VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Jan Strakoš

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

ATRIBUTOVÁ AUTENTIZACE NA PLATFORMĚ ANDROID
ATTRIBUTE AUTHENTICATION ON ANDROID PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jan Strakoš**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Lukáš Malina, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Jan Strakoš ***ID:*** 195171
Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Atributová autentizace na platformě Android

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s atributovou autentizací, tzv. Attribute-based Credentials a dostupnými knihovnami poskytující podporu kryptografie na platformě Android. Zaměřte se na knihovny podporující eliptické křivky na Android platformě včetně možností využívat C knihovny pomocí Android NDK. Proveďte měření paměťové a výpočetní efektivity dílčích operací knihoven a atributových schémat na zvoleném Android zařízení.
V rámci diplomové práce implementujte pilotní systém atributové autentizace na platformě Android (tj. strana uživatele s HCE, strana ověřovatele, vydavatel a revokační manažer). Navrhněte a vytvořte GUI pro jednotlivé entity autentizačního systému a integrujte GUI s kryptografickými a komunikačními funkcemi. Jednotlivé Android aplikace budou rovněž mezi sebou schopné výměny dat autentizačního protokolu. Výsledný implementovaný systém otestujte a případně proveďte odstranění chyb a dodatečnou optimalizaci.

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] CAMENISCH, Jan, et al. Fast keyed-verification anonymous credentials on standard smart cards. IFIP International Conference on ICT Systems Security and Privacy Protection. Springer, Cham, 2019.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: doc. Ing. Lukáš Malina, Ph.D.

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

ABSTRAKT

Diplomová práce se zabývá implementací pilotního systému atributové autentizace na platformě Android. Podpora atributové autentizace na platformě Android je co do počtu implementací velmi slabá a je potřeba jí věnovat zvýšenou pozornost. V teoretické části práce je rozebrána kryptografická podpora na platformě Android, využití nástroje Android NDK (Native Development Kit) a služby HCE (Host-Card Emulation). Součástí teoretické části práce je i popis schémat systému atributové autentizace včetně pilotního systému RKVAC. Praktická část popisuje průběh implementace systému RKVAC na platformě Android společně s implementací vlastního kryptografického jádra založeného na nativní kryptografické knihovně MCL. V závěru práce jsou uvedeny výsledky měření časové, paměťové a výpočetní náročnosti vytvořených mobilních aplikací.

KLÍČOVÁ SLOVA

Android, atributová autentizace, RKVAC, MCL, NDK, HCE, QR kód, Java, C

ABSTRACT

This master's thesis focuses on implementation of ABC (Anonymous attribute-based credential) pilot system on the Android platform. The support for attribute authentication on the Android platform is very weak in terms of the number of implementations and needs a special attention. The theoretical part of the thesis describes the cryptographic support on the Android platform, the use of the Android Native Development Kit (NDK) and the Host-Card Emulation (HCE) service. The theoretical part of the thesis also includes a description of attribute authentication schemes, including a pilot RKVAC system. The practical part describes the implementation of the RKVAC system on the Android platform along with the implementation of a custom cryptographic kernel based on the native MCL cryptographic library. The practical part of this thesis describes implementation proces of RKVAC system on Android plaform, that uses native cryptographic library MCL. The final part shows the results of time, memory and computation difficulty of developed applications.

KEYWORDS

Android, attribute-based authentication, RKVAC, MCL, NDK, HCE, QR code, Java, C

STRAKOŠ, Jan. *Atributová autentizace na platformě Android*. Brno, 2020, 104 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Lukáš Malina, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Atributová autentizace na platformě Android“ jsem vypracoval samostatně pod vedením vedoucího diplomové 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 diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové 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 diplomové práce panu doc. Ing. Lukáši Malinovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Děkuji také panu Ing. Petru Dzurendovi, Ph.D. za věcné připomínky a vstřícnost při konzultacích.

Tato práce vznikla jako součást klíčové aktivity KA6 - Individuální výuka a zapojení studentů bakalářských a magisterských studijních programů do výzkumu v rámci projektu OP VVV Vytvoření double-degree doktorského studijního programu Elektronika a informační technologie a vytvoření doktorského studijního programu Informační bezpečnost, reg. č. CZ.02.2.69/0.0/0.0/16_018/0002575.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Projekt je spolufinancován Evropskou unií.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Rostoucí požadavky uživatelů na ochranu soukromí a zároveň současná evropská nařízení a regulace viz GDPR (General Data Protection Regulation) vnáší do světa informačních technologií a procesu autentizace nebývalou důležitost. Při ověřování proklamované identity subjektu prostřednictvím chytrých zařízení, kterými jsou například dnešní mobilní telefony, je třeba dbát na minimalizaci zpracovaných osobních údajů. Špatnou manipulací s osobními údaji uživatele pak může dojít k tzv. odcizení elektronické identity, pří kterém se útočník vydává za původního majitele údajů. Celý proces tedy musí jak na svém počátku, tak i v průběhu respektovat práva dotčených osob a zasahovat do nich pouze minimálně.

Existují situace, při kterých majitel osobních údajů poskytuje pouze takové údaje, které jsou nezbytně nutné k ověření proklamované identity. Například se může jednat o prokázání věku, ověření příslušnosti k určité organizaci, držení věrnostní nebo čipové karty. V takovém případě pak dokazující odhaluje pouze nezbytně nutné atributy a ostatní zůstávají skryty, stejně jako jeho identita. Vhodným nástrojem, který odpovídá předchozímu popisu a zároveň splňuje současné požadavky, je atributová autentizace.

Jednou z nejpoužívanějších platforem, na které pracují dnešní mobilní telefony je Android. Mobilní telefony si mezi sebou vyměňují obrovské množství dat, jejichž součástí jsou bezpochyby i osobní údaje. Platforma Android poskytuje svým uživatelům celou řadu nástrojů, pomocí kterých mohou svá data chránit, zabezpečit a sdílet. Praktická implementace atributové autentizace na platformě Android, která by splňovala aktuální požadavky na bezpečnost a ochranu soukromí je však co do počtu implementací stále slabá a je ji potřeba věnovat zvýšenou pozornost, vzhledem ke stále rostoucímu trendu chytrých telefonů.

Tato diplomová práce se zabývá atributovou autentizací a její podporu na platformě Android. V teoretické části práce je detailně rozebrána atributová autentizace a kryptografická podpora pro platformu Android včetně možnosti využití Android NDK (Native Development Kit) a HCE (Host card emulation). V praktické části práce je pak popsána implementace pilotního systému atributové autentizace na platformě Android včetně kryptografického jádra využívajícího nativní kryptografickou knihovnu MCL. Práce dále popisuje realizaci návrhu GUI (Graphical User Interface) pro jednotlivé entity systému. V závěru práce je pak zhodnocena časová, paměťová a výpočetní náročnost operací autentizačního schématu a její vliv na uživatelskou přívětivost aplikací.

1 Bezpečnost a kryptografie na platformě Android

Android je mobilní operační systém, který je založený na Linuxovém jádře a zároveň dostupný jako otevřený software (open source). Vývoj platformy Android vede firma Google pod záštitou konsorcia firem Open Handset Alliance[1]. Výrobci různých zařízení mohou Android upravovat při dodržení stanovených podmínek. Dle statistických údajů [1] ze září 2020 je Android nejrozšířenějším operačním systémem používaným na chytrých telefonech (smartphonech), tabletech, chytrých televizích a dalších zařízeních. Dle zdroje [1] platforma Android pokrývá 74 % trhu, na druhém místě je potom operační systém iOS s necelými 25 %. Zbylé procento představují ostatní operační systémy jako například WindowsPhone nebo HarmonyOS.

Pro platformu Android, stejně jako v případě jiných masivně rozšířených operačních systémů, vychází v pravidelných časových intervalech nejnovější verze operačního systému. Novější verze reaguje na chyby v předchozích verzích a zároveň přináší celou řadu nových funkcionalit, rozšíření a bezpečnostních aktualizací. Nejnovější hlavní verzí pro rok 2020 je Android 10, která vyšla 3. září 2019 a nahrazuje svého předchůdce Android 9 Pie.

Nová verze Android 10 se mimo celé řady jiných vylepšení zaměřuje na funkce pro ochranu dat a vyšší bezpečnost. Zdroj [2] uvádí nejdůležitější bezpečnosti změny, mezi které patří žádost o oprávnění a zónové monitorování (geofencing) nebo omezení přístupu aplikací k sériovému číslu a IMEI [2] (International Mobile Equipment Identity), místo kterého jsou vývojáři nuceni využívat obnovitelné identifikátory ke sledování uživatelů. Dalšími změnami, které se dotýkají zejména oblasti šifrování jsou standardní podpora protokolu TLS 1.3 (Transport Layer Security)[3] a vydání nových bezpečnostních knihoven, které posílí ochranu v „sandboxech“[4], které chrání aplikace proti úniku dat.

Vhodným nástrojem pro vývoj aplikací pro platformu Android je Java. Jedná se o populární objektově orientovaný programovací jazyk, pomocí kterého lze vytvářet mobilní aplikace. Výhodou tohoto jazyka je obrovská uživatelská podpora, obsáhlá dokumentace nebo celá řada knihoven a frameworků.[5]

Následující podkapitoly rozebírají bezpečnost platformy Android, kryptografickou podporu pod záštitou programovacího jazyka Java a vysvětlují možné využití Android NDK a HCE.

[1] Uskupení výrobců mobilních telefonů, které stojí za vývojem operačního systému Android
[2] Unikátní číslo přidělené výrobcem mobilnímu telefonu
[3] Kryptografické protokoly poskytující možnost zabezpečené komunikace na Internetu
[4] Označení pro bezpečnostní mechanismus, který slouží pro oddělování běžících procesů
[5] Softwarová struktura pro podporu při programování, vývoji a organizaci

1.1 Bezpečnost platformy Android

Mezi hlavní vlastnosti platformy Android patří otevřenost a rozličnost. Z pohledu otevřenosti nabízí platforma aplikační prostředí, které chrání důvěrnost, integritu a poskytuje dostupnost uživatelům, aplikacím, zařízením a sítím. Zabezpečení otevřené platformy vyžaduje propracovanou bezpečnostní architekturu a přísné bezpečnostní programy. Android využívá vícevrstvou bezpečnostní architekturu, která je dostatečně flexibilní pro podporu otevřené platformy a zároveň poskytuje ochranu pro své uživatele. Samotná otevřenost sebou přináší i celou řadu bezpečnostních rizik [3]. Dostupnost zdrojového kódu umožňuje potenciálním útočníkům zkoumat nedostatky v logice kódu, které mohou následně využít pro svůj prospěch. V uzavřeném systému je zdrojový kód chráněn kryptografickými nástroji, avšak neumožňuje takovou míru kontroly bezpečnosti ze strany komunity.

Z pohledu rozličnosti si většina společností vyvíjí pro platformu Android vlastní nadstavbu, která se od „čisté“ verze Androidu může lišit jak změnou grafického uživatelského rozhraní, tak funkčností. Touto cestou se jednotlivé společnosti dávají, i za cenu méně častých aktualizací, zejména kvůli odlišnosti a jedinečnosti na trhu. Mezi nevýhody patří nemožnost rychle rozšířit nové bezpečnostní aktualizace na všechna dostupná zařízení. Tento problém však Google částečně řeší projektem Mainline [4], který umožňuje aktualizovat jádrové části operačního systému stejným způsobem, jakým se aktualizují aplikace v Google Play[6]. Díky tomu je Google schopen distribuovat AOSP (Android Open Source Project) komponenty do zařízení rychleji, jelikož není plně závislý na plné aktualizaci od výrobce telefonu. Přesto však existuje podstatná část trhu, zejména levnější zařízení, která jsou po dvou až třech letech na trhu bez možnosti dalších systémových nebo bezpečnostních aktualizací. Právě tyto zařízení jsou nejčastějším cílem útoků. Jsou to právě finanční důvody, které výrobce nemotivují udržovat levnější sortiment aktualizovaný. I přesto existují uživatelé, kteří vyvíjejí AOSP komponenty podporující starší zařízení. Jejich počet je však zanedbatelný. Rozličnost systému přináší z bezpečnostního hlediska i určité výhody [3]. Obrovské množství výrobců poskytujících širokou škálu unikátních zařízení společně s různými verzemi systému představují pro útočníka mnohem složitější cíl, než je například platforma homogenní. Důležitým faktem také je, že poměrnou většinu návrhů na zlepšení bezpečnosti iniciují samotní výrobci.

Bezpečnost platformy Android s příchodem hlavní verze 10 obrovským způsobem vzrostla a tím i zájem uživatelů o tuto platformu. Útočníci zároveň čím dál častěji cílí na samotné aplikace než na platformu jako takovou.

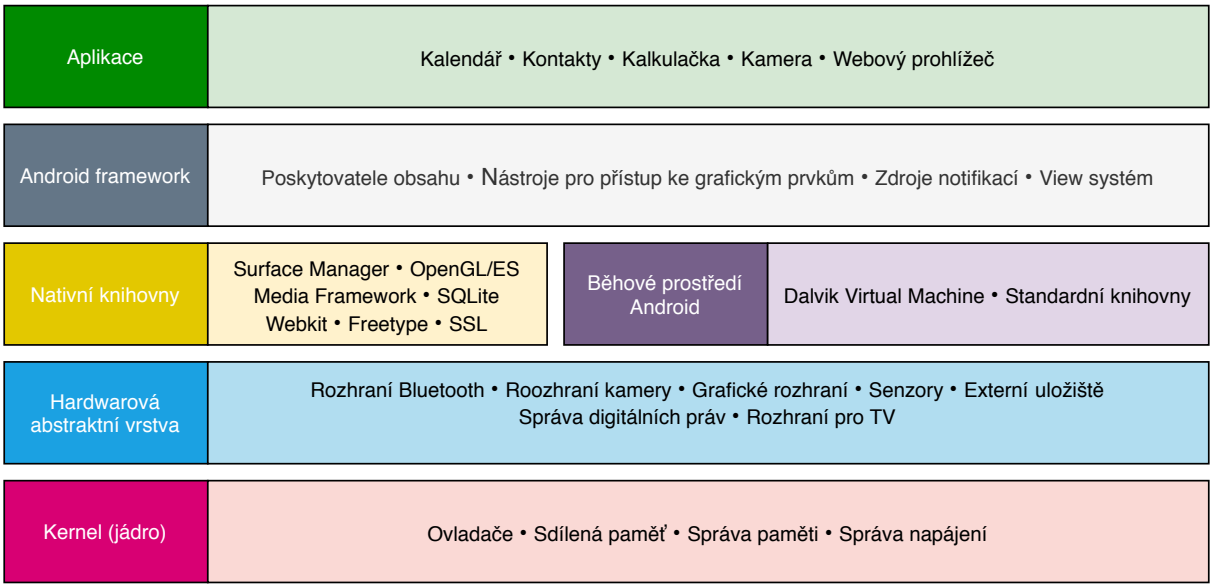
[6] Online distribuční služba, poskytující několik druhů digitálního obsahu

1.1.1 Architektura systému

Stejně jako jiné operační systémy má i Android vlastní architekturu systému, která se odráží i na přístupu k bezpečnosti. V rámci vícevrstvé architektury každá vyšší vrstva předpokládá, že komponenty v příslušné nižší vrstvě zajišťují dostatečnou míru zabezpečení. S výjimkou malého množství zdrojového kódu operačního systému Android běžícího pod oprávněním root[7] je veškerý kód nad linuxovým jádrem omezen sandboxem. Tedy mechanismem, kdy aplikace běží v izolovaném prostoru ve kterém může přistupovat pouze k takovým zdrojům a vykonávat pouze takové úkony, ke kterým má oprávnění. Obrázek 1.1 zobrazuje vícevrstvou architekturu platformy Android. Jednotlivými komponenty dle zdroje [5] jsou:

- **Aplikace**: Aplikace platformy Android, kterými jsou například kalendář, kalkulačka nebo prohlížeč.
- **Android framework**: Knihovny napsané v jazyce Java nebo Kotlin, které tvoří vlastní API (Application Programming Interface) rozhraní. Jsou to zejména knihovny umožňující přístup ke grafickým prvkům systému, kterými jsou například tlačítka, views (pohledy) a layouts (rozložení). Knihovny dále poskytují vhodný nástroj pro práci s notifikacemi nebo obsahem.
- **Běhové prostředí**: Umožňuje běh aplikací napsaných v jazyce Java/Kotlin. Nachází se zde i Dalvik Virtual Machine, který představuje obdobu JVM (Java Virtual Machine) a stará se o převod kódu napsaného v jazyce Java/Kotlin do nativního kódu. Nachází se zde také standardní knihovny.
- **Nativní knihovny**: Knihovny jsou zapsány v programovacím jazyce C/C++ a poskytují základní funkce systému. Surface Manager obstarává správné zobrazování aplikací a jejich vrstvení. OpenGL/ES jsou knihovny pro práci s grafikou. Media framework slouží pro práci s mediálními soubory a obsahuje velkou škálu kodeků pro různé formáty audia a videa. SQLite umožňuje práci s daty a jejich ukládání. Webkit představuje vykreslovací jádro pro webový prohlížeč. Freetype zajišťuje vykreslování písma. SSL (Secure Socket Layer) zajišťuje šifrování a zabezpečení přenosu dat.
- **Hardwarová abstraktní vrstva**: Vytváří jednotné API rozhraní ovládající různě fungující hardware, které zároveň usnadňuje práci při ovládání jednotlivých hardwarových zařízení.
- **Kernel (jádro)**: Tvoří základ operačního systému a zajišťuje komunikaci mezi hardwarem a softwarem. Dále zajišťuje správu procesů, paměti, napájení a síťové spojení. Mezi jeho hlavní součást patří ovladače, které zajišťují právě onu komunikaci. Android nepoužívá vlastní jádro, ale využívá Linuxové jádro.

[7] Nejvyšší oprávnění operačního systému založeného na linuxovém jádře umožňující tvorbu, čtení a modifikaci dat



Obr. 1.1: Vícevrstvá architektura platformy Android.

Základní prvky aplikace

Android poskytuje otevřenou platformu a aplikační prostředí pro mobilní zařízení. Aplikace pro platformu Android jsou nejčastěji napsány v programovacím jazyce Java. Rostoucí popularitu zaznamenal i programovací jazyk Kotlin, který je s Javou kompatibilní. Aplikace mohou být napsány i v nativním kódu, tedy za pomocí programovacího jazyka C/C++. Mobilní aplikace jsou instalovány z jediného souboru formátu APK (Android Application Package) a běží v Dalvik Virtual Machine popsané v podkapitole 1.1.1. Základní prvky Android aplikace dle zdroje [6] jsou:

- **AndroidManifest.xml**: Řídicí soubor, který říká systému, co dělat se všemi komponentami vyšší úrovně, kterými jsou aktivity, služby nebo přijímač všesměrového vysílání. V tomto souboru jsou i definována požadovaná oprávnění a požadované funkce.
- **Aktivity**: Obecný kód, který vykonává uživatelsky zaměřený úkol. Zpravidla se jedná o zobrazení uživatelského rozhraní, ale nemusí tomu tak být. Příkladem aktivit jsou jednotlivé „stránky“ v aplikaci.
- **Služby**: Kód běžící na pozadí. Může běžet ve vlastním procesu nebo v kontextu procesu jiné aplikace. Příkladem služby je hudební přehrávač, který přehrává hudbu i když uživatel opustí uživatelské rozhraní přehrávače.
- **Přijímač všesměrového vysílání**: Umožňuje aplikacím reagovat na zprávy vysílané operačním systémem Android nebo jinou aplikací. Příkladem takové události může být zpráva, která aplikaci informuje i vybité baterii zařízení.

1.1.2 Bezpečnost Android aplikací

V rámci platformy Android jsou řešeny situace, kdy jednotlivé aplikace přistupují ke citlivým zdrojům nebo vykonávají úkony vyžadující odpovídající oprávnění. Například v případě karty SIM (Subscriber identity module) není aplikacím jiných výrobců nízkoúrovňový přístup ke kartě SIM povolen a veškerou komunikaci s kartou SIM zajišťuje operační systém jako takový, včetně přístupu k osobním údajům (kontaktům) v paměti karty [7]. Android v takových případech definuje chráněná rozhraní API, mezi které patří například rozhraní pro funkce fotoaparátu, přenos dat nebo GPS (Global Positioning System). Funkce a prostředky, které tato rozhraní poskytují jsou k dispozici pouze prostřednictvím operačního systému. Pokud chce vývojář v zařízení používat chráněná rozhraní API, musí skrze AndroidManifest.xml definovat funkce, které bude daná aplikace potřebovat [8]. Pokud chce uživatel využít funkci z aplikace, která vyžaduje chráněné rozhraní API, systém zobrazí dialogové okno s výzvou k odepření nebo povolení oprávnění. Mezi chráněné zdroje, ke kterým aplikace může přistupovat a které zároveň souvisí se soukromým uživatele jsou osobní údaje, metadata a zařízení poskytující citlivý obsah.

Osobní údaje

Platforma Android využívá kontrolu oprávnění operačního systému Android. Jedná se o mezilehlý body v komunikaci mezi aplikací a chráněným zdrojem. Při běžném používání shromažďují zařízení Android uživatelská data v aplikacích třetích stran nainstalovaných samotným uživatelem. Proces sdílení těchto informací musí v takovém případě nejdřív projít přes kontrolu oprávnění operačního systému Android, které slouží k ochraně dat před aplikacemi třetích stran. Poskytovatelé systémového obsahu, jako například kontakty nebo kalendáře, taktéž poskytují osobní údaje. Tito poskytovatelé jsou vytvořeni s jasně definovanými oprávněními. Během instalace mohou aplikace třetích stran požádat o přístup k těmto prostředkům. Po udělení oprávnění je možné takovou aplikaci nainstalovat. Kontrolu oprávnění pro přístup aplikace ke chráněným zdrojům a informacím demonstruje obrázek 1.2, vice viz [9].

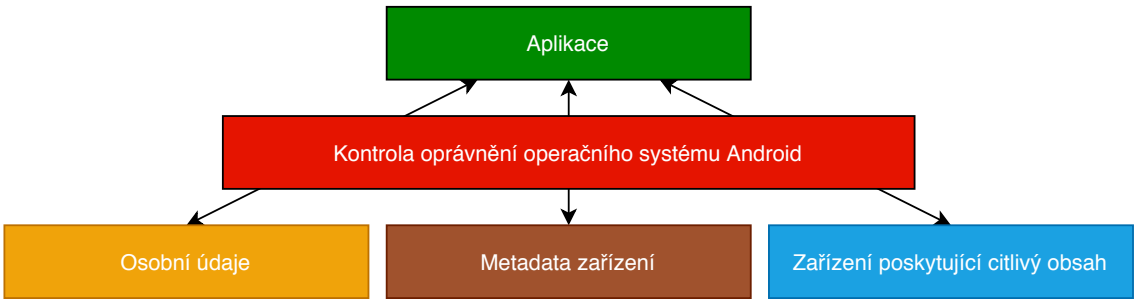
Metadata

Platforma Android také omezuje přístup k metadatům, která nejsou přímo citlivá, ale mohou nepřímo odhalit charakteristiky uživatele, uživatelské předvolby a způsob, jakým zařízení používá [10]. Ve výchozím nastavení nemají aplikace k protokolům operačního systému, historii prohlížeče, telefonnímu číslu nebo síťovým a hardwarovým identifikačním údajům povolený přístup. Jestliže aplikace požaduje přístup

k těmto informacím během instalace, pak se instalační program zeptá uživatele, zdali má aplikace přístup k těmto informacím. Pokud uživatel neudělí přístup, aplikace nebude nainstalována.

Zařízení poskytující citlivý obsah

Podobným způsobem potom platforma Android řeší situaci se zařízeními poskytujícími citlivý obsah. Mezi tyto zařízení patří kamera, mikrofon nebo GPS. Přístup k těmto zařízením systém vyhodnocuje za pomocí oprávnění operačního systému Android a explicitního souhlasu uživatele, více viz. [11].



Obr. 1.2: Kontrola oprávnění pro přístup aplikace k chráněným informacím.

Certifikační autority

Android obsahuje také sadu nainstalovaných systémových CA (certifikační autorita), které jsou důvěrné v celém systému. Zařízení s hlavní verzí 7.0 a novější neumožňují úpravu této sady CA, jako tomu bylo u starších zařízení. V případě přidání nové veřejné CA do sady prostředků systému Android musí nová CA nejdříve projít procesem schválení certifikační autority a poté požádat systém Android o přidání CA do primární sady, která se nastavuje v AOSP (Android Open Source Project). Detailní popis poskytuje zdroj [12].

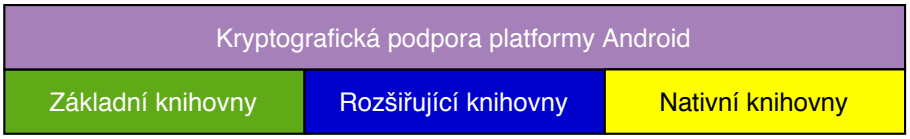
Podpis aplikace

Při instalaci aplikace se zároveň kontroluje její podpis. Každá aplikace, která běží na platformě Android musí disponovat podpisem vývojáře [13]. Aplikace, které se uživatel pokouší nainstalovat bez podpisu, jsou odmítnuty službou Google Play nebo instalačním programem balíčku v zařízení Android. Podpis je realizován za pomocí certifikátů, které mohou vývojáři generovat bez vnější pomoci nebo oprávnění. Podepsaný certifikát aplikace určuje, které ID uživatele je přidruženo ke které aplikaci. Různé aplikace běží pod různými ID uživatelů. Android zároveň neprovádí

ověření certifikační autority pro aplikační certifikáty, což může představovat určitou bezpečnostní hrozbu. Aplikace mohou deklarovat svá bezpečnostní oprávnění na úrovni podpisu tím, že umožní přístup pouze aplikacím podepsaným stejným klíčem (veřejný klíč v certifikátu). V takovém případě pak dvě nebo více aplikací podepsané stejným vývojářským klíčem sdílí společné uživatelské ID deklarované v AndroidManifest.xml. Detailní popis poskytuje zdroj [13].

1.2 Kryptografická podpora

Platforma Android umožňuje vývojářům implementovat celou řadu nástrojů, pomocí kterých lze zajistit standardní, ale i rozšiřující bezpečnostní funkce. Jedná se o balíčky, neboli kolekce souvisejících tříd, které představují souhrn procedur a funkcí. Takovéto balíčky se taktéž označují jako knihovny. Prostřednictvím knihoven vývojáři vytvářejí objekty realizující kryptografické operace, které mohou aplikace platformy Android využívat. Knihovny zajišťující kryptografickou podporu na platformě Android lze rozdělit do třech hlavních skupin. První skupinou jsou knihovny základní, které jsou součástí základní sady balíčků platformy Android a programovacího jazyka Java. Druhou skupinu představují knihovny rozšiřující. Jak už z názvu vyplývá, jedná se o knihovny rozšiřující základní sadu a knihovny třetích stran. Třetí skupinou jsou knihovny nativní. Tyto knihovny jsou napsané v jazyce C/C++ využívající Android NDK. Rozdělení do skupin znázorňuje obrázek 1.3. Následující podkapitoly popisují jednotlivé skupiny knihoven zajišťujících podporu kryptografie na platformě Android.



Obr. 1.3: Kryptografická podpora platformy Android.

1.2.1 Základní knihovny

Na základní podporu kryptografie pro platformu Android lze nahlížet dvěma pohledy. Prvním je základní podpora ze strany programovacího jazyka Java, druhým je potom základní podpora ze strany Androidu.

Java knihovny

Programovací jazyk Java rozlišuje dva typy balíčků z pohledu základní a rozšiřující sady [14]. K rozlišení mezi základní a rozšiřující sadou využívá klíčového slova v názvu balíčku. Balíčky obsahující klíčové slovo „java“ představuje základní sadu. Balíčky obsahující klíčové slovo „javax“ představují sadu rozšiřující. Písmeno x zde zastupuje anglické slovo extension, neboli rozšíření. V průběhu doby se vsak balíčky rozšiřující sady staly součástí standardního Java API. Přesunutí rozšiřující sady do základní sady by však představovalo složitý proces, který by způsobil znehodnocení stávajícího kódu. Prakticky tedy neexistuje rozdíl mezi základní a rozšiřující sadou z pohledu programovacího jazyka Java. Tento rozdíl se však stále uvádí v dokumentaci jednotlivých knihoven poskytujících kryptografickou podporu, a proto je zachován i v této diplomové práci. Zdroj [15] uvádí knihovny poskytující základní kryptografickou podporu z pohledu programovacího jazyka Java, kterými jsou:

- **java.security**: Knihovna poskytuje celou řadu tříd implementujících snadno konfigurovatelnou architekturu zabezpečení přístupu. Knihovna také podporuje generování a ukládání kryptografických párů veřejných a soukromých klíčů, stejně jako celou řadu kryptografických operací pro generování a podpis zpráv. Součástí knihovny jsou i třídy umožňující tvorbu zabezpečených a chráněných objektů nebo bezpečné generování náhodných čísel. Třídy poskytující bezpečné generátory náhodných čísel slouží spíše jako rozhraní, jehož implementace závisí na vývojáři samotném. Mezi rozhraní této knihovny patří rozhraní specifikující velikosti klíčů a jiné parametry kryptografických algoritmů, rozhraní pro veřejné a soukromé klíče a mnohé další.
- **java.security.cert**: Knihovna poskytuje třídy a rozhraní pro parsování a správu certifikátů, seznamy zneplatněných certifikátu a správu certifikačních cest. Knihovna podporuje certifikáty X.509[8] verze 3 a seznamy zneplatněných certifikátů X.509 verze 2.
- **java.security.interfaces**: Knihovna poskytuje rozhraní pro generování RSA[9] (Rivest-Shamir-Adleman) a DSA[10] (Digital Signature Algorithm) klíčů. Tato rozhraní nepodporují operace nad klíči uloženými na externích hardwarových úložištích.
- **java.security.spec**: Knihovna poskytuje třídy a rozhraní pro specifikace klíčů a parametrů kryptografických algoritmů. Specifikace klíče může být dána přímo algoritmem, ve kterém je využit, nebo formátem kódovaní nezávislém na použitém algoritmu. Tato knihovna poskytuje specifikace pro soukromé a veřejné

[8] Standard pro systémy založené na asymetrické kryptografii. Specifikuje formát certifikátů, seznamy zneplatněných certifikátů, parametry certifikátů a metody kontroly platností certifikátů

[9] Šifrovací algoritmus založený na asymetrické kryptografii

[10] Standard americké vlády pro digitální podpis

klíče algoritmů RSA a DSA. Dále obsahuje specifikace pro soukromé klíče dle standardu PKCS[11] #8 (Public Key Cryptographic Standards), popisující formát uložení soukromých klíčů v šifrované i nešifrované formě. Knihovna taktéž poskytuje specifikace soukromých a veřejných klíčů v X.509 certifikátech.

Android knihovny

Podobně jako v případě programovacího jazyka Java rozlišuje platforma Android knihovny, které jsou přímo součástí operačního systému Android a které ne. Pro rozlišení se používá klíčové slovo v názvu balíčku. Knihovny, které jsou dodávány společně s operačním systémem obsahují klíčové slovo „android“. Veškerý další vývoj společně s knihovnami třetích stran uvozuje klíčové slovo v názvu knihovny „androidx“. Základní knihovny se zaměřují spíše na správu přístupu k chráněným zdrojům a datům, než na samotnou implementaci kryptografických algoritmů. Zdroj [15] uvádí základní knihovny zajišťující kryptografickou podporu platformy ze strany Androidu, kterými jsou:

- **android.security**: Knihovna poskytuje přístup k několika mechanismům zabezpečení subsystémů android. Obsahuje třídy poskytující správu politik a přístupu k chráněným kryptografickým klíčům.
- **android.security.identity**: Knihovna poskytuje třídy umožňující tvorbu profilů, na základě kterých lze řídit přístup ke zvoleným zdrojům. Knihovna také zprostředkovává rozhraní pro zabezpečené úložiště osobních dokumentů a citlivých dat uživatele.
- **android.security.keystore**: Knihovna poskytuje informace platnosti a validitě kryptografických klíčů. Knihovna také zprostředkovává proces inicializace generátoru párů kryptografických klíčů.

