VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Michal Moravanský

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

IMPLEMENTACE KRYPTOGRAFICKÝCH PROTOKOLŮ NA ČIPOVÉ KARTY
IMPLEMENTATION OF CRYPTOGRAPHIC PROTOCOLS ON SMART CARDS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Michal Moravanský**
AUTHOR

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

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Informační bezpečnost**

Ústav telekomunikací

Student: Michal Moravanský ***ID:*** 76275
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Implementace kryptografických protokolů na čipové karty

POKYNY PRO VYPRACOVÁNÍ:

Téma práce je zaměřeno na implementaci kryptografických protokolů na platformu současných programovatelných čipových karet (MultOS, JavaCard, BasicCard). Student analyzuje současné platformy čipových karet a kryptografické knihovny z pohledu kryptografické podpory a výkonnosti. Výstupem práce bude funkční implementace přiděleného anonymního pověřovacího schématu na čipovou kartu.

DOPORUČENÁ LITERATURA:

[1] BONEH, Dan; BOYEN, Xavier. Short signatures without random oracles and the SDH assumption in bilinear groups. Journal of Cryptology, 2008, 21.2: 149-177

[2] RANKL, Wolfgang; EFFING, Wolfgang. Smart card handbook. John Wiley & Sons, 2004.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

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

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Bakalářská práce je zaměřena na kryptografická schémata využívající atributová pověření, která se snaží minimalizovat negativní dopady na ochranu soukromí uživatelů při používání autentizačních systémů. Cílem bakalářské práce byla implementace dvou zadaných schémat na čipové karty jakožto zařízení s omezeným výkonem. Schémata se liší pouze ve schopnosti revokovat uživatele. Praktická část práce obsahuje analýzu a výběr platformy čipových karet a kryptografických knihoven v závislosti na výkonnosti. Práce dále popisuje architekturu obou schémat a jednotlivé protokoly včetně probíhající komunikace. Implementace atributového schématu byla provedena na programovatelnou čipovou kartu Multos (strana uživatele) a Raspberry Pi 2 (strana vydavatele a ověřovatele). Je také porovnávána časová náročnost vybraných algoritmů. V závěru jsou formulovány závislosti mající vliv na výslednou efektivitu a rychlost protokolu.

KLÍČOVÁ SLOVA

Autentizace, Anonymita, Atributová Pověření, Eliptická Křivka, Čipové Karty, MultOS

ABSTRACT

The bachelor thesis is focused on cryptographic schemes using Attribute-Based Credentials, which try to minimize the negative impact on the protection of a user's privacy when using authentication systems. The aim of the bachelor's thesis was the implementation of two specified schemes on smart cards as a device with limited performance. Schemes differ only in the ability to revoke user. The practical part of this paper contains the analysis and selection of smart card platform and cryptographic libraries depending on performance. The work also describes the architecture of both schemes and individual protocols, including ongoing communication. The implementation of the Attribute-Based Credentials scheme was performed on a programmable smart card Multos (user side) and Raspberry Pi 2 (issuer and verifier side). The time complexity of the selected algorithms was also compared. In the end, the dependencies affecting the resulting efficiency and speed of the protocol were formulated.

KEYWORDS

Authentication, Anonymity, Attribute-Based Credentials, Elliptic Curve, Smart Cards, MultOS

MORAVANSKÝ, Michal. *Implementace kryptografických protokolů na čipové karty*. Brno, 2020, 64 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Petr Dzurenda, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Implementace kryptografických protokolů na čipové karty“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Petru Dzurendovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. V neposlední řadě také děkuji rodičům a všem přátelům za podporu během celého studia.

Brno ....

podpis autora

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

Jedním z často diskutovaných témat několika posledních let je bezpečnost a ochrana soukromí v kyberprostoru. Toto téma se dostává do popředí díky výraznému nárůstu počtu elektronicky poskytovaných služeb. K zajištění bezpečného ověření proklamované identity v elektronických a fyzických přístupových systémech slouží autentizace. Primárním cílem autentizace je ochrana aktiv, mezi která obvykle patří data, služby nebo prostory. Bez autentizace bychom nebyli schopni poskytovat značný počet služeb jak v kyberprostoru, tak v běžném životě. Příkladem služby v kyberprostoru, pro kterou je autentizace nezbytnou základní službou, je elektronické bankovnictví. Jako další příklady lze uvést informační systémy, mobilní sítě atp. V reálném světě jsou to hlavně systémy řízení fyzického přístupu. Dále v běžném každodenním životě využíváme spoustu jiných nekritických aplikací, jako jsou docházkové systémy, předplatné jízdenky nebo věrnostní karty.

Pro autentizaci se v mnoha praktických aplikacích stále používají tradiční autentizační protokoly, které byly vyvinuté v 80. a 90. letech. Obecný princip těchto protokolů spočívá v tom, že na základě identity uživatele ověřuje autentizační systém znalost tzv. DF (Dokazovací Faktor). Dokazovacím faktorem mohou být unikátní charakteristické rysy nosiče DF (autentizace biometrikou, autentizace průkazem) nebo tajná data uložená v nosiči (autentizace znalostí, autentizace úložištěm). Tímto nosičem může být samotný uživatel žádající o přístup nebo autentizační předmět, který uživatel vlastní. V případě čipových karet je nosičem DF právě tato karta. Problémem tradičních systémů je, že při každém úspěšném pokusu o získání přístupu ke službě uvolňujeme náš osobní identifikátor, čímž zveřejňujeme svoji identitu. Tuto identitu mohou poskytovatelé služeb využít pro sledování chování nebo profilování našeho využívání služby. Na tuto situaci reagují i úřední aparáty na mezinárodní úrovni a vydávají svá nařízení na ochranu soukromí. Příkladem může být obecné nařízení o ochraně osobních údajů GDPR (General Data Protection Regulation) [1] nebo nařízení o elektronické identifikaci eIDAS (electronic Identification, Authentication and Trust Services) [2].

Negativní dopady na soukromí uživatelů se snaží minimalizovat moderní autentizační systémy založené na schématech atributové autentizace. Tyto systémy nejsou založeny na ověřování unikátních osobních identifikátorů, ale na ověření držení konkrétního atributu (např. věk, oprávnění k přístupu). Jako DF slouží právě tyto atributy, které jsou uloženy uvnitř nosiče v certifikované formě. Implementací těchto schémat na čipové karty se zabývá tato práce. Čipové karty jsou vhodným nosičem DF pro moderní autentizační systémy, protože mají integrovaný mikropočítač. Lze na nich tedy realizovat autentizační protokoly založené na kryptografických technikách.

1 Kryptografické metody a primitiva

Pro pochopení atributových schémat, které jsou použity pro implementaci kryptografických protokolů, je nutné nejdříve uvést základní kryptografické metody a principy. Všechny zde popsané metody a primitiva jsou v této práci použity, nicméně největší důraz je kladen na kryptografii na bázi eliptických křivek, která je součástí všech použitých protokolů.

1.1 Kryptografie na bázi eliptických křivek

Kryptografie založená na bázi eliptických křivek ECC (Elliptic Curve Cryptography) spadá do kategorie asymetrických kryptografických systémů. Dosahuje požadované kryptografické bezpečnosti při menší délce klíče ve srovnání s běžnými kryptosystémy na bázi diskrétního logaritmu. Což je velká výhoda u zařízení s omezenou velikostí paměti, jako jsou čipové karty. Pro srovnání lze uvést, že u tradičních struktur je bezpečná délka klíče 3072 bitů, což je ekvivalentní s délkou klíče 256 bitů u struktur založených na eliptických křivkách. Toto porovnání vychází z doporučení Národního institutu standardů a technologií v USA NIST (National Institute of Standards and Technology) [3]. Mezi další výhody lze uvést její rychlost a menší náročnost na hardware i software.

Eliptická křivka EC (Eliptic Curve) může být popsána jako algebraická struktura, která je konstruována nad různými tělesy $\mathbb{K}$. Za obecné těleso $\mathbb{K}$ si můžeme dosadit množinu reálných čísel $\mathbb{R}$, komplexních čísel $\mathbb{C}$, racionálních čísel $\mathbb{Q}$ a konečná tělesa $\mathbb{F}_q$. Příklady vybraných konstrukcí jsou zobrazeny na obrázku 1.1. Výpočty nad tělesem $\mathbb{R}$ nejsou vhodné pro reálné kryptografické systémy. Mohly by totiž vzniknout zaokrouhlovací chyby a tyto systémy vyžadují rychlé a přesné výpočty. Z tohoto důvodu se používají eliptické křivky konstruované nad konečnými tělesy $\mathbb{F}_q$, kde $q = p^m$ (p je prvočíslo a $m \geq 1$). Každá taková eliptická křivka může být zapsána pomocí tzv. Weierstrassovy rovnice (1.1) [4][5]. Eliptická křivka $E(\mathbb{F}_q)$ je algebraická křivka daná rovnicí:

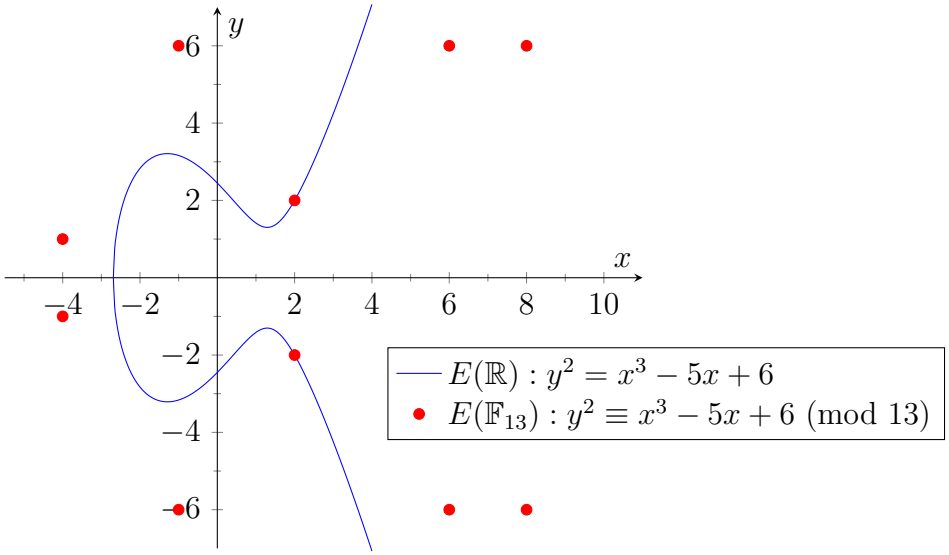
$$E(\mathbb{F}_q) : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{1.1}$$

kde $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_q$ jsou konstantní koeficienty křivky. Dodatečnou podmínkou je, že se jedná o hladkou spojitou křivku, na níž je definován bod $\mathcal{O}$, který leží v nekonečnu. V praxi se pro konečná tělesa $\mathbb{F}_p$ využívá zjednodušená forma Weierstrassovy rovnice (1.2). Můžeme ji definovat jako množinu bodů (x, y), které vyhovují matematické rovnici 3. řádu:

$$E(\mathbb{F}_p) : y^2 = x^3 + ax + b, \tag{1.2}$$

doplněná o podmínky pro hodnoty koeficientů $a, b \in \mathbb{F}_p$ tak, že je polynom třetího řádu $y^2 = x^3 + ax + b$ nerozložitelný a platí $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. To znamená, že daná eliptická křivka může tvořit grupu nad tělesem $\mathbb{F}_p$. EC je symetrická podle osy x (kromě nulového bodu $\mathcal{O}$) a má konečný počet bodů $\#E$ (včetně bodu $\mathcal{O}$). Ten nazýváme řádem křivky $\#E(\mathbb{F}_p)$. Největší prvočíselný dělitel r řádu EC nazýváme kofaktor h a lze ho vyjádřit vztahem 1.3:

$$h = \frac{\#E}{r}. \tag{1.3}$$



Obr. 1.1: Porovnání EC: modře eliptická křivka $E(\mathbb{R})$, kde $(x, y) \in \mathbb{R} \times \mathbb{R}$ a červeně křivka $E(\mathbb{F}_{13})$, kde $(x, y) \in \mathbb{F}_{13} \times \mathbb{F}_{13}$ (bod v nekonečnu není vykreslen).

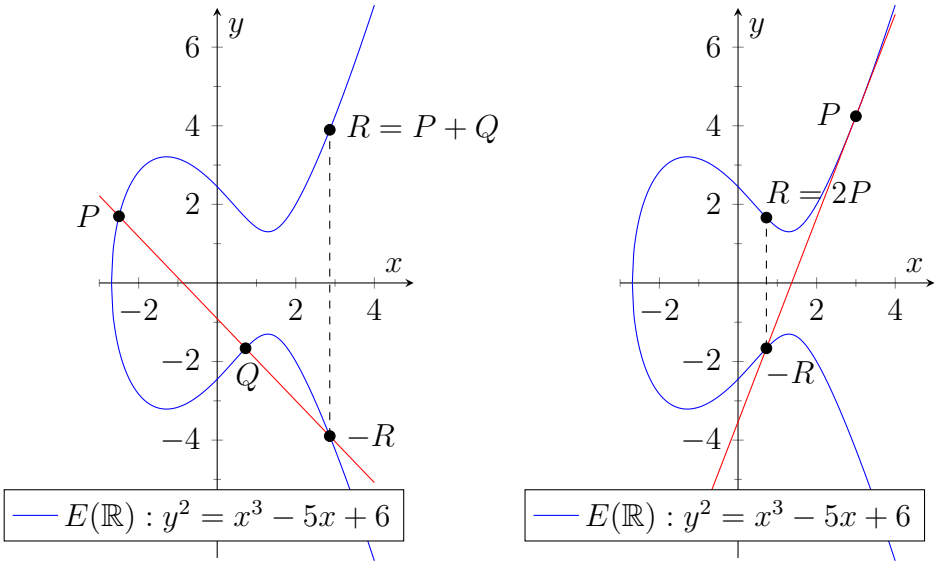
Početní operace na EC

Jednotlivé operace můžeme popsat geometricky nebo algebraicky. K implementaci výpočtů je použit přístup algebraický. Geometrický způsob je názorný a je použit pro popsání operací na eliptických křivkách $E(\mathbb{R})$:

- Základní definovanou operací pro eliptické křivky je **sčítání**. Nechť P, Q jsou dva různé body ležící na EC a $P \neq Q$. Součet těchto bodů $P + Q = R$ lze popsat dvěma kroky. Nejprve je sestrojena přímka procházející body P, Q, která protne EC právě v jednom bodě. Tento bod je možno nazvat $-R$, a je zrcadlovým obrazem hledaného bodu R podle osy x. Aby bylo možno sečíst body P a $Q = -P$, jež jsou umístěné symetricky dle osy x, je k EC přidán bod $\mathcal{O}$, který leží v nekonečnu. Platí tedy $P + (-P) = \mathcal{O}$.
- Pokud je bod P sčítán se sebou samým, tak se tečna k EC dotýká bodu P. V případě, že $y_P \neq 0$, tak tečna protíná EC v bodě $-R$. Zrcadlový obraz

R podle osy x odpovídá rovnici $P + P = 2P = R$. Pokud $y_P = 0$, pak je tečna kolmá na osu x a protíná EC v nekonečnu. Lze tedy říci, že 2P = $\mathcal{O}$. Tato operace se nazývá zdvojnásobení bodu. Opakovaným použitím operace zdvojnásobení bodu je definováno **násobení** bodu celým číslem.

Jednotlivé body EC tvoří spolu s operací sčítání Abelovskou grupu, kde bod v nekonečnu $\mathcal{O}$ je neutrální prvek. Operace jsou graficky znázorněny na obrázku 1.2.



Obr. 1.2: Z leva do prava: sčítání bodů na EC $R = P + Q$, zdvojnásobení bodu (Point Doubling) $R = 2P$.

Bilineární Párování

Kryptografie založená na bilineárním párování nad eliptickými křivkami je v praxi často využívána při vytváření kryptografických schémat. Jako příklad lze uvést schémata využívající atributového pověření ABC (Attribute-Based Credentials). Bilineární párování je mapa $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, která splňuje podmínky bilinearity a nedegenerativnosti. $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ jsou obvykle cyklické grupy řádu q. Bližší pohled na problematiku nabízí [6].

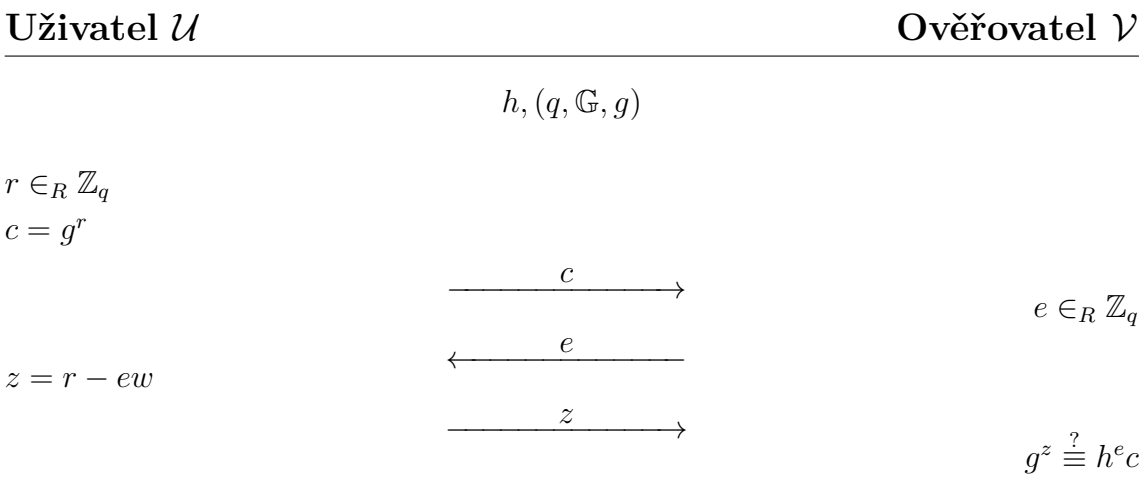
1.2 Sigma protokoly

Sigma protokoly (označované Σ-protokoly) jsou považovány za praktickou formu protokolů s nulovou znalostí ZK (Zero-Knowledge). Více o ZK protokolech lze nalézt v [7]. Σ-protokoly jsou schopny prokazovat vlastnictví tajné informace uživatele bez jejího odhalení [8]. Jejich předností je rychlost a efektivnost, což je důvodem pro jejich použití v reálných systémech, protože stačí tři výměny zpráv a pouze

jedna iterace. Každý Σ-protokol musí splňovat tři základní vlastnosti a oproti ZK protokolům je požadována ještě čtvrtá vlastnost:

- **Úplnost** – oprávněný uživatel, který zná tajnou informaci, musí být vždy schopen vytvořit správnou odpověď a být tedy úspěšně autentizován.
- **Spolehlivost** – naopak útočník jako neoprávněný uživatel bez znalosti potřebné tajné informace nesmí být schopen přesvědčit ověřovatele o opaku a ověřovatel tedy interakci odmítne.
- **Nulová znalost** – během komunikace nesmí být uvolněny žádné soukromé informace. Ověřovatel tedy není schopen zjistit žádné jiné informace o uživateli kromě toho, zda uživatel zná nebo nezná ověřovanou tajnou informaci.
- **Třícestnost** – k vygenerování důkazu znalosti stačí výměna pouze tří zpráv (závazek, výzva, odpověď). V první zprávě se uživatel zavazuje k danému tajemství w. Na tuto zprávu ověřovatel odpovídá výzvou. Poslední zprávou je odpověď, kterou zasílá opět uživatel. Na základě těchto zpráv je možno na straně ověřovatele ověřit znalost tajné informace w uživatele.

