým je dvojce e_v a s_v.
4. Po obdržení důkazu od mobilního telefonu přichází na řadu proces ověření důkazu. Při autentizaci dvěma zařízeními je po obdržení důkazu znalosti na PC zavolána funkce `CreateKeyForBoth(...)`. Ta vytvoří společný veřejný klíč obou zařízení potřebný k ověření důkazu a to sečtením těchto klíčů jako dvou bodů na eliptické křivce. Poté je zavolána funkce `VerifyWithWatch(...)`, té se předávají data přijatá ve zprávě od klienta, potřebná k ověření důkazu znalosti. Tato funkce nejdříve spočítá body t' a t'_k postupem, který byl zobrazen na obrázku 4.3. Poté vytvoří hash ze zřetězení bajtů hodnoty Y a bodů t', t'_k. Body se předávají ve stlačené podobě. Nakonec server zjistí, jestli se tento hash rovná hodnotě e, kterou dostal od uživatele. Pokud ano, vrací funkce hodnotu true a uživatel je vpuštěn do systému. Při ověření důkazu znalosti pro jedno zařízení se po přijetí důkazu od mobilního telefonu zavolá funkce `verifyClientProofSingle()`.

4.6.3 Implementace komunikace přes NFC na PC

Aplikace pro PC zastupuje při NFC komunikaci roli terminálu. Prvním krokem při implementaci NFC komunikace byla instalace ovladačů pro čtečku karet. Čtečka karet, která byla během testování komunikace a systému použita je ACR122U od firmy ACS, ovladače s popsaným postupem instalace pro Windows i Linux jsou dostupné z [41].

Komunikaci přes NFC obstarává třída `Terminal`. Java knihovna, která poskytuje funkce pro odesílání příkazů přes NFC, je kinohovna smartcardio. Pro zahájení komunikace s mobilním telefonem, který se tváří jako čipová karta, je ve zmíněné třídě vytvořena metoda `InitializeCardConnection()`. Program se nejdříve musí spojit se čtečkou a poté čeká na přiložení karty, to lze vidět ve výpise 4.6 na řádku 6. Po přiložení karty a navázání spojení (řádky 8 a 9 výpisu) je zaslán první nejdůležitější příkaz Select AID, který telefonu říká, s jakou aplikací chce komunikovat. Odeslání tohoto příkazu je na řádku 13 výpisu 4.6.

Výpis 4.6: Propojení se čtečkou a odeslání příkazu přes NFC.

```
1 TerminalFactory factory = TerminalFactory.getDefault();
2 List<CardTerminal> terminals = factory.terminals().list
    ();
3 CardTerminal terminal = terminals.get(0);
```

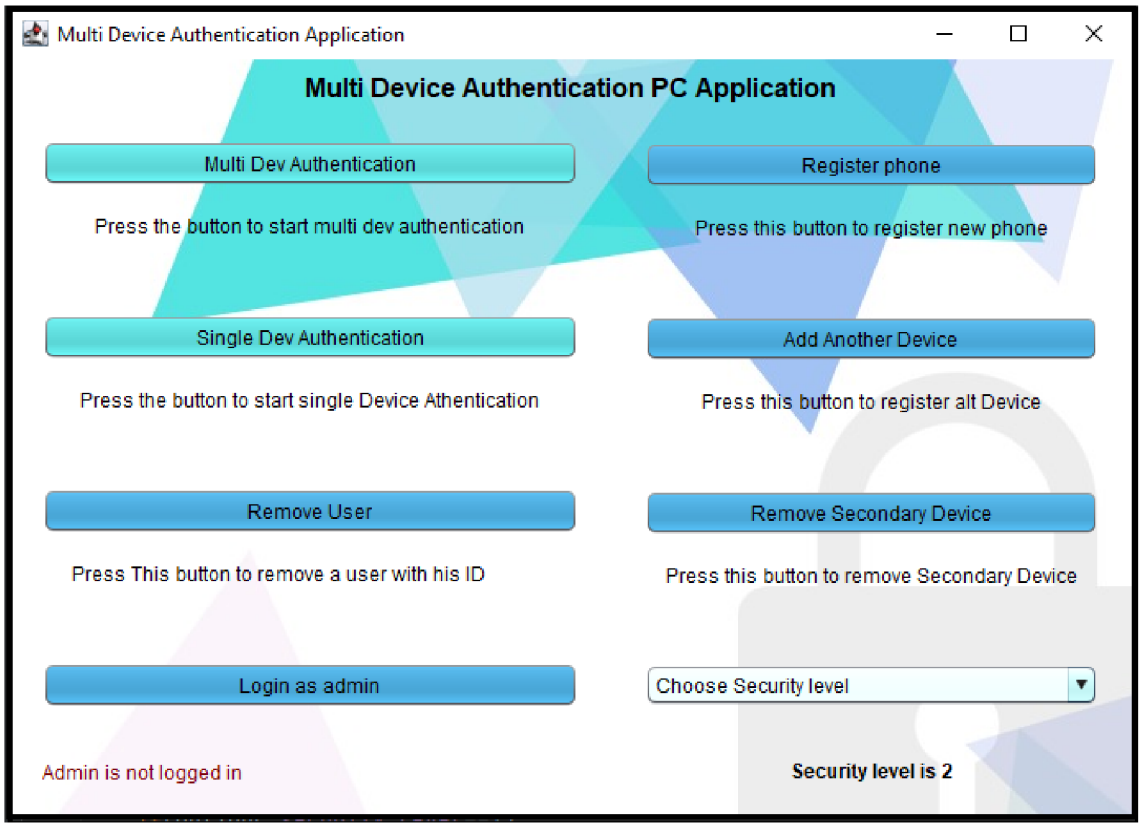
```
 try {
 //terminal is waiting for a card
    while (!terminal.isCardPresent()) ;
    //try to connect to a card
    card = terminal.connect("*");
    channel = card.getBasicChannel();
} catch (CardException e) {
    e.printStackTrace();
}//when the connection with a card has been established
   selectAID command is sent
ResponseAPDU response1 = channel.transmit(new CommandAPDU
   (instructions.getAID()));
byte[] byteResponse1 = null;
byteResponse1 = response1.getBytes();
```