1.2.2 Rozšiřující knihovny

Jak již bylo zmíněno v předchozích podkapitolách, i na rozšiřující knihovny se dá pohlížet z více pohledů. Prvním je pohled ze strany programovacího jazyka Java. Java pro rozšiřující knihovny základní sady zavádí klíčové slovo „javax“ v názvu balíčku. Knihovny třetích stran neuvozuje žádné klíčové slovo a volba je ponechána na autorovi knihovny. Druhým je pohled ze strany platformy Android. Knihovny, které nejsou součástí operačního systému a jsou součástí dalšího vývoje nebo aktualizací se uvozují klíčovým slovem „androidx“ v názvu balíčku. Následující podkapitoly popisují rozšiřující knihovny zajišťující kryptografickou podporu platformy Android.

[11] Skupina standardů pro asymetrickou kryptografii od společnosti RSA Security

Java knihovny

Rozšiřujících knihoven programovacího jazyka Java zajišťující kryptografickou podporu je celá řada. Kromě knihoven rozšiřujících základní sadu přibývají knihovny zajišťující kryptografickou podporu v oblastech autentizace a síťové komunikace. Zdroj [15] uvádí knihovny poskytující rozšiřující kryptografickou podporu z pohledu programovacího jazyka Java, kterými jsou:

- **javax.crypto** Knihovna poskytuje třídy a rozhraní pro kryptografické operace, kterými jsou šifrování, generování a ověřování klíčů, generování kódu MAC (Message Authentication Code) pro ověřování zpráv. Podpora šifrování zahrnuje symetrické, asymetrické, blokové[12] a proudové[13] šifry. Většina tříd definuje rozhraní a parametry, jejichž implementace je ponechána na vývojáři aplikace.
- **javax.crypto.interfaces**: Knihovna poskytuje rozhraní pro DH[14] (Diffie-Hellman) klíče definované dle standardu PKCS #3 popisujícího výměnu klíčů metodou DH. Tato rozhraní nepodporují operace nad klíči uloženými na externích hardwarových úložištích.
- **javax.crypto.spec**: Knihovna poskytuje třídy a rozhraní pro specifikace klíčů a parametrů kryptografických algoritmů. Specifikace klíče může být dána přímo algoritmem, ve kterém je využit, nebo formátem kódovaní nezávislém na použitém algoritmu. Knihovna poskytuje specifikace pro veřejné a soukromé klíče protokolu DH, stejně jako specifikace klíčů použitých v algoritmech DES[15] (Data (Digital) Encryption Standard), 3DES (Triple DES) a PBE (Password Based Encryption). Knihovna taktéž poskytuje specifikace parametrů požívaných v kryptografických algoritmech DH, DES, 3DES, PBE, RC2 a RC5.
- **javax.net.ssl**: Knihovna poskytuje třídy využívající zabezpečený síťový socket[16]. Umožňuje realizovat zabezpečenou komunikaci pomocí protokolu SSL[17] nebo jiného bezpečnostního protokolu a detekovat zavedené chyby do přenášeného proudu dat. Knihovna taktéž umožňuje autentizaci komunikujících stran.
- **javax.security.auth**: Knihovna poskytuje nástroje pro autentizaci a autorizaci. Autentizační komponentu lze realizovat prostřednictvím této knihovny za pomoci modulů „zapojených“ do aplikace. Pomocí autorizační komponenty lze realizovat řízení přístupu na základě oprávnění vyplývajících z informací o dané entitě nebo uživateli.

[12] Typ symetrické šifry, která se pracuje s bloky pevně stanovené délky

[13] Typ symetrické šifry, kde vstupní datový tok je kombinován s pseudonáhodným proudem bitů vytvořeným z šifrovacího klíče a šifrovacího algoritmu

[14] Kryptografický protokol, umožňující přes nezabezpečený kanál vytvořit šifrované spojení

[15] Symetrická bloková šifra

[16] Koncový bod připojení přes počítačovou síť

[17] Protokol pro bezpečnou komunikaci s webovými servery pomocí HTTPS

- **javax.security.auth.callback**: Knihovna poskytuje třídy nezbytné pro služby, které interagují s aplikacemi za účelem načtení citlivých informací, kterými jsou například ověřovací údaje, uživatelské jméno a heslo. Knihovna také zprostředkovává informační kanál poskytující chybové a varovné zprávy.
- **javax.security.auth.login**: Knihovna poskytuje „zásuvný“ autentizační modul.
- **javax.security.auth.x500**: Knihovna umožňuje uložení X.500[18] pověření do skupiny informací o dané entitě nebo uživateli. Takové skupině se jednotně říká subjekt.
- **com.madgag.spongycastle**: Kryptografická knihovna třetích stran zajišťující kryptografickou podporu platformy Android. Knihovna vychází z mateřské knihovny Bouncy Castle, kterou však kvůli problematickému zavedení a konfliktům názvů tříd nelze pro platformu Android plnohodnotně využít. Řešením je právě knihovna Spongy Castle, zajišťující snazší manipulaci. Poslední aktualizaci knihovna zaznamenala v roce 2017.

Android knihovny

Rozšiřující sada knihoven ze strany platformy Android zahrnuje knihovny, které nejsou přímo součástí operačního systému. Tato sada knihoven zároveň spadá do projektu Android Jetpack. Android Jetpack představuje sadu knihoven, jejichž hlavním cílem je podporovat starší verze operačního systému Android. Knihovny poskytují zpětnou kompatibilitu tím, že jsou odděleny od API platformy Android. K aktualizacím dochází častěji, než k aktualizacím samotné platformy Android, tím pádem je zdrojový kód knihoven aktuální a vývojáři mohou pružněji reagovat na bezpečnostní hrozby a rizika. Rozšiřující knihovny spadající do projektu Jetpack jsou uvozovány klíčovým slovem „androidx“ v názvu knihovny. Zdroj [15] uvádí rozšiřující knihovny zajišťující kryptografickou podporu platformy ze strany Androidu, kterými jsou:

- **androidx.security.crypto**: Knihovna poskytuje podporu šifrování a dešifrování souborů, pomocí standardu AES-256 (Advanced Encryption Standard).
- **androidx.security.identity**: Knihovna poskytuje nástroje pro řízení přístupu na základě profilů a informací o dané entitě nebo uživateli. Knihovna taktéž poskytuje objekty, prostřednictvím kterých lze ukládat citlivé osobní údaje nebo dokumenty uživatelů.
- **androidx.biometric**: Knihovna poskytuje třídy a rozhraní pro podporu autentizace pomocí biometriky[19]. Knihovna také řídí správu informací a upozornění spojených s procesem autentizace pomocí biometriky.

[18] Sada mezinárodních standardů vyvinutých Mezinárodní telekomunikační unií
[19] Kvantitativní znak živého organizmu

1.2.3 Nativní knihovny

Nativní knihovny jsou napsané v nativním kódu, tedy za pomocí programovacího jazyka C/C++. Zaujmout co největší část trhu jde ruku v ruce s nutností rozšířit danou aplikaci jak na platformě Android, tak na jiných platformách, kterými jsou například iOS nebo WindowsPhone. Nativní kód je tedy něco jako společný jmenovatel, který umožňuje opakované použití základního kódu napříč platformami. Knihovny napsané v nativním kódu zároveň dosahují lepších výsledků z pohledu výpočetní a paměťové náročnosti. Sada nástrojů, která zprostředkovává komunikaci mezi knihovnami napsanými pomocí programovacího jazyka Java nebo Kotlin a nativními knihovnami je Android NDK popsaná v části 1.3. Pomocí této sady nástrojů lze vyvíjet aplikace, ve kterých se stěžejní paměťově a výpočetně náročné operace realizují prostřednictvím nativního kódu. Nativních knihoven zajišťujících kryptografickou podporu je celá řada. Pro účely této diplomové práce byly vybrány takové, které podporují operace nad eliptickými křivkami a kterým je zajišťována pravidelná podpora ze strany vývojářů. Zdroje [16], [17] uvádí nativní knihovny zajišťující kryptografickou podporu, kterými například jsou:

- **botan**: Kryptografická knihovna napsaná v jazyce C++. Knihovna je vydaná pod zjednodušenou BSD (Berkeley Software Distribution) licencí, u které stačí uvedení copyright notice (poznámky) a licenčních podmínek dané licence, nejčastěji přiložením souboru s názvem „license“. Knihovna implementuje TLS protokol, certifikáty X.509, AHEAD (Authenticated Encryption with Associated Data) šifry, celou řadu hašovacích funkcí, postkvantovou kryptografii a kryptografii založenou na eliptických křivkách. Knihovna taktéž zajišťuje hardwarovou podporu pro PKCS #11 popisující obecně použitelné rozhraní pro kryptografické tokeny a TPM[20] (Trusted Platform Module).
- **cryptlib**: Kryptografická knihovna napsaná v jazyce C. Implementuje SSL, SSH (Secure Shell), TLS, S/MIME[21], PGP[22] (Pretty Good Privacy), OpenPGP[23], PKI (Private Key Infrastrucutre), X.509 certifikáty, protokol správy certifikátů CMP[24] (Certificate Management Protocol), OCSP (Online Certificate Status Protocol) a SCEP[25] (Simple Certificate Enrollment Protocol). Knihovna taktéž podporuje kryptografii založenou na eliptických křivkách.
- **crypto++**: Kryptografická knihovna napsaná v jazyce C++. Knihovna si svou oblibu získala zejména v akademické sféře a studentských projektech. Posky-

[20]Specifikace popisující zabezpečený kryptoprocesor, na který lze ukládat šifrovací klíče
[21]Standard pro kryptografické zabezpečení elektronických zpráv
[22]Počítačový program, který umožňuje šifrování a podepisování založené na RSA
[23]Otevřené PGP eliminující potřebu licencovat
[24]Protokol používaný pro získání seznamu zneplatněných X.509 digitálních certifikátů
[25]Zjednodušený protokol pro zacházení s certifikáty

tuje kompletní kryptografickou implementaci bezpečnostních algoritmů včetně méně často používaných schémat. Knihovna také uchovává soubor nezabezpečených nebo zastaralých algoritmů pro zpětnou kompatibilitu a historickou hodnotu.

- **libsodium**: Kryptografická knihovna napsaná v jazyce C. Poskytuje nástroje pro šifrování, dešifrování, hašování a podepisování. Zjišťuje podporu pro generování náhodných dat, operace s klíči, autentizaci, kryptografii založenou na eliptických křivkách a celou řadu dalších operací.
- **mcl**: Kryptografická knihovna napsaná v jazyce C++ využívající párování, které představuje rozvíjející se typ asymetrické kryptografie založené na eliptických křivkách. Knihovna využívá BN (Barreto-Naehring) křivky, vhodné pro operaci párování, které zároveň zajišťují vysokou úroveň zabezpečení.

I přesto, že komunity vývojářů zajištujících podporu nativních knihoven pružně reagují na aktuální rizika a hrozby světa informačních technologií, je potřeba před použitím dané knihovny vždy zkontrolovat datum poslední aktualizace, popřípadě ověřit bezpečnost použitého protokolu včetně odpovídajících parametrů.

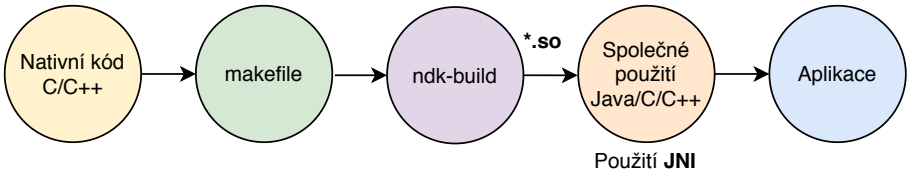
1.3 Android Native Development Kit

Android NDK představuje sadu nástrojů, pomocí kterých lze v rámci platformy Android využívat kód napsaný pomocí programovacích jazyků C/C++. Nástroj dále umožňuje přístup k nativním knihovnám, pomocí kterých lze přistupovat k fyzickým komponentám zařízení, kterými jsou například senzory nebo komponenty citlivé na dotyk.

V případech, kdy chce vývojář aplikace využít plný výkon zařízení nebo realizovat výpočetně náročné operace, představuje Android NDK velmi užitečnou sadu nástrojů. Pomocí Android NDK je pak možné volat funkce nativních knihoven přímo v aplikaci napsané pomocí programovacího jazyka Java a celý systém zahrnout do jednoho spustitelného *.apk* souboru. Možnost vykonávat určité operace za pomocí nativního kódu zvyšuje rychlost celé aplikace, která se se díky tomu stává atraktivnější pro zákazníka [18].

Rychlost nativních programovacích jazyků spočívá v tom, že se přímo kompilují do zdrojového kódu. Vyšší programovací jazyky s větší mírou abstrakce, jako například Java, Kotlin nebo Python stráví delší dobu kompilací do zdrojového kódu [19]. Tato doba je při běžném používání nepodstatná, nicméně při výpočetně náročných operacích znatelná. Kromě delší doby kompilace také nativní jazyky poskytují vývojáři lepší kontrolu nad správou paměti, tedy alokací a dealokací. V případě

Javy to zajišťuje Garbage collector[26]. Komunikaci mezi programovacím jazykem Java bežícím na Dalvik Virtual Machine s nativními jazyky C/C++ využívá Android NDK rozhraní JNI (Java Native Interface). Toto rozhraní zajišťuje konverzi datových typů, referenčních datových typů a propojuje samotný kód. Informace o použitých nativních knihovnách, včetně doplňujících informací nezbytných pro sestavení celého projektu, jsou uloženy v tkzv. *makefilu* ve formátu *.mk*[27]. Android NDK kompiluje nativní kód pomocí *ndk-build*[28] do souborů formátu *.so*. Tyto soubory jsou potom součástí balíčku souborů *.apk* reprezentujícího celou aplikaci. Proces zavedení nativního kódu do Android aplikace zobrazuje obrázek 1.4.



Obr. 1.4: Proces zavedení nativního kódu do Android aplikace.

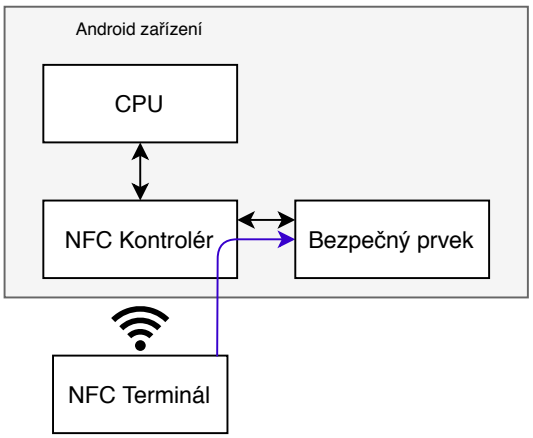
1.4 Emulace hostitelské karty

NFC (Near Field Communication) představuje bezdrátovou technologii umožňující rychlou a zabezpečenou výměnu dat na vzdálenost do 4 cm. Tuto technologii podporuje většina Android zařízení. I přesto se zejména v oblasti chytrých telefonů stále vyskytují i nově vydaná zařízení, která technologii nepodporují. Důvody jsou většinou snížení nákladů na výrobu nebo úspora místa a z toho vyplývající minimalističtější design, viz zdroj [21]. Se zmiňovanými praktikami se lze setkat zejména u levnějších zařízení. I pro tato zařízení však existuje cesta, jak komunikovat se svým okolím, například pomocí čtečky QR (Quick Response) kodů. Technologii NFC lze použít například při bezdrátových platbách, sdílení hesla Wi-Fi (Wireless Fidelity), odemčení dveří, autentizaci a mnohém dalším.

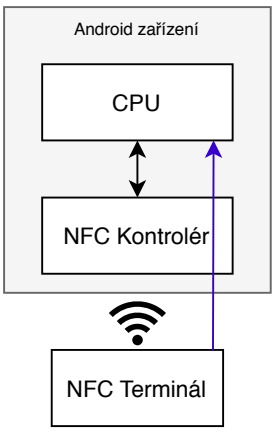
Mnoho Android zařízení podporujících NFC zároveň umožňuje emulaci hostitelské karty. U starších zařízení byla karta emulována prostřednictvím samostatného čipu nazývaného bezpečný prvek. Bezpečný prvek obsahují například SIM karty. Od verze 4.4 lze emulovat kartu i bez přítomnosti bezpečného prvku. To umožňuje jakékoliv aplikaci pro platformu Android komunikovat přímo s NFC terminálem [22]. Rozdílem tedy je, že při emulaci karty pomocí bezpečného prvku NFC kontroler směruje všechna data z NFC terminálu přímo na bezpečný prvek a do transakce není

[26]Způsob automatické správy paměti
[27]Typ souboru popisující zdroje a sdílené knihovny
[28]Nástroj pro sestavování projektů využívajících Android NDK

aplikace vůbec zapojena. Po dokončení transakce se aplikace může dotazovat bezpečného prvku na stav transakce a interagovat s uživatelem. Při emulaci hostitelské karty bez bezpečného prvku jsou data směrována přímo na hostitelské CPU (Central Processor Unit), na kterém běží samotná aplikace. Oba případy demonstrují obrázky 1.5 a 1.6. HCE umožňuje aplikacím vypadat jako chytré karty, které lze využívat při prokazování identity, bezkontaktní platbě, ověřování a mnohém dalším.



Obr. 1.5: Emulace karty s bezpečným prvkem.



Obr. 1.6: Emulace hostitelské karty bez bezpečného prvku.

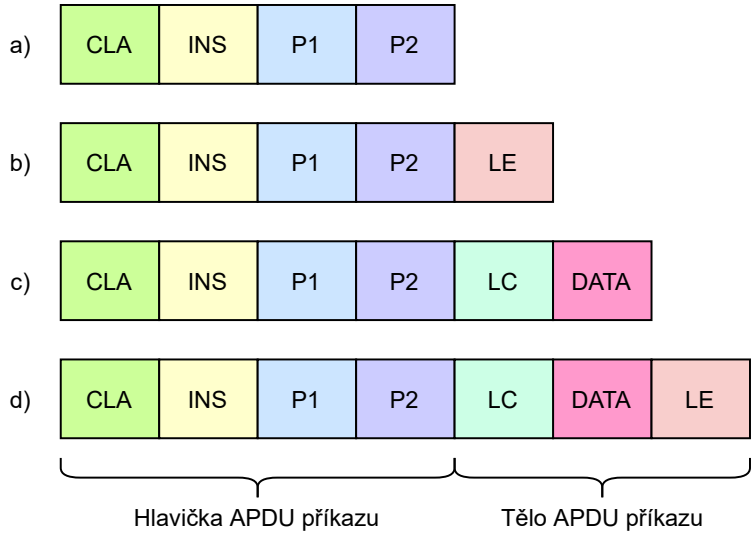
1.4.1 APDU zpráva

APDU (Application Protocol Data Unit) označuje komunikační protokol mezi čipovou kartou a čtecí jednotkou. Struktura APDU je definována normou ISO 7816-4 [20]. V rámci komunikace mezi čipovou kartou a čtecí jednotkou dochází k výměně APDU zpráv, které lze rozdělit do dvou kategorií.

První kategorii představuje APDU příkaz, který je zasílán čtecí jednotkou. Každý APDU příkaz je složen z hlavičky APDU příkazu a těla APDU příkazu. Hlavička obsahuje povinná pole a tělo pole volitelná. Tato pole jsou nositelem určité informace. APDU příkaz existuje v několika variantách. Jednotlivé varianty se od sebe liší absencí určitých polí v příkazu viz. obrázek 1.7. V případě APDU příkazu rozlišujeme tato pole:

- **CLA**: Instrukční třída, která určuje typ příkazu. Za pomocí instrukční třídy lze např. rozlišovat mezi proprietárním a průmyslovým řešením. Průmyslové řešení dodržuje sled zpráv dle určité specifikace. Proprietární řešení, jak již název napovídá, představuje vlastní řešení. Pod vlastním řešením si lze představit například komunikační protokol mezi věrnostní kartou a čtecí jednotkou nebo implementaci vlastního autentizačního protokolu. Délka `CLA` pole je 1 bajt.

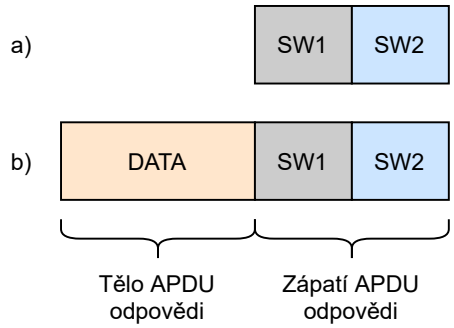
- **INS**: Instrukční kód určuje konkrétní příkaz. Příkazem může být například žádost o zaslání určité informace z čipové karty. Délka `INS` pole je 1 bajt.
- **P1**: První instrukční parametr příkazu. Tento parametr může být nositelem určité doplňující informace specifikující instrukční kód. Může se například jednat o sekvenční číslo. Délka `P1` pole je 1 bajt.
- **P2**: Druhý instrukční parametr příkazu. Tento parametr slouží ke stejnému účelu jako první instrukční parametr. Délka `P2` pole je 1 bajt.
- **LC**: Délka přenášených dat v bajtech. Délka `LC` pole může být 0, 1 nebo 3 bajty, dle použité instrukční třídy.
- **DATA**: Přenášená data APDU příkazu. Délka `DATA` pole je proměnlivá, dle použité instrukční třídy.
- **LE**: Očekávávaná délka APDU odpovědi v bajtech. Délka `LE` pole může být 0, 1, 2 nebo 3 bajty.



Obr. 1.7: Varianty APDU příkazu.

Druhou kategorii představuje APDU odpověď, která je zasílána čipovou kartou. Každá APDU odpověď je složena z těla APDU odpovědi a zápatí APDU odpovědi. Zápatí obsahuje povinná pole a tělo pole volitelná. APDU odpověď existuje ve dvou variantách. Varianty se od sebe liší absencí těla odpovědi viz obrázek 1.8. V případě APDU odpovědi rozlišujeme tato pole:

- **SW1**: První status provedení příkazu. Informuje o provedení přijatého příkazu. Délka `SW1` pole je 1 bajt.
- **SW2**: Druhý status provedení příkazu. Informuje o provedení přijatého příkazu. Délka `SW2` pole je 1 bajt.
- **DATA**: Data APDU odpovědi. Délka `DATA` pole je proměnlivá, dle použité instrukční třídy.

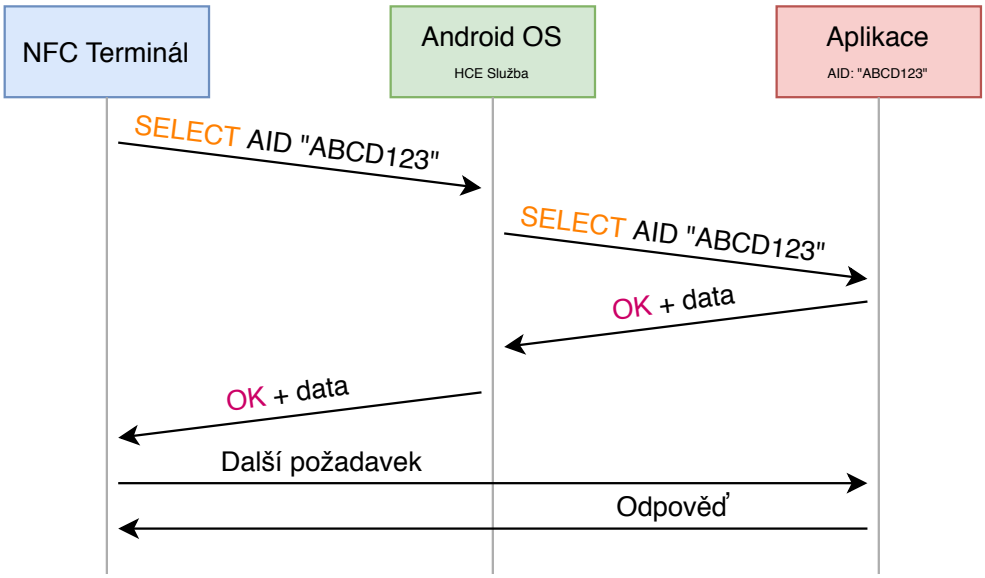


Obr. 1.8: Varianty APDU odpovědi.

1.4.2 Identifikátor aplikace

Čipové karty mohou obsahovat více aplikací. Tyto aplikace jsou označovány jako applety. Stejně jako čipová karta i zařízení s operačním systémem Android obsahuje mnoho aplikací. Každá Android aplikace disponuje unikátním identifikátorem označovaným zkratkou AID (Application Identifier), jehož délka může dosahovat až 16 bajtů, v případě HCE je minimální délka AID 5 bajtů [23]. AID jednoznačně identifikuje aplikaci v zařízení a obchodě Google Play. Dle zdroje [24], v případě že vývojář aktualizuje novou verzi aplikace, musí být AID a certifikát, kterým aplikaci podepisuje, stejné jako původní. Při změně AID by totiž byla aplikace považována za úplně jinou. V rámci HCE lze také využívat skupiny AID. Pomocí těchto skupin lze slučovat identifikátory aplikací, které k sobě patří nebo si vzájemně vyměňují data. Typickým příkladem může být aplikace kalendáře integrovaná do aplikace poskytující bankovní služby. Jestliže by NFC terminál vyžadoval specifickou skupinu AID, pak by HCE služba musela takovou skupinu spravovat. Neexistuje tedy stav, při kterém by NFC terminál komunikoval s HCE službou, která by spravovala pouze některé z potřebných AID. Platforma Android přidružuje skupiny AID do dvou hlavních kategorií. První je kategorie zaštiťující platby, druhou je kategorie ostatních aplikací HCE, do které spadají například aplikace umožňující využití věrnostní a autentizační hostitelské karty.

HCE využívá AID k jednoznačné identifikaci aplikace, se kterou chce NFC terminál komunikovat. NFC terminál nejdříve zasílá požadavek `SELECT` na AID nebo skupinu AID konkrétní aplikace. Operační systém Android pak jednoduše zprostředkuje komunikaci s danou aplikací na základě požadovaného AID a odpovídá zprávou `OK` společně s požadovanými daty. Zjednodušené schéma komunikace NFC terminálu s Android aplikaci zobrazuje obrázek 1.9.



Obr. 1.9: Komunikace NFC terminálu s Android aplikací využívající HCE.

2 Atributová autentizace

Autentizace je proces prokázání proklamované identity dokazujícího ověřovateli. Pojem je často zaměňován s pojmem autorizace, který představuje získání souhlasu o přístupu, na základě oprávnění nebo úspěšné autentizace. Ze zdroje [25] vyplývá, že kvůli požadavkům na ochranu soukromí, kterými jsou dle současných evropských nařízení a regulací zejména požadavky *privacy-by-design* a *privacy-by-default*, není standardní průběh autentizace dostačující. V rámci standardního průběhu autentizace lze profilovat uživatele na základně uživatelských identifikátorů a zároveň je pro svou jednoduchost v přímém rozporu s nejnovějšími požadavky. Z těchto a dalších důvodů byly zavedeny moderní kryptografické technologie rozšiřující standardní bezpečnostní funkce. Jedná se o technologie zajišťující anonymitu, nespojitelnost a nesledovatelnost. Souhrnně se tyto technologie označují zkratkou PET (Privacy Enhancing Technologies).

Autentizace jako taková je založena na jednom či více předpokladech. Dokazující buď zná nějaké heslo či frázi, disponuje nějakým předmětem nebo je reprezentován svými biometrickými či behaviorálními vlastnostmi. Identita uživatele však není ve všech případech dle nejnovějších požadavků skryta. Například při autentizaci prostřednictvím X.509 certifikátů dokazující předkládá ověřovateli svůj certifikát jehož součástí je i jeho identita, což je nežádoucí. Transakce jsou zároveň spojitelné, tím pádem umožňují profilovat daného uživatele.

Autentizace na základě atributů umožňuje dokazujícímu odhalit pouze nutné atributy, kterými je například věk, pohlaví, státní příslušnost nebo zaměstnanecký poměr. Ostatní atributy zůstávají skryty. Právě díky těmto vlastnostem je identita dokazujícího vždy skryta a jednotlivé transakce nejsou jednoduše spojitelné, tím pádem znemožňují proces profilování, viz zdroj [26].

Následující podkapitoly popisují obecný systém atributové autentizace včetně jeho vlastností a entit, existující schémata, stručně uvádějí reálnou implementaci v praxi. Závěrem kapitoly je rozebrán pilotní systém atributové autentizace RKVAC.

2.1 Systém atributové autentizace

Různé způsoby implementace atributové autentizace zaštiťují odpovídající atributové systémy. Dle zdroje [26] patří mezi nejznámější systémy U-Prove od společnosti Microsoft, Identity Mixer od společnosti IBM (International Business Machines Corporation), HM12 nebo CDDH19 navržené na VUT (Vysoké učení technické) v Brně. Každý ze systému přistupuje k implementaci atributové autentizace jinak. V rámci jednotlivých systémů jsou rozdílné role entit, protokoly nebo použité kryptografické algoritmy.

2.1.1 Entity a protokoly systému

Jednotlivé systémy atributové autentizace odlišuje různé použití protokolů nebo role entit. Dle zdroje [27] lze obecně nahlížet na implementaci atributové autentizace z pohledu čtyř entit a čtyř protokolů.

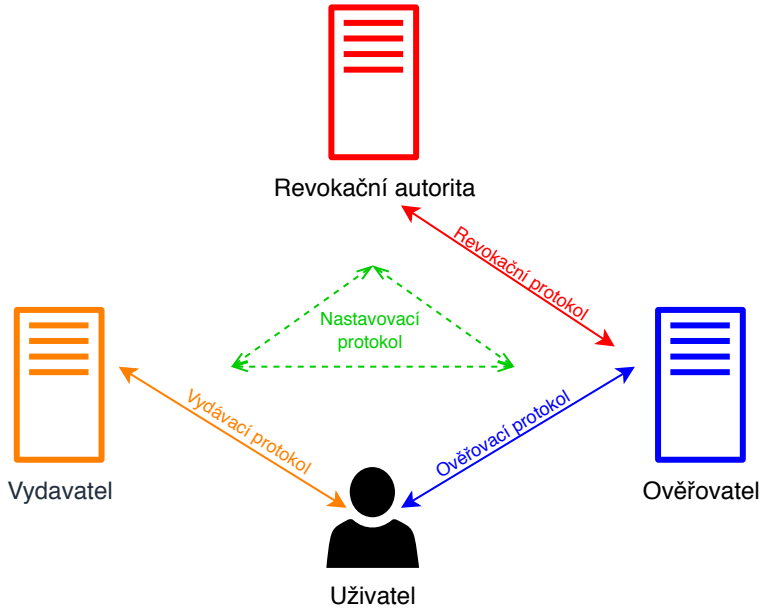
Entity systému představují specifické objekty, které si vzájemně vyměňují informace na základě předem dohodnutých pravidel. Obecně se lze setkat s následujícími entitami systému atributové autentizace:

- **Vydavatel (V)**: Potvrzuje žádosti o vydání atributů a vydává digitální pověření, které představuje množinu společných atributů. Vydavatel spolupracuje s ověřovatelem a revokační autoritou. Vlastní soukromý klíč vydavatele K_V.
- **Uživatel (U)**: Vlastník atributů uložených na vlastním zařízení (čipová karta, mobilní telefon,...), označované jako kryptografický token. Kryptografický token obsahuje soukromý klíč uživatele K_U. Uživatel prostřednictvím kryptografického tokenu anonymně prokazuje vlastnictví atributů ověřovateli.
- **Ověřovatel (O)**: Ověřuje důkaz držení atributů uživatele. Ověřovatel zajišťuje záznamy jednotlivých ověření pro případ porušení podmínek, při kterém kontaktuje revokační autoritu.
- **Revokační autorita (RA)**: Generuje parametry celého systému, revokuje neplatné atributy, digitální pověření nebo samotného uživatele ze systému. Revokační autorita vyřizuje žádosti ověřovatele pro zneplatnění atributů a zaznamenává informace poskytnuté vydavatelem. Revokovat lze i anonymitu uživatele nebo nespojitelnost relací. Vlastní klíč K_{RA}.