Existují *interaktivní* Σ-protokoly, které jsou vhodné a často používané ke konstrukci atributových autentizačních schémat. Nejznámějším reprezentantem takových protokolů je Schnorrův protokol [9], který je zobrazen na obrázku 1.3. Uživatel disponuje tajemstvím $w \in \mathbb{Z}_q$, h je veřejný klíč uživatele a je dán jako $h = g^w$.



Obr. 1.3: Schnorrův protokol.

Druhou skupinou jsou *neinteraktivní* Σ-protokoly, které využívají pro výpočet náhodné výzvy e hašovací funkci na straně uživatele. U těchto protokolů pak proběhne veškerá komunikace v rámci jedné zprávy. Tato varianta je často označována jako SPK (Signature Proof of Knowledge) a je využívána v podpisových schématech.

1.3 Autentizační techniky

Autentizace lze obecně definovat jako proces dokazování a ověřování integrity či zdroje dat nebo identity uživatele. Lze ji provádět za pomoci symetrické i asymetrické kryptografie. Samotný proces autentizace je realizován pomocí autentizačních protokolů, které jsou založené na třech základních metodách [10]:

- **Přenos hesla** – konvenční metoda spadající do kategorie technik symetrické kryptografie, která poskytuje jednostranné ověření. Je založena na sdíleném tajemství mezi uživatelem a ověřovatelem. Důkaz znalosti tohoto tajemství probíhá jeho samotným odhalením.
- **Metoda výzva-odpověď** – jedna entita (uživatel) prokazuje svou identitu vůči jiné entitě (ověřovateli) tím, že prokáže znalost tajemství, aniž by bylo odhaleno v průběhu komunikace. Toho lze docílit tím, že odpověď závisí nejen na samotném tajemství uživatele ale i na výzvě ověřovatele.
- **Metody založené na pokročilé kryptografii** – jsou založeny na myšlenkách interaktivních důkazních systémů a důkazů nulové znalosti. Příkladem protokolů, které využívají tyto metody jsou Σ-protokoly viz kapitola 1.2.

Žádná z výše uvedených základních metod ovšem nenaplňuje stoupající potřebu ochrany soukromí. Často bývá identita uživatele odhalena pomocí uživatelského jména nebo jednoznačného identifikátoru.

Atributová autentizace

Zásadní předností schémat atributové autentizace je, že nejsou postavena na verifikaci identity uživatele. Jednoznačná identita (např. uživatelské jméno, veřejný klíč atp.) není v procesu ověření přímo vyžadována. Tím odstraňují nedostatky základních autentizačních metod a mají potřebné dispozice k ochraně soukromí uživatelů. Jsou založena na ověřování držení konkrétního atributu (vlastnosti). Takových vlastností může být nespočet. Jako příklad lze uvést dosažení určitého věku, držení konkrétního oprávnění (řidičský, zbrojní průkaz), přístup do střežených prostor atp. Během ověřování dochází k **selektivnímu uvolňování** atributů, aniž by došlo k odhalení další informace. Uživatel tedy může prokázáním držení konkrétního atributu získat přístup a nadále zůstat v **anonymitě**. Dalšími pokročilými funkcemi na ochranu soukromí a osobních údajů jsou **nespojitelnost** a **nesledovatelnost**. Může také obsahovat ochranný mechanismus pro případy, kdy konkrétní uživatel porušuje nastavená pravidla. Tento mechanismus se nazývá **revokace** a může uživatele vyloučit či případně odhalit jeho identitu. Schémata atributové autentizace jsou také označována jako ABC schémata, neboli schémata využívající atributového pověření. Jednotlivé schémata se od sebe navzájem odlišují vlastnostmi, a proto bližší popis je vždy uveden u konkrétního schématu.

2 Čipové karty

Čipovou kartu obvykle představuje plastová karta kapesní velikosti s jednoduchým paměťovým čipem nebo mikroprocesorem. Kapesní velikost ovšem není povinná, v praxi se lze setkat i s kartami menších velikostí jako například mini, mikro a nano SIM (Subscriber Identity Module) karty mobilních operátorů. Tyto čipové karty mohou sloužit k ukládání dat nebo jako bezpečný nosič DF pro autentizaci uživatele, čehož je využito v této práci. V případě karet s paměťovým čipem lze využívat pouze jednoduchých autentizačních metod jako je poskytnutí obsahu své paměti (heslo) ověřovateli, protože tato úložiště nejsou vybaveny dokazovacím procesorem. Jsou vhodné pouze pro aplikace s fixními operacemi, např. členská karta, skipas atp. Z tohoto důvodu nebude o kartách s paměťovým čipem dále uvažováno.

Naopak karty s mikroprocesorem obsahují vlastní mikropočítač a jsou označovány jako ICC (Integrated Circuit Card) nebo SC (Smart Card). Slouží jako samotný nosič DF i jako dokazovací procesor, a jsou tedy schopny zpracovávat data. Mohou být na nich realizovány autentizační protokoly, které jsou založené na kryptografických technikách. Tato vlastnost zvyšuje úroveň bezpečnosti, proto se tyto karty staly široce využívané. Jako praktický příklad lze uvést bankovnictví, telekomunikační sítě. Mikroprocesorové karty mohou mít navíc vestavěný kryptografický procesor (co-procesor), který umožňuje akcelerovat některé kryptografické algoritmy, např. AES (Advanced Encryption Standard), RSA (Rivest, Shamir, and Adelman), ECDSA (Elliptic Curve Digital Signature Algorithm) nebo ECDH (Elliptic-curve Diffie-Hellman).

Pokud jsou využívány pouze k jedné aplikaci, a mají pouze statický přístup k paměti, nazýváme je jako jednoaplikační. Existují také multiaplikační programovatelné karty, které disponují vlastním operačním systémem a umožňují vývoj a instalaci uživatelských aplikací. Mezi nejrozšířenější technologie patří JavaCard, MultOS, BasicCard a .NETCard.

2.1 Druhy čipových karet

Dle druhu komunikace mezi přístupovým terminálem a čipovou kartou je definováno základní dělení čipových karet na kontaktní, bezkontaktní a hybridní.

Mezi všeobecně známé zástupce čipových karet s galvanickým rozhraním patří rozhraní typu USB (Universal Serial Bus) nebo standardu ISO 7816 (the International Organization for Standardization) [11]. Použití těchto rozhraní je omezeno spíše na přístupové systémy do virtuálních prostorů z důvodu nepohodlnosti při každodenním používání. Uživatel musí kvůli galvanickému propojení připojit kartu přesně na danou pozici. Mikroprocesor je pomocí vodičů spojen s kontaktní deskou a tato

sestava je vlepena do dutiny v plastové kartě. Obsahuje ROM (Read-Only Memory) paměť, kde je „vypálen“ operační systém přímo od výrobce, EEPROM (Electrically Erasable Programmable Read-Only Memory) neboli flash paměť a RAM (Random Access Memory) paměť. Napájení integrovaného obvodu je řešeno přes kontaktní desku. Stejným způsobem je řešeno i rozhraní pro datový přenos mezi kartou a terminálem (přes sériovou linku). Rychlost datového přenosu je do 223 kb/s.

Pro bezkontaktní čipové karty neboli karty s bezdrátovým rozhraním je definováno více standardů ISO/IEC (standards for International Electrotechnical Commission). Liší se obvykle tím, na jakou vzdálenost jsou tyto karty schopny komunikovat s terminálem. Pro použití v autentizačních systémech jsou nejpoužívanější karty dle standardu ISO/IEC 1443 označované jako karty s vazbou na blízko (Proximity Cards). Komunikační vzdálenost je omezena na 10 cm. U těchto karet je mikroprocesor spolu s anténou zalisován přímo mezi přední a zadní plastovou desku karty. Mohou být napájeny i z autonomní baterie ale v praxi se používají karty bez vlastního zdroje, které jsou napájeny elektromagneticky. V tomto standardu jsou definovány dva přenosové protokoly A a B. V praxi je používanější protokol A. U tohoto protokolu probíhá přenos rychlostí 848 kb/s.

Třetím a posledním zástupcem čipových karet jsou hybridní karty. Tyto karty kombinují oba předešle uvedené typy přenosového rozhraní. V poslední době se dostali hodně do popředí díky tomu, že jsou využívány v bankovnictví.

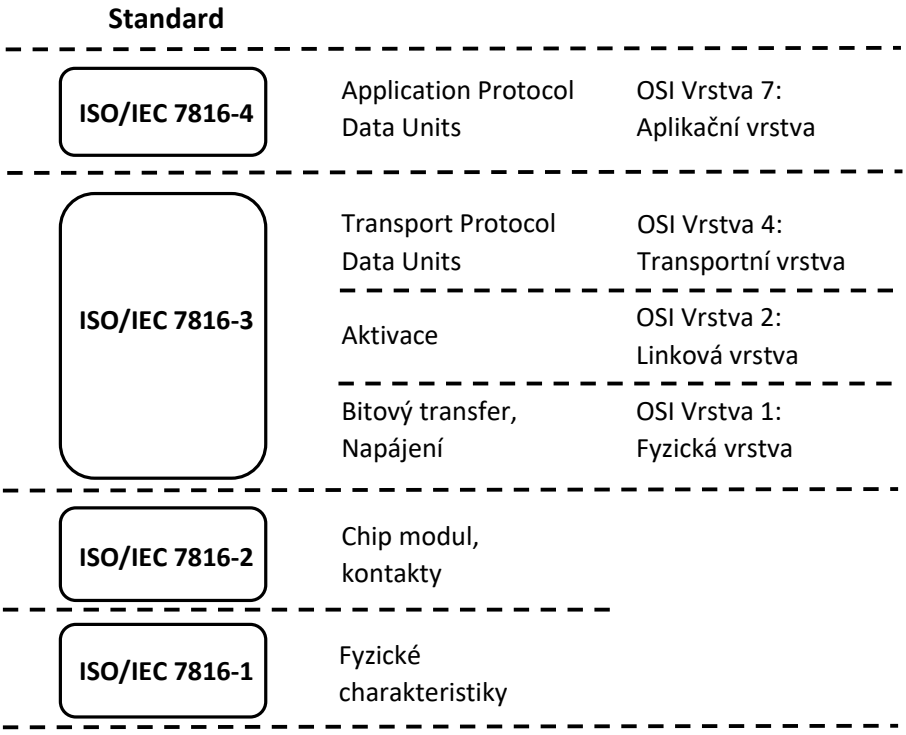
Vzhledem k tomu, že v této práci jsou řešeny autentizační protokoly, tak byly pro vývoj aplikací zvoleny multiaplikační programovatelné mikroprocesorové čipové karty s galvanickým rozhraním. Tudíž v dalších bodech této kapitoly a kapitol následujících je uvažováno pouze o tomto typu karet.

2.2 Komunikace s kartou

Komunikace mezi terminálem a kartou je založena na modelu klient–server, kde klient (terminál) posílá na kartu příkazy a server neboli mikroprocesor čipové karty odpovídá. Samotná komunikace probíhá pomocí zapouzdřování od protokolu nejvyšší aplikační vrstvy až k vrstvě fyzického rozhraní, podobně jako u vrstvového modelu ISO/OSI (Open Systems Interconnection) [12]. Vrstvový model je definován standardy ISO 7816 a je naznačen na obrázku 2.1.

Transportní protokol je definován standardem ISO/IEC 7816-3. Existuje více typů přenosových protokolů, ale pro účely této práce budou popsány pouze dva níže uvedené:

- T = 0 – bajtově orientovaný asynchronní poloduplexní sériový protokol. Jedná se o nejrozšířenější protokol s malými nároky na paměť (300 B).

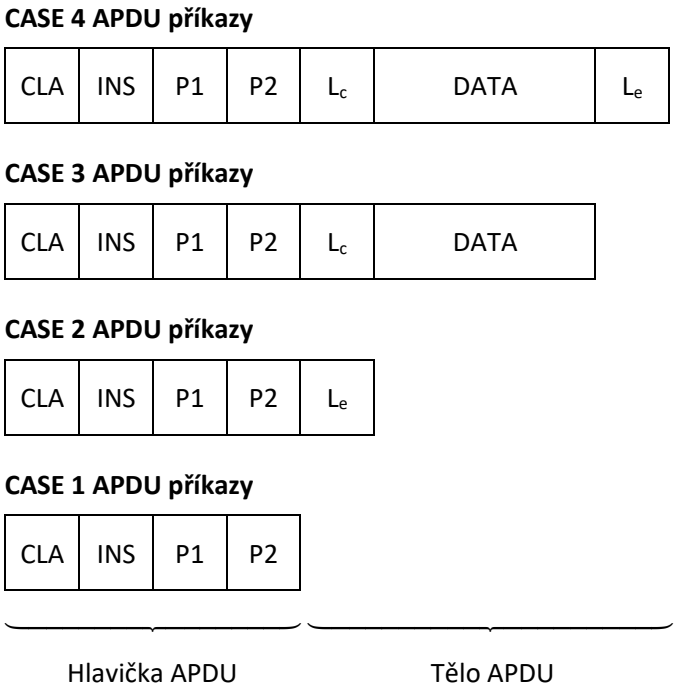


Obr. 2.1: Vrstvový model dle ISO/IEC 7816.

- T = 1 – blokově orientovaný asynchronní poloduplexní sériový protokol. Tento protokol pracuje s kódy pro detekci chyb. Mezi terminálem a kartou probíhá komunikace pomocí celých bloků, což zrychluje výslednou komunikaci, ale zároveň jsou zde kladeny vyšší nároky na paměť (1100 B).

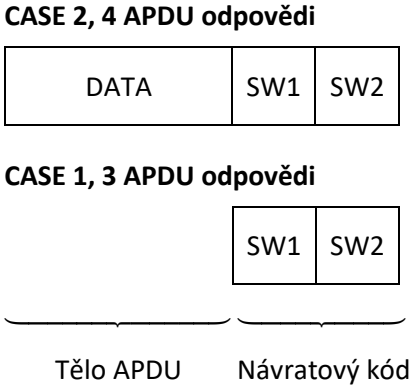
Tyto protokoly řeší pouze komunikaci na nižších vrstvách, protože ale karta neobsahuje vyrovnávací paměť, tak ovlivňují i logiku odesílání příkazů.

Instrukce jsou mezi terminálem a kartou zasílány pomocí tzv. APDU příkazů (Application Protocol Data Unit) a jejich struktura je definována ve standardu ISO/IEC 7816-4. APDU je základní aplikační protokol. Příkazy se označují jako APDU Commands a odpovědi jako APDU Responses. Existuje více typů příkazů, jejich struktura je popsána na obrázku 2.2 a zasílá je vždy terminál. Všechny typy obsahují povinnou 4B hlavičku, která se skládá ze čtyř polí (každé o velikosti 1 B). Pole CLA je třída (Class), která udává typ příkazu (např. proprietární `0x80`). Další pole INS (Instruction) je kód instrukce, určuje konkrétní příkaz ke spuštění (např. „autentizace“ `0x00`). Volitelné parametry P1 a P2 specifikují příkaz (např. „symetrický algoritmus“ `0x01`). Následuje volitelné tělo příkazu. Do pole L_c (Length command) se uvádí velikost dat, které jsou přenášeny v následujícím poli data. Řídící data dosahují velikosti až 255 B pro protokol T = 0 nebo až 65535 B u protokolu T = 1. L_e (Length expected) kóduje velikost dat očekávaných v odpovědi.



Obr. 2.2: Struktura APDU příkazu

Po zpracování přijatého příkazu nám čipová karta vrací odpověď, která se skládá z povinného 2B návratového kódu a nepovinné části, která obsahuje data (viz obrázek 2.3). Návratový kód je označen SW1 a SW2 (Status Word) a označuje výsledek zpracovaného příkazu (např. 90 00 značí úspěšné zpracování příkazu). [1]



Obr. 2.3: Struktura APDU odpovědi

[1] Úplný list APDU odpovědí: https://www.eftlab.com/knowledge-base/complete-list-of-apdu-responses/

3 Analýza a výběr nástrojů

3.1 Analýza platforem čipových karet

Implementaci zadaných kryptografických schémat lze provést na několika odlišných platformách programovatelných čipových karet jako jsou Java Card [13], MultOS [14], Basic Card, .NET Card [15]. Žádná z uvedených platforem není dokonalá, z hlediska jejich využití v kryptografii, tudíž úkolem bylo tyto platformy porovnat a najít vhodný kompromis. Porovnání jednotlivých platforem vychází z jejich podpory EC kryptografie viz tabulka 3.1 (informace převzaty z [16]). Bližší informace o podpoře kryptografických algoritmů a srovnání rychlostí jednotlivých algoritmů lze nalézt v [17].

Tab. 3.1: Podpora kryptografických operací na jednotlivých platformách.

Karta	OS	`ecAdd`	`ecMul`	`ecInv`	`mod`	`modMul`	`modExp`
.NET	.NET	✗	✗	✗	✗	✗	✓
J3A081	JC2.2.2	✓	✓	✗	✗	✗	✓
J3D081	JC3.0.1	✓	✓	✗	✗	✗	✓
Sm@rtCafe 6	JC3.0.1	✗	✗	✗	✗	✗	✓
Sm@rtCafe 5	JC2.2.2	✗	✗	✗	✗	✗	✓
BasicCard 7.6	Basic ZC7	✓	✓	✗	✓	✓	✓
Multos ML4	Multos	✓	✓	✓	✓	✓	✓
Multos ML3	Multos	✗	✗	✗	✓	✓	✓

Vznik platformy Java Card se datuje k roku 1996 a v dnešní době jde o jednu z nejrozšířenějších multiaplikačních platforem. Jedná se o otevřenou platformu poskytující běh aplikací psaných v programovacím jazyku Java Card, který je podmnožinou jazyka Java. Platforma Java Card nebyla zvolena pro vypracování praktické části této práce z toho důvodu, že ve většině verzí OS nepodporuje sčítání bodů na EC a v žádné verzi OS nepodporuje inverzi bodu na EC.

Platforma .NET Card staví na programovacím jazyku C#, jenž je podobný jazyku C a Java. Obsahují implementaci .NET Framework od společnosti Microsoft, avšak s určitými omezeními, které jsou dané mikroprocesorem čipové karty. V porovnání s jinými platformami mají .NET Card k dispozici větší velikost paměti RAM i EEPROM. Avšak jsou v dnešní době již zastaralé a neobsahují podporu pro EC kryptografii.