Stejným způsobem je poté možné posílat další příkazy přes NFC. Tyto příkazy jsou v aplikaci definovány ve třídě `Instruction`. Pro správnou funkčnost aplikace je žádoucí, aby aplikace na mobilním telefonu dokázala zpracovat všechny instrukce, které terminál posílá.

4.6.4 Obsluha a GUI PC Aplikace

Na obrázku 4.12 je vidět výsledný vzhled základní obrazovky aplikace pro PC. Aplikace je spustitelná na počítačích s verzí Javy 11 a vyšší, měla by fungovat jak na zařízeních s operačním systémem Windows, tak i na systémech s Linuxem. Pro správnou funkčnost aplikace je nutné mít k počítači připojenou čtečku karet a mít nainstalovány ovladače pro tuto čtečku. Použití tohoto systému v praxi si lze představit například jako přístupovou aplikaci k ovládání reaktoru v elektrárně.

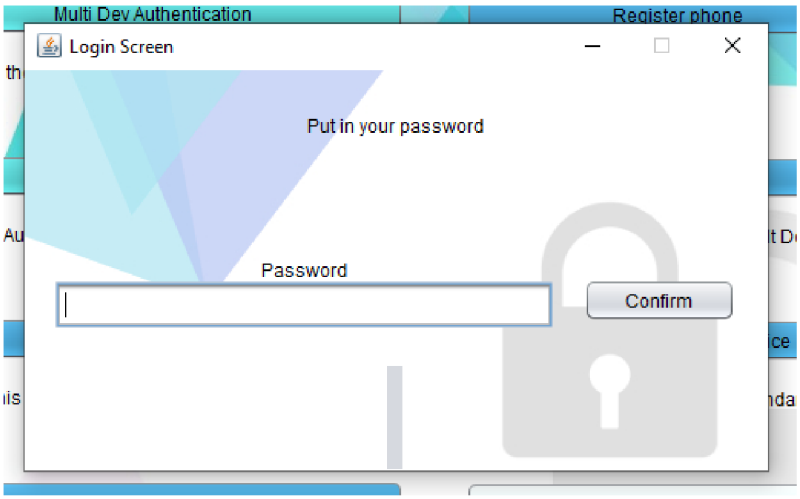
Při prvním spuštění je uživatel vyzván k vytvoření administrátorského hesla, toto heslo se poté používá pro přihlášení administrátora systému. Role administrátora je v systému proto, že pouze administrátor může registrovat nové uživatele. Po vytvoření hesla administrátora je možné se přihlásit jako administrátor tlačítkem **Login as admin** a zadáním hesla. Pokud je administrátor přihlášen, je možné registrovat uživatele, disponující telefony s Android aplikací pro autentizaci. Po stisku tlačítka **Register phone** stačí přiložit telefon se spuštěnou aplikací a měla by proběhnout registrace. Pro dokončení registrace si uživatel ještě musí zvolit své heslo. V dalším kroku je možné si k mobilnímu telefonu přidat sekundární zařízení, jako například chytré hodinky, k tomu slouží tlačítko **Add Another Device**. Poté stačí přiložit telefon, který je spárovaný s chytrými hodinkami, na kterých je také spuštěna autentizační aplikace. Po obdržení klíčů a provedení testovací autentizace je takto k