Elektronická komunikace je v systému atributové autentizace realizována pomocí protokolů stanovující pravidla, která vnáší do komunikace pevně danou podobu a pořadí informací, které jednotlivé entity očekávají na vstupu a výstupu. Obecně se lze setkat s následujícími protokoly systému atributové autentizace:

- **Nastavovací protokol**: Protokol, pomocí kterého dochází ke generování systémových parametrů a soukromých klíčů pro vydavatele a revokační autoritu.
- **Vydávací protokol**: Vydavatel komunikuje prostřednictvím vydávacího protokolu s uživatele, od kterého získává jednotlivé atributy v podobě digitálního pověření. Pověření je následně podepsáno soukromým klíčem K_V.
- **Ověřovací protokol**: Komunikační protokol ověřovatele a uživatele. Uživatel odhalí potřebné atributy a samotné digitální pověření spolu s důkazem vlastnictví skrytých atributů předloženého pověření. Ověřovatel důvěřuje vydavateli a ověřuje důkaz o vlastnictví.
- **Revokační protokol**: Komunikační protokol revokační autority a ověřovatele. Pomocí protokolu dochází k revokaci atributů, pověření, nespojitelnosti relaci nebo uživatelů.

Obecnou architekturu systému atributové autentizace společně s jednotlivými entitami komunikujícími pomocí odpovídajících protokolů demonstruje obrázek 2.1.



Obr. 2.1: Obecná architektura systému atributové autentizace.

2.1.2 Vlastnosti systému

Systém atributové autentizace by měl splňovat požadované vlastnosti. Tyto vlastnosti vyplývají z dlouhodobě zavedených standardů a zároveň reagují na nejnovější evropská nařízení a regulace. Obecnými požadovanými vlastnostmi systému atributové autentizace dle zdroje [28] jsou:

- **Anonymita**: Dokazující uživatel anonymně prokazuje držení potřebných atributů. Identita uživatele zůstává skryta.
- **Nespojitelnost relací**: Všechny transakce jsou vzájemně nespojitelné, což zamezuje procesu profilování. Ověřovatel není schopen sledovat ani profilovat dokazujícího uživatele v systému.
- **Nesledovatelnost**: Pověření vydaná vydavatelem jsou znáhodněná pomocí soukromých klíčů dokazujícího uživatele. To znemožňuje vydavateli sledovaní a profilování uživatele v systému.
- **Selektivní odhalení atributů**: Dokazující uživatel vybírá množinu atributů, kterou chce odkrýt, ostatní zůstávají skryty.
- **Nepřenositelnost**: Unikátní soukromý klíč dokazujícího uživatele uložený na čipové kartě nebo telefonu.

- **Nepadělatelnost**: Dokazující uživatel není schopen vytvořit validní digitální pověření, odpovídající pověření vydané vydavatelem.
- **Revokace**: Schopnost odstranění dokazujícího uživatele ze systému, společně s revokací jeho pověření, anonymity nebo nespojitelnosti relací.
- **Rychlost**: Systém je dostatečně rychlý i na výpočetně omezených zařízeních, kterými jsou například čipové karty.

2.2 Existující schémata

Jak již bylo zmíněno v předchozích kapitolách, mezi nejznámější systémy atributové autentizace patří U-Prove od společnosti Microsoft, Identity Mixer od společnosti IBM, HM12 nebo CDDH19 navržené na VUT v Brně. Následující podkapitola stručně popisuje jednotlivá existující schémata včetně jejich vlastnostní, výhod a nevýhod.

U-Prove

Systém atributové autentizace založený na problému diskrétního logaritmu. Dle zdroje [29] představuje U-Prove systém zajišťující komunikaci dokazujícího uživatele a vydavatele pomocí vydávacího protokolu a uživatele s ověřovatelem pomocí ověřovacího protokolu. Pro nastavení systémových parametrů schéma využívá nastavovací protokol. Jádrem celého systému je zaslepený Schnorrův podpis, který představuje formu digitálního podpisu zajišťujícího anonymitu. Systém taktéž využívá tzv. U-Prove token, který je unikátní pro každého uživatele systému a který uživatel poskytuje ověřovateli pro ověření podpisu vydavatele. Uživatel taktéž dokazuje vlastnictví skrtých atributů pomocí důkazu nulové znalosti realizovaný variantou Schnorrova protokolu. Token jako takový reprezentuje pseudonym, neboli krycí jméno uživatele. Součástí tokenu je podpis vydavatele, veřejný klíč tokenu a parametry TI (Token Informations) a PI (Prover Informations). Schéma zajišťuje anonymitu, nesledovatelnost a selektivní odhalení atributů. Nevýhodou schématu je, že nezajišťuje nespojitelnost relací, kvůli jedinečném pseudonymu.

Identity Mixer

Systém atributové autentizace založený na silném problému prvočíselné faktorizace. Systém podobně jako U-Prove zajišťuje komunikaci dokazujícího uživatele s vydavatelem pomocí vydávacího protokolu a uživatele s ověřovatelem pomocí ověřovacího protokolu. Pro nastavení systémových parametrů schéma využívá nastavovací protokol. Na rozdíl od systému U-Prove je jádrem systému Camenisch-Lysyanskaya

podpis zaručující ochranu soukromí a ověření atributů. Systém zajišťuje kromě anonymity, nesledovatelnosti a selektivního odhalení atributů i nespojitelnost relací, způsobenou znáhodněním pověření. Uživatel taktéž dokazuje vlastnictví skrytých atributů pomocí důkazu nulové znalosti realizovaný variantou Schnorrova protokolu. Systém zavádí časově omezenou platnost pověření, po jejímž vypršení dochází k prodloužení, nebo znehodnocení platnosti pověření. Velkou výhodou Identity Mixer systému je podpora podpisu více zpráv najednou. Více informací o systému Identity Mixer uvádí zdroj [30].

HM12

Systém atributové autentizace založený na problému diskrétního logaritmu. Na rozdíl od předchozích dvou systémů navíc využívá revokační autoritu a s tím spojený revokační protokol, pomocí kterého revokační autorita komunikuje jak s ověřovatelem, tak vydavatelem. Pro nastavení systémových parametrů schéma využívá nastavovací protokol. Prostřednictvím revokace revokační autorita zneplatňuje identitu uživatelů a zajišťuje jejich vyřazení ze systému. Více informací o systému HM12 uvádí zdroj [31].

CDDH19

Systém atributové autentizace založený na problému diskrétního logaritmu využívající operaci bilineárního párování nad eliptickými křivkami. Systém umožňuje klíčovou ověřitelnost, při které vydavatel s ověřovatelem sdílí soukromý klíč označovaný jako Algebraický MAC (Message Authentication Code). Jádrem systému je weak Boneh-Boyen podpis. Tento podpis může uživatel následně přepočítat a tím docílit nespojitelnosti relací a nesledovatelnosti. Systém taktéž poskytuje anonymitu a kompatibilitu s revokačními a identifikačními mechanismy a zároveň umožňuje praktickou implementovatelnost na výpočetně omezených zařízeních, kterými jsou například čipové karty. Více informací o systému CDDH19 uvádí zdroj [32].

2.2.1 Projekt IRMA

Projekt IRMA představuje sadu bezplatných open source projektů založených na Identity Mixer systému atributové autentizace. Dokazující uživatel zahájí komunikaci s důvěryhodným vydavatelem, který podepíše uživatelské atributy. Tyto atributy jsou následně uloženy v mobilní aplikaci a selektivně zveřejňovány dle volby dokazujícího uživatele. Pomocí podpisu uživatele pak ověřovatel ověřuje platnost atributů. Uživatel taktéž dokazuje vlastnictví skrytých atributů pomocí důkazu nulové znalosti. Ochranu atributů uložených pouze v samotné aplikaci zajišťuje taktéž

speciální PIN kód, který zná pouze uživatel aplikace. Implementace tedy umožňuje nesledovatelnost, anonymitu a zamezuje profilování. Jestliže nějaká služba vyžaduje autentizaci uživatele pomocí atributů, potom aplikace komunikuje pouze s danou službou. Řešení je mnohem šetrnější k soukromí uživatele na rozdíl od případů, při kterých se uživatelé přihlásí do svého Facebook nebo Google účtu, který dané službě poskytne všechny potřebné údaje a umožní i následné nežádoucí profilovaní. Řešení umožňuje propojení mobilní aplikace s webovou aplikací, pomocí které lze sledovat provedené autentizační úkony a správu uživatelských atributů. Pomocí webové aplikace lze také zablokovat IRMA účet, například při ztrátě zařízení. Mezi nevýhody aplikace patří fakt, že uživatel musí své atributy aktivně spravovat a po vypršení platnosti znovu obnovit. Mezi výhodu pro uživatele a nevýhodu pro samotný projekt také patří neschopnost vydělávat peníze prostřednictvím profilování uživatelů. Aplikace je volně dostupná jak pro platformu Android tak pro iOS. Více informací o projektu IRMA uvádí zdroj [33].

2.3 RKVAC

RKVAC (Revocable Keyed-Verification Anonymous Credential) je autentizační systém ze třídy ABCs (Attribute-Based Credential schemes). Schéma umožňuje rozsáhlé použití atributové autentizace i na omezených zařízeních, kterými jsou například čipové karty. Signaturou samotného systému je fakt, že entita ověřovatele sdílí soukromý klíč s entitou vydavatele. Tato vlastnost umožňuje při procesu ověřování využít symetrickou kryptografii. Stavební bloky systému jsou wBB (Weak Boneh-Boyen) podpis popsaný v části 2.3.2, anonymní dokazování a ověřování pověření pomocí klíčů. Systém zároveň splňuje všechny vlastnosti popsané v podkapitole 2.1.2. RKVAC systém představuje rozšíření vůči původnímu KVAC (Keyed-Verification Anonymous attribute-based Credentials) schématu založeném na algebraickém autentizačním kódu MAC a BBs (Boneh-Boyen signatures) [34, 35]. Původní schéma však neumožňuje proces revokace, který RKVAC systém umožňuje.

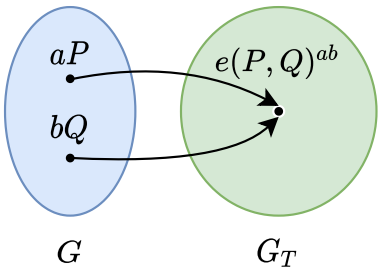
2.3.1 Bilineární párování

Bezpečnosti protokolu RKVAC úzce souvisí s bilineárním párování, jakožto operací nad eliptickou křivkou. Operace bilineární párování pracuje s třemi cyklickými grupami [36]. Cyklická grupa je taková grupa, která může být generována operováním s jedním jediným prvkem. Tento prvek se nazývá generátor cyklické grupy. G_1 a G_2 jsou cyklické aditivní grupy a G_T cyklická multiplikativní grupa. Prvočíselný řád q všech tří grup je stejný, viz [37]. Bilineární párování je ve své podstatě mapování bodů na křivce z cyklických aditivních grup do cyklické multiplikativní grupy. Celou situaci demonstruje obrázek 2.2. Operace párování je definována jako:

$$e : G_1 \times G_2 \rightarrow G_T$$

Takové mapování je pak bilineární, jestliže splňuje následující axiomi:

- **bilinearita**: $\forall P \in G_1, \forall Q \in G_2, \forall a, b \in Z_p$,
  $e(aP, bQ) = e(P, bQ)^a = e(P, bQ)^a = e(aP, Q)^b = e(P, Q)^{ab}$
- **nedegerativnost**: $\forall P \in G_1, Q \in G_2$ platí, že $e(P, Q) \in G_T$
- **spočitatelnost**: existuje algoritmus, který je schopný spočítat $e(P, Q)$ pro všechna $P \in G_1, Q \in G_2$



Obr. 2.2: Zjednodušené schéma operace $e : G_1 \times G_2 \rightarrow G_T$.

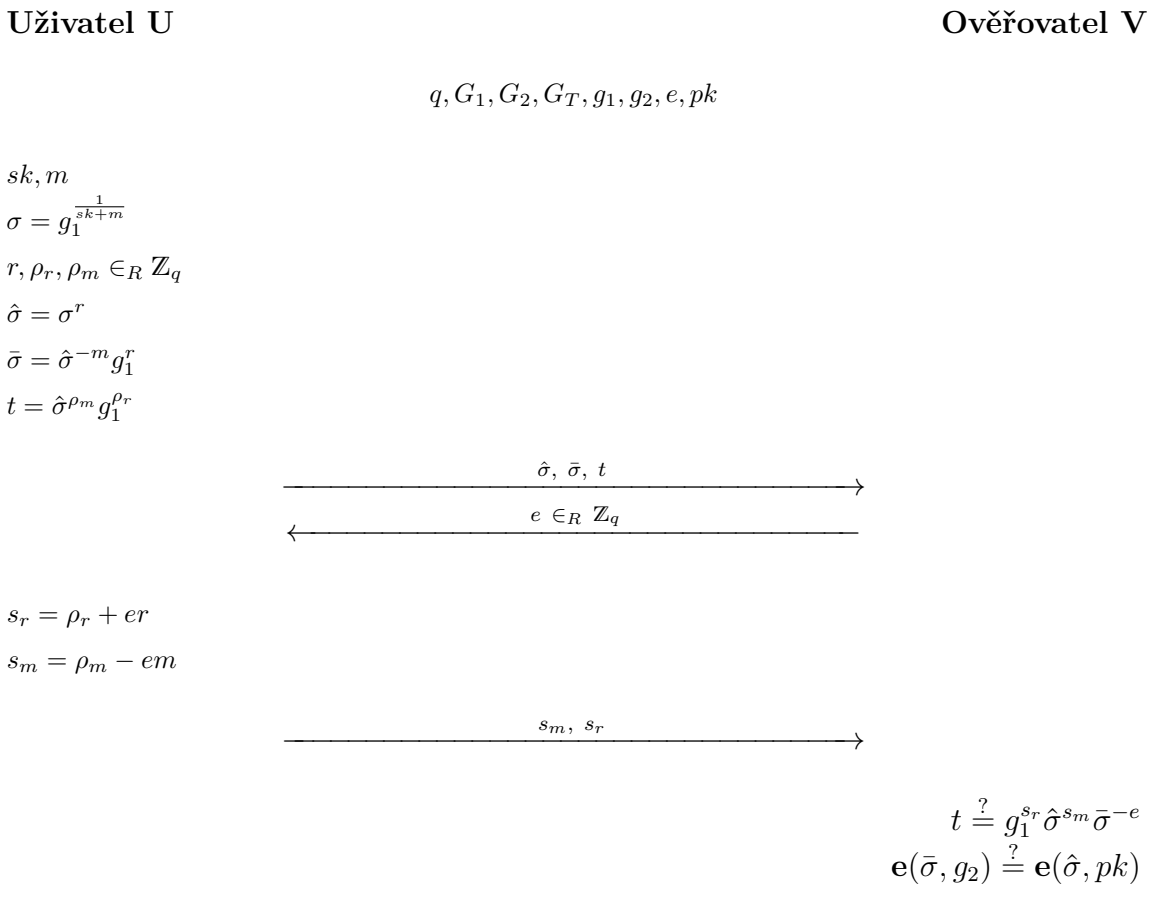
2.3.2 Weak Boneh-Boyen podpis

Weak Boneh-Boyen podpis představuje jeden ze základních stavebních bloků systému RKVAC. Schéma 2.3 je realizované pomocí tří dílčích algoritmů a využívá bilineární párování popsané v části 2.3.1. Obecné algoritmy wBB podpisu jsou:

- **Setup**: Algoritmus sloužící k inicializaci cyklických aditivních grup G_1, G_2 a cyklické multiplikativní grupy G_T. Kromě těchto grup jsou během algoritmu inicializovány generátory g_1, g_2, vytvořena mapa e a vygenerován pár soukromého a veřejného klíče.
- **Sign**: Algoritmus sloužící k tvorbě podpisu. Podpis je vytvořen pomocí zprávy a soukromého klíče. Podpis je realizován jako $\sigma = g_1^{\frac{1}{s+m}}$. V případě RKVAC

aplikace je postup obdobný. Výpočet pseudonymu je $C \leftarrow g_1^{\frac{1}{i-m_r+H(epoch)}}$. Do výpočtu však vstupují i jiné hodnoty popsané v části 2.3.4.

- **Verify**: Algoritmus pro ověření podpisu σ. Uživatel nejprve náhodně zvolí tři prvky ze Z_q, tyto prvky následně použije při znáhodnění podpisu σ. K těmto hodnotám se uživatel následně zaváže a na výzvu od ověřovatele sestrojí odpověď nesoucí důkaz znalosti. Tento důkaz následně zasílá ověřovateli k ověření.



Obr. 2.3: Schéma Weak Boneh-Boyen podpisu.

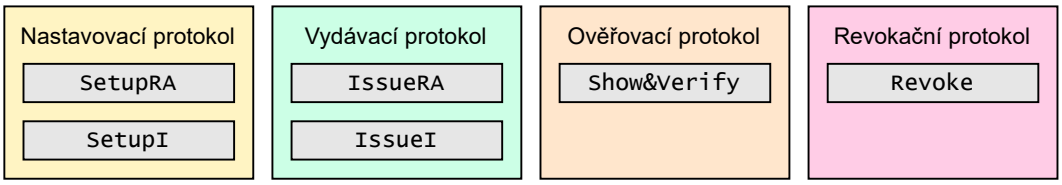
2.3.3 Entity systému

V rámci systému vystupují entity popsané v podkapitole 2.1.1, avšak s nepatrně jinými vlastnostmi. Popis jednotlivých entit systému vychází z odborného článku [38]. Entitami systému RKVAC jsou:

- **Revokační autorita** (RA): Přiřazuje a vydává unikátní revokační handler m_r, který taktéž představuje soukromý atribut každému uživateli. Tento soukromý atribut umožňuje revokační autoritě revokovat uživatele ze systému.
- **Vydavatel** (I): Je zodpovědný za vydávání atributů m_i uživateli prostřednictvím kryptografického pověření *cred*. Kryptografické pověření je podepsáno soukromým klíčem vydavatele.
- **Uživatel** (U): Při zavedení uživatele do systému obdrží uživatel vydavatelem podepsané kryptografické pověření *cred* obsahující vydané atributy m_i. Uživatel pak anonymně dokazuje vlastnictví atributů ověřovateli. Uživatel taktéž musí vypočítat jednorázový pseudonym C, který je propojen s kryptografickým pověřením *cred* pomocí revokačního handleru m_r.
- **Ověřovatel** (V): Ověřuje vlastnictví požadovaných atributů a revokační status revokačního handleru m_r.

2.3.4 Algoritmy systému

RKVAC systém taktéž obsahuje protokoly systému popsané v podkapitole 2.1.1. Tyto protokoly jsou však pro RKVAC systém přizpůsobeny a obsahují dílčí algoritmy. Popis dílčích algoritmů vychází z odborného článku [38]. Rozložení dílčích algoritmů ve vztahu k obecným protokolům systému zobrazuje obrázek 2.4.



Obr. 2.4: Rozložení dílčích algoritmů RKVAC systému mezi systémové protokoly.

Algoritmy atributově autentizačního systému RKVAC jsou:

- **SetupI**: Jedná se o dílčí algoritmus nastavovacího protokolu. Vstupem do algoritmu je bezpečnostní parametr k. Algoritmus generuje veřejné parametry systému, které představují vstupy do jiných dílčích algoritmů protokolů systému. Veřejnými parametry jsou bilineární grupa $params_I = (q, G_1, G_2, G_T, g_1, g_2, e)$, která splňuje $|q| = k$. Kromě veřejných parametrů algoritmus generuje soukromé klíče $sk_I = sk_V = (x_0, \ldots, x_{n-1}, x_r) \xleftarrow{\$} \mathbb{Z}_q$, kde n je počet atributů

v kryptografickém pověření. Tyto klíče jsou bezpečně distribuovány vydavateli a ověřovateli. Algoritmus běží na straně vydavatele.

- **SetupRA**: Jedná se o dílčí algoritmus nastavovacího protokolu. Vstupem do algoritmu je bezpečnostní parametr k a parametr ver_{max}, který slouží k nastavení maximálního počtu nespojitelných relací na uživatele v rámci jedné epochy. Revokační autorita nejdříve vybere svůj soukromý klíč tak, že $sk_{RA} \xleftarrow{\$} \mathbb{Z}_q$ a následně vypočítá svůj veřejný klíč $pk_{RA} = g_2^{sk_{RA}}$. Následně vypočte hodnotu parametru $ver_{max} = k^j$. V rámci algoritmu jsou dále vybrány náhodné hodnoty $(a_1, \ldots, a_j) \xleftarrow{\$} \mathbb{Z}_q$ a vypočteny parametry $h_z = g_1^{a_z}$ pro všechny z od 1 do j. Nakonec je v rámci algoritmu vygenerován prázdný revokační list RL a prázdný list revokačních handlerů RH. Algoritmus běží na straně revokační autority.
- **Issue**: Jedná se o algoritmus vydávacího protokolu. Tento algoritmus se skládá ze dvou dílčích algoritmů, kterými jsou IssueRA a IssueI. Vstupem do algoritmu jsou soukromý klíč revokační autority sk_{RA}, soukromý klíč vydavatele sk_I, seznam atributů $(m_1, \ldots, m_{n-1})$, uživatelský identifikátor ID a seznam revokačních handlerů RH. Výstupem algoritmu je pak podpis pověření σ, revokační handler m_r, podepsané randomizéry $(e_1, \ldots, e_k)$ a aktualizovaný seznam revokačních handlerů RH. Dílčí algoritmy algoritmu Issue jsou:
  - **IssueRA**: Algoritmus běží mezi uživatelem a revokační autoritou. RA si vybírá randomizéry $(e_1, \ldots, e_k) \xleftarrow{\$} \mathbb{Z}_q$ a podepisuje každý z nich pomocí wBB podpisu tak, že $\sigma_{e_z} \leftarrow g_1^{\frac{1}{e_z + sk_{RA}}}$ pro všechna z od 1 do k. RA taktéž podepíše revokační handler m_r společně s uživatelským identifikátorem ID tak, že $\sigma_{RA} = g_1^{\frac{1}{H(m_r||ID) + sk_{RA}}}$. RA aktualizuje seznam revokačních handlerů $RH = RH + (m_r||ID + \{(e_1, \sigma_{e_1}), \ldots, (e_k, \sigma_{e_k})\})$ a zasílá (m_r, σ_{RA}) a páry $\{(e_1, \sigma_{e_1}), \ldots, (e_k, \sigma_{e_k})\})$ zpět uživateli.
  - **IssueI**: Algoritmus běží mezi uživatelem a vydavatelem. Uživatel zasílá všechny své atributy včetně revokačního handleru $(m_1, \ldots, m_{n-1}, m_r)$ společně s podpisem σ_{RA} vydavateli. Vydavatel zkontroluje správnost podpisu, podepíše všechny požadované atributy pomocí soukromého klíče vydavatele tak, že $\sigma = g_1^{\frac{1}{x_o + m_1 x_1 + \cdots + m_{n-1} x_{n-1} + m_r x_r}}$ a zároveň vypočítá pomocné hodnoty $\sigma_{x_i} = \sigma^{x_i}$ pro $1 \leq i \leq n$. Výstupem algoritmu je podepsané pověření σ a pomocné hodnoty $(\sigma_{x_1}, \ldots, \sigma_{x_{n-1}}, \sigma_{x_r})$, které jsou zaslány uživateli.
- **Show&Verify**: Vstupem do algoritmu jsou na uživatelské straně uživatelovi atributy $(m_1, \ldots, m_{n-1}, m_r)$, podpis σ, randomizéry $\{(e_1, \sigma_{e_1}), \ldots, (e_k, \sigma_{e_k})\})$, indície odkrytých atributů $m_{z \in D}$ a identifikátor aktuální epochy *epoch*. Na straně ověřovatele jsou vstupem do algoritmu soukromý klíč ověřovatele sk_V,

revokační seznam RL a identifikátor aktuální epochy *epoch*. Výstupem od uživatele je pseudonym C kryptografický důkaz π o vlastnictví atributů. Výstupem od ověřovatele je logická 1 nebo 0, kde 1 značí povolení přístupu a 0 zamítnutí. Algoritmus běží mezi uživatelem a ověřovatelem. Uživatel nejdříve vypočítá jednorázový pseudonym C hashováním identifikátoru epochy *epoch* s unikátní relační hodnotou $i = \sum_{z=1}^{j} a_z e_z$, kde e_z a a_z jsou uživatelovi tajné parametry bezpečně uložené na čipové kartě, či chytrém zařízení. Do hashe vstupuje i revokační handler m_r. Uživatel následně znáhodní pověření a vypočítá důkaz znalosti všech všech atributů uvnitř pověření. Kromě toho taktéž uživatel dokáže, že jednorázový pseudonym C a podpis $\hat{\sigma}$ byly sestaveny za pomocí stejného revokačního handleru m_r. Nakonec ověřovatel ověří důkaz π a zkontroluje jestli zda-li není jednorázový pseudonym C na revokačním seznamu RL. Celý proces zobrazuje obrázek 2.5

- **Revoke**: Do algoritmu vstupuje seznam revokačních handlerů RH, veřejný revokační seznam RL, soukromý klíč revokační autority sk_{RA} a komunikační předpis (π, C) zaslaný od ověřovatele. Výstupem algoritmu je pak aktualizovaný revokační seznam RL'. Algoritmus běží mezi ověřovatelem a revokační autoritou. RA je schopna zrekonstruovat všechny pseudonymy všech uživatelů za epochu výpočtem $C = g_1^{\frac{1}{i - m_r + H(epoch)}}$, kde $i = a_1 e_I + a_2 e_{II}$ je vypočteno pro všechny možné kombinace a_j a e_k. Porovnáním C přijatého od ověřovatele se všemi sestavenými C uživatelů, je RA schopna identifikovat vlastníka C a umístit všechny jím vygenerované C do aktualizovaného seznamu RL.

Uživatel U **Ověřovatel V**

$$params_I = (q, G_1, G_2, G_T, g_1, g_2, e)$$
$$params_{RA} = (k, j, (h_1, \ldots, h_j), (a_1, \ldots, a_j))$$
$$pk_{RA}, RL$$

$Atributy : (m_1, \ldots, m_{n-1}, m_r)$
$\sigma, (\sigma_{x_1}, \ldots, \sigma_{n-1}, \sigma_{x_r})$ $\qquad sk_V = (x_0, \ldots, x_{n-1}, x_r)$

$\xleftarrow{nonce,\ epoch}$

$Randomizéry : \{(e_1, \sigma_{e_1}), \ldots, (e_k, \sigma_{e_k})\}$
$e_I, e_{II} \xleftarrow{\$} (e_1, \ldots, e_k)$
$\sigma_{e_I}, \sigma_{e_{II}} \xleftarrow{\$} (\sigma_{e_1}, \ldots, \sigma_{e_k})$
$i \leftarrow a_1 e_I + a_2 e_{II}$
$C \leftarrow g_1^{\frac{1}{i - m_r + H(epoch)}}$
$\rho, \rho_v, \rho_i, \rho_{m_r}, \rho_{m_{z \notin D}}, \rho_{e_I}, \rho_{e_{II}} \xleftarrow{\$} \mathbb{Z}_q$
$\hat{\sigma} \leftarrow \sigma^{\rho}$
$\hat{\sigma}_{e_I} \leftarrow \sigma_{e_I}^{\rho}, \hat{\sigma}_{e_{II}} \leftarrow \sigma_{e_{II}}^{\rho}$
$\bar{\sigma}_{e_I} \leftarrow \hat{\sigma}_{e_I}^{-e_I} g_1^{\rho}, \bar{\sigma}_{e_{II}} \leftarrow \hat{\sigma}_{e_{II}}^{-e_{II}} g_1^{\rho}$
$t_{verify} \leftarrow g_1^{\rho_v} \sigma_{x_r}^{\rho_{m_r}\rho} \prod_{z \notin D} \sigma_{x_z}^{\rho_{m_z}\rho}$
$t_{revoke} \leftarrow C^{\rho_{m_r}} C^{\rho_i}$
$t_{sig} \leftarrow g_1^{\rho_i} h_1^{\rho_{e_I}} h_2^{\rho_{e_{II}}}, t_{sigI} \leftarrow g_1^{\rho_v} \hat{\sigma}_{e_I}^{\rho_{e_I}}, t_{sigII} \leftarrow g_1^{\rho_v} \hat{\sigma}_{e_{II}}^{\rho_{e_{II}}}$
$e \leftarrow H(t_{verify}, t_{revoke}, t_{sig}, t_{sigI}, t_{sigII}, \hat{\sigma}, \hat{\sigma}_{e_I}, \bar{\sigma}_{e_I}, \hat{\sigma}_{e_{II}}, \bar{\sigma}_{e_{II}}, C, nonce)$
$\langle s_{m_z} \leftarrow \rho_{m_z} - e m_z \rangle_{z \notin D}$
$s_v \leftarrow \rho_v + e\rho$
$s_{m_r} \leftarrow \rho_{m_r} - e m_r$
$s_i \leftarrow \rho_i + ei$
$s_{e_I} \leftarrow \rho_{e_I} - e e_I, s_{e_{II}} \leftarrow \rho_{e_{II}} - e e_{II}$
$\pi \leftarrow (e, s_{m_{z \notin D}}, s_v, s_{m_r}, s_i, s_{e_I}, s_{e_{II}})$

$\xrightarrow{m_{z\ \in\ D},\ \pi,\ C,\ \hat{\sigma},\ \hat{\sigma}_{e_I},\ \bar{\sigma}_{e_I},\ \hat{\sigma}_{e_{II}},\ \bar{\sigma}_{e_{II}}}$

$t_{verify} \leftarrow \hat{\sigma}^{-e x_0} g_1^{s_v} \hat{\sigma}^{x_r s_{m_r}} \prod_{z \notin D} \hat{\sigma}^{x_z s_{m_z}} \prod_{z \in D} \hat{\sigma}^{-e x_z m_z}$
$t_{revoke} \leftarrow (g_1 C^{-H(epoch)})^{-e} C^{s_{m_r}} C^{s_i}$
$t_{sig} \leftarrow g_1^{s_i} h_1^{s_{e_I}} h_2^{s_{e_{II}}}, t_{sigI} \leftarrow g_1^{s_v} \hat{\sigma}_{e_I}^{s_{e_I}} \bar{\sigma}_{e_I}^{-e}, t_{sigII} \leftarrow g_1^{s_v} \hat{\sigma}_{e_{II}}^{s_{e_{II}}} \bar{\sigma}_{e_{II}}^{-e}$
$e \stackrel{?}{=} H(t_{verify}, t_{revoke}, t_{sig}, t_{sigI}, t_{sigII}, \hat{\sigma}, \hat{\sigma}_{e_I}, \bar{\sigma}_{e_I}, \hat{\sigma}_{e_{II}}, \bar{\sigma}_{e_{II}}, C, nonce)$
$\mathbf{e}(\bar{\sigma}_{e_I}, g_2) \stackrel{?}{=} \mathbf{e}(\hat{\sigma}_{e_I}, pk_{RA})$
$\mathbf{e}(\bar{\sigma}_{e_{II}}, g_2) \stackrel{?}{=} \mathbf{e}(\hat{\sigma}_{e_{II}}, pk_{RA})$
$C \notin RL$

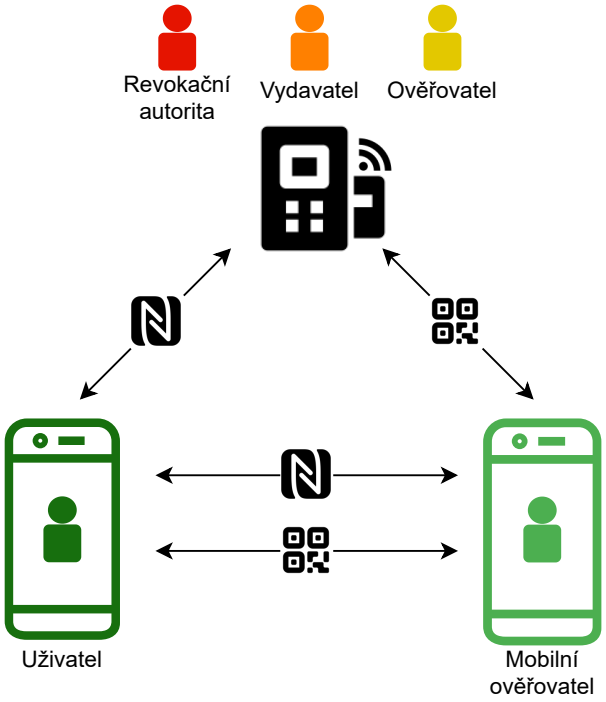
Obr. 2.5: Algoritmus Show&Verify ověřovacího protokolu systému RKVAC.