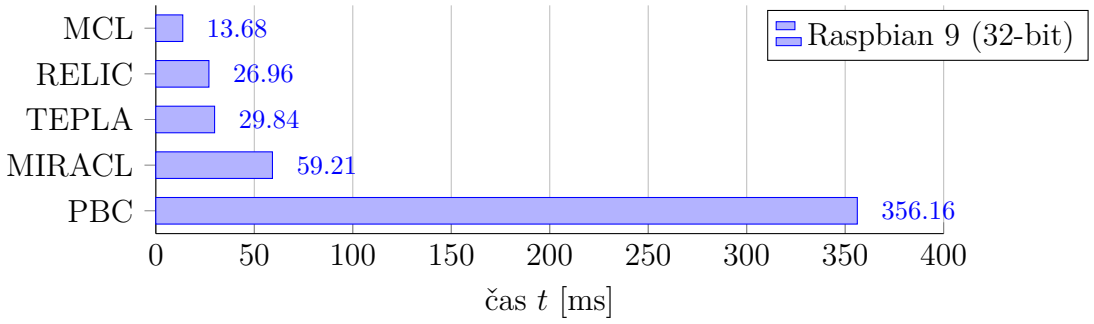
Lépe, než .NET Card jsou na tom Basic Card. Jedná se spíše o experimentální platformu s neověřenou bezpečností. Využívá programovacího jazyka BASIC. Z hlediska podpory EC kryptografie je na tom tato platforma lépe, než obě předcházející

ale v rychlosti provádění kryptografických operací je několikrát pomalejší. Z důvodů bezpečnosti a rychlosti nebyla ani tato platforma zvolena pro praktickou část.

Poslední srovnávanou platformou je MultOS. Programovacím jazykem je jazyk C a Multos assembler, což umožňuje vývojářům širší možnosti a lepší přístup k hardwaru. Momentálně se jedná o nejlepší možnou volbu pro aplikaci autentizačních protokolů na bázi EC. Obsahuje totiž širokou podporu operací na EC a ze všech uvedených platforem je při těchto operacích nejrychlejší. Další výhodou je její „prověřenost“, tzn. že za léta její existence je bezpečnost prověřena vývojáři. Na základě těchto argumentů byla tato platforma zvolena pro implementaci praktické části tohoto projektu.

3.2 Analýza a výběr kryptografických knihoven

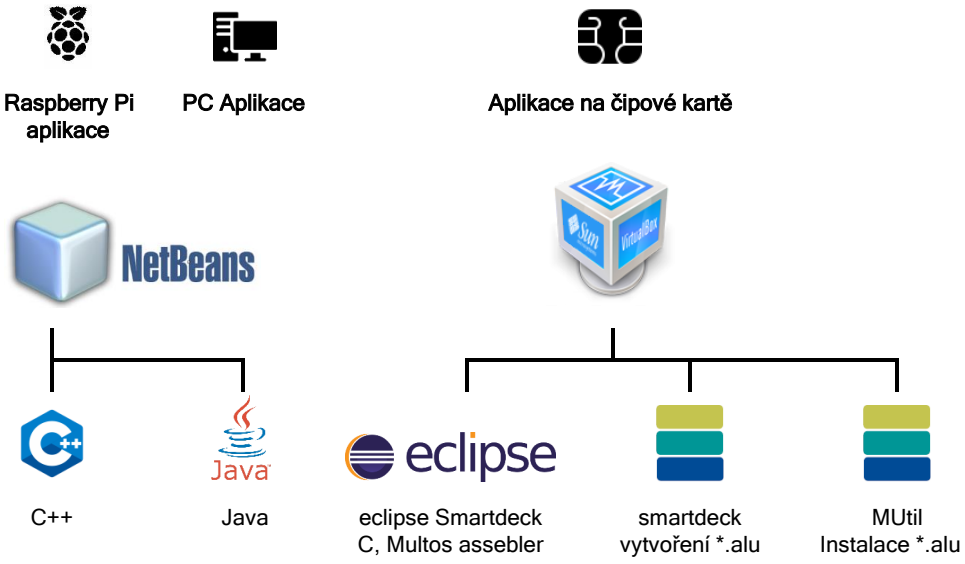
Pro zúžení výběru kryptografických knihoven byly stanoveny dvě základní podmínky. První podmínkou je podpora bilineárního párování a druhou je její dostupnost pro programovací jazyk C/C++. Důvodem pro jejich stanovení je nutnost využití bilineárního párování v rámci jednoho z kryptografických schémat a volba mikropočítače Raspberry Pi 2 Model B pro implementaci některých entit vystupujících v zadaných schématech. Existuje několik knihoven, které tyto podmínky splňují, např. PBC (Pairing Based Cryptography) [18], MIRACL (Multiprecision Integer and Rational Arithmetic Cryptographic Library) [19], TEPLA (University of Tsukuba Elliptic Curve and Pairing Library) [20], RELIC (Efficient LIbrary for Cryptography) [21] a MCL (portable and fast pairing-based Cryptography Library) [22]. Tyto knihovny byly podrobeny analýze výkonu při bilineárním párování. Dle výsledků analýzy, které jsou vyneseny do grafu na obrázku 3.1 (naměřené hodnoty převzaty z [16]), bylo rozhodnuto o využití knihovny MCL, jelikož dosahovala nejlepších časů při výpočtu operace bilineárního párování ze všech analyzovaných knihoven.



Obr. 3.1: Porovnání kryptografických knihoven z pohledu výkonu při bilineárním párování na 256 bitových EC.

4 Nástroje pro vývoj

K vytvoření funkční implementace každého schématu bylo nutné naprogramovat dvě aplikace běžící na rozdílných platformách. Každá z aplikací má na starost výpočty a komunikaci na straně jedné nebo několika entit vystupujících v zadaných schématech. V souladu se zadáním této práce byla jedna z aplikací vyvíjena a implementována na straně počítače s operačním systémem Windows nebo mikropočítače Raspberry Pi 2 Model B s operačním systémem Raspbian GNU/Linux 10. Tato aplikace byla napsána v programovacím jazyce Java nebo C++ (dle implementovaného schématu) a vyvíjena ve vývojovém prostředí Netbeans. Naopak druhá aplikace byla implementována na čipovou kartu. Na základě kapitoly 3.1 byla pro implementaci zadání na čipovou kartu zvolena platforma MultOS ML4 s operačním systémem Multos ve verzi 4.3.1. Tato aplikace byla vyvíjena ve vývojovém prostředí Eclipse, programovacím jazyce C a Multos assembler. Aby bylo možné vytvořenou aplikaci nahrát na kartu, bylo nutné využít nástrojů SmartDeck a MUtil. Pro názornost je výše popsané zobrazeno na obrázku 4.1.



Obr. 4.1: Schéma vývojových nástrojů

4.1 Netbeans

Aplikace lze v podstatě psát v jakémkoliv textovém editoru, který používá pro ukládání prostý text. Po instalaci příslušných nástrojů a kompilátorů může být proveden

překlad. Z důvodů efektivity však bylo zvoleno vývojové prostředí Netbeans. Aplikace na straně PC (Personal Computer) nebo Raspberry Pi byly vyvíjeny kompletně v tomto vývojovém prostředí.

Aby mohla aplikace uživatele vůbec komunikovat s čipovou kartou, musely být nejdříve importovány knihovny, které tuto komunikaci umožňují. Pro programovací jazyk Java byl využit balíček `javax.smartcardio`, který definuje Java API (Application Programming Interface) pro komunikaci s čipovými kartami využívající ISO/IEC 7816-4 APDU [23]. Ekvivalentem pro programovací jazyk C++ je `PCSC` [24].

4.2 Eclipse

Aplikace na straně čipové karty byly vyvíjeny ve vývojovém prostředí Eclipse, které ale nebylo použito ani pro ladění aplikace, ani pro samotné spuštění. Toto vývojové prostředí bylo ale použito pro tvorbu souboru *.hzx. S tímto souborem následně pracují nástroje viz kapitola 4.3.

4.3 SmartDeck a MUtil

Multos SmartDeck je používán ke kompilaci programového kódu vytvořeného pomocí nástrojů Eclipse. Pracuje se s ním za pomoci příkazového řádku. Zavoláním příkazu

```
halugen <nazev_souboru>.hzx
```

dojde k vytvoření ALU (Application Load Unit) souboru `<nazev_souboru>.alu`. Až tento soubor lze nahrát na kartu. Vyzhledem k tomu, že čipová karta neumožňuje dynamickou alokaci paměti za běhu programu, je nutno uvést velikost paměti relace už při nahrávání aplikace na kartu. Aby vývojář nemusel tuto hodnotu počítat vlastními silami, může využít příkazu, který požadovanou hodnotu vypíše.

```
hls -t <nazev_souboru>.hzx
```

Nyní je aplikace připravena k nahrání na čipovou kartu. K tomu je využíván program Mutil. V prvním kroku se vybírá cesta k nahrávanému .alu souboru. Následně je nutné vybrat správné AID (Application Identifier), což je ID (Identifier) aplikace zvolené při psaní kódu. V posledním kroku zbývá nastavit velikost relační paměti, zjištěnou v souladu s předchozím odstavcem. Pokud by nastavená velikost neodpovídala realitě, tak by byla při spuštění aplikace na kartě vrácena chyba `error 0x8010002f`. Stiskem tlačítka Load dojde k nahrání aplikace na čipovou kartu.

5 Základní schémata

Aby mohla být implementována zadaná pokročilejší schémata využívající atributového pověření, je nutné nejdříve definovat a popsat jejich základní stavební bloky. Jelikož pro tyto základní bloky byla provedena implementace v rámci semestrální práce, budou v této kapitole také prezentovány dosažené výsledky.

5.1 Podpisové schéma weak Boneh-Boyen

Prvním stavebním blokem je wBB (weak Boneh-Boyen) podpisové schéma, které lze použít k efektivnímu podepisování zpráv [6]. Toto schéma (označované jako Σ) v základu obsahuje tři algoritmy `Setup`, `Sign` a `Verify`. Lze jej lehce rozšířit o důkaz znalosti PK (Proof of Knowledge) podpisu. Po tomto rozšíření je jeho zásadní vlastností to, že umožňuje podpis, který uživatel získá od vydavatele podepsáním zprávy, randomizovat. Slovo randomizovat by se dalo popsat tak, že je získaný podpis pozměněn takovým způsobem, aby nebyl spojitelný s originálním podpisem, ale zůstal stále platný. Tato vlastnost je velmi podstatná pro zajištění ochrany soukromí, protože je možné prokázat znalost podpisu bez jeho zveřejnění. Především tedy zajišťuje anonymitu, nesledovatelnost a nespojitelnost. Pro důkaz znalosti podpisu bez jeho zveřejnění je nutné základní schéma rozšířit o algoritmy `Prove` a `VerifyRandomized`. Bližší pohled na jednotlivé algoritmy nabízí [25], zde budou popsány pouze vybrané algoritmy, které budou následně součástí složitějších implementací:

$(params, sk, pk) \leftarrow \Sigma.\texttt{Setup}(1^\kappa)$:

- vstupem je bezpečnostní parametr κ, výstupem je soukromý klíč $sk \in_R \mathbb{Z}_q$, veřejný klíč $pk = g_2^{sk}$ a parametry systému $params = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, g_1, g_2, \mathbf{e})$.

$(\sigma) \leftarrow \Sigma.\texttt{Sign}(sk, m)$:

- algoritmus počítá podpis $\sigma = g_1^{\frac{1}{x+m}}$ na základě vstupních parametrů $m \in \mathbb{Z}_q$ a soukromého klíče sk.

$(0/1) \leftarrow \Sigma.\texttt{VerifyRandomized}(\bar{\sigma}, \hat{\sigma}, \pi, m, pk)$:

- ověřovatel prověří důkaz znalosti $\pi = (e, s_m, s_r)$ a dále ověřuje bilineární párování $\mathbf{e}(\bar{\sigma}, g_2) \stackrel{?}{=} \mathbf{e}(\hat{\sigma}, pk)$.
- Pro účely implementace základních schémat (kapitola 5.3) musel být tento algoritmus modifikován a bilineární párování bylo změněno (není podporováno na SC) na ověření $e \stackrel{?}{=} \mathcal{H}(\hat{\sigma}, t', nonce)$.

5.2 Algebraický MAC využívající wBB podpis

Na základě wBB podpisu (viz 5.1) může být sestaven takzvaný algebraický MAC (Message Authentication Code) [25]. Jedná se o symetrický algoritmus. To znamená, že dvě různé entity (vydavatel a ověřovatel) využívají shodný tajný klíč. Tento algoritmus je schopen zajistit autentičnost a integritu bloku zpráv. Algebraický MAC je založen na grupových operacích namísto běžného použití blokových šifer a hašovacích funkcí. Toto schéma (označováno jako $\texttt{MAC}_{\texttt{wBB}}$) obsahuje opět tři základní algoritmy `Setup`, `Sign` a `Verify`. Shodně jako u wBB podpisu lze prokázat znalost autentizačního kódu MAC. K tomu slouží algoritmy `Prove` a `VerifyRandomized`. Níže jsou specifikovány jednotlivé algoritmy:

$(params, sk) \leftarrow \texttt{MAC}_{\texttt{wBB}}.\texttt{Setup}(1^{\kappa})$:

- vstupem je bezpečnostní parametr κ, na výstupu jsou soukromé klíče $sk = (x_0, x_1, \ldots, x_n) \in_R \mathbb{Z}_q$ a parametry systému $params = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}, g_1, g_2, \mathbf{e})$.

$(\sigma, \sigma_{x_0}, \ldots, \sigma_{x_n}) \leftarrow \texttt{MAC}_{\texttt{wBB}}.\texttt{Sign}(sk, (m_1, \ldots, m_n))$:

- algoritmus dostává na vstup množinu zpráv $(m_1, \ldots, m_n) \in \mathbb{Z}_q$ a soukromé klíče sk. Následně je vypočten podpis $\sigma = g_1^{\frac{1}{x_0 + m_1 x_1 + \cdots + m_n x_n}}$ a pomocné hodnoty $(\sigma_{x_0}, \ldots, \sigma_{x_n}) = (\sigma^{x_0}, \ldots, \sigma^{x_n})$.

$(0/1) \leftarrow \texttt{MAC}_{\texttt{wBB}}.\texttt{Verify}(sk, (\sigma, \sigma_{x_0}, \ldots, \sigma_{x_n}), (m_1, \ldots, m_n))$:

- algoritmus vyhodnocuje platnost podpisu σ na základě zpráv $(m_1, \ldots, m_n)$ a soukromého klíče sk. Vrací hodnotu 1 v případě že platí $g = \sigma^{x_0 + m_1 x_1 + \cdots + m_n x_n}$.

$(\hat{\sigma}, \hat{\sigma}_{x_0}, \ldots, \hat{\sigma}_{x_n}, \pi) \leftarrow \texttt{MAC}_{\texttt{wBB}}.\texttt{Prove}((m_1, \ldots, m_n), (\sigma, \sigma_{x_0}, \ldots, \sigma_{x_n}))$:

- algoritmus volí randomizační element $r \in \mathbb{Z}_q$ a vypočítá randomizovaný podpis $\hat{\sigma} = \sigma^r$, pomocné hodnoty $(\hat{\sigma}_{x_0}, \ldots, \hat{\sigma}_{x_n}) = (\sigma_{x_0}^r, \ldots, \sigma_{x_n}^r)$ a je sestrojen důkaz znalosti π.

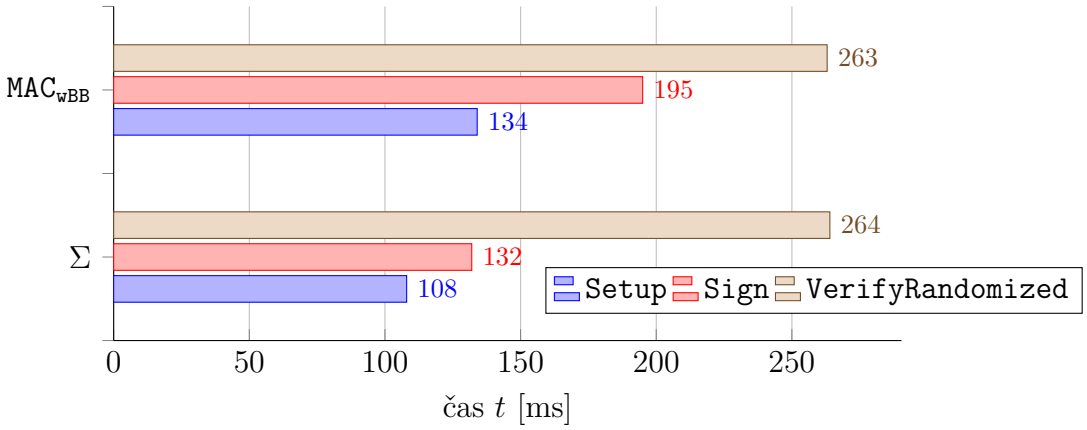
$(0/1) \leftarrow \texttt{MAC}_{\texttt{wBB}}.\texttt{VerifyRandomized}(\hat{\sigma}, \hat{\sigma}_{x_0}, \ldots, \hat{\sigma}_{x_n}, \pi, (m_1, \ldots, m_n))$:

- algoritmus ověřuje důkaz znalosti $\pi = (e, s_{m_i}, s_r)$ a dále ověřuje bilineární párování pro všechny pomocné hodnoty $\mathbf{e}(\hat{\sigma}_{x_i}, g_2) \stackrel{?}{=} \mathbf{e}(\hat{\sigma}, X_i)$.
- Zde musel být tento algoritmus modifikován za účelem implementace základních schémat (kapitola 5.3). Bilineární párování bylo změněno na ověření $e \stackrel{?}{=} \mathcal{H}(\hat{\sigma}, t', nonce)$.

5.3 Implementace základních schémat

Jak bylo zmíněno v úvodu této kapitoly, byla provedena implementace těchto schémat v rámci semestrální práce. Ve schématech vystupují tři entity (uživatel, vydava-

tel a ověřovatel). Aby měla následná měření časů vybraných algoritmů vypovídající hodnotu (z pohledu zařízení s omezeným výkonem), bylo rozhodnuto o implementování entit vydavatele a ověřovatele na programovatelnou čipovou kartu využívající platformu Multos ML4. Entita uživatele byla implementovaná na počítači s operačním systém Microsoft Windows využívající platformu Java. Pro každé schéma byly naprogramovány dvě aplikace, jedna pro čipovou kartu a druhá pro počítač. Detailní popis implementace není součástí této práce, avšak výsledky měření rychlostí jednotlivých implementovaných algoritmů jsou na obrázku 5.1. V implementaci schématu $\texttt{MAC}_{\texttt{wBB}}$ byla množina zpráv zredukována pouze na jednu zprávu m_1. Z naměřených hodnot u algoritmu `Sign` lze odhadnout, jak rychle provádí SC násobení bodu EC. Právě o tuto operaci se liší jednotlivé implementace (navíc je zde ještě operace modulární násobení). U algoritmu `VerifyRandomized` se provádí totožné operace, čemuž odpovídají naměřené hodnoty. Sluší se také uvést, že obě implementace pracovaly se 192b EC.