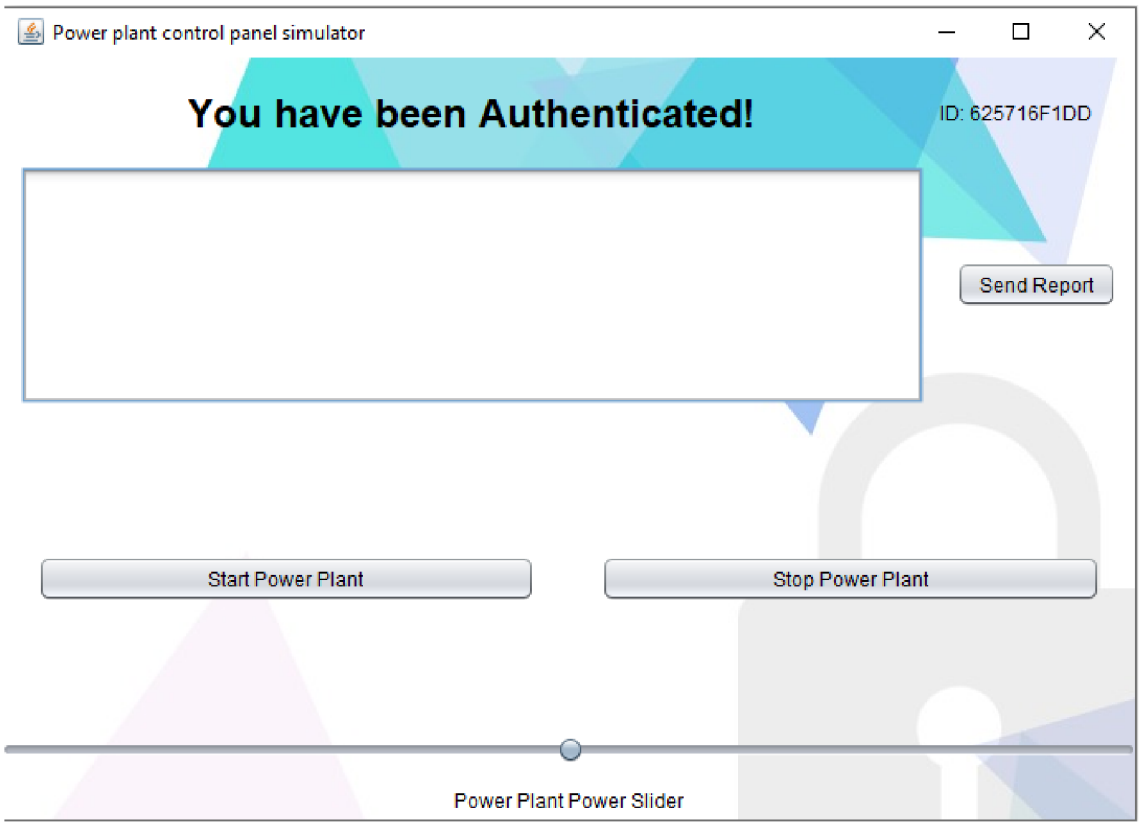
Obr. 4.12: Hlavní okno aplikace pro PC.

uživateli přidáno další zařízení. Tlačítkem **Remove Secondary Device** lze poté takové zařízení odebrat přiložením telefonu, od kterého ho chceme odebrat. Možností **Remove User** lze smazat uživatele podle hodnoty jejich ID.

V nabídce **Choose Security level** je možné zvolit bezpečnostní úroveň, kterou bude aplikace při autentizaci používat. Pro samotnou autentizaci uživatele slouží tlačítka **Multi Dev Authentication** a **Single Dev Authentication**. Pomocí **Multi Dev Authentication** se uživatel přihlašuje více zařízeními a pro vpuštění do systému není třeba zadávat heslo, stačí pouze přiložení telefonu, který je propojen a zaregistrován s hodinkami. U autentizace jedním zařízením uživateli stačí telefon, ale musí navíc zadat i svoje heslo, zadávání hesla je ukázáno na obrázku 4.13. Po úspěšné autentizaci se uživateli otevře okno aplikace, ke které uživatel žádá přístup. Tato aplikace je pouze demonstrativní a ukazuje, jak by mohl být takový systém integrován v praxi. Finální obrazovku po přihlášení do systému lze vidět na obrázku 4.14.