3 Implementace pilotního systému atributové autentizace

V následující kapitole je popsán průběh implementace pilotního systému RKVAC, popsaného v části 2.3, na platformě Android. První část popisuje postup implementace aplikace pro uživatelskou entitu za pomocí jazyka Java. Podkapitoly popisují architekturu uživatelské aplikace, implementaci kryptografického jádra uživatele založeného na nativní kryptografické knihovně MCL, implementaci služby emulující hostitelskou kartu a grafické uživatelské rozhraní aplikace.

Druhá část se věnuje aplikaci ověřovatele, naprogramované taktéž za pomocí jazyka Java. Podkapitoly popisují architekturu aplikace, implementaci kryptografického jádra ověřovatele založeného na nativní kryptografické knihovně MCL, implementaci emulátoru NFC terminálu a grafickému uživatelskému rozhraní aplikace.

Implementace systému vychází ze schématu 3.1, ve kterém vystupuje uživatel, mobilní ověřovatel a čtecí jednotka napojená na terminál s aplikací RKVAC. Terminál s aplikací RKVAC zaštiťuje entitu revokační autority, vydavatele a ověřovatele. Mobilní ověřovatel je ekvivalentní s ověřovatelem na straně terminálu s tím rozdílem, že může fungovat i samostatně, například ve stížených podmínkách. Jednotlivé entity si mezi sebou vyměňují potřebná data pomocí technologie QR kódu nebo NFC.



Obr. 3.1: Schéma systému RKVAC na platformě Android.

3.1 Aplikace uživatele

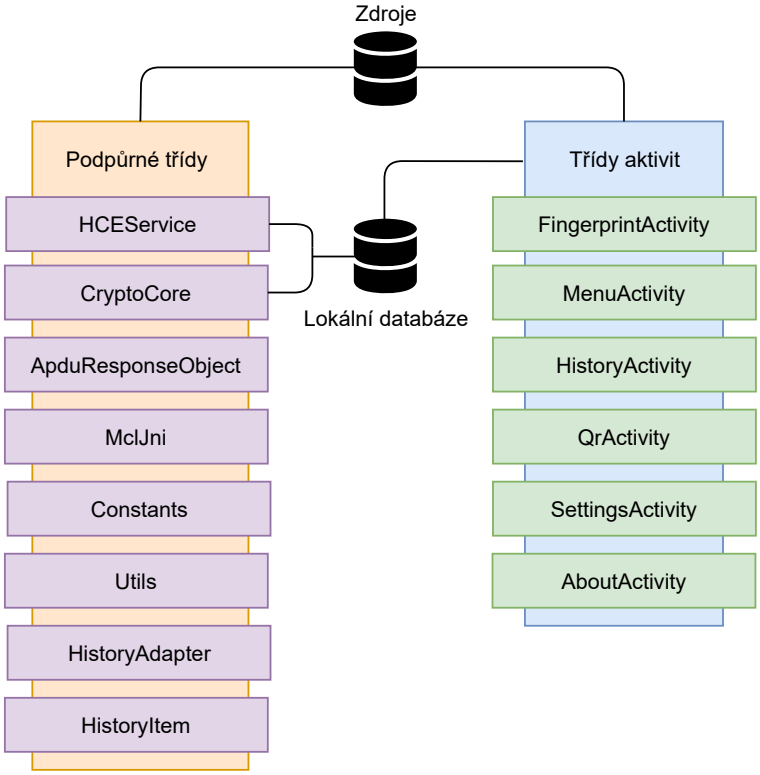
Entita uživatele systému RKVAC komunikuje s ostatními entitami za účelem výměny informací. S každou z entit uživatel komunikuje z jiného důvodu. Všechny tyto případy je třeba při implementaci aplikace zohlednit. Aplikace jako taková emuluje čipovou kartu prostřednictvím služby HCE popsané v části 1.4. Díky této službě je aplikace schopna zasílat informace pomocí APDU zpráv popsaných v části 1.4.1. Důležité je taktéž zohlednit fakt, že čipová karta ve své podstatě nemá žádné grafické uživatelské rozhraní, pomocí kterého by uživatel interagoval se samotnou kartou. Pokud uživatel čipovou kartu nepřiloží k čtecímu zařízení, tak se není schopen dozvědět, které informace karta nese. Obě tyto vlastnosti čipových karet aplikace uživatele zohledňuje. V případě běžného použití stačí mít aplikace spuštěnou na pozadí a telefon přikládat k čtecí jednotce v podobě NFC terminálu. Pokud by se však uživatel chtěl dozvědět o úkonech, ke kterým s aplikací docházelo, pak je mu i toto umožněno prostřednictvím grafického uživatelského rozhraní samotné aplikace.

3.1.1 Architektura aplikace

Architektura aplikace vychází ze schématu 3.2. Aplikace je tvořena z dílčích bloků, které jsou vzájemně propojeny. Stavební bloky aplikace uživatele jsou:

- **Zdroje**: Data, nastavení a konfigurace, které jednotlivé třídy aktivit či podpůrné třídy potřebují pro svůj běh. Pod zdroji si lze představit například zdrojové soubory použité vektorové grafiky, AndroidManifest.xml popsaný v části 1.1.1, layout soubory, které určují uskupení jednotlivých elementů aktivit, zdrojové soubory nativní kryptografické knihovny MCL a mnohé další.
- **Lokální databáze**: Lokální databázi aplikace uživatele realizuje nástroj Shared Preferences. Díky rozhraní tohoto nástroje lze lokálně ukládat data v podobě párů klíčů a jim příslušných hodnot. V případě vypnutí aplikace či samotného zařízení zůstávají data uložena na zařízení. V lokální databázi jsou uloženy hodnoty atributů, aktuální stavy jednotlivých elementů aktivit a systémové parametry. Z podpůrných tříd s lokální databází komunikuje pouze služba emulující hostitelskou kartu a kryptografické jádro uživatele. Ostatní podpůrné třídy s lokální databází přímo nekomunikují.
- **Třídy aktivit**: Každá aktivita má svoji příslušnou třídu. Třída každé aktivity spravuje jednotlivé elementy aktivity. Třídy aktivit komunikují se zdroji, které aktivitám poskytují potřebná data a konfigurace. Třídy aktivit zároveň komunikují s lokální databází, která zprostředkovává aktuální stavy jednotlivých elementů, popřípadě poskytuje další potřebná data.

- **Podpůrné třídy**: Třídy realizující komunikaci v pozadí, včetně všech potřebných výpočtů. Podpůrné třídy slouží k podpoře běhu tříd aktivit. Starají se o konverzi a správu dat, kryptografické výpočty, distribuci dat a výsledků.



Obr. 3.2: Architektura aplikace uživatele.

3.1.2 Zavedení knihovny MCL

Pro zavedení nativní kryptografické knihovny MCL popsané v části 1.2.3 je použita sada nástrojů Android NDK popsaná v části 1.3. Pomocí této sady lze v rámci platformy Android využívat kód napsaný pomocí programovacích jazyků C/C++. Celý proces je realizován na operačním systému Linux Ubuntu ve verzi 18.06.

Prvním krokem je instalace knihovny GMP (GNU Multi-Precision) představující open-source knihovnu umožňující výpočty s libovolnou přesností pro celá, racionální i reálná čísla s pohyblivou desetinnou čárkou [39]. Instalaci knihovny lze realizovat pomocí příkazu uvedeného ve výpisu 3.1

```
$ sudo apt install libgmp-dev
```

Výpis 3.1: Příkaz pro instalaci knihovny GMP.

Dalším krokem je naklonování repozitáře nativní knihovny MCL do námi zvoleného adresáře. Tento krok lze realizovat pomocí nástroje *Git* a příkazem *clone* dle výpisu 3.2.

```
$ sudo apt install git-all
$ git clone git://github.com/herumi/mcl
```

Výpis 3.2: Naklonování repozitáře nativní kryptografické knihovny MCL.

Následuje stažení a instalace sady nástrojů Android NDK z oficiálních zdrojů platformy Android. Po úspěšné instalaci nástroje stačí spustit příkaz dle výpisu 3.3. Autor knihovny poskytuje nativní rozhraní JNI popsané v části 1.3, společně s nezbytnými *makefile* soubory, pomocí kterých lze realizovat kompilaci knihovny do souboru formátu *.so*.

```
$ cd mcl/ffi/java/android/jni
$ ndk-build
```

Výpis 3.3: Kompilace nativní kryptografické knihovny MCL.

Pro budoucí rozšiřitelnost aplikace je vhodné jednotlivé *.so* soubory zkompilovat pro aktuálně používaná nízkoúrovňová rozhraní označovaná ABI (Application binary interface). Různá zařízení běžící na platformě Android disponují jiným CPU s jinou instrukční sadou. Každá kombinace CPU a instrukční sady má své nízkoúrovňové rozhraní ABI. Knihovna byla zkompilována pro všechna aktuálně používaná ABI rozhraní [40].

Posledním krokem je zavedení knihovny do Android aplikace. Pro vygenerované soubory, představující zkompilovanou podobu nativní knihovny, je v projektu aplikace vytvořen adresář s názvem *jniLibs*. Tento adresář slouží jako zdrojový adresář pro jednotlivé *.so* soubory. Adresář obsahuje podadresáře odpovídající příslušným ABI rozhraním. Kromě toho je třeba do projektu zavést soubor Java tříd společně s třídou sloužící jako JNI pro knihovnu MCL. Autor knihovny dané třídy poskytuje na svém veřejném repozitáři. Pro tyto třídy je vytvořen v projektu adresář *herumi.mcl* obsahující třídy kryptografických objektů. Třída sloužící jako JNI pro nativní knihovnu MCL je pojmenována *MclJni* a je součástí bloku podpůrných tříd popsaných v části 3.1.1.

3.1.3 Zavedení služby HCE

Zavedení služby emulující hostitelskou kartu, popsanou v teoretické části 1.4, sestává z několika kroků. Výsledkem zavedení je schopnost aplikace, chovat se jako čipová karta. Taková aplikace je pak schopna odpovídat pomocí APDU odpovědí popsaných v části 1.4.1, na přijaté APDU příkazy. Pro zavedení služby HCE je třeba nejdříve definovat nutné požadavky a oprávnění a následně definovat samotnou službu HCE.

Definice nutných požadavků a oprávnění

Služba HCE vyžaduje od systému Android specifické hardwarové a softwarové funkce. Kromě těchto požadovaných funkcí taktéž pracuje s chráněnými rozhraními, popsanými v části 1.1.2. Pro definici požadavků a oprávnění slouží kořenový soubor AndroidManifest.xml popsaný v části 1.1.1.

Požadované softwarové a hardwarové funkce, které daná aplikace využívá, se deklarují pomocí elementu *<uses-feature>*. Samotný element disponuje několika atributy. Zdroj [56] uvádí atributy tohoto elementu, kterými jsou:

- **android:name**: Definuje jednu hardwarovou nebo softwarovou funkci používanou aplikací. Platné hodnoty atributu jsou uvedeny v dokumentaci dané hardwarové nebo softwarové funkce.
- **android:required**: Definuje míru potřeby dané hardwarové nebo softwarové funkce pro běh samotné aplikace. Atribut nabývá hodnot *true* nebo *false*. Hodnota *true* popisuje situaci, při které aplikace nemůže fungovat nebo není navržena tak, aby fungovala bez dané funkce v systému. Hodnota *false* naopak uvádí situaci, při které aplikace danou funkci preferuje, avšak nevyžaduje.
- **android:glEsVersion**: Definuje verzi OpenGL. Pokud atribut není zadaný, pak se předpokládá, že aplikace vyžaduje pouze OpenGL ES ve verzi 1.0, která je podporována všemi zařízeními se systémem Android.

Použití elementu *<uses-feature>* v kořenovém souboru AndroidManifest.xml zobrazuje výpis 3.4.

```
<uses-feature
 android:name="android.hardware.nfc.hce"
 android:required="true">
</uses-feature>
```

Výpis 3.4: Element *<uses-feature>* v AndroidManifest.xml.

Dalším důležitým krokem je definice systémových oprávnění. Tato oprávnění uživatel uděluje aplikaci při její instalaci. Požadovaná oprávnění, které daná aplikace vyžaduje, se deklarují pomocí elementu *<uses-permission>*. Element se taktéž přidává do kořenového souboru AndroidManifest.xml. Zdroj [57] uvádí atributy tohoto elementu, kterými jsou:

- **android:name**: Definuje název oprávnění. Může se jednat o standardizovaný název nebo o název definovaný jinou aplikací.
- **android:maxSdkVersion**: Definuje nejvyšší úroveň API, na které by mělo být dané aplikaci uděleno toto oprávnění. Nastavení je vhodné v případě, kdy vyšší úroveň API dané oprávnění explicitně nevyžaduje.

Použití elementu<uses-permission>v kořenovém souboru AndroidManifest.xml zobrazuje výpis 3.5.

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

Výpis 3.5: Element *<uses-permission>* v AndroidManifest.xml.

Definice služby HCE

Kromě požadavků a oprávnění je potřeba definovat samotnou službu emulující hostitelskou kartu. Tuto službu je pak vhodným způsobem možné spouštět a vypínat dle potřeb uživatele aplikace. Prvním krokem je přidání elementu *<service>* pod element *<application>* v kořenovém souboru AndroidManifest.xml. Dle zdroje [58] služba vyžaduje určité atributy, kterými se stanovuje jméno služby, oprávnění a podobné. Kromě atributů lze dané službě definovat filtr záměrů. Filtr záměrů představuje element *<intent-filter>*, který stanovuje typy záměrů, na které může aktivita, služba nebo přijímač vysílání reagovat. Vedle filtru záměrů lze službě definovat meta data pomocí elementu *<meta-data>*. Element slouží k definici dat, se kterými daná služba pracuje společně s odkazem na samotná data. Pro účely této diplomové práce byla vytvořena metadata, která obsahují AID popsané v části 1.4.2. Metadata je vhodné uložit do kořenové složky zdrojů projektu dané aplikace. Použití elementu *<service>* v kořenovém souboru AndroidManifest.xml zobrazuje výpis 3.6.

```
<service
 android:name=".HCEService"
 android:exported="true"
 android:permission="android.permission.BIND_NFC_SERVICE">
 <intent-filter>
  <action android:name="android.nfc.cardemulation.action.HOST_APDU_SERVICE"/>
  <category android:name="android.intent.category.DEFAULT"/>
 </intent-filter>
 <meta-data
  android:name="android.nfc.cardemulation.host_apdu_service"
  android:resource="@xml/apduservice">
 </meta-data>
</service>
```

Výpis 3.6: Element *<service>* v AndroidManifest.xml.

Metadata uložená v kořenové složce zdrojů projektu dané aplikace zobrazuje výpis 3.7. Jedná se o definici AID, které NFC terminál použije při komunikaci s aplikací. Pro účely této diplomové práce byl zvolen řetězec číslic 76757432313031. V rámci

metadat se definuje atribut ovlivňující požadavek na odemčení zařízení při používání služby nebo kategorie AID skupiny.

```
<?xml version="1.0" encoding="utf-8"?>
<host-apdu-service
 xmlns:android="http://schemas.android.com/apk/res/android"
 android:requireDeviceUnlock="false">
  <aid-group android:category="other">
   <aid-filter android:name="76757432313031"></aid-filter>
  </aid-group>
</host-apdu-service>
```

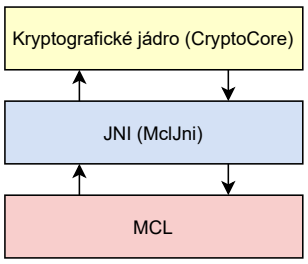
Výpis 3.7: Metadata definující AID pro NFC terminál.

3.1.4 Implementace kryptografického jádra

Po úspěšném zavedení nativní kryptografické knihovny MCL, lze implementovat samotné kryptografické jádro aplikace uživatele. Kryptografické jádro je realizováno podpůrnou třídou *CryptoCore*, která komunikuje s knihovnou MCL prostřednictvím nativního rozhraní realizovaného podpůrnou třídou *MclJni*. Blokové schéma komunikace kryptografického jádra s nativní knihovnou MCL zobrazuje obrázek 3.3. Toto rozhraní zajišťuje konverzi datových typů, referenčních datových typů a propojuje samotný kód. Rozhraní obsahuje deklarace metod knihovny MCL. Příklad JNI nativní kryptografické knihovny MCL uvádí výpis 3.8.

```
public class MclJni
{
  public final static native void SystemInit(...);
  public final static native void Add(...);
  public final static native void Mul(...);
  ...
}
```

Výpis 3.8: Příklad JNI pro nativní kryptografickou knihovnu MCL.



Obr. 3.3: Blokové schéma komunikace kryptografického jádra s knihovnou MCL.

Následně je třeba naimportovat balíček knihovny MCL a pak už stačí pomocí statického inicializačního bloku knihovnu načíst prostřednictvím metody *LoadLibrary()* a provolat v konstruktoru třídy *CryptoCore* metodu *SystemInit()*, pomocí které se nastaví Barreto-Naehring křivka. Kromě toho do konstruktoru vstupuje i *Context*. Pomocí kontextu je možné kryptografické jádro volat napříč aplikací. Demonstraci procesu načtení nativní kryptografické knihovny MCL uvádí výpis 3.9.

```
import com.herumi.mcl.*;
public class CryptoCore
{
 CryptoCore(Context context) {
 this.context = context;
 MclJni.SystemInit(MclConstants.BN254); }

 static { System.loadLibrary("mcljava"); }
 ...
}
```

Výpis 3.9: Načtení nativní kryptografické knihovny MCL.

Kryptografické jádro obstarává dílčí výpočty strany uživatele schématu 2.5. Při implementaci jádra je využita Barreto-Naehring křivka BN-254 knihovny MCL. Pro účely této diplomové práce je jako příklad uvedeno použití kryptografického jádra pro výpočet hodnoty pseudonymu C.

Postup výpočtu C

Metoda třídy *CryptoCore* s názvem *computePseudonym()* zajišťuje výpočet dané hodnoty parametru. Výpočet je realizován dle následující rovnice:

$$C \leftarrow g_1^{\frac{1}{i-m_r+H(epoch)}}$$

Význam jednotlivých parametrů uvádí část 2.3. Výstupem metody je bod na eliptické křivce. V první kroku dojde k inicializaci objektu *SharedPreferences*. Objekt zprostředkovává komunikaci s lokální databází. Inicializace objektu je realizována následujícím způsobem:

```
SharedPreferences sP = context.getSharedPreferences("UserData", 0);
```

V dalším kroku dojde k inicializaci hodnot parametrů vstupujících do výpočtu. Jednotlivé parametry jsou namapovány na objekt třídy *Fr*. Jedná se o konečné prostory řádu q. Proměnné *tmp_1,tmp_2* slouží jako kontejnery pro mezivýpočty. Tento krok je realizován následujícím způsobem:

```
Fr multiplier = new Fr();
Fr tmp_1 = new Fr();
Fr tmp_2 = new Fr();
Fr m_r = new Fr(sP.getString("m_r", "00"), 16);
```

V dalším kroku již dochází k mezivýpočtu $i - m_r$. K tomuto mezivýpočtu je použita metoda knihovny MCL. Výsledek je uložen do proměnné *tmp_1* Krok je proveden následovně:

```
MclJni.sub(tmp_1, i, m_r);
```

V dalším kroku dochází k výpočtu hashe. Nejdříve se načte hodnota *epoch* z lokální databáze aplikace. Pro výpočet je použita hashovací funkce SHA-1 [41]. Výstupem funkce jsou data o velikosti 20 bajtů. Tyto data je třeba doplnit paddingem (doplňkem), kvůli kompatibilitě s knihovnou MCL, která v případě mapování na objekty třídy *Fr* očekává vstupy o délce 32 bajtů. Takovýto výstup je následně převeden na objekt typu *BigInteger*, kvůli nutné modulární redukci řádem q. Výsledek je potom namapován na objekt třídy *Fr*. Průběh daného kroku je následovný:

```
String epoch_hash = new BigInteger(SHA1_PADDING +
        SHA1(sP.getString("epoch", "00")), 16)
       .mod(BN256_q).toString(16);
Fr epoch_tmp = new Fr(epoch_hash, 16);
```

V dalším kroku je možné realizovat výpočet exponentu, tedy $\frac{1}{i-m_r+H(epoch)}$. Nejedná se ve své podstatě o exponent, nýbrž o činitele bodu na křivce skrytého pod proměnnou *multiplier*. Výpočet je tedy realizován tímto způsobem:

```
MclJni.add(tmp_2, tmp_1, epoch_tmp);
MclJni.div(multiplier, new Fr(1), tmp_2);
```

Posledním krokem je pak výpočet samotného pseudonymu C. Ten je následně namapován na objekt třídy *G1*. Třída reprezentuje cyklickou aditivní grupu G_1. Výsledek výpočtu je i návratovou hodnotou celé metody *ComputePseudonym()*. Poslední krok je proveden tímto způsobem:

```
G1 C = new G1();
MclJni.mul(C, g1, multiplier);
return C;
```

3.1.5 Implementace služby HCE

Po tvorbě souboru obsahujícího metadata a definici nezbytných požadavků, oprávnění a služby jako takové, lze implementovat samotnou službu emulující hostitelskou kartu. Pro tyto účely umožňuje vývojové prostředí Android studio tvorbu komponenty s názvem *Service*. Jedná se o běžnou třídu, která pro účely implementace služby HCE dědí ze třídy *HostApduService*. Abstraktní třída *HostApduService* deklaruje dvě abstraktní metody, které musí třída potomka přepsat a implementovat [42]. Jedná se o metody:

- **processCommandApdu()**: Tato metoda je volána vždy, když NFC terminál pošle službě HCE APDU zprávu popsanou v části 1.4.1. Komunikace je střídavě obousměrná (poloduplexní): NFC terminál zašle APDU příkaz a čeká na odpověď aplikace v podobě APDU odpovědi. Vstupem do metody jsou přijatá data v podobě pole bajtů. Výstupem je pak odpověď taktéž v podobě pole bajtů.
- **onDeactivated()**: Metoda je volána ve dvou případech. V prvním případě AID v APDU nesedí s AID definovaném v meta-datech služby HCE. Druhým případem je situace, při které se komunikace mezi NFC terminálem a službou HCE přeruší.

Podpůrnou třídu služby HCE společně s implementovanými metodami zobrazuje zjednodušený výpis 3.10. Metoda *processCommandApdu()* obsahuje switch, který reaguje na přijaté APDU příkazy. Důležitou roli hraje taktéž instrukční třída `CLA`. Na základě této instrukční třídy služba HCE pracuje s přijatými daty. V rámci implementace aplikace RKVAC je zvolena proprietární instrukční třída. Pole `CLA` je tedy nastaveno na hodnotu `0x07`. Všechny další APDU příkazy mají toto pole nastaveno stejně.

Zpráva je nejdříve převedena do hexadecimální soustavy z důvodu jednodušší manipulace s daty. Následně je ze zprávy vyextrahován druhý bajt, který dle popisu v části 1.4.1 odpovídá poli instrukčního kódu. Na základě této hodnoty je pak zvolen odpovídající *case* switche. APDU příkaz je následně dle instrukčního kódu zpracován a aplikace APDU odpověď zasílá zpět k adresátovi. V případě, že aplikace daný instrukční kód nezná, vrací APDU odpověď s nastavenými poli `SW1=6D`, `SW2=00`. Tato hodnota odpovídá statusu `INS NOT SUPPORTED`, označující nepodporovaný instrukční kód. V případě jakéhokoliv jiného neúspěchu vrací aplikace APDU odpověď s nastavenými poli `SW1=6F`, `SW2=00`. Takto nastavená pole odpovídají statusu `STATUS FAILED`.

```
public class HCEService extends HostApduService {
    @Override
    public byte[] processCommandApdu(byte[] commandApdu, Bundle bundle) {
        String hexCommandApdu = Utils.byteArrayToHexString(commandApdu);
        switch (hexCommandApdu.substring(2, 4)) {
            case: Constants.INS_SET_USER_IDENTIFIER:
                  ...
            case: Constants.INS_GET_USER_IDENTIFIER:
                  ...
            ...
            default:
                return new ApduReponseObject(null, Constants.INS_NOT_SUPPORTED);
        }
    }
    @Override
    public void onDeactivated(int i) {
        ...
    }
}
```

Výpis 3.10: Třída HCEService dědící ze třídy HostApduService.

Aplikace uživatele reaguje na celou řadu instrukčních kódu. Tyto kódy vychází ze systému RKVAC popsaného v části 2.3. Následující části popisují jednotlivé instrukční kódy, které aplikace uživatele podporuje a jakým způsobem na ně reaguje, potažmo odpovídá.

Příkaz `APDU SCARD SELECT APPLICATION`

APDU příkaz, pomocí kterého čtecí jednotka v podobě NFC terminálu začíná komunikaci. Hodnota pole `CLA=0x00` nespecifikuje konkretní instrukční třídu. Pole `INS=A4` značí výběr appletu, v našem případě samotné aplikace uživatele. Parametry `P1` a `P2` jsou u tohoto příkazu nevyužity a nastaveny na `0x00`. Pole `LC=0x07` stanovuje délku AID přenášeného v poli `DATA`. AID přenášené v poli `DATA` musí odpovídat hodnotě AID nastaveném při definici služby HCE popsané v části 3.1.3. V případě úspěšného výběru, aplikace uživatele odpovídá APDU odpovědí s nastavenými poli `SW1=0x90`, `SW2=0x00`. Tato hodnota odpovídá statusu `SUCCESS`. Příklad APDU příkazu s tímto instrukčním kódem zobrazuje obrázek 3.4.

CLA	INS	P1	P2	LC	DATA
0x00	0xA4	0x00	0x00	0x07	*AID*

Obr. 3.4: APDU příkaz `APDU SCARD SELECT APPLICATION`.

Příkaz `INS SET USER IDENTIFIER`

APDU příkaz s instrukčním kódem `0x0B`. Jedná se o instrukci sloužící k procesu personalizace čipové karty, potažmo mobilní aplikace. APDU příkaz s takto nastaveným polem `INS` zasílá entita vydavatele. APDU příkaz nevyužívá parametry `P1` a `P2`, a proto jsou oba nastaveny na hodnotu `0x00`. Vydavatel zasílá uživateli unikátní identifikátor `ID`. Tento identifikátor má délku 8 bajtů. Z tohoto důvodu je i pole `LC` nastaveno na hodnotu `0x08`. Pole `DATA` pak nese samotný identifikátor. V případě úspěšné personalizace aplikace dojde k uložení identifikátoru do lokální databáze aplikace, vytvoření záznamu o provedeném úkonu a aplikace odpovídá APDU odpovědí s nastavenými poli `SW1=0x90`, `SW2=0x00`. Tato hodnota odpovídá statusu `SUCCESS`. Příklad APDU příkazu s tímto instrukčním kódem zobrazuje obrázek 3.5

CLA	INS	P1	P2	LC	DATA
0x07	0x0B	0x00	0x00	0x08	*ID*

Obr. 3.5: APDU příkaz `INS SET USER IDENTIFIER`.

Příkaz `INS GET USER IDENTIFIER`

APDU příkaz s instrukčním kódem `0x01`. Příkaz slouží k získání identifikátoru `ID`, který byl aplikaci přiřazen vydavatelem. Parametry `P1` a `P2` nejsou v tomto případě využity, a proto jsou nastaveny na hodnotu `0x00`. Aplikace nejdřív zkontroluje přítomnost identifikátoru v lokální databázi. V případě, že se zde nachází, zasílá odpověď signalizující úspěšný status společně s daty v podobě ID. APDU příkaz s takto nastaveným instrukčním kódem společně s odpovědi zobrazuje obrázek 3.6.

CLA	INS	P1	P2	LE
0x07	0x01	0x00	0x00	0x07

DATA	SW1	SW2
ID	0x90	0x00

Obr. 3.6: APDU příkaz `INS GET USER IDENTIFIER` a APDU odpověď.

Příkaz `INS SET REVOCATION AUTHORITY DATA`

APDU příkaz s polem `INS=0x02`. Příkaz složí k zaslání systémových parametrů revokační autority uživateli. Jedná se o revokační handler m_r, podpis revokačního handleru σ_{RA} a parametry $params_{RA}$. Význam jednotlivých parametrů popisuje část 2.3. Všechny hodnoty jsou řazeny postupně do bajtového pole. Pořadí jednotlivých parametrů je následující:

$$m_r, \sigma_{RA}, k, j, a_1 \dots a_j, h_1 \dots h_j, e_1 \dots e_k, \sigma_{e_1} \dots \sigma_{e_k}$$

V případě, že velikost aplikačních dat přesáhne 250 bajtů, je dané pole rozděleno do několika APDU zpráv a zasláno postupně. Pokud by byla velikost aplikačních dat například 1273 bajtů, pak by bylo třeba zaslat celkem 6 APDU zpráv. Parametr `P2` značí celkový počet zpráv, parametr `P1` označuje aktuálně posílanou zprávu tj. její sekvenční číslo. V případě, že je `P1<P2` očekává aplikace další data a vrací APDU odpověď s nastavenými poli `SW1=0x91`, `SW2=0xAF`. Tato hodnota odpovídá statusu `MORE DATA`. V případě úspěšného doručení všech zpráv odpovídá aplikace APDU odpovědí s nastavenými poli `SW1=0x90`, `SW2=0x00`. Tato hodnota odpovídá statusu `SUCCESS`. Aplikace si všechna přijatá data rozdělí a uloží do lokální databáze a vytvoří záznam o provedeném úkonu. Příklad sledu dvou APDU příkazů s tímto instrukčním kódem společně s odpověďmi zobrazuje obrázek 3.7.

CLA	INS	P1	P2	LC	DATA
0x07	0x02	0x01	0x02	0xFA	$m_r, \sigma_{RA}, k, j, a_1, a_2, h_1$

SW1	SW2
0x91	0xAF

CLA	INS	P1	P2	LC	DATA
0x07	0x02	0x02	0x02	0xF9	$h_2, e_1, e_2, \sigma_{e_1}, \sigma_{e_2}$

SW1	SW2
0x90	0x00

Obr. 3.7: APDU příkazy `INS SET REVOCATION AUTHORITY DATA` a APDU odpovědi.