Obr. 5.1: Časová náročnost vybraných algoritmů pro základní schémata.

6 Schéma využívající anonymní atributové pověření – KVAC

Popisované schéma, využívající anonymní atributové pověření, s označením KVAC (Keyed-Verification Anonymous Credentials) je detailně popsáno v [25] a bylo zadáno školitelem k implementaci. Disponuje níže uvedenými vlastnostmi na ochranu soukromí:

- **Anonymita** – uživatel je schopen prokázat držení požadované vlastnosti (atributu) bez odhalení své identity.
- **Nespojitelnost** – každá relace ověřovacího protokolu je unikátní a nespojitelná s předchozí relací. Nelze určit, zdali zkoumané relace patří jednomu či více různým uživatelům.
- **Nesledovatelnost** – každé vydavatelem vydané pověření je randomizováno. Z čehož vyplývá, že ani vydavatel si nemůže spojit konkrétní přístupy a identifikovat uživatele systému.
- **Selektivní odhalení atributů** – uživatel má kontrolu nad tím, které atributy během ověřování odhalí.

Velkou předností tohoto schématu je, že jsou na straně uživatele eliminovány složitější kryptografické operace, které nebývají v současné době podporovány na zařízeních s omezeným výkonem jako jsou čipové karty. Příkladem takovýchto operací může být bilineární párování.

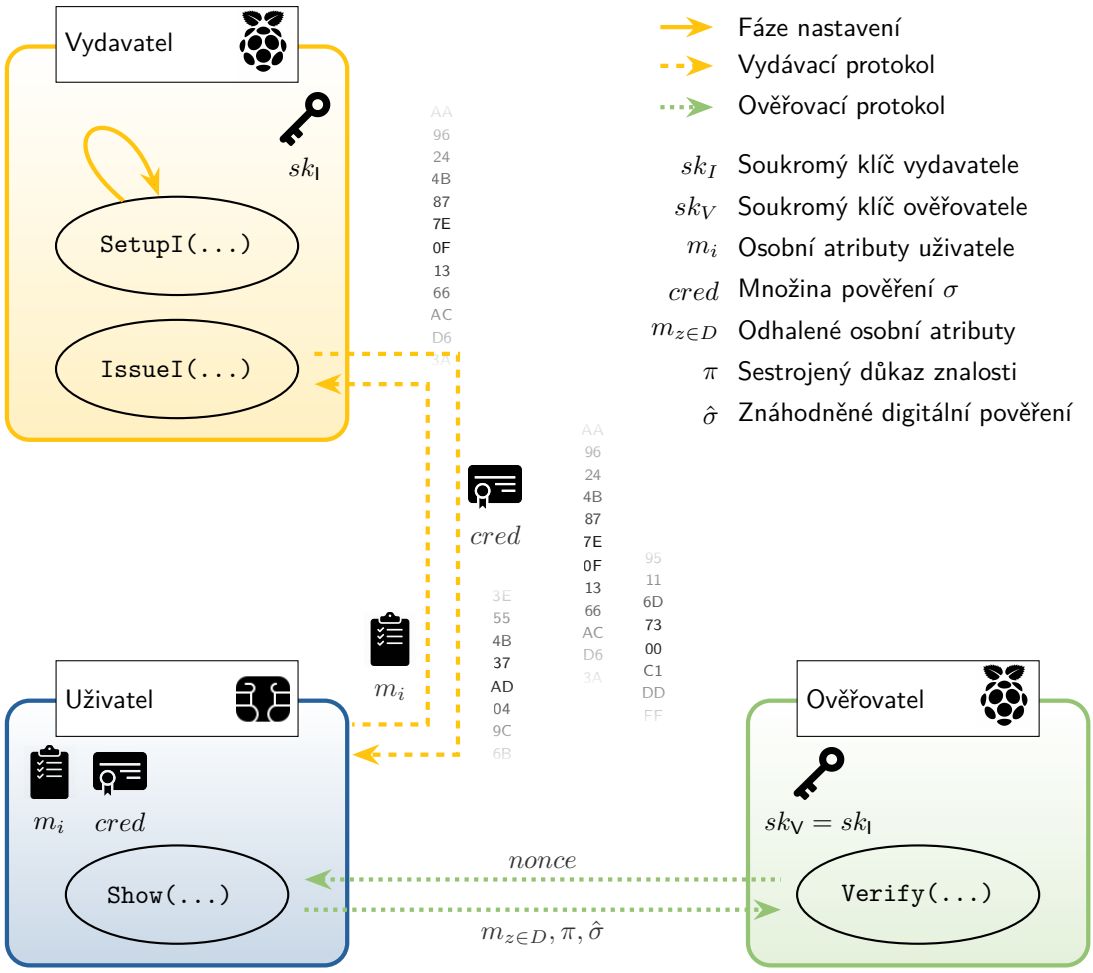
6.1 Obecná architektura KVAC

Konstrukce popisovaného KVAC schématu je postavena na algebraickém MAC (označeno jako $\mathtt{MAC}_{\mathtt{wBB}}$), který je popsán v kapitole 5.2, a na důkazech nulové znalosti. Ve srovnání s tradičními schématy, využívající anonymního atributového pověření (ABC schémata), musí ověřovatel znát tajné soukromé klíče vydavatele. V opačném případě by nebyl schopen ověřit atributy uživatele. Což předurčuje jeho použití v systémech, kde jsou vydavatelé atributů stejné entity jako ověřovatelé. Struktura KVAC schématu je v souladu s obecným schématem atributové autentizace a obsahuje níže uvedené entity:

- **Uživatel** ($\mathcal{U}$): je držitelem atributů (vlastností), které prokazuje ověřovateli.
- **Vydavatel** ($\mathcal{I}$): je důvěryhodná autorita, která jako jediná může verifikovat uživatelské atributy a generuje parametry systému.
- **Ověřovatel** ($\mathcal{V}$): ověřuje držení konkrétních atributů uživatelem při požadavku o službu nebo o přístup.

Uvedené entity v rámci systému interagují pomocí specifických kryptografických protokolů a algoritmů (viz obrázek 6.1). V rámci implementace KVAC schématu se jedná konkrétně o tři základní protokoly (fáze), které již obsahují konkrétní kryptografické algoritmy:

- **Nastavení** – v této fázi dochází ke generování parametrů systému, nastavení soukromých a jejich distribuci. Obsahuje kryptografický algoritmus `SetupI`.
- **Vydávací protokol** – protokol zajišťuje vydání digitálního pověření. Opět se zde vyskytuje kryptografický algoritmus, tentokrát je pojmenovaný `IssueI`.
- **Ověřovací protokol** – protokol umožňující prokázání držení atributů uživatele vůči ověřovateli. Jeho součástí jsou algoritmy `Show` a `Verify`, které spolu navzájem komunikují. V případě validního ověření je obvykle uživateli povolen přístup.



Obr. 6.1: Architektura Keyed-Verification Anonymous Credentials.

6.2 Popis protokolů a kryptografických algoritmů

V této podkapitole jsou detailněji specifikovány a popsány jednotlivé protokoly a kryptografické algoritmy KVAC včetně vstupních a výstupních proměnných.

6.2.1 Nastavení

V první fázi je nezbytné nastavit parametry systému označované jako $params_{\mathrm{I}}$, které obsahují cyklickou grupu $\mathbb{G}$, bod eliptické křivky g_1 včetně jeho řádu q. Tyto parametry budou použity v celé implementaci a jsou generovány algoritmem `SetupI`. Dále tento algoritmus vygeneruje množinu klíčů vydavatele sk_{I}. Algoritmus spouští vydavatel. Tento algoritmus odpovídá algoritmu `setup` u $\mathtt{MAC}_{\mathtt{wBB}}$ viz kapitola 5.2.

$(sk_{\mathrm{I}}, params_{\mathrm{I}}) \leftarrow \mathtt{SetupI}(1^{\kappa})$:

- vstupem je bezpečnostní parametr κ,
- výstupem algoritmu jsou soukromé klíče $sk_{\mathrm{I}} = (x_0, x_1, \ldots, x_n) \in_R \mathbb{Z}_q$ a parametry systému $params_{\mathrm{I}} = (q, \mathbb{G}, g_1)$.

6.2.2 Vydávací protokol

Vydávací protokol zahrnuje kryptografický algoritmus `IssueI`, který běží mezi vydavatelem a uživatelem. U tohoto algoritmu je analogie s algoritmem `sign` u $\mathtt{MAC}_{\mathtt{wBB}}$ (viz kapitola 5.2). Vstupem je soukromý klíč vydavatele sk_{I} a parametry systému $params_{\mathrm{I}}$. Uživatel zasílá své osobní atributy, pro které chce získat digitální pověření. Vydavatel vypočítá množinu digitálních pověření, kterou zasílá nazpět uživateli. Detaily algoritmu zobrazeny na obrázku 7.3. Obrázek je společný pro KVAC i schéma RKVAC, jenž je popisováno v další kapitole.

$(cred) \leftarrow \mathtt{IssueI}(params_{\mathrm{I}}, sk_{\mathrm{I}}, (m_1, \ldots, m_n))$:

- uživatel odesílá vydavateli své atributy $(m_1, \ldots, m_n)$, na jejichž základě vydavatel vydává množinu pověření: $cred = (\sigma, \sigma_{x_0}, \sigma_{x_1}, \ldots, \sigma_{x_n})$.

6.2.3 Ověřovací protokol

V rámci procesu ověření uživatel odhaluje vůči autoritě ověřovatele některé své atributy, jiné naopak zůstávají skryté. Schopností a účelem ověřovacího protokolu je spolehlivě ověřit, zda tyto odhalené atributy jsou správné, při splnění základních požadavků na ochranu soukromí. Komunikace v rámci protokolu probíhá mezi uživatelem a ověřovatelem.

Entita uživatele využívá kryptografického algoritmu `Show`. Na vstupu má parametry systému $params_{\mathrm{I}}$, množinu osobních atributů $(m_1, \ldots, m_n)$ a jejich pověření

cred. Dále od ověřovatele obdrží náhodnou hodnotu *nonce*. V dalším kroku generuje několik pomocných hodnot, které slouží k vystavení znáhodněného digitálního pověření $\hat{\sigma}$ a sestrojení důkazů znalosti π. Znáhodněné digitální pověření a důkaz znalosti zasílá zpět ověřovateli včetně odhalených atributů $m_{z \notin D}$.

Ověřovatel využívá algoritmu `Verify` na jehož vstup přichází hodnoty od uživatele. Kromě parametrů systému je dalším vstupem množina klíčů vydavatele $sk_{\mathrm{V}} = sk_{\mathrm{I}}$. Ověřovatel prověří platnost důkazu znalosti. V případě pozitivního výsledku jsou odhalené atributy považovány za validní. Kompletní notace protokolu je zobrazena na obrázku 7.4.

$[(\hat{\sigma}, \pi) \leftarrow \texttt{Show}(params_{\mathrm{I}}, (m_1, \ldots, m_n), cred)$

$\leftrightarrow \texttt{Verify}(params_{\mathrm{I}}, sk_{\mathrm{I}}, m_{z \in D}, \hat{\sigma}, \pi) \rightarrow (0/1)] :$

- uživatel randomizuje své pověření a sestavuje důkaz znalosti: $(\hat{\sigma}, \pi)$,
- ověřovatel prověří zaslaný důkaz $\pi = (e, s_{m \notin D}, s_v)$.

6.3 Implementace KVAC

V rámci realizace KVAC je entita uživatele implementována na programovatelnou čipovou kartu využívající platformu Multos ML4. Vzhledem k možnostem využití tohoto schématu bylo pro zbylou dvojici entit systému rozhodnuto o použití platformy Raspbian GNU/Linux běžící na mikropočítači Raspberry Pi. Uvedené platformy byly zvoleny na základě analýzy provedené v kapitole 3 a bližší informace jsou k nalezení v kapitole 4.

6.3.1 Zadané a zvolené parametry

Implementované schéma pracuje s eliptickými křivkami definovanými přes konečná pole $\mathbb{F}_p$. Tyto křivky jsou obvykle reprezentovány v krátké Weierstrassově formě s pomocí doménových parametrů $(a, b, p, G = (x_g, y_g), q, h$, kde $a, b \in \mathbb{F}_p$, $G \in E(\mathbb{F}_p)$, q je řád bodu G a h je kofaktor. Doménové parametry nebyly generovány, ale jsou dány křivkou BN254 z MCL knihovny [22]. Jejich vyjádření v hexadecimální podobě je zobrazeno ve výpisu 6.1.

Jako hašovací funkce byla zvolena SHA1 (Secure Hash Algorithm), jejímž výstupem je 160 bitů dlouhý otisk zprávy. Použití této funkce je v současné době omezováno (i když je považována stále za bezpečnou) na úkor bezpečnějších hašovacích funkcí. O použití této funkce bylo rozhodnuto kvůli její podpoře na čipové kartě (výpočty lze akcelerovat na kryptografickém procesoru).

Výpis 6.1: Doménové parametry EC.

```
SETECdom ecDomain = {
   0x00, // Format domenovych parametru
   0x20, // Delka prvecisla p v bytech
   0x25, 0x23, 0x64, 0x82, 0x40, 0x00, 0x00, 0x01, 0xba, 0x34, 0x4d,
   0x80, 0x00, 0x00, 0x00, 0x08, 0x61, 0x21, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x13, 0xa7, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x13, // p
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // a
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, // b
   0x25, 0x23, 0x64, 0x82, 0x40, 0x00, 0x00, 0x01, 0xBA, 0x34, 0x4D,
   0x80, 0x00, 0x00, 0x00, 0x08, 0x61, 0x21, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x13, 0xA7, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x12, //x_g
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, //y_g
   0x25, 0x23, 0x64, 0x82, 0x40, 0x00, 0x00, 0x01, 0xba, 0x34, 0x4d,
   0x80, 0x00, 0x00, 0x00, 0x07, 0xff, 0x9f, 0x80, 0x00, 0x00, 0x00,
   0x00, 0x10, 0xa1, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0d, // q
   0x01 // h
}; // mcl BN254
```

6.3.2 Vytvořené aplikace a knihovny

V souladu se zadáním práce a architekturou tohoto schématu využívajícího anonymního atributového pověření byly naprogramovány dvě aplikace. Jako první bude popsána aplikace KBCT_eloyalty obsluhující entitu uživatele, která běží na platformě Multos ML4 a je naprogramována v jazyce C a Multos assembler. Zdrojový kód této aplikace je uložen v souboru `eloyalty.c`. V tomto zdrojovém kódu musí být nadefinován hexadecimální identifikátor aplikace, který slouží pro zavolání aplikace pomocí APDU příkazu. Pro korektní průběh kompilace aplikace je nezbytný hlavičkový soubor `scAppSetup.h`, ve kterém jsou pomocí maker nadefinovány konstanty zásadní pro běh celé aplikace. Mezi tyto konstanty patří číselné kódy instrukcí, návratové kódy APDU příkazů a bajtové konstanty definující velikost proměnných ve zdrojovém kódu. K provádění operací na čipové kartě je efektivní (pro kryptografické operace přímo nezbytné) psát instrukce v Multos assembler kódu. Z důvodu zachování čitelnosti kódu programu je vhodné si často se opakující instrukce nadefinovat pomocí maker. Příklad takové definice je uveden ve výpisu 6.2. Právě pro tyto definice byl napsán druhý nezbytný hlavičkový soubor `multosecc.h`.

Výpis 6.2: Definice makra pro sčítání bodů EC.

```
#define EcAddition(EcdomainParameters, EcPoint1, EcPoint2, Result) \
do {                                                               \
 __push(__typechk(unsigned char *, EcdomainParameters));           \
 __push(__typechk(unsigned char *, EcPoint1));                     \
 __push(__typechk(unsigned char *, EcPoint2));                     \
 __push(__typechk(unsigned char *, Result));                       \
 __code(PRIM, 0xd0);                                               \
} while (0)
```

V hlavním zdrojovém kódu jsou pro lepší orientaci a přístup nadefinovány datové struktury sjednocují množiny parametrů systému jako například množina pověření. Dále jsou zde definované proměnné aplikace, které jsou rozděleny do skupin dle umístění v jednotlivých pamětech čipové karty (veřejná paměť RAM, RAM relace, EEPROM). Aplikace je za běhu ovládána pomocí APDU příkazů a kódů jednotlivých instrukcí.

Druhá aplikace zvaná KBCT_KVAC zabezpečuje funkce pro zbývající dvě entity systému (vydavatele a ověřovatele). Je psána v jazyce C++ a je určena pro platformu Raspbian GNU/Linux. Hlavní část kódu aplikace se nachází v souboru `main.cpp`. Jsou zde opět nadefinovány číselné konstanty (např. instrukce pro obsluhu čipové karty), datové struktury, proměnné a všechny důležité funkce určené k obsluze dílčích částí programu. Obsahuje také nezbytnou funkci `main`. Aby mohla aplikace jako celek komunikovat s čipovou kartu, tak musely být naprogramovány dílčí funkce zajišťující tuto funkcionalitu. Právě tyto důležité funkce obsahuje soubor `pcscRem.cpp`, který je do hlavního kódu zahrnut pomocí hlavičkového souboru `pcscRem.h`.

Vybrané funkce a metody

Aplikace KBCT_eloyalty obsahuje kromě funkce `main` pouze jednu další nadefinovanou funkci `getRandom`, která slouží ke generování pseudonáhodných čísel o délce 24 nebo 32 bajtů $\in \mathbb{Z}_q$ (délka generovaného čísla závisí na definici konstanty velikosti EC křivky). Tato pseudonáhodná čísla jsou hlavně využita pro randomizační elementy ρ. Ve zbytku kódu jsou hojně využívána již nadefinovaná makra jako např. `ModularReduction` viz výpis 6.3.

Aplikace KBCT_KVAC naopak používá velké množství nadefinovaných funkcí. V rámci rozsahu této práce není možné popsat tyto funkce detailně a kompletně všechny. Z tohoto důvodu je výběr autorem zúžen na několik nejdůležitějších funkcí:

- Sada 4 základních funkcí `byteArrayToMclECpoint`, `byteArrayToMclBigint`, `mclECpointToByteArray`, `mclBigintToByteArray` zajišťuje odstranění nekompatibility mezi platformami při výměně dat v digitální podobě a jejich interpretaci. Platforma Multos reprezentuje data v syrové formě v polích bajtů a

používá kódování big-endian. Naopak knihovna MCL využívá vlastní definované datové struktury, a navíc v kódování little-endian. Právě pro převod mezi formáty slouží tyto definované funkce.