Obr. 4.13: Výzva pro zadání hesla v aplikaci



Obr. 4.14: Odemčená aplikace po přihlášení uživatele

4.7 Srovnání časů vykonání protokolu

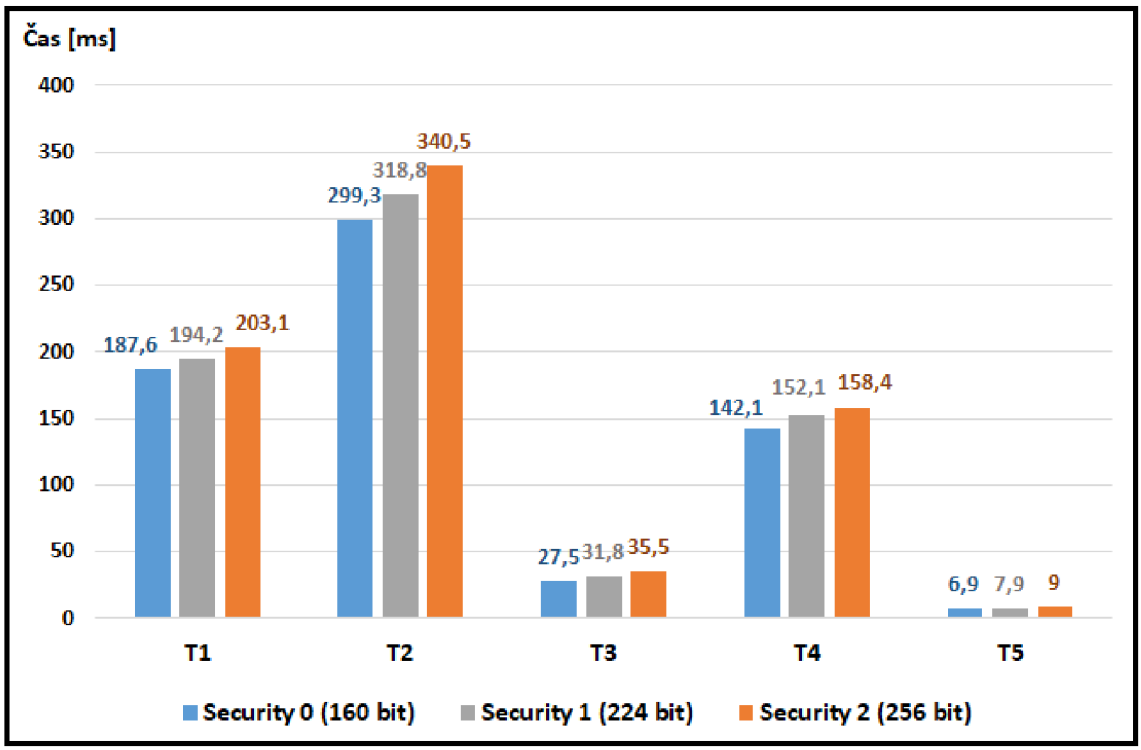
V závěru praktické části této práce byly proměřeny časy různých úseků finálního protokolu pro zjištění rozdílů u různých eliptických křivek a hashovacích funkcí a také

pro ověření, zda by byla rychlost protokolu s použitými komunikačními rozhraními dostačující v reálném systému.

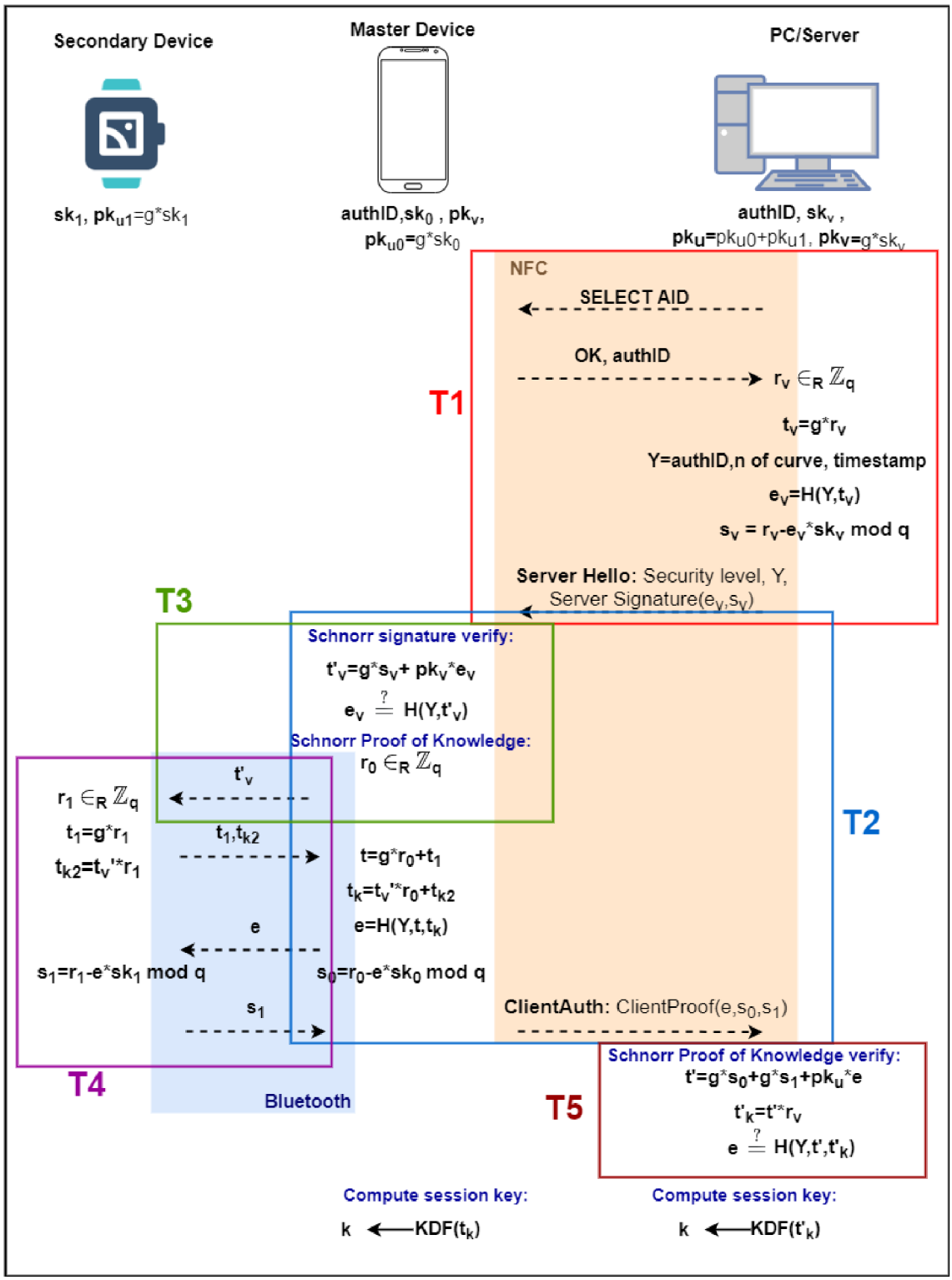
Všechny časy byly měřeny desetkrát. Pro měření byla použita tato zařízení: **PC**- HP Pavilion 15-bc5xxx, Intel i5-9300H (4cores, 2,4-4,1GHz), 16GB RAM, Windows 10 Home 20H2, **mobilní telefon** - Xiaomi Redmi note 8 pro, procesor Mediatek Helio G90T (2x2.05 GHz/6x2.0GHz), RAM 6GB, Android 10, **chytré hodinky** - Huawei Watch 2 4G LTE (LEO-DLXX), procesor Qualcomm Snapdragon 2100 (4 core, up to 1,2Ghz), 768MB RAM, Wear-OS 2. Aby bylo zřetelné, v jakých úsecích byly v protokolu jednotlivé časy měřeny, byly tyto úseky zaznačeny do schématu protokolu na obrázku 4.16.