Příkaz `INS GET USER IDENTIFIER ATTRIBUTES`

APDU příkaz s instrukčním kódem `0x04`. Příkaz slouží k získání `ID` uživatele, revokačního handleru m_r, podpisu revokačního handleru σ_{RA} a hodnoty n udávající počet atributů uložených v aplikaci. Význam jednotlivých parametrů popisuje část 2.3. Parametry `P1` a `P2` nejsou tímto příkazem využívány, avšak z důvodu původní implementace aplikace RKVAC jsou oba nastaveny na hodnotu `0x01`. Očekávaná délka odpovědi je v tomto případě konstantní, a to `LE=6A`. Všechny hodnoty jsou řazeny postupně do bajtového pole. Pořadí jednotlivých parametrů je následující:

$$ID, m_r, \sigma_{RA}, n$$

Aplikace nejdříve zkontroluje přítomnost požadovaných parametrů v lokální databázi. V případě úspěchu aplikace zasílá APDU odpověď signalizující úspěšný status společně s příslušnými daty. Příklad APDU příkazu s tímto instrukčním kódem společně s odpovědí zobrazuje obrázek 3.8.

CLA	INS	P1	P2	LE
0x07	0x04	0x01	0x01	0x6A

DATA	SW1	SW2
ID, m_r, σ_{RA}, n	0x90	0x00

Obr. 3.8: APDU příkaz `INS GET USER IDENTIFIER ATTRIBUTES` a APDU odpověď.

Příkaz `INS SET USER ATTRIBUTES`

APDU příkaz s polem `INS=0x03`. Příkaz slouží k zaslání atributů $m_1, \ldots, m_n$ aplikaci uživatele. Atributy zasílá na aplikaci vydavatel. První bajt aplikačních dat je rezervován pro parametr n udávající počet vydaných atributů. Parametr n je společně s atributy řazen do bajtového pole za sebe. Pořadí je následující:

$$n, m_1, \ldots, m_n$$

Maximální délka APDU zprávy je 250 bajtů. Pokud by byla aplikační data větší jak 250 bajtů, pak jsou data rozdělena do více APDU zpráv a aplikace pak vrací APDU odpověď s nastavenými poli `SW1=0x91`, `SW2=0xAF` značící status `MORE DATA`. Parametr `P2` určuje celkový počet zpráv, parametr `P1` aktuální číslo zprávy tj. sekvenční číslo. Aplikace nejdříve zkontroluje zda-li na kartě nějaké atributy již uloženy jsou. Jestliže na kartě žádné atributy uloženy nejsou, pak aplikace kontroluje zda-li není počet vydaných atributů vyšší jak 9. Pokud tomu tak není, tak přijaté data aplikace rozdělí a uloží do lokální databáze. Příklad APDU příkazu nesoucího 5 atributů s tímto instrukčním kódem společně s odpovědí zobrazuje obrázek 3.9.

CLA	INS	P1	P2	LC	DATA
0x07	0x03	0x01	0x01	0xA1	$n, m_1, m_2, m_3, m_4, m_5$

SW1	SW2
0x90	0x00

Obr. 3.9: APDU příkaz `INS SET USER ATTRIBUTES` a APDU odpověď.

Příkaz `INS SET ISSUER SIGNATURES`

APDU příkaz s polem `INS=0x05`. Příkaz slouží k zaslání kryptografického pověření σ a pomocných hodnot $\sigma_{x_r}, \sigma_{x_1}, \ldots, \sigma_{x_n}$. Význam jednotlivých parametrů popisuje část 2.3. Parametry zasílá na aplikaci vydavatel. Pořadí parametrů je následující:

$$\sigma, \sigma_{x_r}, \sigma_{x_1}, \ldots, \sigma_{x_n}$$

Pokud by byla aplikační data větší jak 250 bajtů, pak jsou data rozdělena do více APDU zpráv a aplikace pak vrací APDU odpověď s nastavenými poli `SW1=0x91`, `SW2=0xAF` značící status `MORE DATA`. Parametr `P2` určuje celkový počet zpráv, parametr `P1` aktuální číslo zprávy tj. sekvenční číslo. Aplikace přijatá data rozdělí a uloží do lokální databáze aplikace. Kromě toho taktéž vytvoří záznam o provedeném úkonu. Příklad APDU příkazu s tímto instrukčním kódem nesoucím kryptografické pověření a pomocné hodnoty pro 5 vydaných atributů společně s odpověďmi zobrazuje obrázek 3.10.

CLA	INS	P1	P2	LC	DATA
0x07	0x05	0x01	0x02	0xFA	$\sigma, \sigma_{x_r}, \sigma_{x_1}$

SW1	SW2
0x91	0xAF

CLA	INS	P1	P2	LC	DATA
0x07	0x05	0x02	0x02	0xCD	$\sigma_{x_2}, \sigma_{x_3}, \sigma_{x_4}, \sigma_{x_5}$

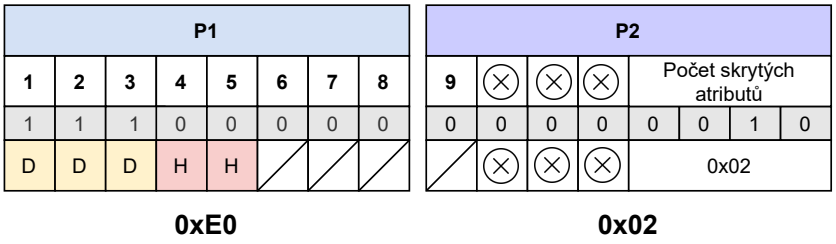
SW1	SW2
0x90	0x00

Obr. 3.10: APDU příkaz `INS SET ISSUER SIGNATURES` a APDU odpověď.

Příkaz `CMD TEST BIT CHECKER`

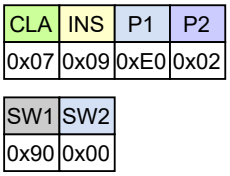
APDU příkaz s instrukčním kódem `0x09`. Pomocný příkaz sloužící k zaslání indície, která popisuje pozice odhalených a skrytých atributů. Příkaz zasílá ověřovatel uživateli. Na základě této informace bude aplikace vědět, které atributy má zahrnout do výpočtu důkazu znalosti a které ne.

Parametry `P1` a `P2` slouží k specifikaci dané pozice. Příklad stanovení hodnoty parametrů `P1` a `P2` při žádosti o odkrytí atributů na pozicích 1, 2 a 3 zobrazuje obrázek 3.11. Z obrázku vyplývá, že bylo vydáno celkem pět atributů. Ověřovatel žádá o odhalení atributů na pozici 1, 2 a 3. Atributy na pozici 4 a 5 zůstávají skryty. Obě pole mají velikost 1 bajt. V případě parametru `P1`, jednotlivé bity nastavené na hodnotu 1 znamenají žádost o odhalení. Písmeno D značí anglický výraz pro odhalení (Disclose). V opačném případě je bit nastaven na hodnotu 0. Písmeno H značí anglický výraz skrytí (Hidden). V případě parametru `P2` je první bit rezervován pro případný atribut na deváté pozici. Následující 3 bity nejsou v tomto příkazu využívány, a proto jsou nastaveny na hodnotu 0. Zbylé 4 bity slouží k definici celkového počtu skrytých atributů, kterých může být maximálně 9.



Obr. 3.11: Parametry `P1` a `P2` u APDU příkazu `CMD TEST BIT CHECKER`.

Při přijetí příkazu aplikace zkontroluje přítomnost požadovaných atributů. Jestliže jsou hodnoty atributů přítomny v lokální databázi aplikace, pak aplikace daným atributům nastaví příslušnou vlastnost, tedy odhalen či skryt. V případě úspěšného nastavení odpovídá aplikace APDU odpovědí s nastavenými poli `SW1=0x90`, `SW2=0x00`. Tato hodnota odpovídá statusu `SUCCESS`. Příklad APDU příkazu s tímto instrukčním kódem společně s odpovědí zobrazuje obrázek 3.12.

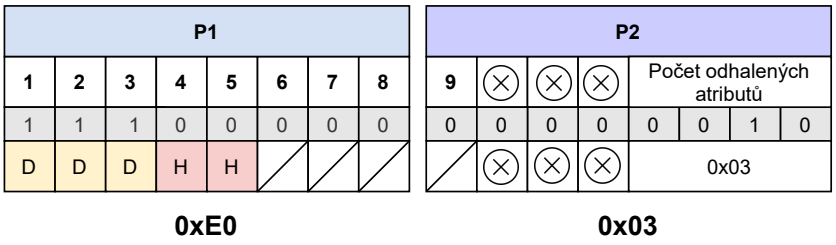


Obr. 3.12: APDU příkaz `CMD TEST BIT CHECKER` a APDU odpověď.

Příkaz `INS GET USER DISCLOSED ATTRIBUTES`

APDU příkaz s instrukčním kódem `0x0C`. Pomocný příkaz slouží k zaslání indície, která popisuje pozice odhalených a skrytých atributů. Příkaz taktéž slouží k získání parametru n udávajícího počet uložených atributů na kartě a hodnot odhalených atributů z aplikace $m_{z \in D}$. Příkaz zasílá ověřovatel uživateli. Na základě této informace bude aplikace vědět, které atributy má zahrnout do výpočtu důkazu znalosti a které ne. Ověřovatel zároveň bude schopen použít přijaté hodnoty atributů při lokálním ověření.

Parametry `P1` a `P2` slouží k specifikaci dané pozice. Příklad stanovení hodnoty parametrů `P1` a `P2` při žádosti o odkrytí atributů na pozicích 1, 2 a 3 zobrazuje obrázek 3.13. Z obrázku vyplývá, že bylo vydáno celkem pět atributů. Ověřovatel žádá o odhalení atributů na pozici 1, 2 a 3. Atributy na pozici 4 a 5 zůstávají skryty. Obě pole mají velikost 1 bajt. V případě parametru `P1`, jednotlivé bity nastavené na hodnotu 1 znamenají žádost o odhalení. Písmeno D značí anglický výraz pro odhalení (Disclose). V opačném případě je bit nastaven na hodnotu 0. Písmeno H značí anglický výraz skrytí (Hidden). V případě parametru `P2` je první bit rezervován pro případný atribut na deváté pozici. Následující 3 bity nejsou v tomto příkazu využívány, a proto jsou nastaveny na hodnotu 0. Zbylé 4 bity slouží k definici celkového počtu odhalených atributů, kterých může být maximálně 9.



Obr. 3.13: Parametry `P1` a `P2` u APDU příkazu `INS GET USER DISCLOSED ATTRIBUTES`.

Při přijetí příkazu aplikace zkontroluje přítomnost požadovaných atributů. Jestliže jsou hodnoty atributů přítomny v lokální databázi aplikace, pak aplikace daným atributům nastaví příslušnou vlastnost, tedy odhalen či skryt. V případě úspěšného nastavení odpovídá aplikace APDU odpovědí nesoucí aplikační data v podobě hodnoty parametru n, hodnot odhalených atributů $m_{z \in D}$ a nastavenými poli `SW1=0x90`, `SW2=0x00`. Tato hodnota odpovídá statusu `SUCCESS`. Příklad APDU příkazu s tímto instrukčním kódem společně s odpovědí nesoucí hodnoty třech odkrytých atributů zobrazuje obrázek 3.14.

CLA	INS	P1	P2	LE
0x07	0x0C	0xE0	0x03	0x61

DATA	SW1	SW2
n, m_1, m_2, m_3	0x90	0x00

Obr. 3.14: APDU příkaz `INS GET USER DISCLOSED ATTRIBUTES` a APDU odpověď.

Příkaz `INS COMPUTE PROOF OF KNOWLEDGE SEQ DISCLOSED`

APDU příkaz s instrukčním kódem `0x0A`. Příkaz slouží k zaslání hodnoty parametru *nonce* a *epoch*. Význam jednotlivých parametrů popisuje část 2.3. Parametry zasílá na aplikaci vydavatel. Pořadí parametrů je následující: *nonce*, *epoch*. Tomuto příkazu předchází buď APDU příkaz `CMD TEST BIT CHECKER`, nebo `INS GET USER DISCLOSED ATTRIBUTES`. Hodnoty parametrů `P1` a `P2` odpovídají hodnotám, které byly nastaveny v příkazu předešlém. Aplikace daný příkaz zpracuje a prostřednictvím kryptografického jádra, které obstarává podpůrná třída *CryptoCore*, realizuje dle obrázku 2.5 dílčí výpočty strany uživatele. Výsledky dílčích výpočtů jsou uloženy do lokální databáze aplikace. V případě úspěšného průběhu všech výpočtů zasílá aplikace odpověď s nastavenými poli `SW1=0x90`, `SW2=0x00`. Tato hodnota odpovídá statusu `SUCCESS`. Příklad APDU příkazu s tímto instrukčním kódem společně s odpovědí zobrazuje obrázek 3.15.

CLA	INS	P1	P2	LC	DATA
0x07	0x0A	0xE0	0x03	0x24	*nonce*, *epoch*

SW1	SW2
0x90	0x00

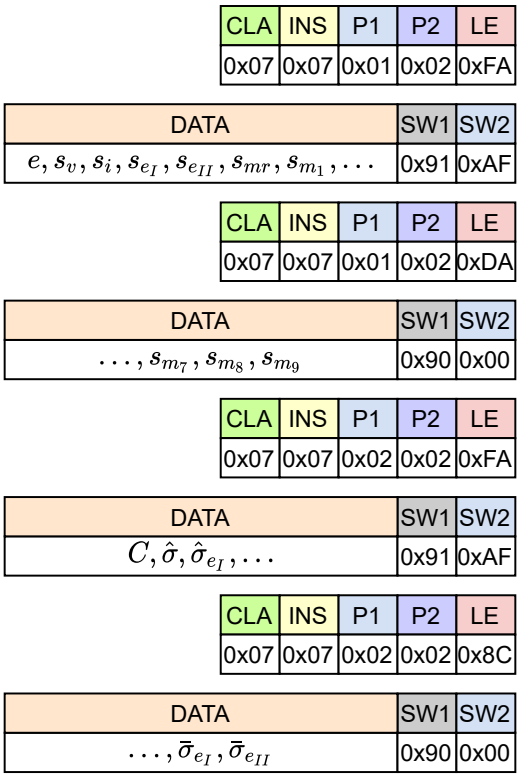
Obr. 3.15: APDU příkaz `INS COMPUTE PROOF OF KNOWLEDGE SEQ DISCLOSED` a APDU odpověď.

Příkaz `INS GET PROOF OF KNOWLEDGE`

APDU příkaz s instrukčním kódem `0x07`. Příkaz slouží k získání hodnot parametrů, které uživatel vypočítal prostřednictvím podpůrné třídy *CryptoCore*. Význam jednotlivých parametrů popisuje část 2.3. Dílčí výpočty uživatele byly iniciovány APDU příkazem `INS COMPUTE PROOF OF KNOWLEDGE SEQ DISCLOSED` a realizovány dle obrázku 2.5. Příkaz zasílá ověřovatel uživateli. Maximální délka APDU odpovědi je 250 bajtů. V případě, že jsou aplikační data větší jak 250 bajtu, jsou data rozdělena do více APDU zpráv. Hodnoty parametrů aplikace poskládá do bajtového pole za sebe dle definovaného pořadí. Pořadí vypočtených hodnot parametrů je:

$$e, s_v, s_i, s_{e_I}, s_{e_{II}}, s_{m_r}, \langle s_{m_{z \notin D}} \rangle, C, \hat{\sigma}, \hat{\sigma}_{e_I}, \hat{\sigma}_{e_{II}}, \bar{\sigma}_{e_I}, \bar{\sigma}_{e_{II}}$$

Aplikace nejdříve zasílá ověřovateli důkaz znalosti $\pi = e, s_v, s_i, s_{e_I}, s_{e_{II}}, s_{m_r}, \langle s_{m_{z \notin D}} \rangle$. Po odeslání π následuje zaslání parametrů $cred = C, \hat{\sigma}, \hat{\sigma}_{e_I}, \hat{\sigma}_{e_{II}}, \bar{\sigma}_{e_I}, \bar{\sigma}_{e_{II}}$. Pro rozlišení mezi daty spadajícími pod π a *cred* slouží parametr `P1` určující typ odpovědi. Jestliže je parametr `P1` nastaven na hodnotu `0x01`, pak se jedná o data důkazu znalosti. V případě, že je parametr `P1` nastaven na hodnotu `0x02`, pak se jedná o data parametru `cred`. Parametr `P2` určuje celkový počet požadovaných odpovědí daného typu. Pokud aplikace zasílá ověřovateli více dat, pak jsou v APDU odpovědi nastavena pole `SW1=0x91`, `SW2=0xAF` značící status `MORE DATA`. V okamžiku, kdy aplikace zašle všechna data, dojde k nastavení polí APDU odpovědi na `SW1=0x90`, `SW2=0x00`. Tato hodnota odpovídá statusu `SUCCESS`. Příklad APDU komunikace společně s odpověďmi zobrazuje obrázek 3.16.



Obr. 3.16: APDU komunikace příkazu `INS GET PROOF OF KNOWLEDGE` a APDU odpovědi.

3.1.6 Autentizace biometrikou

Předpokladem pro používání aplikace je zajištění základního zabezpečení, kterým může být zadání tajného hesla nebo pinu. Pokročilý způsob zabezpečení představuje autentizace biometrikou. Používané zařízení lokálně drží biometrická data, která do zařízení nahrál vlastník zařízení prostřednictvím odpovídajícího senzoru. Data zařízení neopouští a nejsou součástí procesu ověřování u pilotního systému RKVAC. Biometrikou je nejčastěji otisk prstu nebo obličej. Dražší zařízení poskytují obě možnosti a je jen na uživateli, kterou možnost si vybere.

Autentizace obličeje rozlišuje 2D (Two-dimensional) a 3D (Three-dimensional) systémy. Nevýhodou 2D systémů je, že je lze poměrně snadno oklamat například vytištěnou fotografií. V případě 3D systémů výše popsaný útok není možné použít, neboť se zde měří i hloubka. Metoda je obecně založena na měření vzdáleností mezi specifickými body. Měří se tak například vzdálenost mezi očima, nosem, ústy a jejich velikost, více viz [43].

Autentizace otiskem prstu představuje rozšířenější řešení. Stejně jako v předchozím případě existuje i zde několik různých technologií, které se používají pro snímání biometriky. Pro snímání otisků prstů lze využívat optické skenery, které zachycují

obraz otisku jen ve 2D podobě a tím pádem představují méně bezpečné řešení. Další řešení představují kapacitní skenery, které jsou schopné uchovat elektrický náboj ve shluku drobných kondenzátorů, které mohou po přiblížení prstu ke snímači velice přesně zjistit tvar papilárních linií. S tímto řešením se lze setkat nejčastěji. Poslední a nejpokročilejší technologii představují ultrazvukové skenery. Zařízení mají senzor integrován do displeje, jenž je vybaven ultrazvukovým vysílačem a přijímačem. Senzor rozpozná papilární linie podle paprsku ultrazvukového signálu odraženého zpět k senzoru. Jedná se o stále častěji používané řešení, více viz [44].

Definice nezbytných oprávnění a závislostí

Autentizace biometrikou pracuje s chráněnými rozhraními, popsanými v části 1.1.2. Pro definici oprávnění slouží kořenový soubor AndroidManifest.xml popsaný v části 1.1.1. Do kořenového souboru byl přidán element *uses-permission* dle výpisu 3.11.

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

Výpis 3.11: Element <uses-permission> v AndroidManifest.xml.

Kromě definice nezbytných oprávnění je třeba přidat tzv. závislost. Závislost představuje odkaz na používaný doplněk nebo knihovnu v aplikaci. K práci se závislostmi slouží Gradle. Gradle je nástroj pro automatizaci sestavování programu, kompilaci kódu a správu jednotlivých balíčků. Vývojové prostředí Android studio poskytuje vývojářům aplikace rozhraní v podobě konfiguračních souborů nástroje Gradle [45]. Konfigurační soubor *build* obsahuje jednotlivé závislosti aplikace. Pro zavedení knihovny poskytující objekty a funkce umožňující implementaci autentizace biometrikou byla přidána závislost dle výpisu 3.12

```
dependencies
{
  ...
  implementation 'androidx.biometric:biometric:1.0.1'
  ...
}
```

Výpis 3.12: Přidání závislosti do konfiguračního souboru nástroje Gradle.

Implementace autentizace pomocí otisku prstu

Po definici nezbytných oprávnění a zavedení potřebných závislostí lze implementovat autentizaci biometrikou. Klíčovým objektem je *BiometricManager* [46]. Pomocí tohoto objektu lze zjistit informaci o funkčnosti a dostupnosti biometrického

senzoru. Inicializaci výše zmíněného objektu včetně rozhraní realizovaného pomocí stavového přepínače zobrazuje výpis 3.13. Rozhraní stavového přepínače zajišťuje kontrolu funkčnosti a dostupnosti biometrického senzoru pomocí metody *canAuthenticate()*.

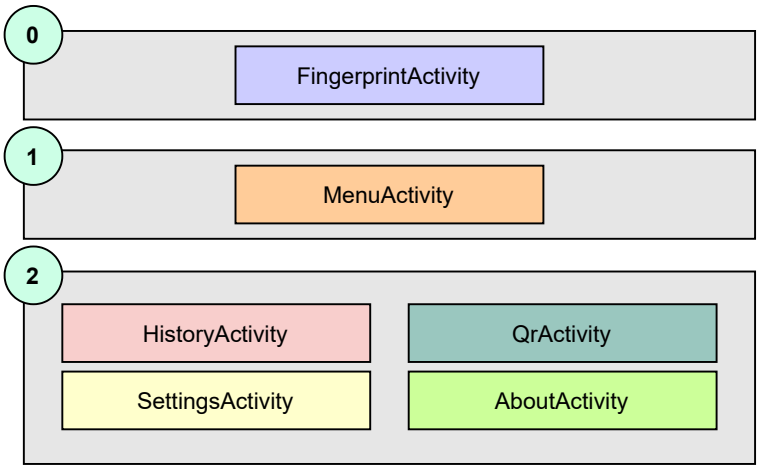
Jestliže dané zařízení disponuje funkčním a dostupným senzorem otisku prstu, pak je možné realizovat proces autentizace. Pro tyto účely slouží objekt *BiometricPrompt* [47], který vyzve uživatele k přiložení prstu na biometrický senzor. Objekt zároveň vyžaduje přepsání a implementaci tří klíčových metod, kterými jsou *onAuthenticationError()*, *onAuthenticationSucceeded()* a *onAuthenticationFailed()*, které jsou volány v případě úspěšné, nebo neúspěšné autentizace.

```
BiometricManager biometricManager = BiometricManager.from(this);
    switch (biometricManager.canAuthenticate())
    {
        case BiometricManager.BIOMETRIC_SUCCESS:
            break;
        case BiometricManager.BIOMETRIC_ERROR_NO_HARDWARE:
            break;
        case BiometricManager.BIOMETRIC_ERROR_HW_UNAVAILABLE:
            break;
        case BiometricManager.BIOMETRIC_ERROR_NONE_ENROLLED:
            break;
    }
```

Výpis 3.13: Objekt BiometricManager společně se stavovým přepínačem.

3.1.7 Grafické uživatelské rozhraní

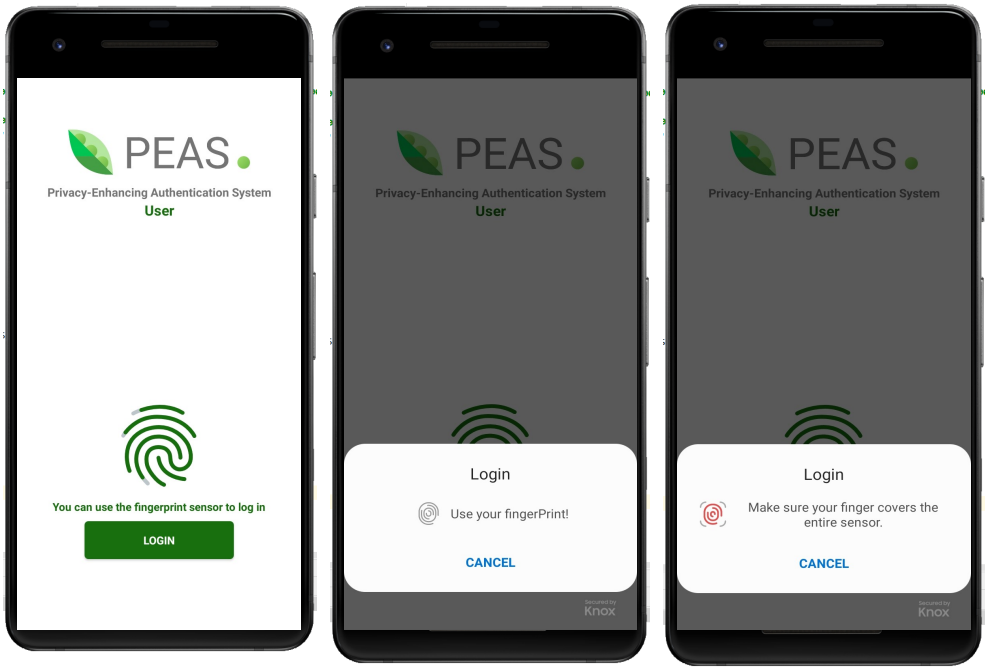
GUI (Graphic User Interface) umožňuje uživateli ovládat aplikaci prostřednictvím interaktivních grafických ovládacích prvků. Těmito prvky jsou tlačítka, přepínače, textová pole a mnohé další. Interaktivní grafické ovládací prvky se vkládají do tzv. rozložení. Rozložení definuje strukturu pro GUI. Jak zdroj [48] uvádí, Android rozlišuje tři základní typy rozložení, kterými jsou *LinearLayout*, *RelativeLayout* a *ConstraintLayout*. Každé z těchto rozložení disponuje jinými vlastnostmi, které je třeba při návrhu GUI zvážit. Prvním krokem při tvorbě návrhu GUI je samotné rozvržení aktivit a jejich vzájemné propojení. V této práci je použito rozložení aktivit, které vychází ze schématu třívrstvé architektury aplikace, kterou zobrazuje obrázek 3.17. Po vhodném rozvržení aktivit následuje samotný návrh dílčích aktivit aplikace. Při návrhu jsou dodržovány základní pravidla pro tvorbu GUI spočívající v praktičnosti, jednoduchosti, intuitivnosti a estetičnosti. Animace jednotlivých aktivit zprostředkovává knihovna Lottie [49].



Obr. 3.17: Třívrstvá architektura aktivit aplikace uživatele.

Aktivita nulté vrstvy

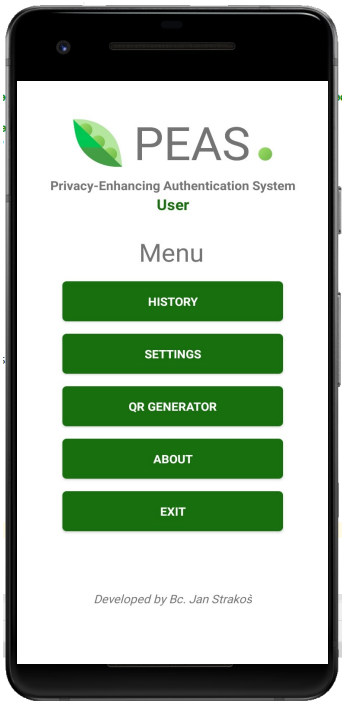
Nultou vrstvu tvoří úvodní aktivita zajišťující zabezpečení aplikace pomocí autentizace otiskem prstu. O skutečnosti, kdy senzor v zařízení chybí nebo je nedostupný, je uživatel aplikace informován pomocí informačního textového pole. V ostatních případech je vyzván k přiložení prstu na snímač otisků prstů, po kterém dochází k ověření identity. V případě úspěšné autentizace je uživatel aplikace přesměrován do první vrstvy. Aktivitu nulté vrstvy zobrazuje obrázek 3.18.



Obr. 3.18: Aplikace uživatele - aktivita nulté vrstvy.

Aktivita první vrstvy

Vrstva obsahuje aktivitu, která slouží jako uživatelský rozcestník. Obsahuje tlačítka, pomocí kterých uživatel přechází mezi jednotlivými aktivitami a zároveň umožňuje aplikaci opustit prostřednictvím tlačítka *Exit*. Po opuštění první vrstvy aktivit je uživatel znovu vyzván k autentizaci pomocí otisku prstu. Aktivit první vrstvy zobrazuje obrázek 3.19.



Obr. 3.19: Aplikace uživatele - aktivita první vrstvy.

Aktivity druhé vrstvy

Za pomocí aktivit druhé vrstvy uživatel komunikuje s podpůrnými třídami, ovlivňuje stav lokální databáze aplikace, poskytuje hodnoty odkrytých atributů mobilnímu ověřovateli prostřednictvím QR kódu nebo kontroluje historii záznamů aplikace. Aktivity druhé vrstvy jsou:

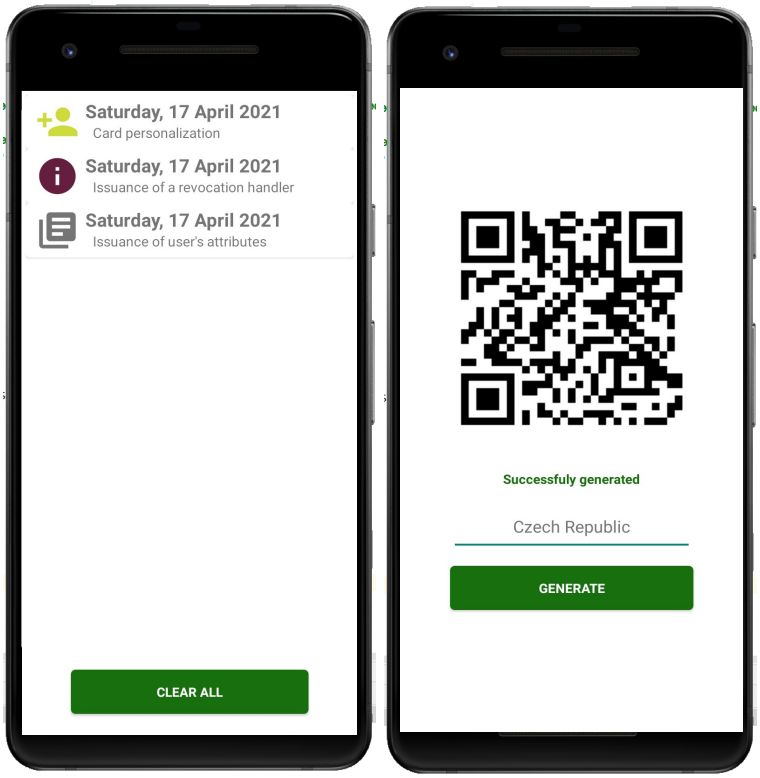
- ***HistoryActivity***: K zobrazení aktivity slouží tlačítko HISTORY. Aktivita poskytuje historii záznamů. Záznamy představují důležité události, ke kterým během běhu aplikace došlo. Danou událost aktivita uživateli zobrazí pomocí rozložení *RecyclerView* [50].