- Většina níže uvedených funkcí musí komunikovat s čipovou kartou. K tomu slouží APDU příkazy. Pro tvorbu těchto APDU příkazů byla napsána funkce `CreateAPDUComm`.
- Vytvořené APDU příkazy je nutné následně odesílat na čipovou kartu a také přijmout APDU odpovědi. Za tímto účelem byla napsána funkce `SendAPDUComm`.
- Pro prvotní nastavení čipové karty slouží funkce `setAttributestoSC`, jejímž jediným účelem je nahrát uživateli na čipovou kartu množinu atributů. Maximální počet atributů je v rámci této implementace omezen na 10.
- Funkce zvaná `signI` v rámci schématu spadá do vydávacího protokolu. Počítá množinu digitálních pověření *cred* a zasílá je na čipovou kartu. Zastává tedy funkce kryptografického algoritmu `IssueI`.
- Nezbytnou funkcí je `proveI`, která zajišťuje největší množství výpočetních operací na platformě Raspbian GNU/Linux. Pro entitu ověřovatele zajišťuje kryptografický algoritmus `Verify`. Výstupem funkce je hodnota reprezentována logickým datovým typem boolean, tedy pravda/nepravda.

V rámci aplikace KBCT_KVAC jsou používány navíc funkce pro připojení a odpojení terminálu čipových karet a také funkce pro měření času algoritmů.

Výpis 6.3: Funkce pro generování pseudonáhodných číslel.

```
void getRandom(BYTE *firstOutputAddress, BYTE _numberOfAttributes,
        unsigned int offset){
    BYTE cntr = 0;
    //_numberOfAttributes++;
    while (cntr<_numberOfAttributes){
        GetRndNumber(firstOutputAddress);
        cntr++;
        ModularReduction(EC_SIZE, EC_SIZE, firstOutputAddress,
                (BYTE*) &ecDomain.N);
        firstOutputAddress = firstOutputAddress + offset;
        }
}
```

6.3.3 APDU komunikace

V této podkapitole je popsáno časový sled zasílání APDU příkazů a také je zde přiblížena jejich struktura a obsah včetně očekávaných APDU odpovědí. Tyto příkazy a odpovědi jsou zde reprezentovány slovním označením instrukce. Toto označení je také používáno ve zdrojovém kódu a pomocí definice konstant jsou jim přiřazeny číselné hodnoty.

Na obrázku 6.2 je naznačen vývojový diagram úplně prvního kroku, kde dochází ke spuštění aplikací na obou zařízeních a následně k odeslání atributů uživateli. Pro tento krok jsou využity instrukce:

- `INS_SELECT` – Výběr KBCT_KVAC aplikace na čipové kartě pomocí AID aplikace. Očekávaná odpověď je 90 00.

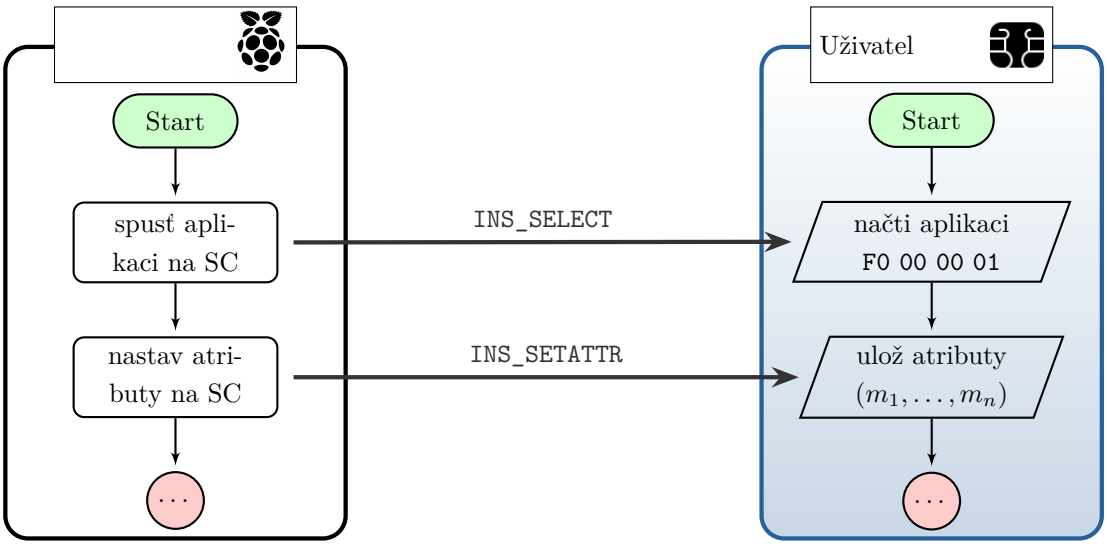
CLA	**INS**	**P1**	**P2**	**Lc**	**Data**
0x00	0xA4	0x04	0x00	0x04	0xF0 0x00 0x00 0x01

W1	**W2**
0x90	0x00

- `INS_SETATTR` – Odeslání atributů $m_1, \ldots, m_n$ na čipovou kartu. Proměnná *no* představuje celkový počet atributů n. Hodnota offset je v prvním kroku rovna 0x00. V případě, že je počet atributů $n > 7$, probíhá druhý krok a hodnota $offset$ se inkrementuje o 7. Hodnotu proměnné *size* je možné spočítat jako $(no - offset) \cdot EC_{SIZE}$, kde EC_{SIZE} reprezentuje velikost eliptické křivky v bajtech. Očekávaná odpověď je 90 00.

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**
0x80	0x00	no	offset	size	$(m_1, \ldots, m_n)$

W1	**W2**
0x90	0x00



Obr. 6.2: Načtení aplikace a uložení atributů – vývojový diagram.

Nyní lze vydat uživateli potřebná pověření. Vše je vyobrazeno ve vývojovém diagramu na obrázku 6.3 a pro komunikaci jsou využity následující instrukce:

- `INS_INIT` – návrat osobních atributů $(m_1,\ldots,m_n)$. Hodnota proměnné $offset$ vždy začíná na 0x00. Pokud je atributů méně či rovno sedmi, tak celá komunikace proběhne pomocí jednoho APDU příkazu. V případě více atributů je hodnota $offset$ nastavena na 0x07. Očekávaná velikost příchozích dat $size$ je dána počtem atributů a jejich velikostí.

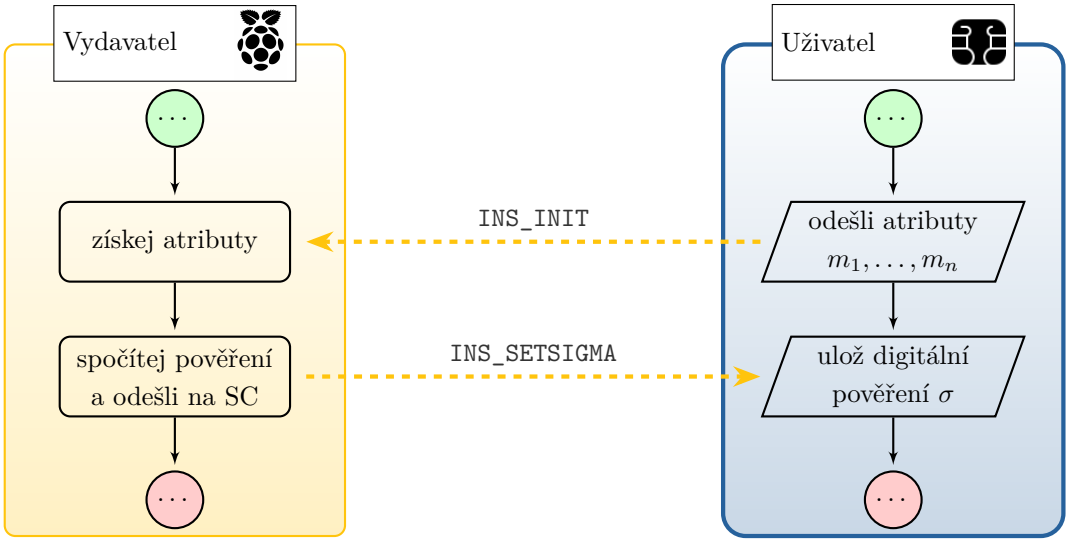
CLA	**INS**	**P1**	**P2**	**Le**
0x80	0x05	offset	0x00	size

Data	**W1**	**W2**
$(m_1,\ldots,m_n)$	0x90	0x00

- `INS_SETSIGMA` – odeslání pověření σ, $\sigma_{x_1},\ldots,\sigma_{x_n}$ na čipovou kartu. Proměnná no představuje počet vystavených pověření σ, tedy celkový počet atributů $n+1$. Hodnota $offset$ vždy začíná na 0x00. Pokud je počet pověření σ menší či roven třem, celá komunikace proběhne pomocí jednoho APDU příkazu. V případě více pověření je hodnota $offset$ inkrementována o tři. Velikost $size$ je dána součtem velikostí jednotlivých přenášených parametrů.

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**
0x80	0x01	no	offset	size	$\sigma \mid (\sigma_{x_1},\ldots,\sigma_{x_n})$

W1	**W2**
0x90	0x00



Obr. 6.3: Vydávací protokol KVAC (algoritmus `IssueI`) – vývojový diagram.

Uživatel má v této fázi téměř vše potřebné pro vytvoření důkazu znalosti. Poslední data na čipovou kartu odesílá autorita, která si žádá ověření atributů. Těmito daty je myšleno náhodné číslo *nonce*. V rámci této implementace nebylo ošetřeno odhalování konkrétních atributů, ale ověřovatel zasílá v rámci APDU komunikace pouze to, kolik atributů má zůstat skrytých. Není zde také řešeno, zdali má ověřovatel právo požadovat odhalení zbylých atributů. V tomto kroku již probíhají kryptografické výpočty i na čipové kartě a lze ho prohlásit za časově nejnáročnější. Jakmile karta dokončí výpočty znáhodněného digitálního pověření $\hat{\sigma}$ a sestrojí důkaz znalosti π, odesílá tyto údaje ověřovateli. V rámci této implementace nejsou odhalené atributy zasílány zpětným kanálem ověřovateli, ale je uvažováno, že se jedná o atributy veřejné. Ověřovatel následně provede ověření těchto údajů a vyhodnotí, zdali je uživatel oprávněným držitelem odhalovaných atributů, a tedy zda je ověřen či nikoli. Vzájemná komunikace je vykreslena do vývojového diagramu na obrázku 6.4 a lze ji popsat následujícími instrukcemi:

- `INS_GETPROVE` – odeslání *nonce* na čipovou kartu. Proměnná *no* představuje počet skrytých atributů $m_{z \notin D}$. V APDU odpovědi se očekává návrat $\hat{\sigma}$, e, s_v, $s_{m_{z \notin D}}$, kde $z \in \langle 0, 4 \rangle$. V případě, že je počet skrytých atributů menší nebo roven čtyřem, není již zasílána následující instrukce. Velikost *size* je dána součtem velikostí jednotlivých přenášených parametrů.

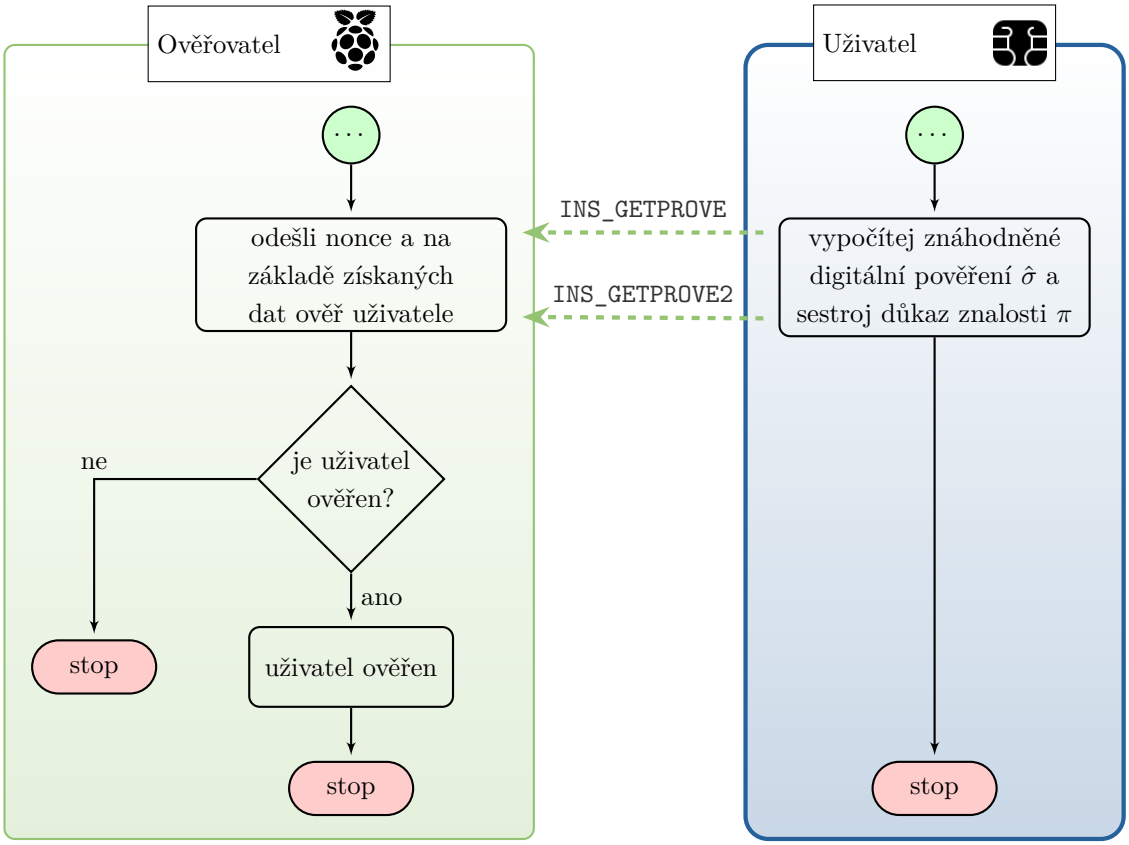
CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**
0x80	0x03	no	0x00	0x20	*nonce*	*size*

Data	**W1**	**W2**
$\hat{\sigma} \mid e \mid s_v \mid s_{m_{z \notin D}}$	0x90	0x00

- `INS_GETPROVE2` – tato instrukce se použije pouze v případě, že je počet skrytých atributů větší než čtyři. V tomto případě se již žádná data na čipovou kartu neodesílají. Tentokráte hodnota proměnné *no* má význam jako počet již navrácených atributů $m_{z \notin D}$, tedy za předpokladu maximálního počtu deseti uživatelských atributů je vždy rovna hodnotě 0x04. Očekávanou velikost *size* APDU odpovědí lze spočítat jako počet skrytých atributů $m_{z \notin D} - 4$, celé vynásobeno velikostí eliptické křivky.

CLA	**INS**	**P1**	**P2**	**Le**
0x80	0x04	no	0x00	size

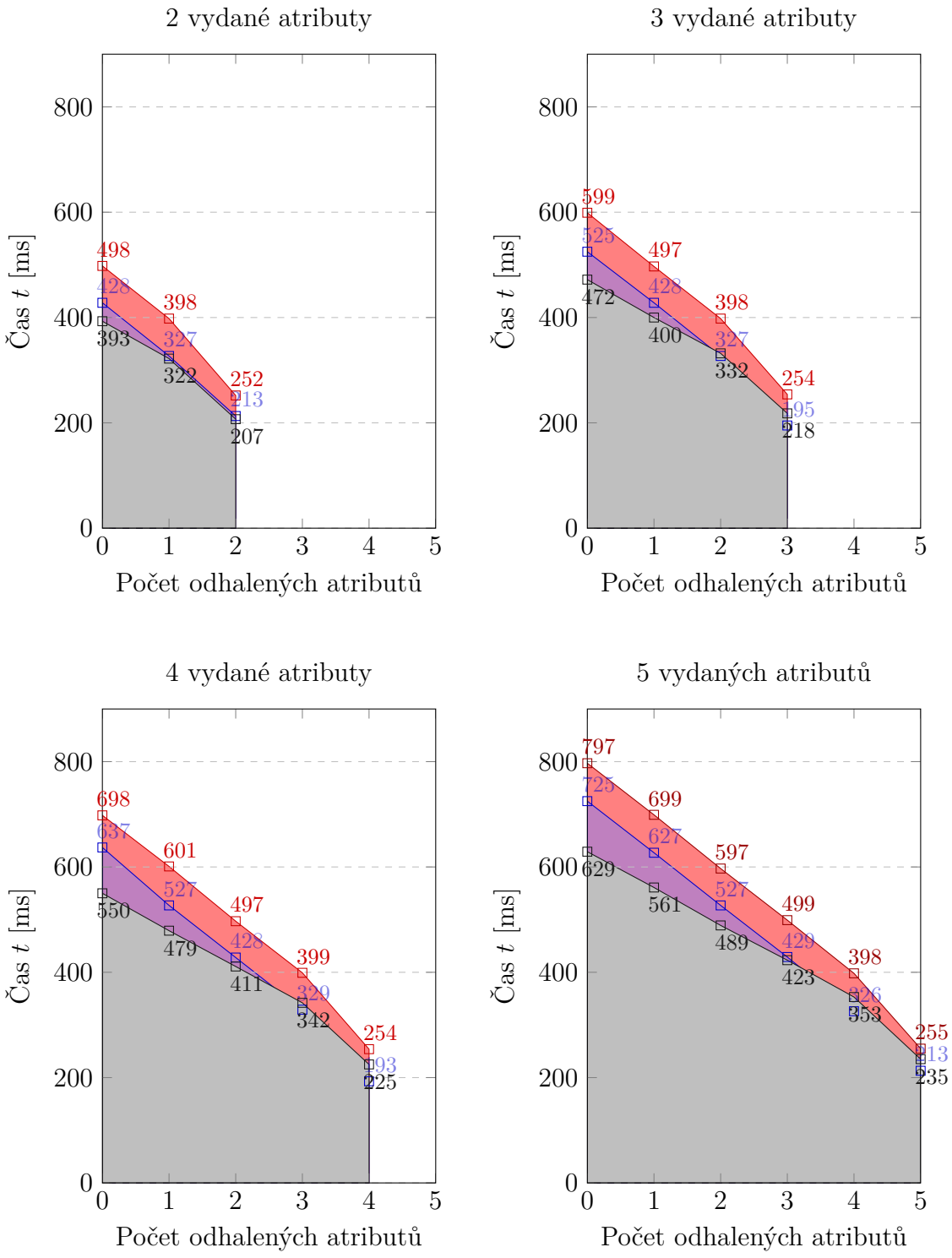
Data	**W1**	**W2**
$s_{m_{z \notin D}}$	0x90	0x00



Obr. 6.4: Ověřovací protokol KVAC – vývojový diagram.