	T1	T2	T3	T4	T5	Celkový čas protokolu
Security 2	203,1 ms	340,5 ms	35,5 ms	158,4 ms	9 ms	752,2 ms
Security 1	194,5 ms	318,8 ms	31,8 ms	152,1 ms	7,9 ms	709,6 ms
Security 0	187,6 ms	299,3 ms	27,5 ms	142,1 ms	6,9 ms	675,4 ms

Tab. 4.3: Časy protokolu pro různé nastavení bezpečnostních parametrů.



Obr. 4.15: Graf časů jednotlivých částí protokolu pro různé nastavení bezpečnosti.



Obr. 4.16: Protokol s vyznačenými měřenými časy.

Z tabulky 4.3 a grafu na obrázku 4.15 je patrné, že protokol je skutečně rychlejší pro nižší nastavení bezpečnosti, které používá menší eliptickou křivku a starší hasho-

vací funkci. Z grafu lze vyčíst, že rozdíl časů na těchto zařízeních mezi bezpečností úrovni 2 a bezpečností úrovni 0 není tak velký, aby se v praxi vyplatilo používat méně bezpečnou úroveň 0. V některých časech je zahrnut i čas komunikace. Například v čase T1 je zahrnuta jedna komunikace tam i zpět přes rozhraní NFC, kdy PC posílá SELECT AID a mobilní telefon na tuto zprávu odpovídá. Čas T2 kromě ověření podpisu serveru a spočítání důkazu obsahuje i dvě komunikace tam a zpět přes rozhraní Bluetooth s chytrými hodinkami.

Výsledné časy se tedy hodně odvíjí od časů samotné komunikace. Výpočty v systému zaberou menší část celkového času. Časy komunikace můžou být hodně proměnlivé, například doba, kterou trvá první komunikace přes NFC, kdy PC posílá SELECT AID, se pohybuje v rozmezí 140 až 270 ms. Po bližším zkoumání a měření časů jednotlivých výpočtů v protokolu bylo také zjištěno, že odeslání zprávy přes Bluetooth a doručení na druhé zařízení trvá během protokolu přibližně 50 ms. V případě NFC komunikace odeslání a doručení zprávy trvalo průměrně 95 ms.