Aktivita ke svému provozu využívá podpůrné třídy *HistoryAdapter* a *HistoryItem*. Aplikace uživatele rozlišuje mezi několika událostmi, kterými jsou:

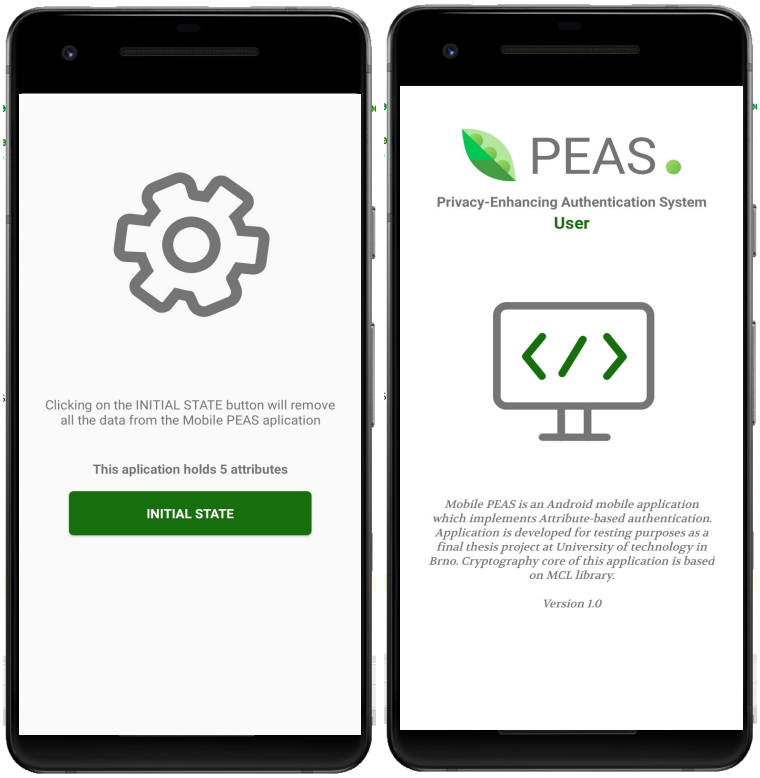
  - **Personalizace aplikace**: Událost vyvolaná úspěšným zpracováním a odpovědí na příkaz `INS SET USER IDENTIFIER`.
  - **Vydání revokačního handleru**: Událost vyvolaná úspěšným zpracováním a odpovědí na příkaz `INS SET REVOCATION AUTHORITY DATA`
  - **Vydání atributů**: Událost vyvolaná přijetím vydaných atributů od entity vydavatele. Vyvolána je úspěšným zpracováním a odpovědí na příkaz `INS SET ISSUER SIGNATURES`.
  - **Výpočet důkazu znalosti**: Událost vyvolaný úspěšným zpracováním a odpovědí na `INS COMPUTE PROOF OF KNOWLEDGE SEQ DISCLOSED`
  - **Error**: Událost vyvolaná jakýmkoliv jiným neúspěchem během APDU komunikace. Název příkazu, při jehož zpracování vznikl problém je taktéž propsán do záznamu.

- ***QrActivity***: K zobrazení aktivity slouží tlačítko QR GENERATOR. Aktivita zajišťující generování QR kódu pro mobilního ověřovatele. Pomocí tohoto kódu si mobilní ověřovatel nastaví hodnotu příslušného odkrytého atributu, kterou následně použije při ověření důkazu znalosti. Hodnota atributu je nejdříve zhashována pomocí hashovací funkce SHA-256 [41] a následně převedena do podoby QR kódu za pomocí knihovny ZXing [51].
- ***SettingsActivity***: K zobrazení aktivity slouží tlačítko SETTINGS. Aktivita pomocí které může uživatel uvést aplikaci do iniciálního stavu. Tato možnost tu je pro případ, že by se aplikace dostala do stavu, který by znemožňoval její správnou funkčnost. Aktivita zároveň informuje uživatele o počtu atributů, které aplikace drží v lokální databázi.
- ***AboutActivity***: K zobrazení aktivity slouží tlačítko ABOUT. Aktivita poskytuje uživateli základní informace o aplikaci, kterými jsou verze a účel aplikace.

Aktivity *HistoryActivity* a *QrActivity* zobrazuje obrázek 3.20. Aktivity *SettingsActivity* a *AboutActivity* pak obrázek 3.21.



Obr. 3.20: Aktivity HistoryActivity (vlevo) a QrActivity (vpravo).



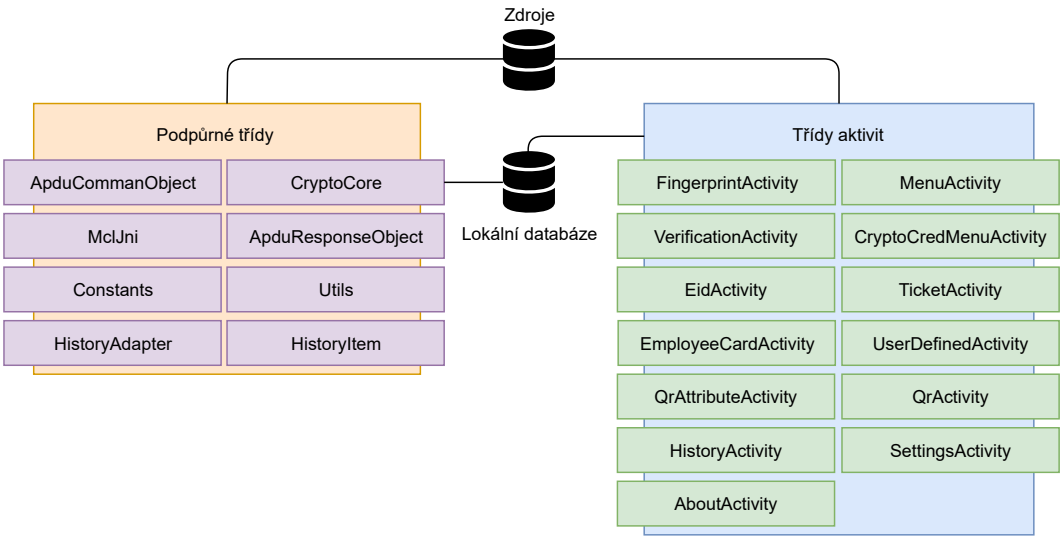
Obr. 3.21: Aktivity SettingsActivity (vlevo) a AboutActivity (vpravo).

3.2 Aplikace mobilního ověřovatele

Aplikace mobilního ověřovatele je oproti aplikaci uživatele popsané v části 3.1 více interaktivní. Aplikace umožňuje mobilnímu ověřovateli základní nastavení parametrů systému, na základě kterých ověřuje vlastnictví držených atributů. Mobilní ověřovatel zajišťuje základní funkce ověřovatele aplikace RKVAC, popsaného v části 2.3, s tím rozdílem, že může fungovat i samostatně. Výhodou samostatnosti je možnost provádět proces ověřovaní ihned a takřka kdekoliv, například ve stížených podmínkách. Mobilní ověřovatel komunikuje s ostatními entitami systému dle schématu 3.1. Komunikaci pomocí APDU zpráv, popsaných v části 1.4.1, zajišťuje aplikaci emulátor NFC terminálu. Díky tomuto emulátoru se aplikace chová jako čtecí jednotka, která umí komunikovat jak s aplikací uživatele, tak čipovými kartami. Výměnu informací v podobě QR kódů umožňuje skener QR kódů realizovaný knihovnou Budiyev [52]. Mobilní ověřovatel podporuje proces ověřování pro atributové pověření typu elektronická identita, tiket, zaměstnanecká karta nebo pověření definované uživatelem. Události jako povolení či zamezení přístupu aplikace zaznamenává do lokálních záznamů. Následující podkapitoly popisují průběh implementace aplikace mobilního ověřovatele.

3.2.1 Architektura aplikace

Architektura aplikace vychází se schématu 3.22. Stejně jako tomu bylo u aplikace uživatele, je i tato aplikace tvořena z dílčích, vzájemně propojených bloků. Jednotlivé bloky svým účelem odpovídají blokům popsaným v části 3.1.1. Rozdílem jsou pouze jiné hodnoty, třídy či zdrojová data, které aplikace mobilního ověřovatele používá.



Obr. 3.22: Architektura aplikace mobilního ověřovatele.

3.2.2 Zavedení emulátoru NFC terminálu

Zavedení emulátoru NFC terminálu sestává z jediného kroku. Výsledkem zavedení je schopnost aplikace, chovat se jako čtecí jednotka. Taková aplikace je pak schopna zasílat APDU příkazy popsané v části 1.4.1. Pro zavedení emulátoru NFC terminálu je třeba definovat nutné požadavky a oprávnění. Pak je již možné samotný NFC emulátor implementovat.

Definice nutných požadavků a oprávnění

Emulátor NFC terminálu vyžaduje od systému Android specifické hardwarové a softwarové funkce. Kromě těchto požadovaných funkcí taktéž pracuje s chráněnými rozhraními, popsanými v části 1.1.2. Pro definici požadavků a oprávnění slouží kořenový soubor AndroidManifest.xml popsaný v části 1.1.1.

Požadované softwarové a hardwarové funkce, které daná aplikace využívá, se deklarují pomocí elementu *<uses-feature>*. Samotný element disponuje několika atributy. Použití elementu *<uses-feature>* v kořenovém souboru AndroidManifest.xml zobrazuje výpis 3.14.

```
<uses-feature
 android:name="android.hardware.nfc.hce"
 android:required="true">
</uses-feature>
```

Výpis 3.14: Element *<uses-feature>* v AndroidManifest.xml.

Dalším důležitým krokem je definice systémových oprávnění. Tato oprávnění uživatel uděluje aplikaci při její instalaci. Požadovaná oprávnění, které daná aplikace vyžaduje, se deklarují pomocí elementu *<uses-permission>*. Element se taktéž přidává do kořenového souboru AndroidManifest.xml. Použití elementu *<uses-permission>* v kořenovém souboru AndroidManifest.xml zobrazuje výpis 3.15.

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

Výpis 3.15: Element *<uses-permission>* v AndroidManifest.xml.

3.2.3 Implementace kryptografického jádra

Kryptografické jádro realizuje podpůrná třída *CryptoCore*, která komunikuje s knihovnou MCL prostřednictvím nativního rozhraní. Blokové schéma komunikace kryptografického jádra s nativní knihovnou MCL zobrazuje obrázek 3.3. Komunikaci mezi jádrem a knihovnou MCL zprostředkovává podpůrná třída *MclJni*. Stejně jako tomu

bylo v případě aplikace uživatele, i zde je třeba naimportovat balíček knihovny MCL, knihovnu načíst pomocí statického inicializačního bloku prostřednictvím metody *LoadLibrary()* a provolat v konstruktoru třídy *CryptoCore* metodu *SystemInit()*, pomocí které se nastaví Barreto-Naehring křivka. *Context* v konstruktoru umožňuje využívat kryptografické jádro napříč aplikací. Celý proces načtení kryptografického jádra demonstruje výpis 3.9.

Kryptografické jádro mobilního ověřovatele obstarává dílčí výpočty strany ověřovatele schématu 2.5. Při implementaci jádra je využita Barreto-Naehring křivka BN-254 [53] knihovny MCL. Pro účely této diplomové práce je uvedeno použití kryptografického jádra pro výpočet parametru t_{revoke}.

Postup výpočtu parametru t_{revoke}

Metoda podpůrné třídy *CryptoCore* s názvem *computeT_revoke()* zajišťuje výpočet hodnoty parametru t_{revoke}. Výpočet je realizován dle následující rovnice:

$$t_{revoke} = (g_1 C^{-H(epoch)})^{-e} C^{s_{m_r}} C^{s_i}$$

Význam jednotlivých parametrů uvádí část 2.3. Výstupem metody je bod na eliptické křivce. V první kroku dojde k inicializaci objektu *SharedPreferences*. Objekt zprostředkovává komunikaci s lokální databází. Inicializace objektu je realizována následovně:

```
SharedPreferences sP = context.getSharedPreferences("VerifierData", 0);
```

V dalším kroku dochází k výpočtu hashe hodnoty parametru *epoch*. Pro výpočet je použita hashovací funkce SHA-1. Výstupem této funkce jsou data o velikosti 20 bajtů. Z důvodu kompatibility s knihovnou MCL a následného mapování na objekty třídy *Fr*, dochází k přidání paddingu (doplnění) nul. Hodnota je modulárně zredukována řádem q a namapována na proměnnou *tmp1*. Číslo 16 značí hexadecimální soustavu. Celý postup je proveden tímto způsobem:

```
String epoch_hash = new BigInteger(SHA1_PADDING +
                  SHA1(sP.getString("epoch","00"), 16)
                  .mod(BN256_q).toString(16);
Fr tmp1 = new Fr(epoch_hash, 16);
```

Dalším krokem je výpočet inverzního prvku k hodnotě hashe. Tuto hodnotu jádro počítá za pomocí metody *neg()* knihovny MCL. Výsledná hodnota se uloží do proměnné *tmp2*. Postup je realizován následovně:

```
Fr tmp2 = new Fr();
MclJni.neg(tmp2, tmp1);
```

Nyní je třeba vynásobit obdržený pseudonym C hodnotou inverzního prvku. Pseudonym je třeba načíst z lokální databáze a namapovat na objekt třídy *G1*. Následně je možné využít metodu *mul()* knihovny MCL a výsledek uložit do proměnné *tmp3*. K danému výsledku se došlo následujícím způsobem:

```
G1 tmp3 = new G1();
G1 C = new G1();
C.setStr(databaseCurvePointToMcl("C", sP), 16);
MclJni.mul(tmp3, C, tmp2);
```

V dalším kroku je možné sečíst body na křivce, kterými jsou generátor g_1 a *tmp3*. Generátor je třeba nejdříve inicializovat. Následně je možné použít metodu *add()* knihovny MCL a výsledek uložit do proměnné *tmp4*. Číslo 16 značí hexadecimální soustavu. Postup je následovný:

```
G1 tmp4 = new G1();
G1 g1 = new G1();
g1.setStr(BN256_g1_hex, 16);
MclJni.add(tmp4, g1, tmp3);
```

Další částí výpočtu je vynásobení inverzním prvkem e součet bodů z předchozího výpočtu. Hodnota parametru je načtena z lokální databáze a namapována na objekt třídy *Fr*. Inverzní prvek parametru je pak uložen do proměnné *tmp5*. Tímto číslem je pak vynásoben součet skrytý pod proměnnou *tmp4* a výsledek uložen do proměnné *tmp6*. Číslo 16 značí hexadecimální soustavu. Postup výpočtu je proveden tímto způsobem:

```
Fr tmp5 = new Fr();
Fr e = new Fr(sP.getString("e", "00"), 16);
MclJni.neg(tmp5, e);
G1 tmp6 = new G1();
MclJni.mul(tmp6, tmp4, tmp5);
```

V dalším kroku je třeba vynásobit pseudonym C hodnotou parametru s_{m_r}, to stejné pak zopakovat pro hodnotu parametru s_i. Hodnoty zmiňovaných parametrů jsou načteny z lokální databáze a namapovány na objekty třídy *Fr*. Výsledky dílčích výpočtu jsou pak uloženy do proměnných *tmp7* a *tmp8*. V případě mapování hodnoty s_i je třeba počítat s nadbytečným prvním bajtem. Tento bajt se zde nachází kvůli kompatibilitě s čipovou kartou. Číslo 16 značí hexadecimální soustavu. Realizace kroku je provedena následovně (další strana).

```
Fr s_m_r = new Fr(sP.getString("s_m_r", "00"), 16);
G1 tmp7 = new G1();
MclJni.mul(tmp7, C, s_m_r);
G1 tmp8 = new G1();
Fr s_i = new Fr(sP.getString("s_i","00").substring(2, 66), 16);
MclJni.mul(tmp8, C, s_i);
```

Posledním krokem výpočtu je součet výsledků dílčích mezivýpočtů. Výsledkem je pak hodnota parametru t_{revoke} představující bod na eliptické křivce. Poslední krok je proveden tímto způsobem:

```
G1 tmp9 = new G1();
MclJni.add(tmp9, tmp6, tmp7;
G1 t_revoke = new G1();
MclJni.add(t_revoke, tmp9, tmp8);
return t_revoke;
```

3.2.4 Implementace emulátoru NFC terminálu

Po definici nezbytných požadavků a oprávnění zajišťující funkčnost emulátoru NFC terminálu, lze implementovat emulátor jako takový. Pro tyto účely poskytuje platforma Android podporu v podobě tříd *NfcAdapter*, *Tag* a *IsoDep*. Každý z objektů těchto tříd přímo ovlivňuje komunikaci pomocí APDU zpráv popsaných v části 1.4.1. Třída, pod kterou jednotlivé objekty vystupují je třída aktivit *VerificationActivity*. V rámci této třídy jsou objekty inicializovaný a využívány.

Pro správnou funkčnost musí třída *VerificationActivity* implementovat rozhraní *NfcAdapter.ReaderCallback*. Rodičovská třída *NfcAdapter* deklaruje rozhraní *ReaderCallback* obsahující metodu *onTagDiscovered()*, kterou musí třída *VerificationActivity* přepsat a implementovat. Pro správnou funkčnost emulátoru NFC terminálu a běh samotné aplikace je tedy třeba přepsat a implementovat tyto metody:

- ***onTagDiscovered()***: Metoda, která je volána vždy, když aplikace detekuje přítomnost čipové karty, NFC tagu nebo aplikaci emulující hostitelskou kartu. Jedná se taktéž o klíčovou metdu třídy *VerificationActivity*. Daná metoda obsahuje logiku a pravidla, na základě kterých aplikace mobilního ověřovatele komunikuje se svým okolím.
- ***onResume()***: Metoda vyvolaná znovuspuštěním aktivity *VerificationActivity*. Každá aktivita má totiž životní cyklus představující posloupnost stavů, ve kterých se aktivita nachází. Jednotlivé stavy iniciují metody jako *onCreate()*, *onStart()*, *onDestroy()* a jiné. Metoda *onResume()* je volána v situaci, kdy byla daná aktivita opuštěna a znovu navštívena. Implementací této metody

je docíleno toho, že v případě kdy mobilní ověřovatel znovu navštíví aktivitu pro ověřování, automaticky dojde k inicializaci objektů zajišťujících APDU komunikaci a spuštění čtecího módu.

- ***onPause()***: Metoda vyvolaná opuštěním aktivity *VerificationActivity*. Metoda vypne čtecí mód, ve kterém se aplikace při svém běhu nacházela.

Metoda *onTagDiscovered()* obsahuje zřetězenou sekvenci if-else bloků, na základě kterých aplikace mobilního ověřovatele generuje APDU příkazy. Aplikace mobilního ověřovatele podporuje následující APDU příkazy.

- `APDU SCARD SELECT APLICATION`
- `CMD TEST BIT CHECKER`
- `INS GET USER DISCLOSED ATTRIBUTES`
- `INS COMPUTE PROOF OF KNOWLEDGE SEQ DISCLOSED`
- `INS GET PROOF OF KNOWLEDGE`

Jednotlivé APDU příkazy detailně popisuje část 3.1.5. Hodnoty polí APDU příkazů přímo ovlivňuje mobilní ověřovatel pomocí grafického uživatelského rozhraní. Každý z podporovaných APDU příkazů má odpovídající metodu v třídě *VerificationActivity*. Příklad implementace metody *onTagDiscovered()* uvádí výpis 3.16. Třída využívá podpůrných tříd *ApduResponseObject()* a *ApduCommandObject()*, které slouží ke generování APDU příkazů nebo k mapování APDU odpovědí na objekty. Objekt třídy *IsoDep* prostřednictvím metody *transceive()* zajišťuje APDU komunikaci dle normy ISO 14443-4 [54].

```
@Override
public void onTagDiscovered(Tag tag) {
 IsoDep idoDep = IsoDep.get(tag);
 try {
 isoDep.connect();
 ApduResponseObject response_1 = APDU_SCARD_SELECT_APLICATION();
 if (response_1.getS1_S2().equals(Constants.SUCCESS)) {
    ApduResponseObject response_2 = CMD_TEST_BIT_CHECKER(isoDep, apduCommand);
    if (response_2.getS1_S2().equals(Constants.SUCCESS))
        CryptoCore cryptoCore = new CryptoCore(this);
        ApduResponseObject response_3 =
        INS_COMPUTE_PROOF_OF_KNOWLEDGE_SEQ_DISCLOSED(
            isoDep, apduCommand, cryptoCore);
            ... }
    else {
         showProblemMessage(Constants.INS_CMD_BIT_CHECKER_SIGN); }
  else {
        showProblemMessage(Constants.SELECT_AID_SIGN); }}}
```

Výpis 3.16: Příklad implementace metody onTagDiscovered().

Pro účely této diplomové práce je uveden příklad implementace APDU příkazu `INS COMPUTE PROOF OF KNOWLEDGE SEQ DISCLOSED`. Pomocí tohoto APDU příkazu mobilní ověřovatel žádá uživatele o výpočet důkazu znalosti.

Implementace příkazu `INS COMPUTE PROOF OF KNOWLEDGE SEQ DISCLOSED`

Implementaci příkazu `INS COMPUTE PROOF OF KNOWLEDGE SEQ DISCLOSED` zajišťuje stejně pojmenovaná metoda třídy *VerificationActivity*. Při tvorbě APDU příkazu dochází ke komunikaci s lokální databází aplikace a kryptografickým jádrem. Jednotlivá pole APDU příkazu lze nastavit pomocí příslušných setterů. Celý proces demonstruje výpis 3.17.

Metoda vrací odpověď na zaslaný APDU příkaz. Příkaz je zaslán objektem třídy *IsoDep* prostřednictvím metody *transceive()*. V prvním kroku je inicializován objekt třídy *SharedPreferences* pro komunikaci s lokální databází. Kromě toho je i inicializován editor, pomocí kterého je možné editovat data lokální databáze. V dalším kroku je načtena hodnota parametru *epoch* a vygenerována hodnota parametru *nonce*, která je pak následně nahrána do lokální databáze. Posledním krokem je nastavení příslušných polí APDU příkazu.

```
private void INS_COMPUTE_PROOF_OF_KNOWLEDGE_SEQ_DISCLOSED (
            IsoDep isoDep,
            ApduCommandObject apduCommand,
            CryptoCore cryptoCore) {
  SharedPreferences sP = getApplicationContext().
                            getSharedPreferences("VerifierData", 0);
  SharedPreferences.Editor editor = sP.edit();
  String EPOCH = sP.getString("epoch", "00000000");
  String NONCE = cryptoCore.generateNonce().toUpperCase();
  editor.putString("nonce", NONCE);
  editor.commit();
  apduCommand.setINS("INS_COMPUTE_PROOF_OF_KNOWLEDGE_SEQ_DISCLOSED");
  apduCommand.setLC(Utils.dataLengthCounter(NONCE + EPOCH));
  apduCommand.setDATA(NONCE + EPOCH);
  apduCommand.setLE(EMPTY);
  return new ApduResponseObject(
              Utils.byteArrayToHexString(isoDep.transceive(
                    Utils.hexStringToByteArray(apduCommand.toString())
                )
               )
          );
}
```

Výpis 3.17: APDU příkaz `INS COMPUTE PROOF OF KNOWLEDGE SEQ DISCLOSED`.

3.2.5 Zavedení QR skeneru

Mobilní ověřovatel komunikuje s entitou vydavatele a revokační autority prostřednictvím technologie QR kódu. Pomocí této technologie si také mobilní ověřovatel načítá hodnoty odkrytých atributů od uživatele. Pro schopnost aplikace číst QR kódy je potřeba definovat nutné požadavky a oprávnění a následně zavést vhodnou knihovnu, která skener implementuje.

Definice nutných požadavků a oprávnění

QR skener pracuje s chráněnými rozhraními, popsanými v části 1.1.2, která přistupují k hardwaru zařízení v podobě kamery. Pro přístup je potřeba definovat nutná oprávnění. Tato oprávnění uživatel uděluje aplikaci při její instalaci. Požadovaná oprávnění, které daná aplikace vyžaduje, se deklarují pomocí elementu *<uses-permission>*. Element se taktéž přidává do kořenového souboru AndroidManifest.xml. Použití elementu *<uses-permission>* v kořenovém souboru AndroidManifest.xml zobrazuje následující výpis:

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

Výpis 3.18: Element *<uses-permission>* v AndroidManifest.xml.

Zavedení knihovny Budiyev

Kromě definice nezbytných oprávnění je třeba přidat tzv. závislost. Závislost představuje odkaz na používaný doplněk nebo knihovnu v aplikaci. K práci se závislostmi slouží Gradle. Zdroje obsahují konfigurační soubor *build*, který definuje jednotlivé závislosti aplikace. Knihovna Budiyev [52] představuje volně dostupnou open-source knihovnu napsanou pomocí programovacího jazyka Java. Pro zavedení knihovny poskytující objekty a funkce umožňující implementaci QR skeneru je přidána závislost dle výpisu 3.19

```
dependencies
{
  ...
  implementation 'com.budiyev.android:code-scanner:2.1.0'
  ...
}
```

Výpis 3.19: Přidání závislosti do konfiguračního souboru nástroje Gradle.

3.2.6 Implementace QR skeneru

Aktuální implementace aplikace RKVAC neumožňuje distribuci dat v podobě APDU zpráv mobilnímu ověřovateli. Tento nedostatek řeší QR kódy, pomocí kterých se potřebná data rychle a efektivně načtou do aplikace mobilního ověřovatele. Při procesu ověřování, které popisuje schéma 2.5, je nezbytnou součástí výpočtu na straně ověřovatele znalost určitých systémových parametrů. Tyto parametry si musí mobilní ověřovatel nejdřív načíst pomocí QR kódu od ostatních entit systému. Až pak je mobilní ověřovatel schopen samostatně ověřovat vlastnictví atributů. Pro ověřování vlastnictví atributů mobilní ověřovatel potřebuje tyto systémové parametry:

$$epoch, (x_0, x_1, \ldots, x_n), pk_{RA}, h_1, h_2$$

Význam jednotlivých parametrů uvádí část 2.3. Mobilní ověřovatel očekává hodnoty parametrů, které jsou poskládány za sebe v pořadí odpovídající výše uvedenému. Data mají konstantní velikost 550 bajtů. Všechna data lze převést do podoby jednoho QR kódu. Takovýto kód pak mobilní ověřovatel načte a hodnoty uloží do lokální databáze. K načtení aplikace využívá knihovnu Budiyev. Během prvotního spuštění aktivity zajišťující skenování QR kódu je uživatel aplikace dotázán k povolení oprávnění používat kameru. Tento dotaz během běhu aplikace zajišťuje open-source knihovna Dexter [55]. Knihovna Budiyev poskytuje celou řadu tříd, pomocí kterých lze vytvářet objekty, prostřednictvím kterých lze načíst požadovaná data v podobě QR kódu. Nejdůležitější je třída *CodeScanner*. Jednou z poskytovaných metod dané třídy je metoda *setDecodeCallback()*, představující callback funkci[1]. Tato metoda vyžadje přepsání a implementaci metody *onDecode()*. Metoda *onDecode()* je volána v případě, že byl předložený QR kód úspěšně dekódován. S daty je pak možné manipulovat dle záměru vývojáře aplikace. Příklad uvádí výpis 3.20.

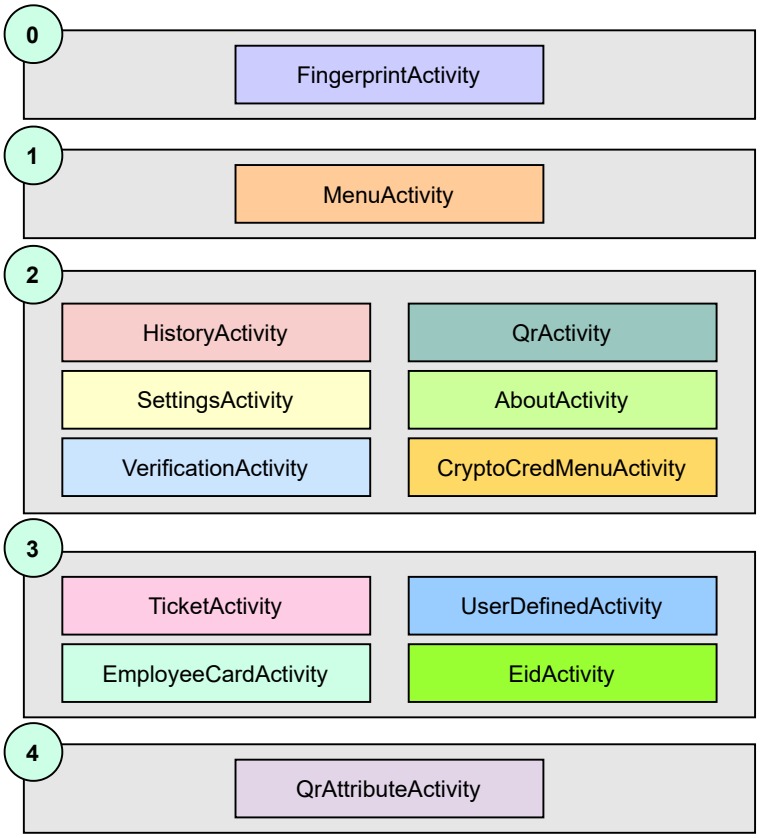
```
...
CodeScanner codeScanner = new CodeScanner(this, scannerView);
codeScanner.setDecodeCallback(new DecodeCallback() {
    @Override
    public void onDecode(Result result) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                String receivedData = result.getText();
                ...
                }}}}
```

Výpis 3.20: Příklad implementace QR skeneru.

[1]Metoda předaná jako argument jiné metodě

3.2.7 Grafické uživatelské rozhraní

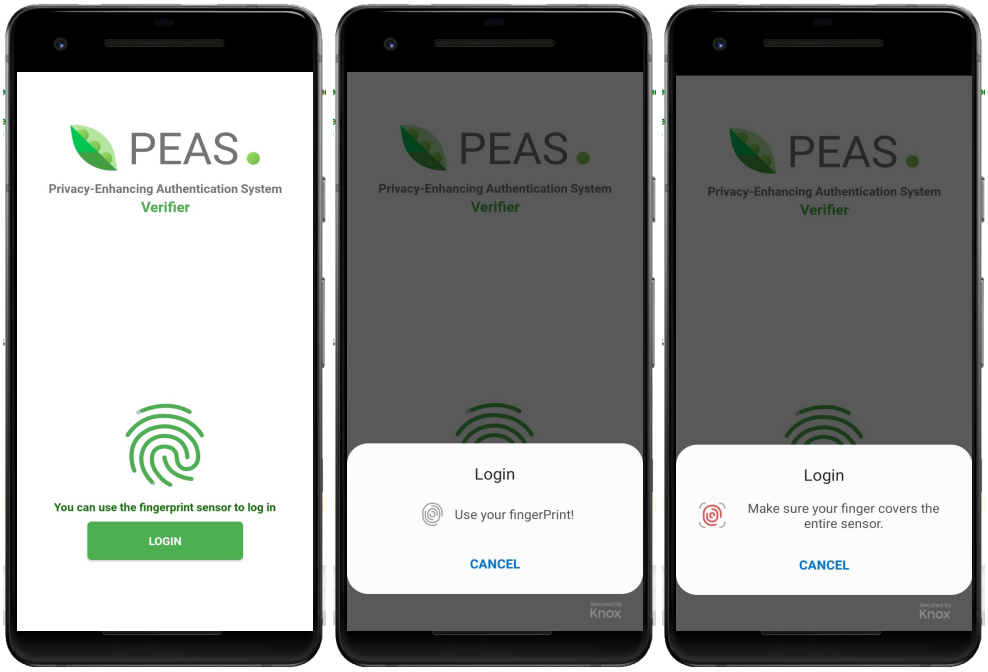
Aplikace mobilního ověřovatele je po stránce grafického uživatelského rozhraní, oproti aplikaci uživatele, komplexnější. Větší množství aktivit zvyšuje počet stavů do kterých se samotná aplikace může dostat. I při vyšším počtu aktivit je třeba dbát na základní pravidla pro tvorbu GUI, kterými jsou praktičnost, jednoduchost, intuitivnost a estetičnost. V rámci aplikace jsou taktéž využívány animace zprostředkované knihovnou Lottie. Prvním krokem při tvorbě návrhu GUI je samotné rozvržení aktivit a jejich vzájemné propojení. V této práci je použito rozložení aktivit, které vychází ze schématu pětivrstvé architektury aplikace, kterou zobrazuje obrázek 3.23. Aktivity jsou rozděleny na aktivity nulté, první, druhé, třetí a čtvrté vrstvy. Následující podkapitoly rozebírají účel a rozvržení jednotlivý aktivit.