6.3.4 Zhodnocení výsledků implementace

Schéma využívající anonymního atributového pověření KVAC bylo implementováno v souladu se zadáním práce. Byla provedena měření času běhu kryptografického algoritmu `Show`, který je počítán na čipové kartě. Jedna sada měření zahrnovala přenos dat na čipovou kartu a zpět a druhá nikoli. Variabilními parametry byly celkový počet atributů a počet odhalených atributů. Naměřené hodnoty byly vyneseny do grafu viz obrázek 6.5 a porovnány s implementací Ing. Dzurendy, Ph.D. [16]. Vzhledem k tomu, že implementace Ing. Dzurendy, Ph.D. využívá EC o velikosti 192 bitů, byla pro účely porovnání použita taktéž 192bitová EC. V grafu jsou šedou barvou vyznačeny časy, kterých dosáhl algoritmus Ing. Dzurendy, Ph.D. (bez APDU komunikace). Srovnatelné hodnoty časů dosažené v rámci implementace této práce jsou zobrazeny modře. Algoritmus je pomalejší pro větší počet skrytých algoritmů, což může být dáno rozdílným přístupem k samotné implementaci. Červeně vyznačené hodnoty odpovídají časům algoritmu včetně potřebné APDU komunikace. Tyto hodnoty již nebyly podrobeny srovnání.



Obr. 6.5: Porovnání rychlostí ověřovacího protokolu KVAC ve srovnání s implementací Ing. Dzurendy, Ph.D. Modrá barva – čas algoritmu, červená – čas algoritmu včetně komunikace, šedá – čas algoritmu Ing. Dzurendy, Ph.D.

7 Revokovatelné schéma využívající atributové pověření – RKVAC

Zadání tohoto revokovatelného schématu, které využívá anonymního atributového pověření RKVAC (Revocable Keyed-Verification Anonymous Credentials) bylo předloženo školitelem a vychází z [25], [26].

Základní požadavky jsou shodné s předchozími schématy, tedy zajistit prokazatelnou bezpečnost a ochranu soukromí (poskytovat anonymitu uživatelů, nesledovatelnost, nespojitelnost). Dále selektivní odhalení atributů a efektivní implementaci, která umožní použití na výkonově omezených zařízeních jako jsou čipové karty. Novinkou je požadavek na revokovatelnost uživatele, kde může nadřazená revokační autorita odvolávat uživatele ze systému, případně odhalit jeho identitu.

7.1 Obecná architektura RKVAC

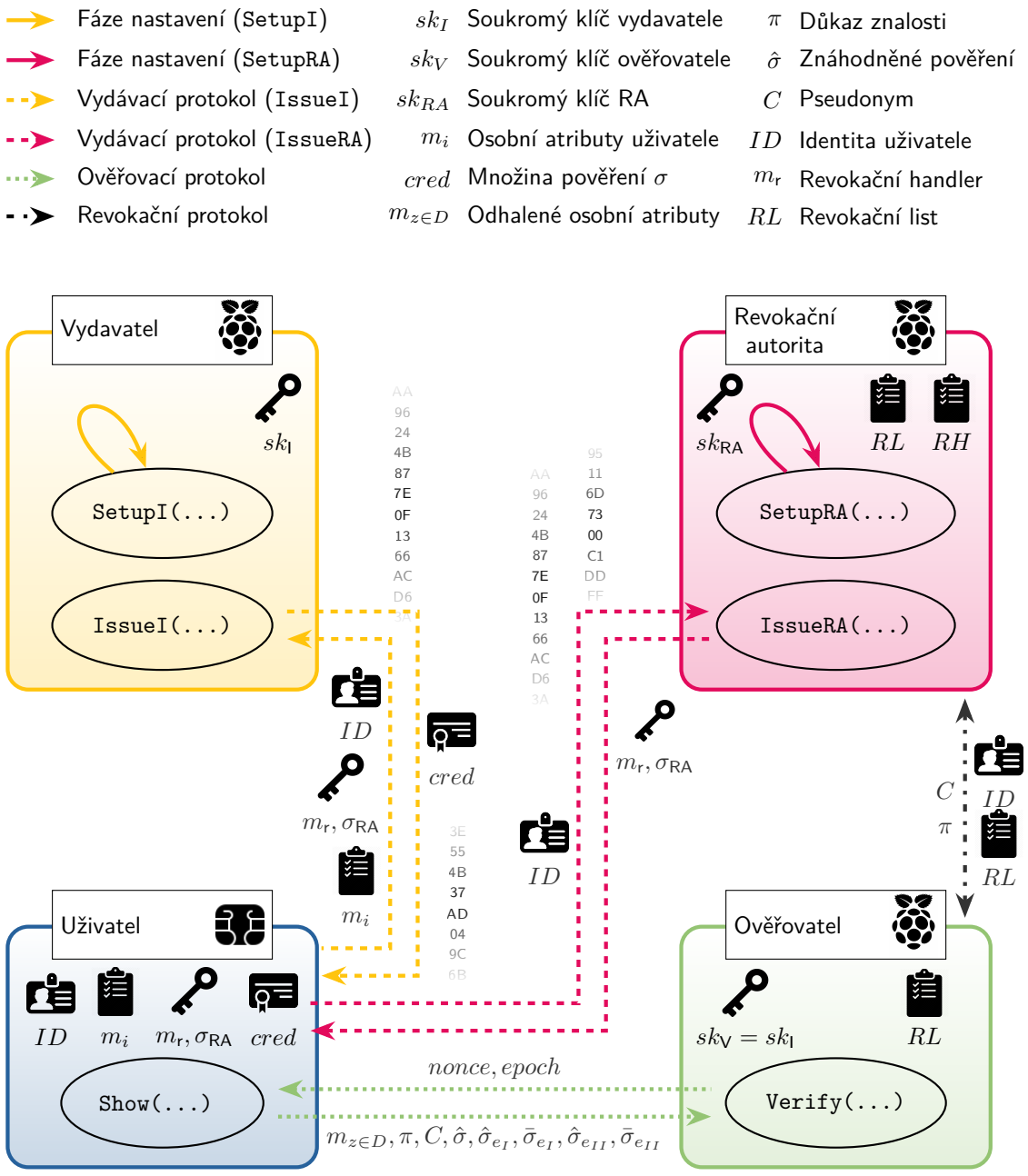
Konstrukce popisovaného revokovatelného schématu vychází ze schématu KVAC, které je popsáno v kapitole 6 a je založeno na podpisu wBB (označeno jako Σ), algebraické funkci MAC (označeno jako $\texttt{MAC}_{\texttt{wBB}}$) a důkazech nulové znalosti. Ve srovnání s dříve uvedeným schématem jsou zde navíc implementovány efektivní mechanismy pro přímou revokaci uživatele ze systému či jeho případnou deanonymizaci. Opět je zde na straně uživatele, který používá čipovou kartu (nebo jiné výkonově omezené zařízení), omezen počet náročných výpočetních operací. Struktura RKVAC vychází ze schématu KVAC. Tři základní entity jsou pro obě schémata shodné (Uživatel, Vydavatel, Ověřovatel) a proto zde již nebudou rozepisovány. Níže je popsána entita, se vyskytuje pouze u RKVAC:

- **Revokační autorita ($\mathcal{RA}$)**: je důvěryhodná třetí strana, která je schopna revokovat a zneplatnit uživatelské atributy a případně odkrýt identitu uživatele.

Všechny čtyři entity v rámci systému opět interagují pomocí specifických kryptografických protokolů a algoritmů (viz obrázek 7.1). Implementace RKVAC obsahuje celkem čtyři základní protokoly (fáze):

- **Nastavení** – generování parametrů systému, nastavení soukromých a veřejných klíčů a jejich distribuce. Obsahuje algoritmy `SetupI`, `SetupRA`.
- **Vydávací protokol** – protokol pro vydání a verifikaci atributů. Uživatel získává digitální pověření. Dělí se na algoritmy `IssueI`, `IssueRA`.
- **Ověřovací protokol** – protokol umožňující prokázání držení atributů uživatele vůči ověřovateli. Jeho součástí jsou algoritmy `Show` a `Verify`, které spolu navzájem komunikují. V případě validního ověření je uživateli povolen přístup.

- **Revokační protokol** – pokud některý z uživatelů porušuje pravidla systému, předává ověřovatel pseudonym uživatele a důkaz znalosti revokační autoritě, která může vyloučit uživatele ze systému tím, že zneplatní jeho pověření, nebo může odhalit uživatelovu identitu. [1]



Obr. 7.1: Architektura Revocable Keyed-Verification Anonymous Credentials.

[1] Revokační protokol není v rámci této práce implementován.

7.2 Popis protokolů a kryptografických algoritmů

V této podkapitole jsou detailněji specifikovány a popsány jednotlivé protokoly a kryptografické algoritmy RKVAC včetně vstupních a výstupních proměnných.

7.2.1 Nastavení

Tak jako u KVAC je nezbytné nastavit parametry systému označované jako $params_{\mathrm{I}}$. Obsah těchto parametrů není 100% shodný s KVAC a proto zde bude rozepsán. Parametry systému obsahují cyklické grupy $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathrm{T}}$ stejného řádu, body eliptické křivky g_1, g_2 včetně jejich řádu q a nedegenerativní mapu $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_{\mathrm{T}}$ pro bilineární párování. Tyto parametry budou použity v celé implementaci a jsou generovány algoritmem `SetupI`. Dále tento algoritmus vygeneruje množinu klíčů vydavatele sk_{I}. Algoritmus spouští vydavatel. Jedná se o algoritmus shodný s algoritmem $\mathtt{MAC}_{\mathtt{wBB}}.\mathtt{Setup}$ (viz kapitola 5.2), proto zde nebude podrobně rozepisován.

Protože se v systému vyskytuje také entita zvaná revokační autorita, je nutné ji také nastavit parametry, označované jako $params_{\mathrm{RA}}$, které přebírají grupové parametry z $params_{\mathrm{I}}$. V tomto případě je vše řízeno algoritmem `SetupRA`, jehož vstupem je také parametr $ver_{\max}$, jenž určuje maximální počet ověřovacích relací, pro které je zajištěna nespojitelnost. Hodnota tohoto parametru je vyjádřena vztahem $ver_{\max} = k^j$, kde celočíselná hodnota j určuje počet náhodných celočíselných proměnných α. Hodnota k určuje počet randomizérů e a jejich pověření σ_e. Proměnné e spolu s pověřením σ_e slouží na straně uživatele k výpočtu pseudonymu. Správnou volbou proměnných j, k balancujeme výpočetní a paměťové nároky. Pokud inkrementujeme hodnotu proměnné k o jedničku, zvýšíme tak paměťovou náročnost o $3 \cdot |p|$. Naopak inkrementací proměnné j zvyšujeme výpočetní náročnost, což může být v případě výpočtů na straně čipové karty způsobit značné zdržení. Aby nemusela být hodnota $ver_{\max}$ vysoká, a přitom nebyla omezující co do počtu ověření, je zde začleněn časový element zvaný epocha. V tom případě musí být každá ověřovací relace unikátní pouze po dobu trvání jedné epochy. Algoritmus `SetupRA` také generuje a počítá pár klíčů $sk_{\mathrm{RA}}, pk_{\mathrm{RA}}$ a je spouštěn revokační autoritou.

$(params_{\mathrm{RA}}, sk_{\mathrm{RA}}, pk_{\mathrm{RA}}, RL) \leftarrow \mathtt{SetupRA}(1^{\kappa}, ver_{\max})$:

- výběr proměnných typu integer $(k, j) : ver_{\max} = k^j$, v rámci implementace zvoleno ($j = 2$, $k = 10$), z čehož vychází 100 pseudonymů na jednu epochu.
- Výběr náhodných celočíselných proměnných $(\alpha_1, \ldots, \alpha_j) \in_R \mathbb{Z}_q$ a výpočet $h_z = g_1^{\alpha_z}$ pro všechna z od 1 do j, tedy (α_1, α_2), (h_1, h_2),
- zvolení soukromého a výpočet veřejného klíče RA: $(sk_{\mathrm{RA}} \in_R \mathbb{Z}_q, pk_{\mathrm{RA}} = g_2^{sk_{\mathrm{RA}}})$,
- výběr randomizérů $(e_1, \ldots, e_k) \in_R \mathbb{Z}_q$, konkrétně $(e_1, \ldots, e_{10})$ a výpočet pověření pro každý z nich: $\sigma_{e_z} \leftarrow g_1^{\frac{1}{e_z + sk_{\mathrm{RA}}}}$. Toto pověření vychází z wBB podpisu

(viz kapitola 5.1), konkretně z algoritmu Σ.`Sign`.

– Výstupem algoritmu je pár klíčů $(sk_{\mathrm{RA}}, pk_{\mathrm{RA}})$ a parametry revokační autority $params_{\mathrm{RA}} = (q, \mathbb{G}_1, g_1, k, j, (h_1, \ldots, h_j), (\alpha_1, \ldots, \alpha_j), \{(e_1, \sigma_{e_1}, \ldots, \{(e_k, \sigma_{e_k})\})$.

7.2.2 Vydávací protokol

Vydávací protokol je kombinací dvou kryptografických algoritmů. V první fázi běží protokol mezi revokační autoritou a uživatelem, což zajišťuje algoritmus `IssueRA`. Uživatel zasílá revokační autoritě svůj jednoznačný identifikátor neboli ID, který je součástí pověření. Dalším vstupem jsou dříve vygenerované parametry revokační autority $params_{\mathrm{RA}}$ a soukromý klíč revokační autority sk_{RA}. Následně je revokační autoritou generován revokační handler m_{r}, který bude součástí uživatelem počítaného pseudonymu. Revokační autorita také vystavuje pověření σ_{RA} pro tento handler. Revokační handler a jeho pověření je odesláno zpět k uživateli. Znázornění tohoto algoritmu je na obrázku 7.2.

$(m_{\mathrm{r}}, \sigma_{\mathrm{RA}}) \leftarrow$ `IssueRA`$(params_{\mathrm{RA}}, sk_{\mathrm{RA}}, \mathrm{ID})$:

– uživatel zasílá svůj jednoznačný identifikátor ID, který je součástí podpisu revokačního handleru,

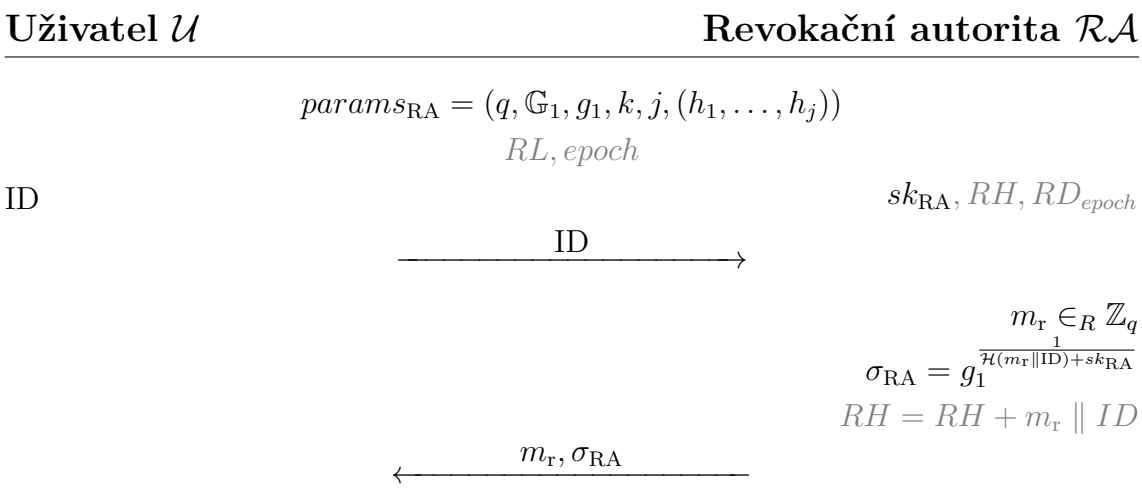
– revokační autoritou je náhodně zvolen revokační handler m_{r} a následně je pro něj spočítán podpis $\sigma_{\mathrm{RA}} = g_1^{\frac{1}{\mathcal{H}(m_{\mathrm{r}} \| \mathrm{ID}) + sk_{\mathrm{RA}}}}$.

V rámci druhé fáze běží protokol mezi vydavatelem a uživatelem a tuto fázi zabezpečuje algoritmus `IssueI`. Vstupem je soukromý klíč vydavatele sk_{I}, veřejný klíč revokační autority pk_{RA} a parametry systému $params_{\mathrm{I}}$. Uživatel zasílá ID, revokační handler, jeho pověření a své atributy, pro které chce získat digitální pověření. Aby nenastala situace, kdy uživatel podvrhne revokační handler a v podstatě se stane nerevokovatelný, musí uživatel nejdříve prověřit platnost pověření tohoto revokačního handleru. Po úspěšném ověření vypočítá vydavatel množinu digitálních pověření, kterou zasílá nazpět uživateli. Detaily algoritmu zobrazeny na obrázku 7.3.

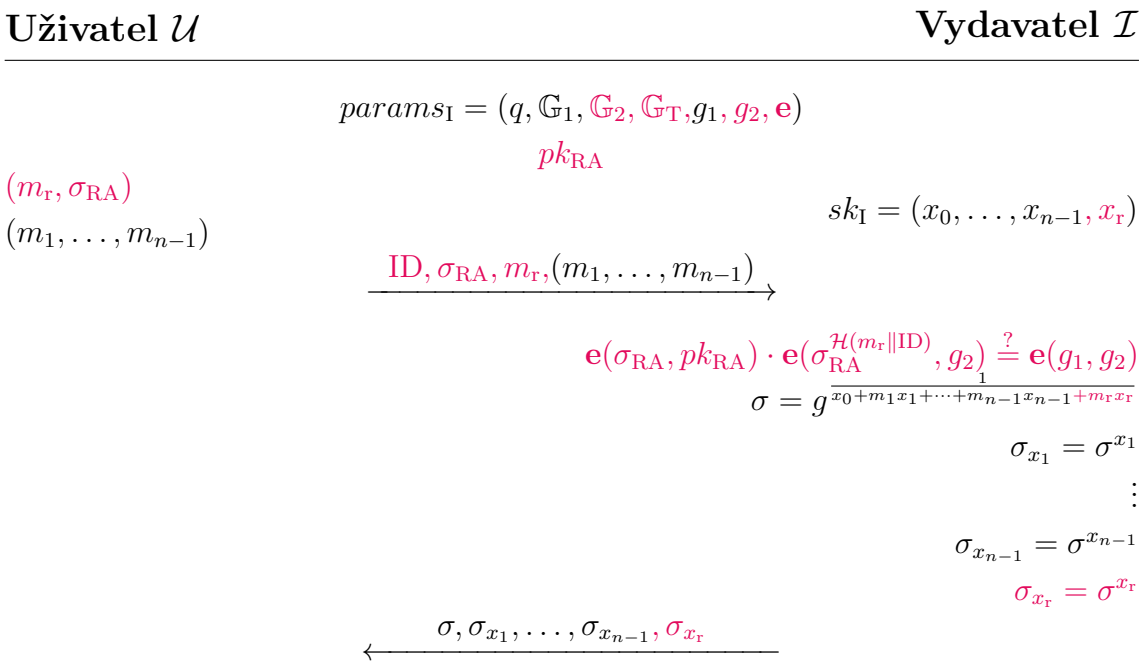
$(cred) \leftarrow$ `IssueI`$(params_{\mathrm{I}}, pk_{\mathrm{RA}}, sk_{\mathrm{I}}, m_{\mathrm{r}}, (m_1, \ldots, m_{n-1}), \mathrm{ID}, \sigma_{\mathrm{RA}})$:

– uživatel odesílá vydavateli své atributy $(m_1, \ldots, m_{n-1})$ a trojici ověřovacích údajů, která obsahuje jednoznačný identifikátor ID, revokační handler m_{r} a jeho pověření σ_{RA},

– vydavatel provede ověření uživatele na základě zaslané trojice ověřovacích údajů a veřejného klíče RA: $\mathbf{e}(\sigma_{\mathrm{RA}}, pk_{\mathrm{RA}}) \cdot \mathbf{e}(\sigma_{\mathrm{RA}}^{\mathcal{H}(m_{\mathrm{r}} \| \mathrm{ID})}, g_2) \stackrel{?}{=} \mathbf{e}(g_1, g_2)$ a v případě shody vydá množinu pověření: $cred = (\sigma, \sigma_{x_{\mathrm{r}}}, \sigma_{x_0}, \sigma_{x_1}, \ldots, \sigma_{x_{n-1}})$.