Závěr

Cílem bakalářské práce bylo vytvořit systém, který umožní přihlášení uživatele mobilním telefonem spolu s chytrými hodinkami. Tohoto cíle se podařilo dosáhnout. Byl vytvořen autentizační systém, který umožňuje přihlášení uživatele pomocí mobilního telefonu s operačním systémem Android spolu s chytrými hodinkami s operačním systémem WearOS. Autentizace probíhá za pomoci protokolu vycházejícího ze Schnorrova protokolu a využívá kryptografie nad eliptickými křivkami. Systém také obsahuje funkční registraci, nástroj pro odebrání zařízení uživatele ze systému a možnost výběru ze tří různých bezpečnostních úrovní. Celkem byly vyvinuty tři aplikace, tedy jedna pro každé zařízení (PC, mobilní telefon, chytré hodinky). PC aplikace slouží jako terminál pro přihlašování uživatele, zbylé dvě aplikace mají v systému roli autentizačního tokenu uživatele. V systému je použita komunikace přes rozhraní NFC a přes Bluetooth. Na mobilních zařízeních se podařilo z jazyka Java volat nativní funkce jazyka C, které poskytují větší rychlost výpočtů při některých operacích, jako je násobení bodu na eliptické křivce.

Zatím autentizační systém odemyká pouze aplikace demonstrativní, v budoucnu by mohla být práce rozšířena o specifickou funkční aplikaci, ke které by systém umožňoval přístup. Touto aplikací by mohla být například webová služba, jako je online bankovnictví. Při takovémto použití by mohl být autentizační systém rozšířen také o různé role a práva uživatelů.

Literatura

[1] Nakamoto. *Hash Functions. Nakomoto* [online]. 2019-12-29 [cit. 2021-5-8]. Dostupné z: <https://nakamoto.com/hash-functions/>.

[2] TutorialsPoint. *Cryptography Hash functions. TutorialsPoint* [online]. 2015-09-30 [cit. 2021-5-8]. Dostupné z: <https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm>.

[3] GRIMES, Roger. *Why aren't we using SHA-3? CSOonline* [online]. 2018-08-21 [cit. 2021-5-8]. Dostupné z: <https://www.csoonline.com/article/3256088/why-arent-we-using-sha3.html>.

[4] MENEZES, Alfred, Paul C. VAN OORSCHOT a Scott A. VANSTONE. *Handbook of Applied Cryptography: Discrete Mathematics and Its Applications.* Boca Raton: CRC Press, 1997, s. 191-192. ISBN 0-8493-8523-7.

[5] SIMPLILEARN. *What is DES?: Understanding DES Algorithm and Operation. Simplilearn* [online]. 2021-05-10 [cit. 2021-5-23]. Dostupné z: <https://www.simplilearn.com/what-is-des-article>.

[6] NIST. *Advanced Encryption Standard (AES). Nist* [online]. 2001-11-26 [cit. 2021-5-22]. Dostupné z: <https://www.nist.gov/publications/advanced-encryption-standard-aes>.

[7] MUSTAFEEZ, Anusheh Zohair. *What is the AES algorithm? Educative* [online]. [cit. 2021-5-22]. Dostupné z: <https://www.educative.io/edpresso/what-is-the-aes-algorithm>.

[8] NAKOV, Svetlin. *Practical Cryptography for Developers: Cipher Block Modes. Cryptobook.nakov* [online]. Sofia, 2018 [cit. 2021-5-22]. Dostupné z: <https://cryptobook.nakov.com/symmetric-key-ciphers/cipher-block-modes>.

[9] BRUSH, Kate. *Asymmetric cryptography. SearchSecurity* [online]. 2020-03 [cit. 2021-5-23]. Dostupné z: Dostupné z: <https://searchsecurity.techtarget.com/definition/asymmetric-cryptography>.

[10] LAKE, Josh. *What is the Diffie–Hellman key exchange and how does it work? Comparitech* [online]. 2021-03-21 [cit. 2021-5-27]. Dostupné z: <https://www.comparitech.com/blog/information-security/diffie-hellman-key-exchange/>.

[11] RUSSELL, Aaron. *Co je kryptografie eliptické křivky (ECC)? SSL.com* [online]. 2019-08-07 [cit. 2021-5-8]. Dostupné z: <https://www.ssl.com/cs/

Nej%C4%8Dast%C4%9Bj%C5%A1%C3%AD-dotazy/co-je-eliptick%C3%A1-k%C5%99ivka-kryptografie-ecc/>.

[12] CERTICOM RESEARCH. *Standards for Efficient Cryptography: SEC 2: Recommended Elliptic Curve Domain Parameters. Certicom Corp* [online]. 2000-09-20 [cit. 2021-5-8]. Dostupné z: <https://www.secg.org/SEC2-Ver-1.0.pdf>.

[13] KOHLI, Kerman. *Learning Cryptography, Part 3: Elliptic Curves. Medium* [online]. 2019-08-15 [cit. 2021-5-9]. Dostupné z: <https://medium.com/loopring-protocol/learning-cryptography-elliptic-curves-4cfd0bdcb05a>.