Obr. 3.23: Pětivrstvá architektura aplikace mobilního ověřovatele.

Aktivita nulté vrstvy

Stejně jako tomu bylo v případě aplikace uživatele, i zde tvoří úvodní aktivitu aktivita pro autentizaci otiskem prstu. V případě úspěšné autentizace je uživatel aplikace přesměrován do první vrstvy. Aktivitu nulté vrstvy zobrazuje obrázek 3.24.



Obr. 3.24: Aplikace mobilního ověřovatele - aktivita FingerprintActivity.

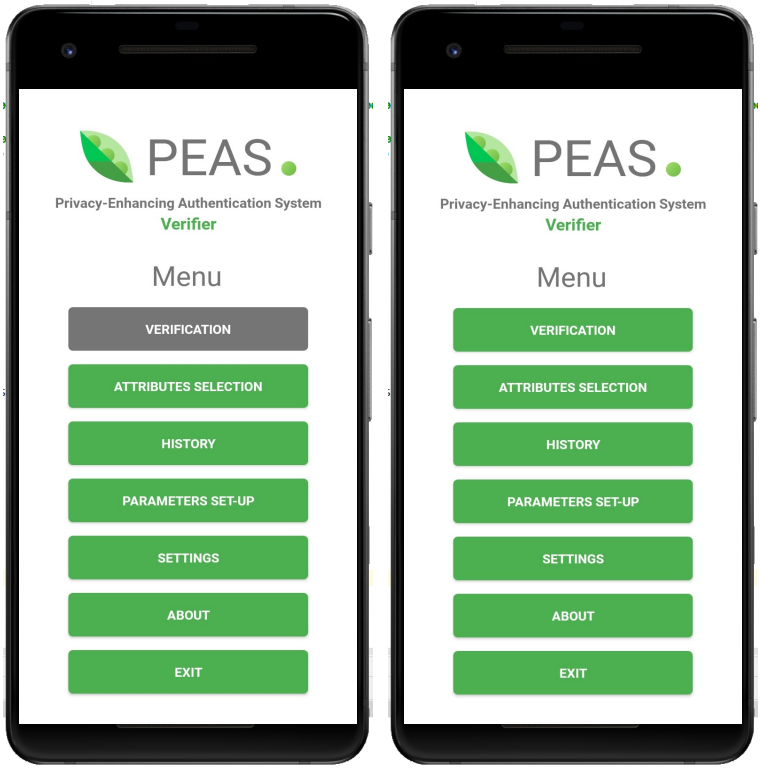
Aktivita první vrstvy

První vrstvu tvoří aktivita Menu, která slouží jako uživatelský rozcestník. Aktivita má v iniciálním stavu neaktivní tlačítko pro přechod do aktivity zajišťující ověřování. Tlačítko se stane stlačitelným po úspěšném nastavení systémových parametrů a volbě typu pověření. Po opuštění první vrstvy aktivit je ověřovatel znovu vyzván k autentizaci pomocí otisku prstu. Aktivitu první vrstvy zobrazuje obrázek 3.25.

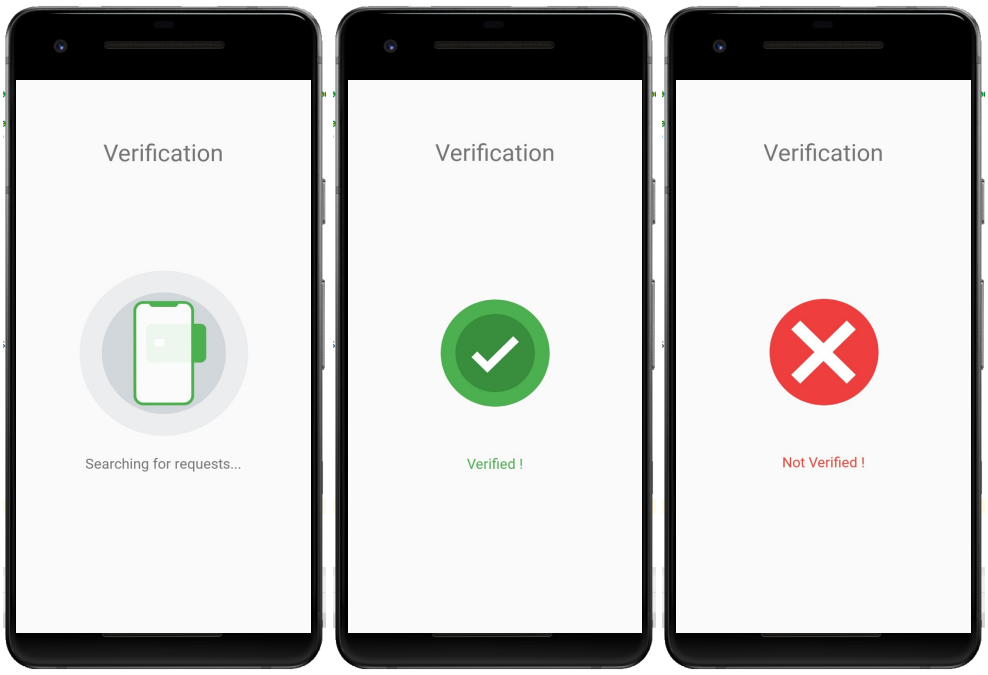
Aktivity druhé vrstvy

Za pomocí aktivit druhé vrstvy ověřovatel komunikuje s podpůrnými třídami, ovlivňuje stav lokální databáze aplikace, načítá systémové parametry a hodnoty odkrytých atributů prostřednictvím QR kódu, volí typ pověření a ověřuje vlastnictví předložených atributů. To vše pak může kontrolovat pomocí historie záznamů aplikace. Aktivity druhé vrstvy jsou:

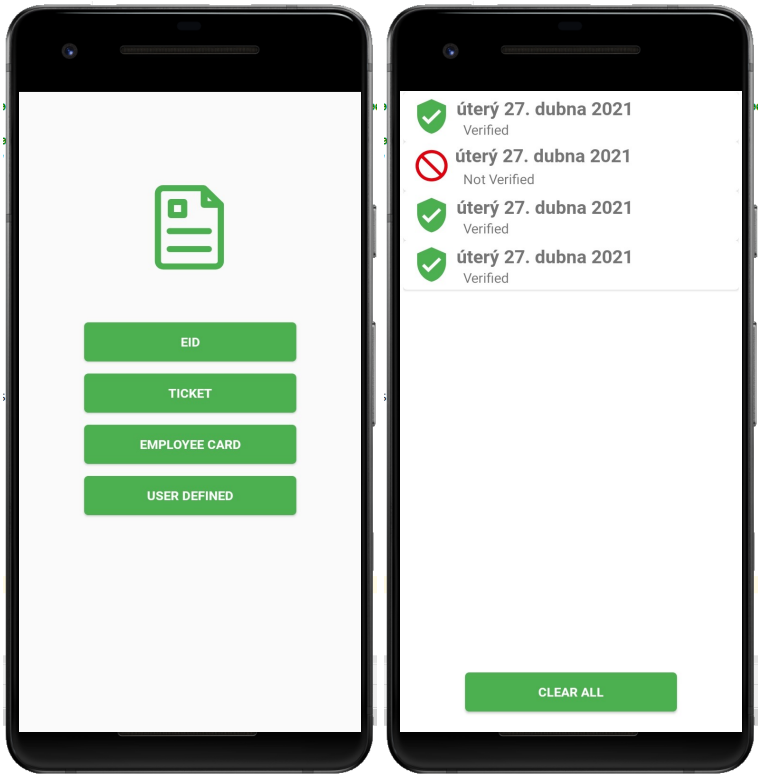
- ***VerificationActivity***: K zobrazení aktivity slouží tlačítko VERIFICATION. Aktivita zajišťující proces ověřování, která při své inicializaci spouští čtecí mód. Aplikace je následně připravena komunikovat pomocí APDU zpráv. Aktivita informuje ověřovatele o výsledku ověření nebo problémech, ke kterým během komunikace došlo. Jednotlivé události zároveň zaznamenává do lokální databáze. Při opuštění aktivity je čtecí mód vypnut. Aktivitu zajišťující proces ověření zobrazuje obrázek 3.26.



Obr. 3.25: Aplikace mobilního ověřovatele - aktivita MenuActivity.



Obr. 3.26: Aplikace mobilního ověřovatele - aktivita VerificationActivity.



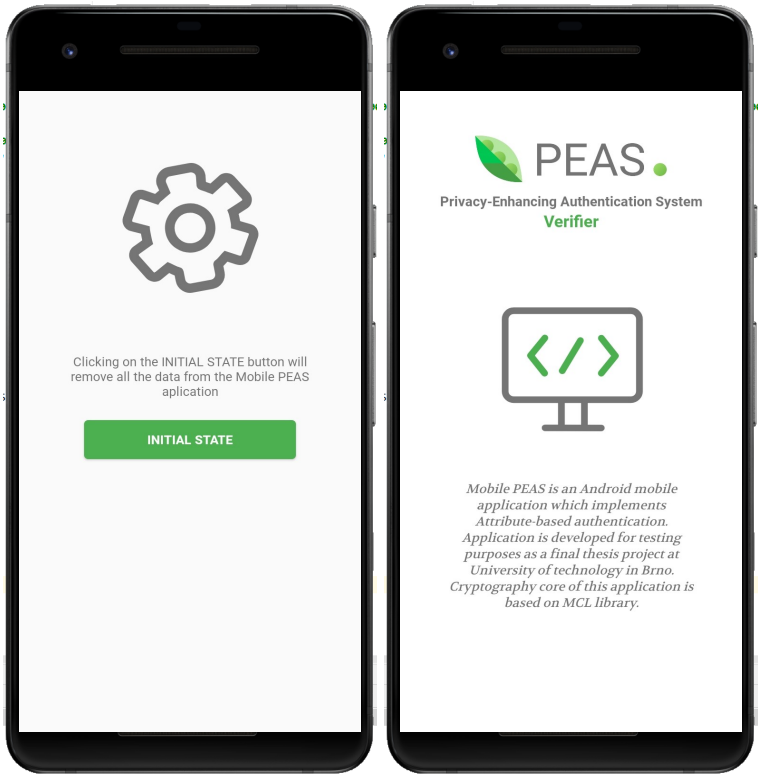
Obr. 3.27: Aktivity CryptoCredMenuActivity (vlevo) a HistoryActivity (vpravo).

- ***CryptoCredMenuActivity***: K zobrazení aktivity slouží tlačítko ATTRIBUTES SELECTION. Aktivita představuje rozcestník, na základě kterého se ověřovatel rozhoduje o typu pověření, které chce ověřovat. Aplikace mobilního ověřovatele podporuje pověření typu elektronická identita, tiket, zaměstnanecká karta nebo uživatelsky definované pověření. V jeden okamžik může mobilní ověřovatel ověřovat pouze jeden typ pověření. Aktivitu zajišťující rozcestník pro volbu typu pověření zobrazuje obrázek 3.27.
- ***HistoryActivity***: K zobrazení aktivity slouží tlačítko HISTORY. Aktivita poskytuje historii záznamů. Záznamy představují důležité události, ke kterým během běhu aplikace došlo. Aktivita ke svému provozu využívá podpůrné třídy *HistoryAdapter* a *HistoryItem*. Aplikace ověřovatele rozlišuje mezi několika událostmi, kterými jsou:
  - **Úspěšné ověření**: Událost vyvolaná úspěšným ověřením.
  - **Neúspěšné ověření**: Událost vyvolaná neúspěšným ověřením.
  - **Error**: Událost vyvolaná jakýmkoliv jiným neúspěchem během APDU komunikace. Název příkazu, při jehož zpracování vznikl problém je taktéž propsán do záznamu.

  Aktivitu zajišťující historii záznamů zobrazuje obrázek 3.27.

Obr. 3.28: Aplikace mobilního ověřovatele - aktivita QrActivity.

- ***QrActivity***: K zobrazení aktivity slouží tlačítko PARAMETERS SET-UP. Aktivita zajišťující čtení QR kódu, pomocí kterého si mobilní ověřovatel nastaví hodnoty systémových parametrů, které následně použije při ověření důkazu znalosti. V rámci čtení dochází ke kontrole velikosti dat, která je 550 bajtů. V případě jiné velikosti načtení neproběhne a ověřovatel je o této skutečnosti informován. Aktivitu zajišťující čtení systémových parametrů pomocí QR kódu zobrazuje obrázek 3.28.
- ***SettingsActivity***: K zobrazení aktivity slouží tlačítko SETTINGS. Aktivita pomocí které může ověřovatel uvést aplikaci do iniciálního stavu. Tato možnost tu je pro případ, že by se aplikace dostala do stavu, který by znemožňoval její správnou funkčnost. Aktivitu zobrazuje obrázek 3.29.
- ***AboutActivity***: K zobrazení aktivity slouží tlačítko ABOUT. Aktivita poskytuje uživateli základní informace o aplikaci, kterými jsou verze a účel aplikace. Aktivitu zobrazuje obrázek 3.29.

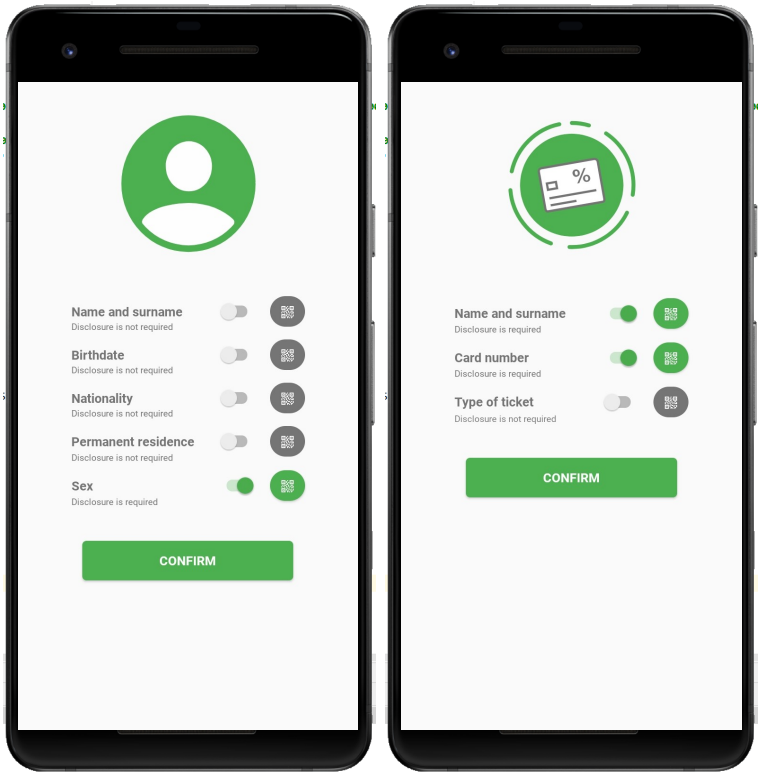


Obr. 3.29: Aktivity SettingsActivity (vlevo) a AboutActivity (vpravo).

Aktivity třetí vrstvy

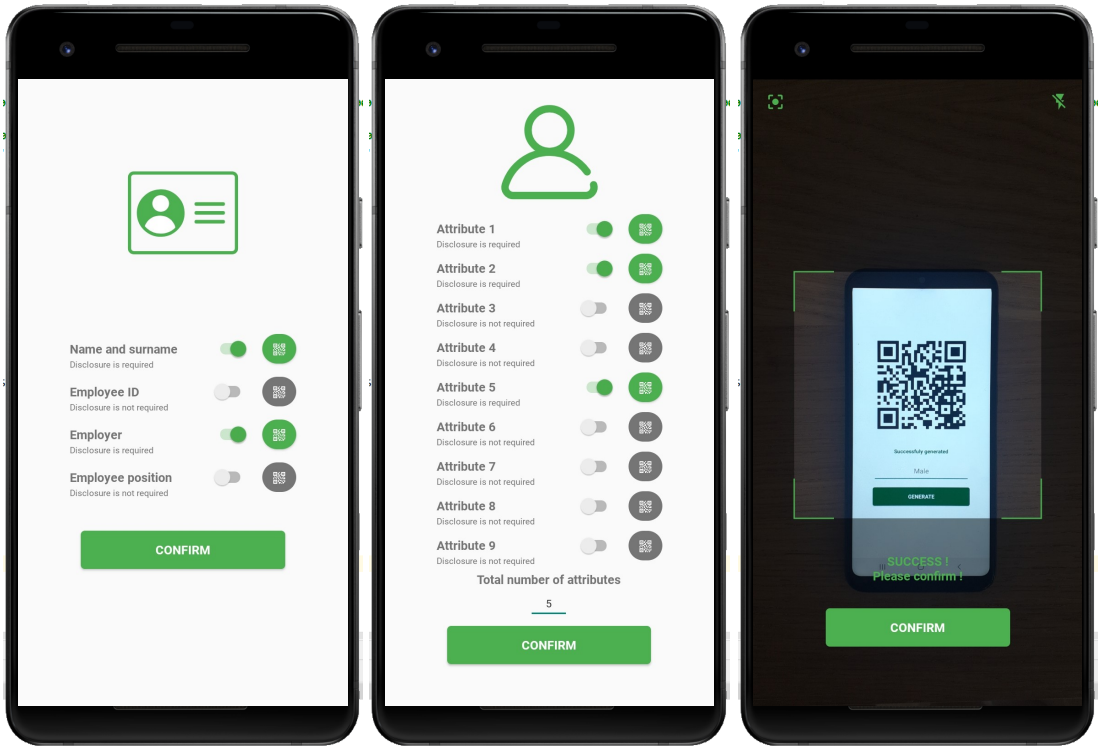
Aktivity třetí vrstvy komunikují s aktivitami nižších vrstev, podpůrnými třídami a lokální databází. Do aktivit třetí vrstvy se lze dostat pouze přes aktivitu *CryptoCredMenuActivity*. Třetí vrstvu tvoří aktivity jednotlivých typů pověření. V případě, že chce ověřovatel zaslat žádost o výpočet důkazu znalosti neodhalených atributů daného pověření, musí hodnoty odhalených atributů načíst pomocí QR sekenru. K tomu slouží tlačítko s obrázkem QR kódu. Iniciální stav jednotlivých atributů je skrytý. Aktivity jednotlivých typů pověření jsou:

- ***EidActivity***: K zobrazení aktivity slouží tlačítko EID. Aktivita umožňuje konfiguraci atributů pověření elektronická identita. Tento typ pověření obsahuje atributy jméno a příjmení, datum narození, národnost, trvalé bydliště a pohlaví. Aktivitu zajišťující konfiguraci pověření elektronická identita zobrazuje obrázek 3.30 (vlevo).
- ***TicketActivity***: K zobrazení aktivity slouží tlačítko TICKET. Aktivita umožňuje konfiguraci atributů pověření tiket. Tento typ pověření obsahuje atributy jméno a příjmení, číslo karty a typ tiketu. Aktivitu zajišťující konfiguraci pověření elektronická identita zobrazuje obrázek 3.30 (vpravo).



Obr. 3.30: Aktivity EidActivity (vlevo) a TicketActivity (vpravo).

- ***EmployeeCardActivity***: K zobrazení aktivity slouží tlačítko EMPLOYEE CARD. Aktivita umožňuje konfiguraci atributů pověření zaměstnanecká karta. Tento typ pověření obsahuje atributy jméno a příjmení, ID zaměstnance, zaměstnavatel, zaměstnanecká pozice. Aktivitu zajišťující konfiguraci pověření elektronická identita zobrazuje obrázek 3.31 (vlevo).
- ***UserDefinedActivity***: K zobrazení aktivity slouží tlačítko USER DEFINED. Aktivita umožňuje konfiguraci atributů pověření, které si uživatel sám definuje. Tento typ pověření obsahuje až 9 atributů definovaných samotným uživatelem. V tomto případě je potřeba definovat celkový počet atributů v systému. Aktivitu zajišťující konfiguraci pověření elektronická identita zobrazuje obrázek 3.31 (uprostřed).



Obr. 3.31: Aktivity EmployeeCardActivity (vlevo), UserDefinedActivity (uprostřed) a QrAttributeActivity (vpravo).

Aktivita čtvrté vrstvy

Aktivitu čtvrté vrstvy tvoří jediná aktivita, která zajišťuje načtení hodnoty odkrytého atributu pomocí QR skeneru. Do aktivity se lze dostat pouze pomocí tlačítka pro načtení QR kódu, které se nachází vedle příslušného atributu. Tlačítko je v případě skrytého atributu neaktivní. Aktivita komunikuje s lokální databází, do které v případě úspěšného načtení, hodnotu atributu uloží. V rámci čtení dochází ke kontrole velikosti dat, která je 32 bajtů. V případě jiné velikosti načtení neproběhne a ověřovatel je o této skutečnosti informován. Aktivitu *QrAttributeActivity* zobrazuje obrázek 3.31 (vpravo).

4 Testování aplikací a měření náročnosti

Testování představuje nedílnou součást procesu vývoje mobilní aplikace. Výstupem testování jsou výsledky vypovídající jak o kvalitě, tak o současném stavu provedení. Pro účely této diplomové bylo provedeno testování aplikace uživatele, popsané v části 3.1, mobilního ověřovatele, popsané v části 3.2 a provedeno měření časové, paměťové a výpočetní náročnosti.

V rámci testování byly aplikace nasazeny na dvě reálná zařízení spadající do kategorie nižší střední třídy. Zařízení byla zvolena z důvodu své cenové dostupnosti a rozšířenosti. Obě zařízení jsou si svými parametry velmi podobna, rozdílem je pouze jiná verze operačního systému a velikost operační paměti RAM. Parametry testovacích zařízení popisuje tabulka 4.1. Následující podkapitoly popisuji průběh testování dílčích aplikací společně s výsledky měření časové, paměťové a výpočetní náročnosti.

Model	CPU	Počet jader CPU	RAM	OS
Samsung Galaxy A20e	Exynos 7884	8	3 GB	Android 10
Samsung Galaxy A10	Exynos 7884	8	2 GB	Android 9.0

Tab. 4.1: Parametry testovacích zařízení.

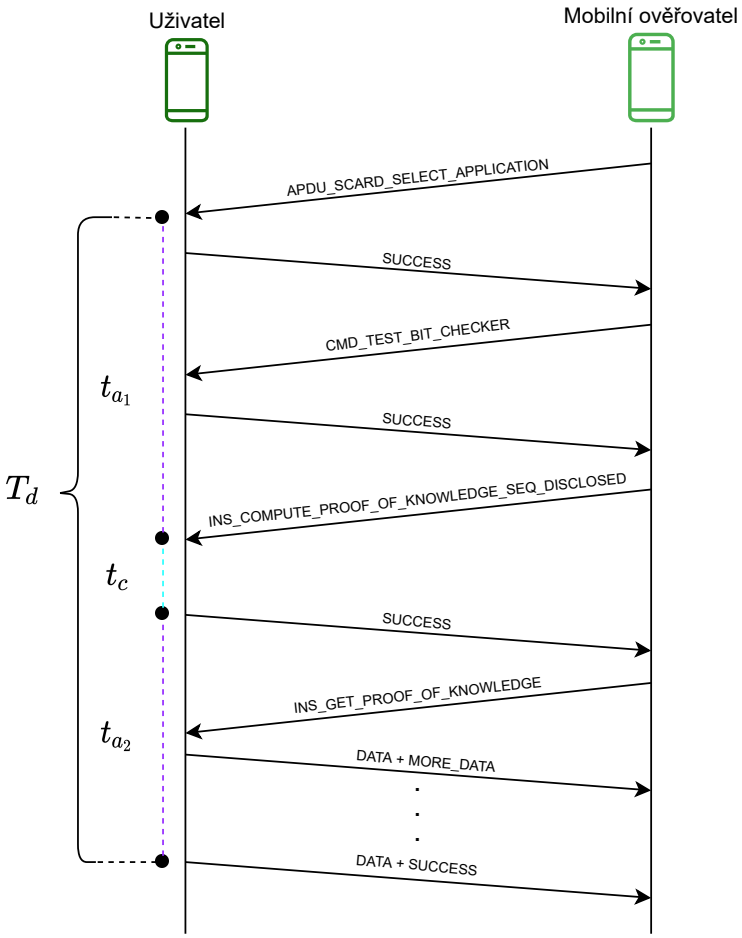
4.1 Časová náročnost

Časová náročnost představuje dobu nutnou pro inicializaci všech nutných vstupů, jejich následné zpracování a převedení na odpovídající výstupy. Časová náročnost se v případě obou aplikací skládá z časových úseků, během kterých probíhala APDU komunikace ($t_{a_i} \dots t_{a_n}$), nebo docházelo ke kryptografickým výpočtům t_c. Součtem těchto úseků pak dostaneme výslednou časovou náročnost T_d. V rámci této diplomové práce byl pro výpočet časové náročnosti použit tento vzorec:

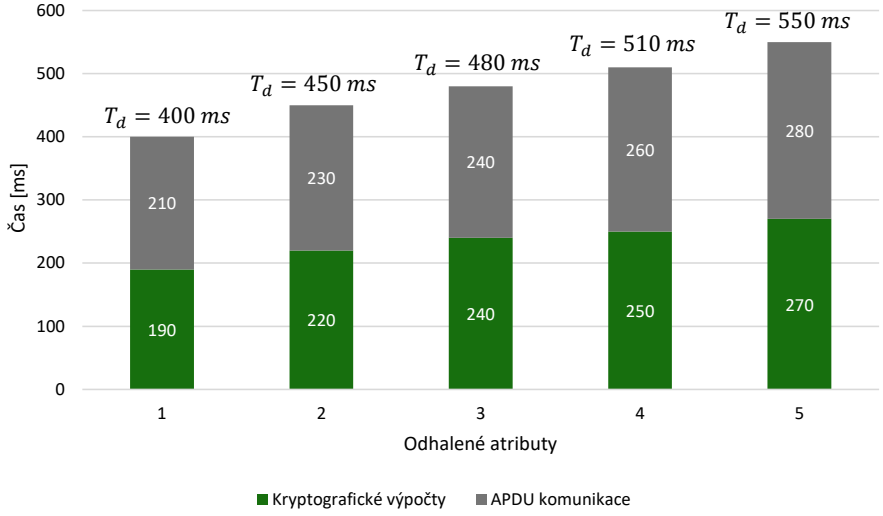
$$T_d = t_c + \sum_{i=1}^{n} t_{a_i}$$

Aplikace uživatele

Výslednou časovou náročnost aplikace uživatele tvoří tři časové úseky t_{a_1}, t_c a t_{a_2}. Význam jednotlivých úseků demonstruje schéma komunikace 4.1. Vliv počtu odhalených atributů na výsledné časové náročnosti zobrazuje sloupcový graf na obrázku 4.2. Pro každý z případů bylo provedeno deset měření a výsledné hodnoty zprůměrovány. Z grafu vyplývá, že s rostoucím počtem odhalených atributů roste i výsledná časová náročnost. Stále se však jedná o nepostřehnutelnou prodlevu.



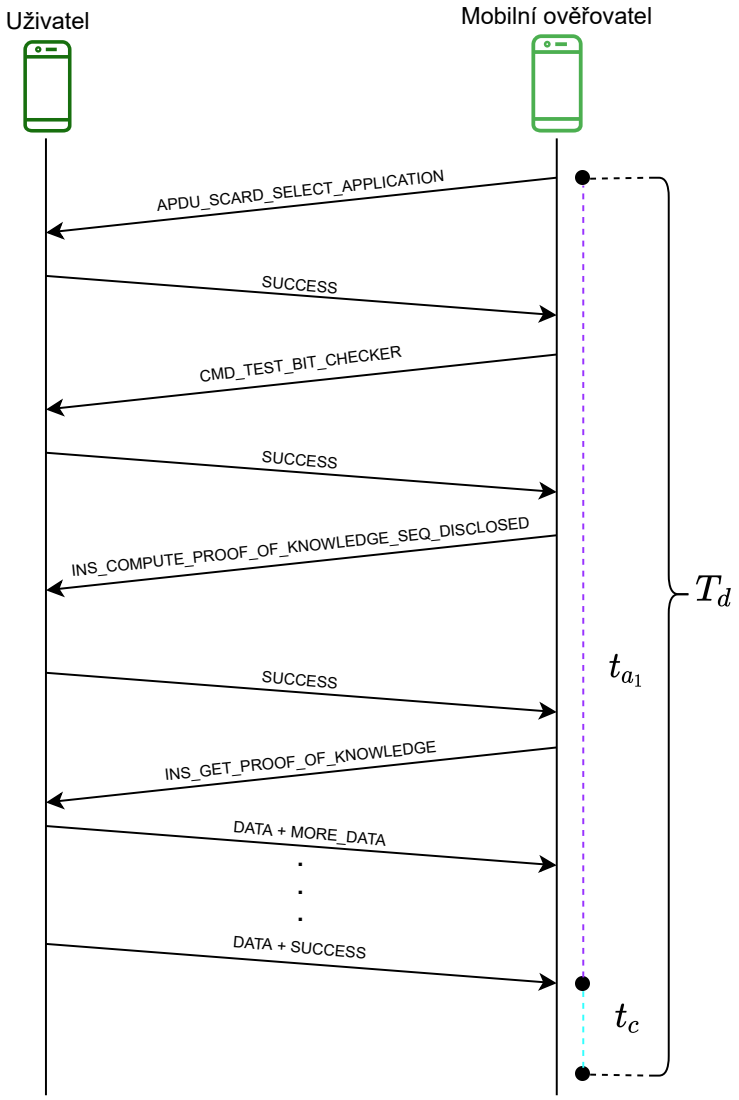
Obr. 4.1: Časové úseky komunikace u aplikace uživatele.



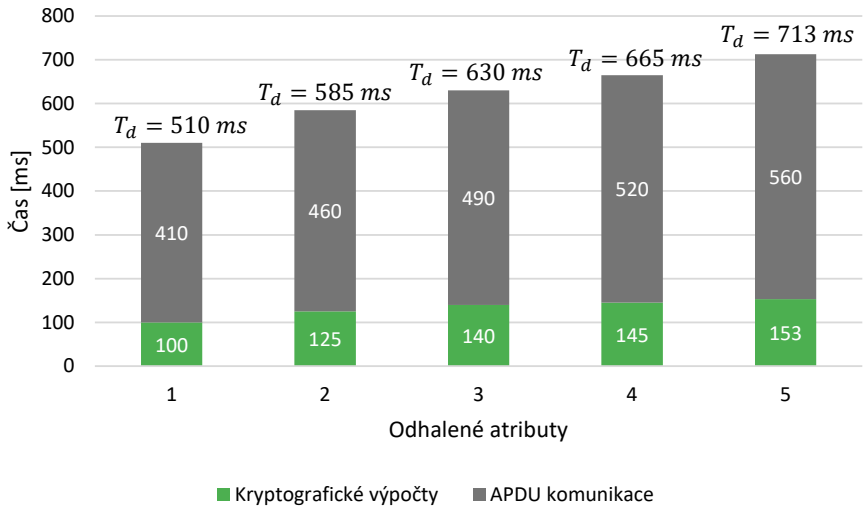
Obr. 4.2: Aplikace uživatele - vliv počtu odhalených atributů na časové náročnosti operací aplikace u zařízení Samusung Galaxy A20e.

Aplikace mobilního ověřovatele

Výslednou časovou náročnost aplikace mobilního ověřovatele tvoří dva časové úseky t_{a_1} a t_c. Význam obou úseků demonstruje schéma komunikace 4.3. Vliv počtu odhalených atributů na výsledné časové náročnosti zobrazuje sloupcový graf na obrázku 4.4. Pro každý z případů bylo provedeno deset měření a výsledné hodnoty zprůměrovány. Z grafu vyplývá, že s rostoucím počtem odhalených atributů roste i výsledná časová náročnost. Stále se však jedná o nepostřehnutelnou prodlevu. Kromě toho lze z grafu vyčíst celkovou dobu procesu ověřování, která se v případě pěti odhalených atributů pohybuje kolem 713 ms.