Obr. 7.2: Vydávací protokol – algoritmus `IssueRA`. Šedou barou vyznačeny části, které nebyly implementovány v rámci této práce.



Obr. 7.3: Vydávací protokol – algoritmus `IssueI`. Červeně jsou vyznačeny rozdíly oproti schématu KVAC.

7.2.3 Ověřovací protokol

Shodně jako u KVAC uživatel odhaluje vůči autoritě ověřovatele některé své atributy, jiné naopak zůstávají skryté. Srovnatelný je i účel protokolu, kde se prověřuje jestli jsou odhalené atributy vydané důvěryhodnou autoritou. Klíčovým požadavkem tohoto schématu je revokovatelnost uživatele. Aby mohla být tato podmínka dodržena, musí uživatel zasílat svůj pseudonym ovšem při splnění zásad pro anonymitu a nesledovatelnost. Což je proveditelné za předpokladu, že nebude docházet k opakování pseudonymu. Pro splnění neméně důležitého požadavku nespojitelnosti nesmí dojít k odhalení uživatelova pověření, které by mohlo být zpětně trasovatelné k jeho majiteli. Toto je zajištěno prozrazením takzvaného znáhodněného digitálního pověření.

Z výše uvedeného je zřejmé, že komunikace v rámci protokolu probíhá mezi uživatelem a ověřovatelem. Entita uživatele využívá kryptografického algoritmu `Show`. Na vstupu má parametry systému $params_{\mathrm{I}}$, parametry revokační autority $params_{\mathrm{RA}}$, množinu svých atributů $(m_1, \ldots, m_{n-1})$, revokační handler m_{r} a jejich pověření $cred$. Dále od ověřovatele obdrží náhodnou hodnotu $nonce$ a identifikátor časového období neboli hodnotu $epoch$. Na jejich základě uživatel sestaví pseudonym C. V dalším kroku generuje několik pomocných hodnot, které slouží k vystavení znáhodněného digitálního pověření $\hat{\sigma}$ a sestrojení důkazů znalosti π (včetně důkazu znalosti pseudonymu). Znáhodněné pověření, důkaz znalosti a pseudonym zasílá zpět ověřovateli včetně odhalených atributů $m_{z \notin D}$.

Entita ověřovatele využívá modifikovaného algoritmu `Verify`, na jehož vstup přichází hodnoty od uživatele. Kromě parametrů systému a revokační autority je dalším vstupem veřejný klíč revokační autority pk_{RA} a množina klíčů vydavatele $sk_{\mathrm{V}} = sk_{\mathrm{I}}$. Ověřovatel prověří platnost důkazu znalosti a důkazu znalosti pseudonymu a zkontroluje, zdali se zaslaný pseudonym nevyskytuje v seznamu revokovaných pseudonymů. V případě potvrzení platnosti důkazu znalosti jsou odhalené atributy považovány za validní. Úplný popis protokolu je zobrazen na obrázku 7.4.

$[(C, \hat{\sigma}, \hat{\sigma}_{e_I}, \hat{\sigma}_{e_{II}}, \bar{\sigma}_{e_I}, \bar{\sigma}_{e_{II}}, \pi) \leftarrow \mathtt{Show}(params_{\mathrm{I,\,RA}}, m_{\mathrm{r}}, (m_1, \ldots, m_{n-1}), cred, epoch)$
$\leftrightarrow \mathtt{Verify}(params_{\mathrm{I,\,RA}}, sk_{\mathrm{I}}, pk_{\mathrm{RA}}, m_{z \in D}, C, \hat{\sigma}, \hat{\sigma}_{e_I}, \hat{\sigma}_{e_{II}}, \bar{\sigma}_{e_I}, \bar{\sigma}_{e_{II}}, \pi, epoch) \rightarrow (0/1)]$:

- uživatel vytvoří pseudonym C pomocí hashe z aktuální epochy $\mathcal{H}(epoch)$, jedinečné hodnoty i vypočtené z $params_{\mathrm{RA}}$ (randomizérů e_z a hodnot α_z) jako $i = \sum_{z=1}^{j} \alpha_z e_z$ a revokačního handleru m_{r}: $C = g_1^{\frac{1}{i - m_{\mathrm{r}} + \mathcal{H}(epoch)}}$,
- dále uživatel randomizuje své pověření a sestavuje důkaz znalosti: $(\hat{\sigma}, \hat{\sigma}_{e_I}, \hat{\sigma}_{e_{II}}, \bar{\sigma}_{e_I}, \bar{\sigma}_{e_{II}}, \pi)$,
- ověřovatel prověří zaslaný důkaz π a zkontroluje, zda není pseudonym C na revokačním listu RL.

Uživatel $\mathcal{U}$ **Ověřovatel $\mathcal{V}$**

$$params_{\mathrm{I}} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathrm{T}}, g_1, g_2, \mathbf{e})$$
$$params_{\mathrm{RA}} = (q, \mathbb{G}_1, g_1, k, j, (h_1, \ldots, h_j), (\alpha_1, \ldots, \alpha_j))$$
$$pk_{\mathrm{RA}}, RL$$

$$params_{\mathrm{RA}} = \{(e_1, \sigma_{e_1}, \ldots, \{(e_k, \sigma_{e_k})\}$$

$(m_1, \ldots, m_{n-1}, m_{\mathrm{r}})$
$\sigma, (\sigma_{x_1}, \ldots, \sigma_{x_{n-1}}, \sigma_{x_{\mathrm{r}}})$
$sk_{\mathrm{V}} = sk_{\mathrm{I}} = (x_0, \ldots, x_{n-1}, x_{\mathrm{r}})$

$\xleftarrow{nonce, epoch}$

$e_I, e_{II} \in_R (e_1, \ldots, e_k)$
$\sigma_{e_I}, \sigma_{e_{II}} \in_R (\sigma_{e_1}, \ldots, \sigma_{e_k})$
$i \leftarrow \alpha_1 e_I + \alpha_2 e_{II}$
$C \leftarrow g_1^{\frac{1}{i - m_{\mathrm{r}} + \mathcal{H}(epoch)}}$
$\rho, \rho_v, \rho_i, \rho_{m_{\mathrm{r}}}, \rho_{m_{z \notin D}}, \rho_{e_I}, \rho_{e_{II}} \in_R \mathbb{Z}_q$
$\hat{\sigma} \leftarrow \sigma^{\rho}$
$\hat{\sigma}_{e_I} \leftarrow \sigma_{e_I}^{\rho}, \hat{\sigma}_{e_{II}} \leftarrow \sigma_{e_{II}}^{\rho}$
$\bar{\sigma}_{e_I} \leftarrow \hat{\sigma}_{e_I}^{-e_I} g_1^{\rho}, \bar{\sigma}_{e_{II}} \leftarrow \hat{\sigma}_{e_{II}}^{-e_{II}} g_1^{\rho}$

$t_{\mathrm{verify}} \leftarrow g_1^{\rho_v} \sigma^{\rho_{m_{\mathrm{r}}} \rho} \prod_{z \notin D} \sigma_{x_z}^{\rho_{m_z} \rho}$
$t_{\mathrm{revoke}} \leftarrow C^{\rho_{m_{\mathrm{r}}}} C^{\rho_i}$
$t_{\mathrm{sig}} \leftarrow g_1^{\rho_i} h_1^{\rho_{e_I}} h_2^{\rho_{e_{II}}}, t_{\mathrm{sigI}} \leftarrow g_1^{\rho_v} \hat{\sigma}_{e_I}^{\rho_{e_I}}, t_{\mathrm{sigII}} \leftarrow g_1^{\rho_v} \hat{\sigma}_{e_{II}}^{\rho_{e_{II}}}$
$e \leftarrow \mathcal{H}(t_{\mathrm{verify}}, t_{\mathrm{revoke}}, t_{\mathrm{sig}}, t_{\mathrm{sigI}}, t_{\mathrm{sigII}}, \hat{\sigma}, \hat{\sigma}_{e_I}, \bar{\sigma}_{e_I}, \hat{\sigma}_{e_{II}}, \bar{\sigma}_{e_{II}}, C, nonce)$

$\langle s_{m_z} \leftarrow \rho_{m_z} - e m_z \rangle_{z \notin D}$
$s_v \leftarrow \rho_v + e\rho$
$s_{m_{\mathrm{r}}} \leftarrow \rho_{m_{\mathrm{r}}} - e m_{\mathrm{r}}$
$s_i \leftarrow \rho_i + ei$
$s_{e_I} \leftarrow \rho_{e_I} - e e_I, s_{e_{II}} \leftarrow \rho_{e_{II}} - e e_{II}$
$\pi \leftarrow (e, s_{m_{z \notin D}}, s_v, s_{m_{\mathrm{r}}}, s_i, s_{e_I}, s_{e_{II}})$

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

$t_{\mathrm{verify}} \leftarrow \hat{\sigma}^{-e x_0} g_1^{s_v} \hat{\sigma}^{x_{\mathrm{r}} s_{m_{\mathrm{r}}}} \prod_{z \notin D} \hat{\sigma}^{x_z s_{m_z}} \prod_{z \in D} \hat{\sigma}^{-e x_z s_{m_z}}$
$t_{\mathrm{revoke}} \leftarrow (g_1 C^{-\mathcal{H}(epoch)})^{-e} C^{s_{m_{\mathrm{r}}}} C^{s_i}$
$t_{\mathrm{sig}} \leftarrow g_1^{s_i} h_1^{s_{e_I}} h_2^{s_{e_{II}}}, t_{\mathrm{sigI}} \leftarrow g_1^{s_v} \hat{\sigma}_{e_I}^{s_{e_I}} \bar{\sigma}_{e_I}^{-e}, t_{\mathrm{sigII}} \leftarrow g_1^{s_v} \hat{\sigma}_{e_{II}}^{s_{e_{II}}} \bar{\sigma}_{e_{II}}^{-e}$

$e \leftarrow \mathcal{H}(t_{\mathrm{verify}}, t_{\mathrm{revoke}}, t_{\mathrm{sig}}, t_{\mathrm{sigI}}, t_{\mathrm{sigII}}, \hat{\sigma}, \hat{\sigma}_{e_I}, \bar{\sigma}_{e_I}, \hat{\sigma}_{e_{II}}, \bar{\sigma}_{e_{II}}, C, nonce)$
$\mathbf{e}(\bar{\sigma}_{e_I}, g_2) = \mathbf{e}(\hat{\sigma}_{e_I}, pk_{\mathrm{RA}})$
$\mathbf{e}(\bar{\sigma}_{e_{II}}, g_2) = \mathbf{e}(\hat{\sigma}_{e_{II}}, pk_{\mathrm{RA}})$
$C \notin RL$

Obr. 7.4: Ověřovací protokol (červeně označeny rozdíly oproti KVAC).

7.3 Implementace RKVAC

Vzhledem k tomu, že RKVAC je rozšířením schématu KVAC byly vytvořené entity také shodně implementovány na jednotlivé platformy (viz 6.3). V tomto schématu se navíc vyskytuje entita zvaná revokační autorita, která byla implementována na platformu Raspbian GNU/Linux běžící na mikropočítači Raspberry Pi.

7.3.1 Zadané a zvolené parametry

I tato implementace pracuje s eliptickými křivkami definovanými přes konečná pole $\mathbb{F}_p$, které jsou obvykle reprezentovány v krátké Weierstrassově formě s pomocí doménových parametrů. Tyto doménové parametry opět nebyly generovány, ale jsou dány křivkou BN254 z MCL knihovny [22] a jsou shodné s předchozí implementací.

7.3.2 Vytvořené aplikace a knihovny

Základ aplikací, které byly programované pro předchozí KVAC schéma, byl použit také pro tuto implementaci. Navíc bylo přidáno patřičné rozšíření, jenž umožní vykonávat funkce potřebné pro splnění RKVAC schématu. První aplikace, obsluhující entitu uživatele, byla přejmenována na KBCT_RKVACeloyalty a zdrojový kód této aplikace je uložen v souboru `eloyalty.c`. Hexadecimální identifikátor aplikace zůstal nezměněn. Pro korektní průběh kompilace aplikace je nezbytný hlavičkový soubor `scAppSetup.h`, v němž jsou pomocí maker nadefinovány konstanty zásadní pro běh celé aplikace. Druhá aplikace byla přejmenována na KBCT_RKVAC a zabezpečuje funkce pro zbývající tři entity systému. Hlavní části, funkce i pomocné hlavičkové soubory zůstaly zachovány. Byly doplněny funkce potřebné pro obsluhu revokační autority a některé stávající funkce byly modifikovány.

Vybrané funkce a metody

Jelikož jsou některé funkce totožné s předchozí implementací, nebudou zde popisovány. Budou popsány jen vybrané části kódu, které jsou využity pro účely revokace nebo byly v rámci této implementace změněny. Za zmínku stojí algoritmus (aplikace KBCT_RKVACeloyalty) pseudonáhodné inicializace proměnných I, II pro výběr randomizérů $e_I, e_{II} \in_R (e_1, \ldots, e_{10})$ a jejich pověření $\sigma_{e_I}, \sigma_{e_{II}} \in_R (\sigma_{e_1}, \ldots, \sigma_{e_{10}})$. Jako základ je využito makro `getRND4bits`, které vygeneruje pseudonáhodné číslo o velikosti čtyř bitů. Toto číslo může nabývat hodnot 0-F šestnáctkové soustavy. Avšak pro zvolené hodnoty protokolu musíme následně toto číslo redukovat modulo 10. Na základě tohoto 4b čísla, které nyní může nabývat hodnot $\langle 0, 9 \rangle$ je proveden prvotní výběr randomizéru. Detail definice zmíněného makra je zobrazen ve výpisu 7.1.

Výpis 7.1: Definice makra pro generování 4bitových pseudonáhodných čísel.

```
#define getRND4bits(byteOutAddr)                                    \
do                                                                  \
{                                                                   \
    __code (PRIM, 0xc4);                                            \
    __code (POPN, 7);                                               \
    __code (PRIM, __PRIM_BIT_MANIPULATE_BYTE, 0x83, 0x0F);          \
    __code (STORE, __typechk(BYTE *, byteOutAddr), 1);              \
} while (0)
```

Aplikace KBCT_RKVAC opět používá velké množství nadefinovaných funkcí a nově přidané nebo modifikované jsou popsány níže:

- Novou definovanou funkcí je `setupRA`, která by se dala popsat tak, že zabezpečuje část fáze nastavení a vydávacího protokolu entity zvané revokační autorita. Ještě přesněji tak, že obstarává kryptografické algoritmy `setupRA` a `IssueRA`.
- Funkce zvaná `signI` v rámci schématu spadá do vydávacího protokolu. Počítá množinu digitálních pověření *cred* a zasílá je na čipovou kartu. Zastává tedy funkce kryptografického algoritmu `IssueI`. Nově prověřuje platnost pověření revokačního handleru.
- Nezbytnou funkcí je `proveI`, která zajišťuje největší množství výpočetních operací na platformě Raspbian GNU/Linux. Pro entitu ověřovatele zajišťuje kryptografický algoritmus `Verify`. Výstupem funkce je hodnota reprezentována logickým datovým typem boolean tedy pravda/nepravda, což vypovídá o úspěchu ověření. V rámci této implementace byla přidána kontrola platnosti pseudonymu C.

7.3.3 APDU komunikace

V této podkapitole je popsán časový sled zasílání APDU příkazů a také je přiblížena jejich struktura a obsah včetně očekávaných APDU odpovědí. Tyto příkazy a odpovědi jsou zde reprezentovány slovním označením instrukce. Toto označení je také používáno ve zdrojovém kódu a pomocí definice konstant jsou jim přiřazeny číselné hodnoty. Popis prvního kroku, tedy spuštění aplikace a odeslání atributů na čipovou kartu, nebyl výrazně modifikován, proto zde nebude znovu uváděn. Jedinou změnou je, že nyní lze na čipovou kartu odeslat maximálně devět uživatelských atributů. Desátý je totiž rezervován pro revokační handler. Instrukce `INS_SELECT` a `INS_SETATTR` jsou podrobně popsány v kapitole 6.3.3.

V rámci druhé fáze je nezbytné vydat uživateli jeho revokační handler včetně pověření a na čipovou kartu nahrát parametry revokační autority. Tento postup je

vykreslen do vývojového diagramu na obrázku 7.5 (barevné označení entit figurujících v následujících diagramech je shodné s architekturou systému 7.1) a slouží k tomu následující instrukce:

- `INS_GETID` – žádost o návrat jednoznačného identifikátoru (ID) uživatele. Očekávaná velikost příchozích dat 21B.

CLA	**INS**	**P1**	**P2**	**Le**
0x80	0x30	0x00	0x00	0x15

Data	**W1**	**W2**
ID	0x90	0x00

- `INS_SETPRA` – odeslání revokačního handleru m_r, jeho pověření σ_{RA} a parametrů $params_{RA}$ na čipovou kartu. Jako první jsou odeslány hodnoty m_r, σ_{RA} a (α_1, α_2). Následně jsou v pěti krocích odeslány atributy $\{(e_1, \sigma_{e_1}), \ldots, (e_{10}, \sigma_{e_{10}})\}$. Hodnota proměnné $offset$ je v prvním kroku rovna 0x00, následně se iteruje vždy o dvě až do hodnoty 0x08. Atributy (e_k, σ_{e_k}) se vždy odesílají po dvojicích. Z toho lze vyvodit, že hodnota offset se dá matematicky vyjádřit jako dvojnásobek kroku k: $offset = 2k$.

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**
0x80	0x31	0x00	0x02	0xA1	$m_r \mid \sigma_{RA} \mid (\alpha_1, \alpha_2)$

W1	**W2**
0x90	0x00

...