[14] NUVRENI, Juan. *Zero Knowledge Proofs. Medium* [online]. 2018-01-31 [cit. 2021-5-10]. Dostupné z: <https://medium.com/coinmonks/zero-knowledge-proofs-14bb012c1ce9>.

[15] DAMGARD, Ivan. *On Σ-protocols. Aarhus University* [online]. 2010 [cit. 2021-5-12]. Dostupné z: <https://www.cs.au.dk/~ivan/Sigma.pdf>

[16] KOGAN, Dima. *Lecture 5: Proofs of Knowledge, Schnorr's protocol, NIZK. Stanford Crypto* [online]. 2019 [cit. 2021-5-12]. <Dostupnéz:https://crypto.stanford.edu/cs355/19sp/lec5.pdf>.

[17] HAJNÝ, Jan. *TCPT - Cryptologic Protocol Theory: Sigma Protocols. Brno: VUT* [přednáška]. 2021-4-24 [cit. 2021-5-12].

[18] TECHTERMS. *Authentication. Techterms* [online]. 2018-07-13 [cit. 2021-5-23]. Dostupné z: <https://techterms.com/definition/authentication>.

[19] TECHOPEDIA. *Authorization. Techopedia* [online]. [cit. 2021-5-23]. Dostupné z: <https://www.techopedia.com/definition/10237/authorization>.

[20] COMPUTER HOPE. *Access control system. ComputerHope* [online]. 2020-02-07 [cit. 2021-5-23]. Dostupné z: <https://www.computerhope.com/jargon/a/acs.htm>.

[21] EDUCBA: *Java vs Kotlin: Differences Between Java and Kotlin. EDUCBA* [online]. 2018-19-07 [cit. 2020-12-05]. Dostupné z: <https://www.educba.com/java-vs-kotlin/>.

[22] ANDROID DEVELOPERS. *Get started with the NDK. Android Developers* [online]. Poslední aktualizace 2020-09-30 [cit. 2020-12-05]. Dostupné z: <https://developer.android.com/ndk/guides>.

[23] TYLEY, Roberto. *Spongy Castle. GitHub* [online]. 2018 [cit. 2021-5-22]. Dostupné z: <https://rtyley.github.io/spongycastle/>.

[24] MACKAY, Ken. *Micro-ecc. GitHub* [online]. 2014, poslední aktualizace 2020-11-01 [cit. 2020-12-05]. Dostupné z: <https://github.com/kmackay/micro-ecc>.

[25] CVRČEK, Tadeáš. *Kryptografie na platformě Arduino* [online]. Brno, 2020 [cit. 2020-11-21]. Dostupné z: <http://hdl.handle.net/11012/190258>. Bakalářská práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Petr Dzurenda.

[26] KOKKE. *Tiny-AES. GitHub* [online]. 2013, poslední aktualizace 2020-07-01 [cit. 2020-12-11]. Dostupné z: <https://github.com/kokke/tiny-AES-c>.

[27] RWEATHER. *Crypto library. GitHub* [online]. 2015 [cit. 2021-5-11]. Dostupné z: <https://github.com/rweather/arduinolibs/tree/master/libraries/Crypto>.

[28] TUSHIE, David. *An Introduction to NFC Standards. ICMA* [online]. 2012-10-16 [cit. 2020-12-05]. Dostupné z: Dostupné z: <http://www.icma.com/ArticleArchives/StandardsOct12.pdf>.

[29] NFC-FORUM. *NFC - Specification Releases. NFC-Forum* [online]. 2021-03-01 [cit. 2021-5-22]. Dostupné z: <https://nfc-forum.org/our-work/specification-releases/>.

[30] NFC-FORUM. *What Are The Operating Modes Of NFC Devices? NFC-Forum* [online]. 2013-12-17 [cit. 2020-12-11]. Dostupné z: <https://nfc-forum.org/resources/what-are-the-operating-modes-of-nfc-devices/>.

[31] Android Developers. *Host-based card emulation overview* [online]. Android Developers, 2019 [cit. 2020-11-21]. Dostupné z: <https://developer.android.com/guide/topics/connectivity/nfc/hce>.

[32] Wikiedia. *APDU. In: Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2013, poslední aktualizace 2019-11-25 [cit. 2020-11-23]. Dostupné z: <https://cs.wikipedia.org/wiki/APDU>.

[33] Wikipedia. *Bluetooth. In: Wikipedia: the free encyclopedia* [online]. Wikimedia Foundation, 2005, 2020-06-04 [cit. 2020-12-11]. Dostupné z: <https://cs.wikipedia.org/wiki/Bluetooth>.

[34] KRUEGER, Brian. *Understanding Bluetooth Technology. ABR* [online]. 2020-10-11 [cit. 2021-5-20]. Dostupné z: <https://www.abr.com/understanding-bluetooth-technology/>

[35] Bluetooth SIG. *Bluetooth Radio Versions. Bluetooth* [online]. [cit. 2020-12-02]. Dostupné z: <https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/radio-versions/>.