Obr. 4.3: Časové úseky komunikace u aplikace mobilního ověřovatele.



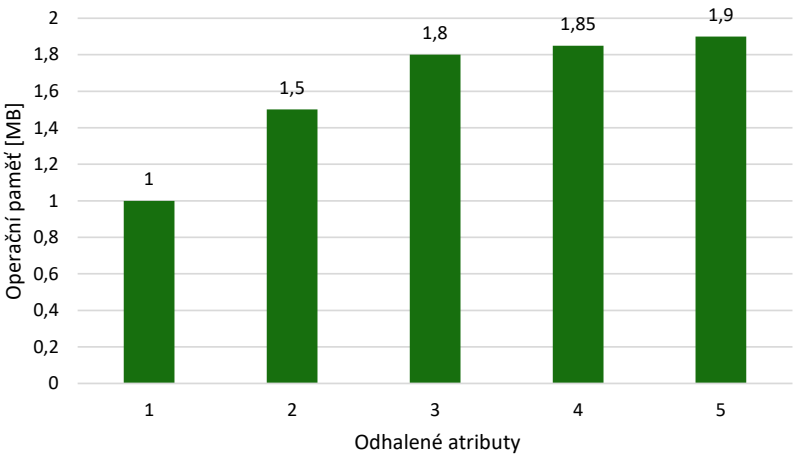
Obr. 4.4: Aplikace mobilního ověřovatele - vliv počtu odhalených atributů na časové náročnosti operací aplikace u zařízení Samusung Galaxy A10.

4.2 Paměťová náročnost

Na paměťovou náročnost lze pohlížet z několika pohledů. Prvním pohledem je zohlednění množství informací, které samotná aplikace nese a z toho vyplývající alokace interní paměti zařízení. Druhým pohledem je velikost operační paměti nutné pro běh jednotlivých aplikací. Třetí pohled představuje vliv výpočetní náročnosti na alokaci operační paměti.

Aplikace uživatele

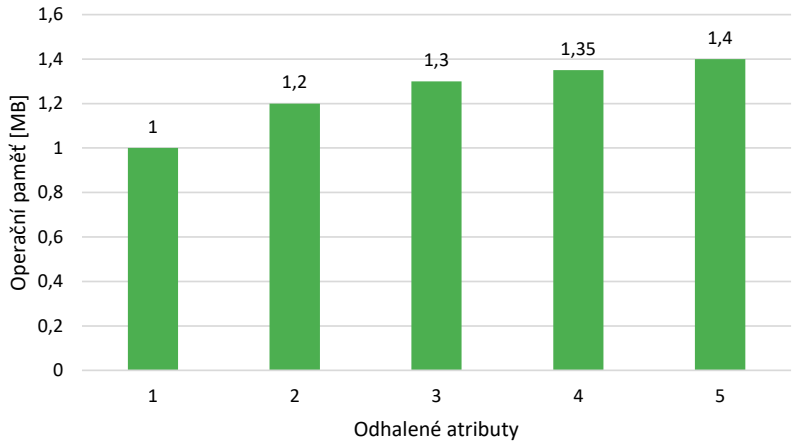
Množství informací, které aplikace uživatele nese je 6,2 MB. Jedná se o standardní velikost mobilní aplikace a vliv velikosti na potřebnou alokaci interní paměti je zanedbatelná. Aplikace při svém běhu zároveň vyžaduje alokaci zhruba 62 MB operační paměti. Tato hodnota je taktéž, vzhledem k velikostem operačních pamětí mobilních zařízení, zanedbatelná. Při výměně APDU zpráv a jejich následném zpracování, je zátěž na operační paměť nepatrně vyšší. Vliv počtu odhalených atributů při APDU komunikaci a kryptografických výpočtech na alokaci operační paměti zobrazuje sloupcový graf 4.5.



Obr. 4.5: Aplikace uživatele - vliv počtu odhalených atributů na alokaci operační paměti u zařízení Samusung Galaxy A20e.

Aplikace mobilního ověřovatele

Množství informací, které aplikace mobilního ověřovatele nese je 6,7 MB. Velikost je nepatrně vyšší oproti aplikaci uživatele. Tento rozdíl je způsoben větším počtem aktivit. Aplikace při svém běhu zároveň vyžaduje alokaci zhruba 67 MB operační paměti. Tato hodnota je taktéž vyšší. Rozdíl je způsoben vyšším výskytem animací aktivit. Při výměně APDU zpráv a jejich následném zpracování, je naopak zátěž na operační paměť nepatrně nižší. Tento rozdíl způsoben menším množstvím kryptografických výpočtů. Vliv počtu odhalených atributů při APDU komunikaci a kryptografických výpočtech na alokaci operační paměti zobrazuje graf 4.8.

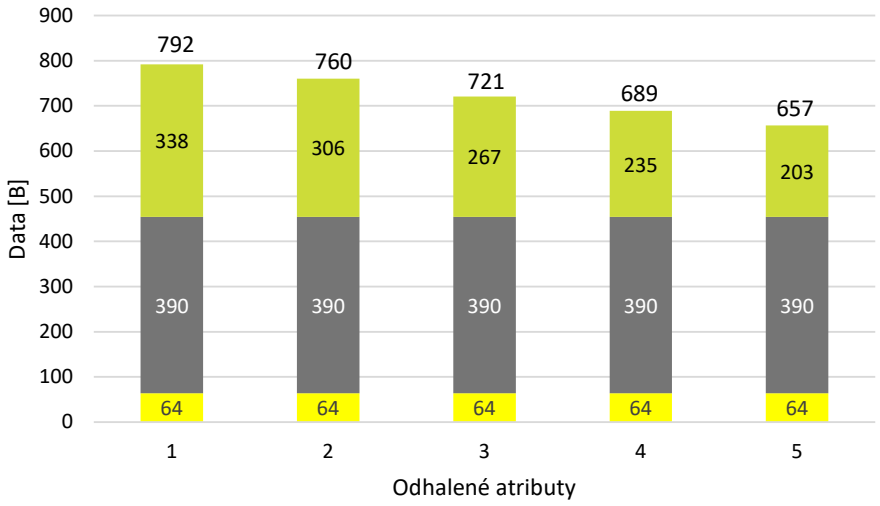


Obr. 4.6: Aplikace mobilního ověřovatele - vliv počtu odhalených atributů na alokaci operační paměti u zařízení Samusung Galaxy A10.

4.3 Výpočetní náročnost

Výpočetní náročnost byla v rámci této diplomové práce zohledněna z pohledu procentuálního zatížení procesoru zařízení. Obě aplikace využívají část výpočetní kapacity procesorů pro svůj běh. Během procesu ověřování je zatížení procesorů vyšší. Obě aplikace však využívají jen malou část výpočetní kapacity svých procesorů, a proto je taková zátěž vzhledem ke starším procesorům testovacích zařízení zanedbatelná.

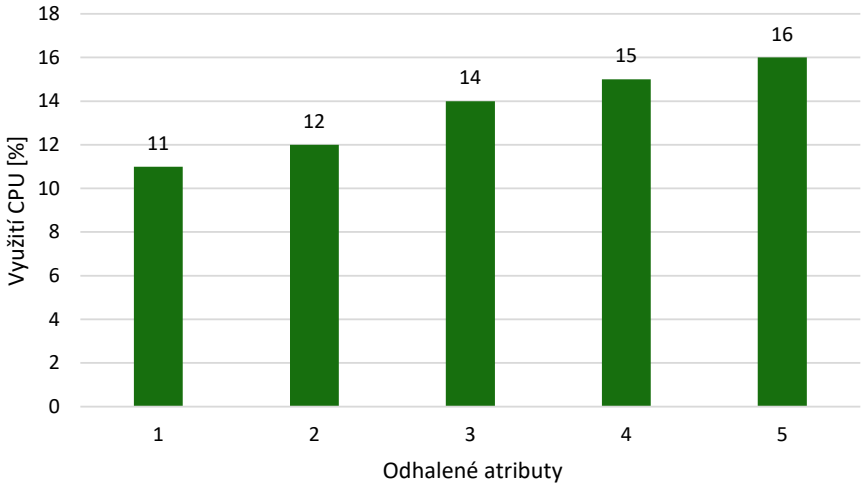
Na výpočetní náročnost má vliv i množství informací, které si mezi sebou jednotlivé aplikace vyměňují během APDU komunikace. Vliv množství vyměněných dat na počtu odhalených atributů mezi aplikacemi uživatele a mobilního ověřovatele zobrazuje graf 4.7. Z grafu vyplývá, že během komunikace docházelo k přenosu dat tří kategorií, z nichž pouze jedna má proměnlivou délku. 64 bajtů představuje množství dat, které byly mezi aplikacemi vyměněny během zahájení protokolu ověřování. Druhou kategorií jsou data nesoucí hodnotu parametru *cred*, popsanou v části 2.3.4. Tyto data mají vždy konstantní délku 390 bajtů. Poslední kategorií jsou data důkazu znalosti proměnlivé délky. Délka se odvíjí od počtu odhalených atributů. S rostoucím počtem odhalených atributů klesá počet přenášených dat. Tento úkaz je zapříčiněn postupnou absencí hodnot $\langle s_{m_{z \notin D}} \rangle$, popsaných v části 2.3.4.



Obr. 4.7: Vliv počtu odhalených atributů na množství přenášených dat při procesu ověřování.

Aplikace uživatele

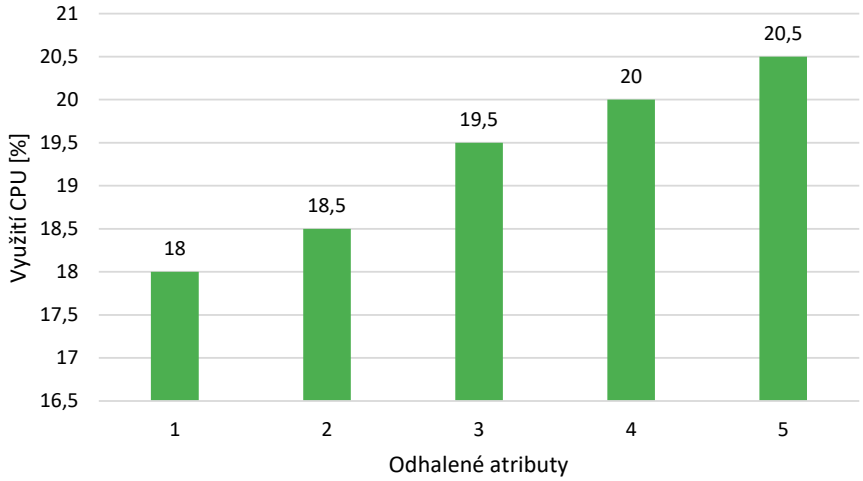
Aplikace uživatele během procesu ověřování využívá zhruba jednu šestinu výpočetní kapacity procesoru. Taková hodnota je při běžném používání zařízení nižší střední třídy zanedbatelná a žádným způsobem neovlivňuje plynulý chod zařízení. Vliv počtu odhalených atributů při APDU komunikaci a kryptografických výpočtech na využití výpočetní kapacity procesoru zařízení zobrazuje graf 4.8.



Obr. 4.8: Aplikace mobilního ověřovatele - vliv počtu odhalených atributů na procentuálním využití CPU u zařízení Samusung Galaxy A20e.

Aplikace mobilního ověřovatele

Aplikace mobilního ověřovatele během procesu ověřování využívá zhruba jednu pětinu výpočetní kapacity procesoru. Tato hodnota je oproti aplikaci uživatele nepatrně vyšší. Tento rozdíl je způsoben použitými animacemi aktivit. Takováto hodnota je však při běžném používání zařízení nižší střední třídy zanedbatelná a žádným způsobem neovlivňuje plynulý chod zařízení. Vliv počtu odhalených atributů při APDU komunikaci a kryptografických výpočtech na využití výpočetní kapacity procesoru zařízení zobrazuje graf 4.9.



Obr. 4.9: Aplikace mobilního ověřovatele - vliv počtu odhalených atributů na procentuálním využití CPU u zařízení Samusung Galaxy A10.

Závěr

Zadáním diplomové práce bylo seznámení se s atributovou autentizací jako takovou společně s dostupnými knihovnami poskytujícími podporu kryptografie na platformě Android. Dílčím úkolem bylo prozkoumat dostupné kryptografické knihovny podporující kryptografii nad eliptickými křivkami včetně možnosti využití nativních kryptografických knihoven pomocí sady nástrojů Android NDK. Úkolem praktické části diplomové práce pak byla implementace pilotního systému atributové autentizace na platformě Android (tj. strana uživatele s HCE, strana ověřovatele, vydavatele a revokačního manažera). Dílčím úkolem pak byl návrh a tvorba grafického uživatelského rozhraní pro jednotlivé entity systému a jeho následná integrace společně s kryptografickými funkcemi. Další dílčí úkol pak představovalo otestování a optimalizace implementovaného systému.

V teoretické části této diplomové práce byla nejdříve detailně rozebrána problematika bezpečnosti na platformě Android společně s analýzou dostupných kryptografických knihoven. V práci jsou popsány základní, rozšiřující, ale i nativní knihovny zajišťující bezpečnost na platformě Android. Práce také rozebírá sadu nástrojů Android NDK a službu HCE. Teoretická část se dále věnovala studiu atributové autentizace jako takové. Část popisuje entity a vlastnosti systému atributové autentizace, existující schémata systému atributové autentizace včetně reálné implementace v podobě projektu IRMA. Závěrem je rozebrán pilotní systém RKVAC, který je předmětem praktické části této diplomové práce.

Praktická část diplomové práce se zabývala implementací pilotního systému Atributové autentizace na platformě Android. Z důvodu nekompatibilního kryptografického jádra aplikace RKVAC s platformou Android bylo nutné implementovat vlastní kryptografické jádro založené na nativní kryptografické knihovně MCL. Tato skutečnost představovala podstatnou změnu při procesu vývoje jednotlivých aplikací. Během práce byly implementovány entity uživatele s HCE a mobilního ověřovatele. Implementované entity jsou s entitami pilotního systému RKVAC kompatibilní a schopny výměny dat autentizačního protokolu. Obě aplikace jsou zároveň použitelné v reálném prostředí. Schopnost mobilního ověřovatele provádět proces ověřovaní ve stížených podmínkách zapříčiňuje zvýšení atraktivnosti systému RKVAC pro další potenciální uživatele. V rámci praktické části bylo pro implementované entity navrženo a otestováno grafické uživatelské rozhraní. Závěrem bylo provedeno měření časové, paměťové a výpočetní náročnosti dílčích aplikací na cenově dostupných zařízeních.

Literatura

[1] *Mobile Operating System Market Share Worldwide.* [online]. [cit. 2020-08-10]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[2] *Android 10: Jaké přináší bezpečnostní změny?* [online]. [cit. 2020-10-10]. Dostupné z: https://www.datahelp.cz/clanky/android-10–jake-prinasi-bezpecnostni-zmeny.

[3] *Secure an Android Device.* [online]. [cit. 2020-23-10]. Dostupné z: https://source.android.com/security.

[4] HILDENBRAND, Jerry. *Project Mainline for Android 10: What it is, what it isn't, and how it works.* [online]. [cit. 2020-25-10]. Dostupné z: https://www.androidcentral.com/project-mainline.

[5] *Platform Architecture.* [online]. [cit. 2020-15-11]. Dostupné z: https://developer.android.com/guide/platform.

[6] *Application security - Elements of Applications.* [online]. [cit. 2020-15-11]. Dostupné z: https://source.android.com/security/overview/app-security.

[7] *Device Identifiers.* [online]. [cit. 2020-15-11]. Dostupné z: https://source.android.com/devices/tech/config/device-identifiers.

[8] HAYTON, Richard. *The Benefits of Trusted User Interface (TUI)* [online]. [cit. 2020-20-11]. Dostupné z: https://www.trustonic.com/technical-articles/benefits-trusted-user-interface/.

[9] *Personal Information*. [online]. [cit. 2020-20-11]. Dostupné z: https://source.android.com/security/overview/app-security.

[10] *Device Metadata*. [online]. [cit. 2020-20-11]. Dostupné z: https://source.android.com/security/overview/app-security.

[11] *Sensitive Data Input Devices*. [online]. [cit. 2020-25-11]. Dostupné z: https://source.android.com/security/overview/app-security.

[12] *Certificate authorities*. [online]. [cit. 2020-25-11]. Dostupné z: https://source.android.com/security/overview/app-security.

[13] *Application Signing*. [online]. [cit. 2020-26-11]. Dostupné z: https://source.android.com/security/overview/app-security.

[14] *Trail: The Extension Mechanism*. [online]. [cit. 2020-28-11].
Dostupné z: https://docs.oracle.com/javase/tutorial/ext/index.html.

[15] *Android-dev, Package Index*. [online]. [cit. 2020-28-11].
Dostupné z: https://developer.android.com/reference/packages.

[16] *Comparison of cryptography libraries*. [online]. [cit. 2020-29-11].
Dostupné z: https://en.wikipedia.org/wiki/Comparisonofcryptographylibraries.

[17] MITSUNARI, Shiego. *MCL Library*. [online]. [cit. 2020-29-11].
Dostupné z: https://github.com/herumi/mcl.

[18] *5 Key Benefits of Native Mobile App Development*. [online]. [cit. 2020-29-11].
Dostupné z: https://clearbridgemobile.com/benefits-of-native-mobile-app-development/.

[19] *Java vs C++: Which Language is Right for Your Software Project?* [online]. [cit. 2020-29-11].
Dostupné z: https://www.upwork.com/resources/java-vs-c-which-language-is-right-for-your-software-project.

[20] *ISO-7816*. [online]. [cit. 2020-02-12].
Dostupné z: https://cardwerk.com/smart-card-standard-iso7816-4-section-2-normative-references/.

[21] *NFC phones: The definitive list*. [online]. [cit. 2020-29-11]. Dostupné z:
https://en.wikipedia.org/wiki/List_of_NFC-enabled_mobile_devices.

[22] *Android-dev, Host-based card emulation overview*. [online]. [cit. 2021-02-05].
Dostupné z: https://developer.android.com/guide/topics/connectivity/nfc/hce.

[23] *Android NFC implementation*. [online]. [cit. 2021-02-05]..
Dostupné z: https://android.googlesource.com/platform/packages/apps/Nfc

[24] *Host-based card emulation overview - AID*. [online]. [cit. 2020-30-11].
Dostupné z: https://developer.android.com/guide/topics/connectivity/nfc/hce#AidGroups.

[25] *GDPR - Privacy by Design*. [online]. [cit. 2020-30-11].
Dostupné z: https://gdpr-info.eu/issues/privacy-by-design/.

[26] CHWASTKOVÁ, Šárka. *Atributová autentizace*. [online]. [cit. 2020-01-12]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/110195,
Vysoké učení technické v Brně. Vedoucí práce Petr Dzurenda.

[27] HAJNÝ, Jan. *Autentizace a ochrana soukromí*. [online]. [cit. 2020-01-12]. Dostupné z: https://www.slideshare.net/OKsystem/ smart-cards-devices-forum-2012-autentizace-a-ochrana-soukrom,

[28] HAJNÝ, Jan. *Autentizační protokoly a ochrana soukromí*. [online]. [cit. 2020-02-12]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/59396?zpid=59396, Vysoké učení technické v Brně. Vedoucí práce Karel Burda.

[29] *U-Prove*. [online]. [cit. 2020-02-12]. Dostupné z: https://www.microsoft.com/en-us/research/project/u-prove/

[30] *What is Identity Mixer?* [online]. [cit. 2020-02-12]. Dostupné z: https://idemix.wordpress.com/

[31] CELENG, M. *Bezpečnostní aplikace pro Android* [online]. [cit. 2020-02-12]. Dostupné z: https://dspace.vutbr.cz/handle/11012/34212. Vysoké učení technické v Brně. Vedoucí práce Jan Hajný.

[32] BRODA, Jan. *Aplikace pro programovatelné čipové karty*. [online]. [cit. 2020-02-12]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/125911, Vysoké učení technické v Brně. Vedoucí práce Jan Hajný.

[33] *IRMA App documentation*. [online]. [cit. 2020-02-12]. Dostupné z: https://irma.app/.

[34] Boneh-boyen signatures and the strong diffie-hellman problem. In Hovav Shacham and Brent Waters, editors, *Pairing-Based Cryptography - Pairing 2009*, pages 1-16, Berlin, Heidelberg, 2009. Springer Berling Heidelberg.

[35] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008. Dostupné z: https://link.springer.com/article/10.1007/s00145-007-9005-7.

[36] Chin-Ling Chen, Tzay-Farn Shih, Yu-Ting Tsai, and De-Kui Li. A Bilinear Pairing-Based Dynamic Key Management and Authentication for Wireless Sensor Networks. Journal of Sensors, 2015:14, 2015. Dostupné z: https://www.researchgate.net/publication/276344489.

[37] CAMENISCH, Jan., LYSYANSKAYA, Anna. (2004) Signature Schemes and Anonymous Credentials from Bilinear Maps. In: Franklin M. (eds) Advances in Cryptology – CRYPTO 2004. CRYPTO 2004. Lecture Notes in Computer Science, vol 3152. Springer, Berlin, Heidelberg. Dostupné z: https://doi.org/10.1007/978-3-540-28628-8_4.

[38] HAJNÝ, J.; DZURENDA, P.; CASANOVA MARQUÉS, R.; MALINA, L. Privacy ABCs: Now Ready for Your Wallets!. In Proceedings of The 19th International Conference on Pervasive Computing and Communications (IEEE PerCom 2021). 2021. s. 686-691. ISBN: 978-0-7381-4348-4.

[39] *GMP Library documentation.* [online]. [cit. 2020-05-12]. Dostupné z: https://gmplib.org/#WHAT.

[40] *Android ABIs.* [online]. [cit. 2020-05-12].. Dostupné z: https://developer.android.com/ndk/guides/abis.

[41] *Android-dev, MessageDigest.* [online]. [cit. 2020-02-12]. Dostupné z: https://developer.android.com/reference/java/security/MessageDigest.

[42] *HostApduService documentation.* [online]. [cit. 2020-05-12]. Dostupné z: https://developer.android.com/reference/android/nfc/HostApduService.

[43] *Rozpoznání obličeje nový standard bezpečnosti.* [online]. [cit. 2020-05-12] Dostupné z: https://www.viakom.cz/rozpoznani-obliceje-novy-standard-bezpecnosti/article-192.

[44] *Autentizace operátorů podle otisků prstů.* [online]. [cit. 2020-05-12] Dostupné z: https://automa.cz/cz/casopis-clanky/autentizace-operatoru-podle-otisku-prstu-200312290282557/.

[45] *Gradle User Manual.* [online]. [cit. 2020-05-12] Dostupné z: https://docs.gradle.org/current/userguide/userguide.html.

[46] *Android dev, BiometricManager.* [online]. [cit. 2020-05-12]. Dostupné z: https://developer.android.com/reference/androidx/biometric/BiometricManager.

[47] *Android dev, BiometricPrompt.* [online]. [cit. 2020-05-12]. Dostupné z: https://developer.android.com/reference/android/biometrics/BiometricPrompt.

[48] *Android dev, layouts.* [online]. [cit. 2020-05-12].. Dostupné z: https://developer.android.com/guide/topics/ui/declaring-layout.

[49] Lottie Docs - Android. [online]. [cit. 2020-02-05].. Dostupné z: https://airbnb.io/lottie/#/android.

[50] *Android-dev, RecyclerView.* [online]. [cit. 2020-02-05]. Dostupné z: https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.

[51] Zxing, A barcode image processing library. [online]. [cit. 2020-02-05]. Dostupné z: https://github.com/zxing/zxing.

[52] Budiyev, Code scanner library for Android, based on ZXing. [online]. [cit. 2020-02-05]. Dostupné z: https://github.com/yuriy-budiyev/code-scanner.

[53] P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In Selected Areas in Cryptography – SAC'2005, volume 3897 of LNCS, pages 319–331, Kingston, 2006. Springer-Verlag. Dostupné z: https://eprint.iacr.org/2005/133.

[54] ISO/IEC 14443, Identifikační karty - Bezkontaktní karty s integrovanými obvody - Karty s vazbou na blízko. [online]. [cit. 2020-02-05]. Dostupné z: https://en.wikipedia.org/wiki/ISO/IEC_14443.

[55] Dexter, Android library that simplifies the process of requesting permissions at runtime. [online]. [cit. 2020-02-05]. Dostupné z: https://github.com/Karumi/Dexter.

[56] *Android-dev, Uses-feature element.* [online]. [cit. 2020-02-12]. Dostupné z: https://developer.android.com/guide/topics/manifest/uses-feature-element.

[57] *Uses-permission element.* [online]. [cit. 2020-02-12]. Dostupné z: https://developer.android.com/guide/topics/manifest/uses-permission-element.

[58] *Android Service.* [online]. [cit. 2020-04-12]. Dostupné z: https://developer.android.com/guide/components/services.

Seznam symbolů, veličin a zkratek

ABC Attribute-based Credentials, Atributová autentizace

RKVAC Revocable Keyed-Verification Anonymous Credential, pilotní systém Atributové autentizace

NDK Native Development Kit, sada nástrojů, pomocí kterých lze v rámci platformy Android využívat kód napsaný pomocí programovacích jazyků C/C++

HCE Host–Card Emulation, emulace hostitelské karty

NFC Near Field Communication, technologie pro bezdrátovou komunikaci

GUI Graphical User Interface, grafické uživatelské rozhraní

MCL Nativní kryptografická knihovna

QR Quick Response, technologie pro komunikaci pomocí QR kódu

GDPR General Data Protection Regulation, obecné nařízení o ochraně osobních údajů

OS Operating System, operační systém

PIN Personal identification number, osobní identifikační číslo

IMEI International Mobile Equipment Identity, mezinárodní mobilní identifikátor

TLS Transport Layer Security, protokol transportní vrstvy

AOSP Android Open Source Project, otevřený systém platformy Android

JVM Java Virtual Machine, sada počítačových programů a datových struktur, která využívá modul virtuálního stroje ke spuštění dalších počítačových programů a skriptů vytvořených v jazyce Java

API Application Interface, aplikační rozhraní

SSL Secure Sockets Layer, vrstva poskytující zabezpečení komunikace šifrováním a autentizaci komunikujících stran

SIM Subscriber identity module, identifikační modul

GPS Global Positioning System, globální družicový polohový systém

CA	Certifikační autorita
ID	Identifier, identifikátor
RSA	Rivest–Shamir–Adleman Algorithm, algoritmus asymetrické kryptografie
DSA	Digital Signature Algorithm, algoritmus pro digitání podpis
PKCS	Public Key Cryptographic Standards, standard veřejných klíčů
DH	Diffie-Hellman Algorithm, algoritmus asymetrické kryptografie
DES	Data Encryption Standard, symetrická šifra
3DES	Triple DES, trojnásobný DES
PBE	Password Based Encryption, šifrování založené na hesle
RC2	Rivest Cypher 2, symetrická bloková šifra
RC5	Rivest Cypher 5, symetrická bloková šifra
HTTPS	Hypertext Transfer Protocol Secure, protokol umožňující zabezpečenou komunikaci v počítačové síti
BSD	Berkeley Software Distribution, licence pro svobodný software
AHEAD	Authenticated Encryption with Associated Data, autentizované šifrování
TPM	Trusted Platform Module, specifikace, která popisuje zabezpečený kryptoprocesor
SSH	Secure Shell, zabezpečený komunikační protokol v počítačových sítích
PGP	Pretty Good Privacy, program umožňující šifrování a podepisování
CMP	Certificate Management Protocol, protokol používaný pro získávání digitálních certifikátů X.509 v infrastruktuře veřejného klíče
OCSP	Online Certificate Status Protocol, protokol pro získání seznamu zneplatněných X.509 digitálních certifikátů
SCEP	Simple Certificate Enrollment Protocol, protokol pro pragmatické poskytování digitálních certifikátů většinou pro síťová zařízení

BN	Barreto-Naehring, typ pairing-friendly eliptické křivky
JNI	Java Native Interface, nativní rozhraní
WIFI	Wireless Fidelity, standard popisujících bezdrátovou komunikaci v počítačových sítích
CPU	Central Processor Unit, centrální procesorová jednotka
APDU	Application Protocol Data Unit, základní přenosová jednotka aplikačního protokolu
CLA	Pole instrukční třídy APDU příkazu
INS	Pole instrukčního kódu APDU příkazu
P1	První pole parametru APDU příkazu
P2	Druhé pole parametru APDU příkazu
LC	Pole velikosti přenášených dat APDU příkazu
LE	Pole velikosti očekávané odpovědi APDU příkazu
SW1	První pole statusu APDU odpovědi
SW2	Druhé pole statusu APDU odpovědi
PET	Privacy Enhancing Technologies, technologie zvyšující soukromí
IBM	International Business Machines Corporation, mezinárodní technologická společnost
V	Vydavatel
U	Uživatel
O	Ověřovatel
RA	Revokační autorita
EID	Electronic Identity, elektronická identita
VUT	Vysoké učení technické

A Obsah přiloženého CD

- `/` ..... kořenový adresář přiloženého CD
  - `AAAndroid_DP.pdf` ..... elektronická verze práce ve formátu PDF
  - `README.txt` ..... návod k použití
  - `HowToUserApp-en.mp4` ..... instruktážní video - uživatel (EN)
  - `HowToVerifierApp-en.mp4` ..... instruktážní video - mobilní ověřovatel (EN)
  - `HowToUserApp-cz.mp4` ..... instruktážní video - uživatel (CZ)
  - `HowToVerifierApp-cz.mp4` ..... instruktážní video - mobilní ověřovatel (CZ)
  - `User` ..... kořenový adresář aplikace uživatele
    - `gradle` ..... kořenový adresář pro gradle wrapper
    - `.gradle` ..... kořenový adresář pro gradle build proces
    - `UserPeas.apk` ..... zkompilovaná aplikace uživatele formátu apk
    - `app` ..... kořenový adresář aplikace uživatele (třídy, zdroje,...)
    - `build`
    - `.git` ..... kořenový adresář pro git
    - `.idea` ..... kořenový adresář pro jednotlivá nastavení projektu (IntelliJ IDE)
    - `.gitignore.txt` ..... soubor pro práci s git
    - `build.gradle` ..... kořenový adresář pro definici konfigurace nástroje gradle
    - `gradlew.bat` ..... skript pro práci s wrapperem nástroje gradle
    - `settings.gradle` ..... skript pro nástroj gradle a Groovy
    - `gradle.properties` ..... parametry pro gradle build
    - `local.properties` ..... parametry lokálního prostředí (sdk,...)
  - `Verifier` ..... kořenový adresář aplikace mobilního ověřovatele
    - `gradle` ..... kořenový adresář pro gradle wrapper
    - `.gradle` ..... kořenový adresář pro gradle build proces
    - `VerifierPeas.apk` . zkompilovaná aplikace mobilního ověřovatele formátu apk
    - `app` ..... kořenový adresář aplikace uživatele (třídy, zdroje,...)
    - `build`
    - `.git` ..... kořenový adresář pro git
    - `.idea` ..... kořenový adresář pro jednotlivá nastavení projektu (IntelliJ IDE)
    - `.gitignore.txt` ..... soubor pro práci s git
    - `build.gradle` ..... kořenový adresář pro definici konfigurace nástroje gradle
    - `gradlew.bat` ..... skript pro práci s wrapperem nástroje gradle
    - `settings.gradle` ..... skript pro nástroj gradle a Groovy
    - `gradle.properties` ..... parametry pro gradle build
    - `local.properties` ..... parametry lokálního prostředí (sdk,...)