[36] WELBES, William. *What is Bluetooth Low Energy and how does it work? Centare* [online]. 2019-03-07 [cit. 2021-5-27]. Dostupné z: <https://www.centare.com/insights/what-is-bluetooth-low-energy>.

[37] Standards for Efficient Cryptography Group: *Standards for Efficient Cryptography 2: Recommended Elliptic Curve Domain Parameters* [online]. 210-01-27 [cit. 2020-12-09]. Dostupné z: <https://www.secg.org/sec2-v2.pdf>.

[38] THORNSBY, Jessica. *Get Wear OS and Android Talking: Exchanging Information via the Wearable Data Layer. Tutsplus* [online]. 2018-04-27 [cit. 2020-11-29]. Dostupné z: <https://code.tutsplus.com/tutorials/get-wear-os-and-android-talking-exchanging-information-via-the-wearable-data-layer--cms-30986>.

[39] ROTH, Soren. *LocalBroadcastManager modification. GitHub* [online]. 2018-03-08 [cit. 2021-5-17]. Dostupné z: <https://github.com/sorenoid/LocalBroadcastManager>.

[40] The Legion of the Bouncy Castle. *Bouncy Castle library: Latest JAVA Releases. Bouncycastle* [online]. 2018-03-15 [cit. 2021-5-25]. Dostupné z: <https://www.bouncycastle.org/latest_releases.html>.

[41] ACS: *ACR122U software. Advanced Card Systems Ltd.* [online]. 2017 [cit. 2021-5-17]. Dostupné z: <https://www.acs.com.hk/en/driver/3/acr122u-usb-nfc-reader/>.

Seznam symbolů, veličin a zkratek

ECDH	Elliptic-Curve Diffie–Hellman
DES	Data Encryption Standard
AES	Advanced Encryption Standard
ECB	Electronic Codebook
CBC	Cipher Block Chaining
CFB	Cipher FeedBack
OFB	Output FeedBack
CTR	Counter Mode
GCM	Galois/Counter Mode
HCE	Host Card Emulation
SHA	Secure Hash Algorithm
RSA	Rivest–Shamir–Adleman
RFID	Radio Frequency Identification
NFC	Near Field Communication
BLE	Bluetooth Low Energy
AID	Application Identifier
APDU	Application Protocol Data Unit
GATT	Generic Attribute
FHSS	Frequency Hopping Spread Spectrum
NDK	Native Development Kit
JNI	Java Native Interface
GUI	Graphical User Interface
UI	User Interface
PAN	Personal Area Network
ECDSA	Elliptic Curve Digital Signature Algorithm

A Struktura archivu se zdrojovými soubory

- `Source Codes BP Klasovitý` ..... Kořenový adresář zdrojových souborů
  - `README` ..... Soubor s popisy složek a důležitými informacemi o souborech
  - `Návod k instalaci a obsluze aplikací.docx`
  - `app-release.apk` ..... Apk soubor pro instalaci aplikace pro mobilní telefon
  - `WatchWithMobile` . Adresář projektu aplikací pro mobilní telefon a chytré hodinky
    - `app` ..... Adresář části projektu pro mobilní telefon
      - `src` ..... Adresář se zdrojovými soubory
        - `main`
          - `cpp` . Adresář se soubory jazyka C, včetně importovaných knihoven
          - `java` ..... Adresář se zdrojovými soubory Javy
          - `res` ..... Adresář se soubory GUI
          - `AndroidManifest.xml`
      - `build.gradle` ..... build.gradle soubor modulu pro mobilní telefon
    - `gradle` ..... Složka s graddle wrapperem
    - `wear` ..... Adresář části projektu pro chytré hodinky
      - `src` ..... Adresář se zdrojovými soubory
        - `main`
          - `cpp` . Adresář se soubory jazyka C, včetně importovaných knihoven
          - `java` ..... Adresář se zdrojovými soubory Javy
          - `AndroidManifest.xml`
      - `buildgradle` ..... build.gradle soubor modulu pro chytré hodinky
    - `build.gradle` ..... build.gradle soubor projektu
  - `client_java_jar` ..... Adresář spustitelné aplikace pro PC
    - `client java` ..... Spustitelný soubor aplikace pro PC
    - `bcprov-jdk15on-167` ..... Soubor knihovny Bouncy Castle
    - `background2` ..... Soubor pozadí aplikace
    - `backgroundSmall` ..... Soubor pozadí aplikace
  - `client java` ..... Adresář projektu PC aplikace
    - `idea` ..... Adresář se soubory studia Intellij idea
    - `files` ..... Adresář s uloženými daty aplikace spouštěné v projektu
    - `libs` ..... Adresář knihoven (sem je potřeba vložit kopii souboru bcprov-jdk15on-167, soubor musel být z adresáře odstraněn, aby byly soubory menší než 15MB)
    - `src` ..... Adresář se zdrojovými soubory aplikace pro PC