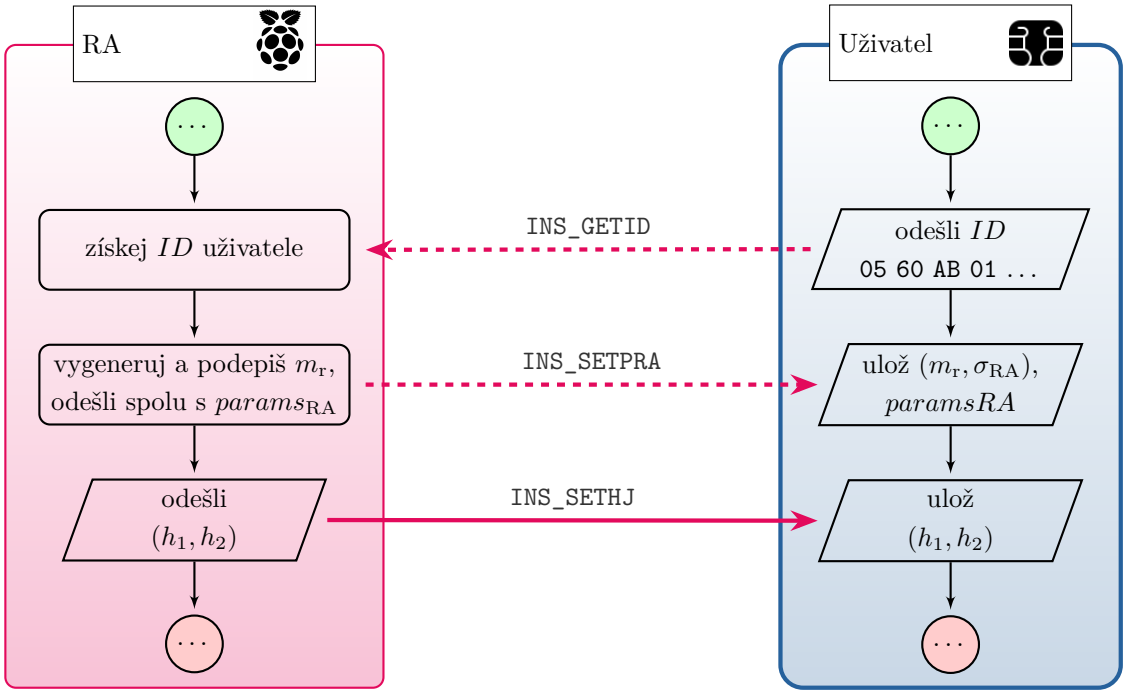
CLA	**INS**	**P1**	**P2**	**Lc**	**Data**
0x80	0x31	0x0A	offset	0xC2	$(e_k, \sigma_{e_k}) \mid (e_{k+1}, \sigma_{e_{k+1}})$

W1	**W2**
0x90	0x00

- `INS_SETHJ` – odeslání atributů h_1, h_2 na čipovou kartu. Tyto atributy by bylo také možno spočítat až na čipové kartě, ale z důvodu menšího výpočetního výkonu jsou na kartu přeneseny.

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**
0x80	0x32	0x00	0x00	0x82	$h_1 \mid h_2$

W1	**W2**
0x90	0x00



Obr. 7.5: Vydávací protokol RKVAC (algoritmus `IssueRA`) – vývojový diagram.

V této fázi jsou všechna nezbytná nastavení již uložena na čipové kartě a vydavatel může, po ověření identity, vydat uživateli potřebná pověření. V případě, že není identita uživatele ověřena, dojde k ukončení algoritmu a žádná další komunikace již neprobíhá. Vše je vyobrazeno ve vývojovém diagramu na obrázku 7.6 a pro komunikaci jsou využity tyto instrukce:

- `INS_INIT` – návrat jednoznačného identifikátoru uživatele (ID), pověření revokační autority σ_{RA}, revokačního handleru a atributů (m_r, m_1,...,m_{n-1}). Hodnota proměnné $offset$ vždy začíná na 0x00. Pokud je atributů méně či rovno pěti, tak celá komunikace proběhne pomocí jednoho APDU příkazu. V případě více atributů je hodnota $offset$ nastavena na 0x05. Očekávaná velikost příchozích dat $size$ je dána součtem velikostí jednotlivých přenášených parametrů.

CLA	**INS**	**P1**	**P2**	**Le**
0x80	0x05	offset	0x00	size

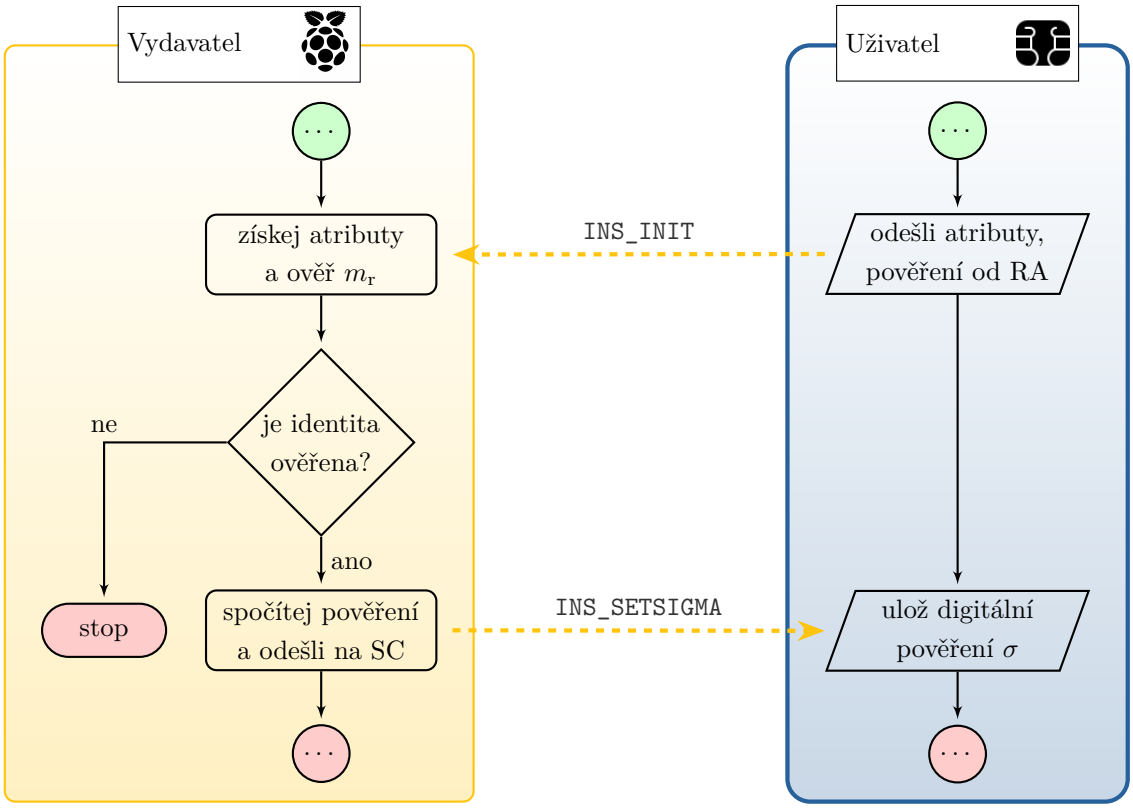
Data	**W1**	**W2**
$ID \mid \sigma_{RA} \mid (m_r, m_1,\ldots,m_{n-1})$	0x90	0x00

- `INS_SETSIGMA` – odeslání pověření σ, σ_{x_r}, $\sigma_{x_1},\ldots,\sigma_{x_{n-1}}$ na čipovou kartu. Proměnná no představuje počet vystavených pověření σ, tedy celkový počet atributů $n+1$. Hodnota $offset$ vždy začíná na 0x00. Pokud je počet pověření σ

menší či roven třem, celá komunikace proběhne pomocí jednoho APDU příkazu. V případě více pověření je hodnota $offset$ inkrementována o tři. Velikost $size$ je dána součtem velikostí jednotlivých přenášených parametrů.

CLA	INS	P1	P2	Lc	Data
0x80	0x01	no	offset	size	$\sigma \mid \sigma_{x_r} \mid (\sigma_{x_1},\ldots,\sigma_{x_{n-1}})$

W1	W2
0x90	0x00



Obr. 7.6: Vydávací protokol RKVAC (algoritmus `IssueI`) – vývojový diagram.

Uživatel má v této fázi téměř vše potřebné pro ověření. Poslední data na čipovou kartu odesílá autorita, která si žádá ověření atributů. Těmito daty jsou náhodné číslo *nonce* a identifikátor časového období *epoch*. V rámci této implementace nebylo ošetřeno odhalování konkrétních atributů, ale ověřovatel zasílá v rámci APDU komunikace pouze to, kolik atributů má zůstat skrytých. Není zde také řešeno, zdali má ověřovatel právo požadovat odhalení zbylých atributů. V tomto kroku již probíhají kryptografické výpočty i na čipové kartě a lze ho prohlásit za časově nejnáročnější. Jakmile karta dokončí výpočty pseudonymu C, znáhodněného digitálního pověření $\hat{\sigma}$

a sestrojí důkaz znalosti π, odesílá tyto údaje ověřovateli. V rámci této implementace nejsou odhalené atributy zasílány zpětným kanálem ověřovateli, ale je uvažováno, že se jedná o atributy veřejné. Ověřovatel následně provede ověření těchto údajů a vyhodnotí, zdali je uživatel oprávněným držitelem odhalovaných atributů, a tedy zda je ověřen či nikoli. Vzájemná komunikace je vykreslena do vývojového diagramu na obrázku 7.7 a lze ji popsat následujícími instrukcemi:

- `INS_GETPROVE` – odeslání *nonce*, *epoch* na čipovou kartu. Proměnná *no* představuje počet skrytých atributů $m_{z \notin D}$. V APDU odpovědi se očekává návrat C, $\hat{\sigma}$, $\hat{\sigma}_{e_I}$.

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**
0x80	0x03	no	0x00	0x24	*nonce* | *epoch*	0xC3

Data	**W1**	**W2**
$C \mid \hat{\sigma} \mid \hat{\sigma}_{e_I}$	0x90	0x00

- `INS_GETPROVE2` – proměnná *iter* představuje číslo iterace, které se vždy inkrementuje o 1 a začíná na hodnotě 0x01. Celkový počet iterací je dán počtem skrytých atributů (minimálně dvě, maximálně tři iterace). V první iteraci se očekává návrat $\hat{\sigma}_{e_{II}}$, $\bar{\sigma}_{e_I}$, $\bar{\sigma}_{e_{II}}$. Ve druhé iteraci je očekáván návrat e, s_v, s_i, s_{e_I}, $s_{e_{II}}$, s_{m_r}, $s_{m_{z \notin D}}$, kde $z \in \langle 1, 3 \rangle$. Pokud existuje třetí iterace, bude z karty zaslán zbytek s_{m_z}. Proměnná *no* určuje, kolik atributů $s_{m_{z \notin D}}$ je očekáváno v odpovědi. V první iteraci tedy 0, ve druhé maximálně tři a ve třetí zbytek. Očekávaná velikost *size* APDU odpovědí je dána součtem velikostí jednotlivých parametrů.

CLA	**INS**	**P1**	**P2**	**Le**
0x80	0x04	iter	no	0xC3

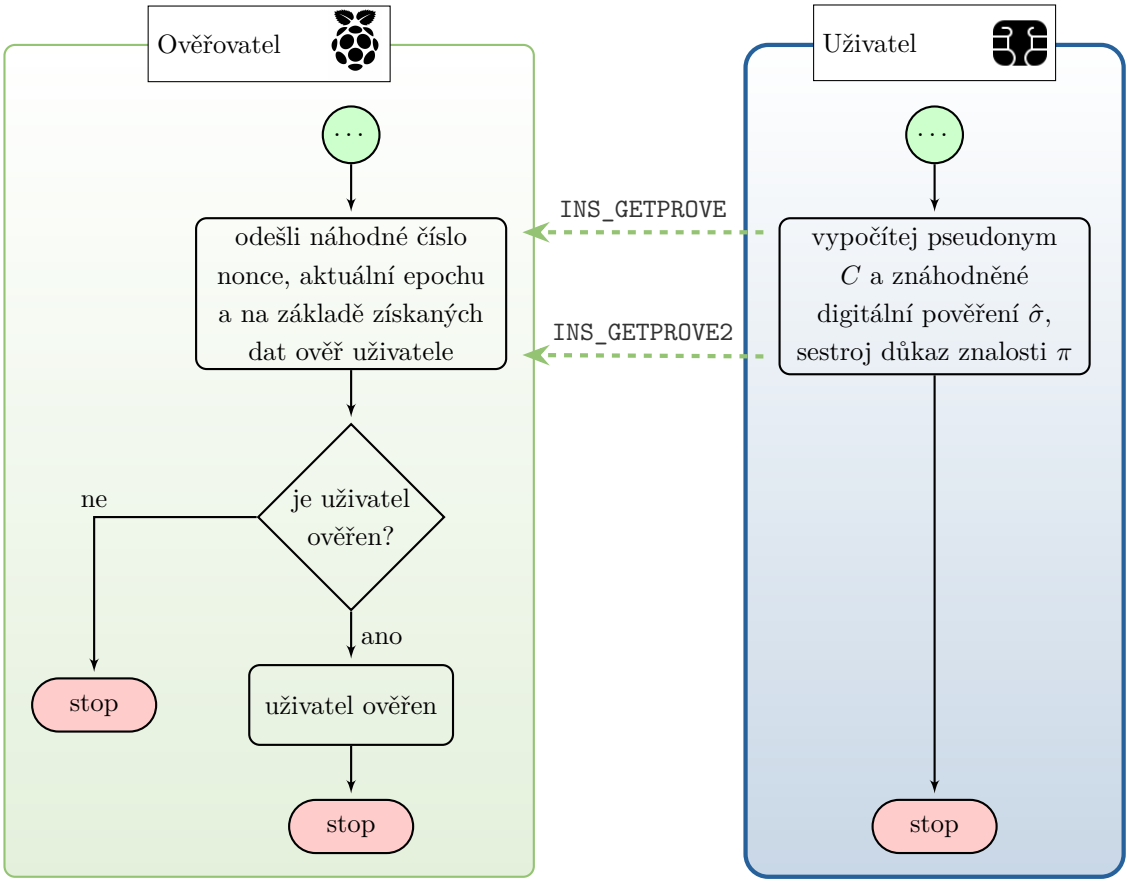
Data	**W1**	**W2**
$\hat{\sigma}_{e_{II}} \mid \bar{\sigma}_{e_I} \mid \bar{\sigma}_{e_{II}}$	0x90	0x00

CLA	**INS**	**P1**	**P2**	**Le**
0x80	0x04	iter	no	size

Data	**W1**	**W2**
$e \mid s_v \mid s_i \mid s_{e_I} \mid s_{e_{II}} \mid s_{m_r} \mid s_{m_{z \notin D}}$	0x90	0x00

CLA	**INS**	**P1**	**P2**	**Le**
0x80	0x04	iter	no	size

Data	**W1**	**W2**
$s_{m_{z \notin D}}$	0x90	0x00

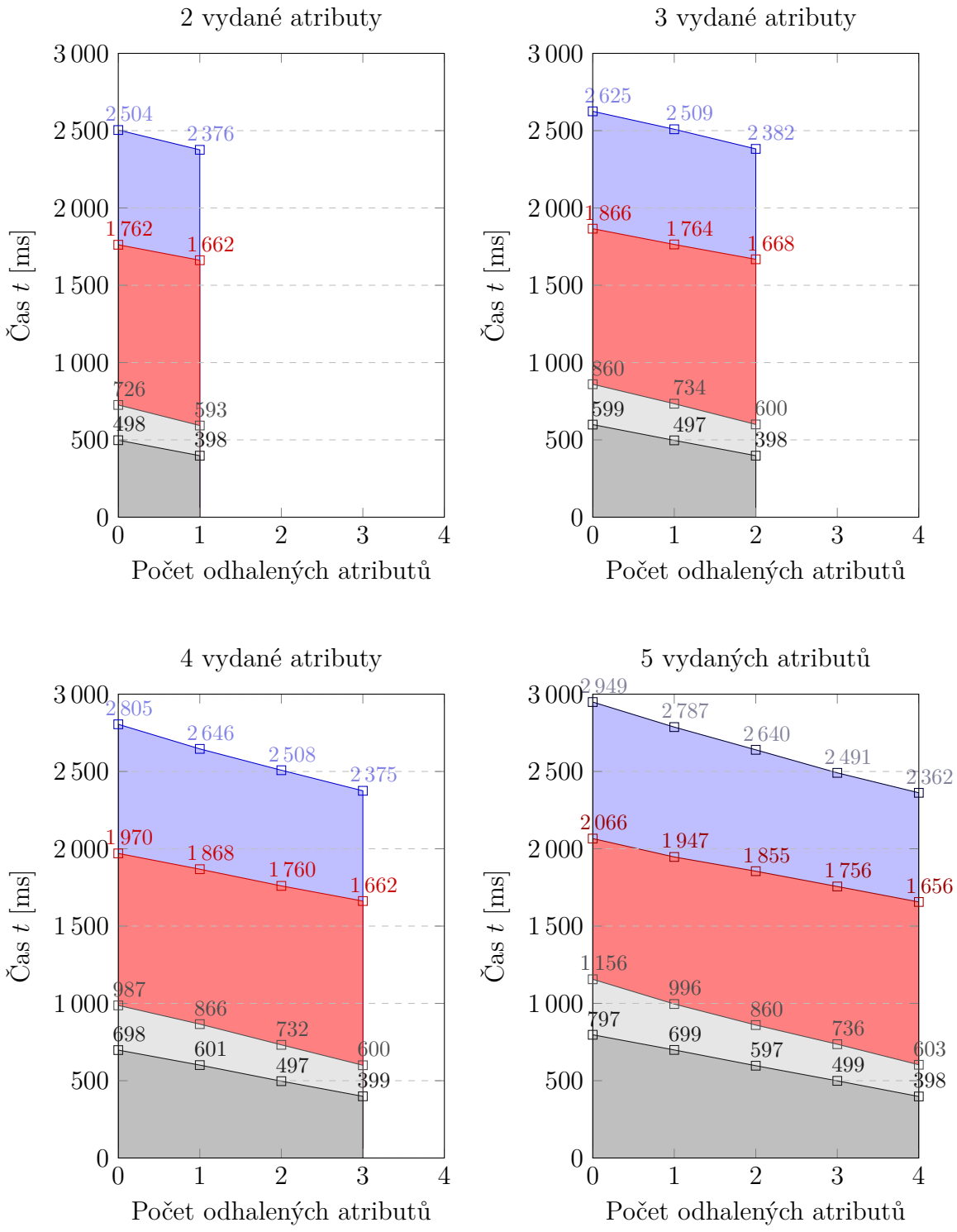


Obr. 7.7: Ověřovací protokol RKVAC – vývojový diagram.

7.3.4 Zhodnocení výsledků implementace

Revokovatelné schéma využívající anonymního atributového pověření RKVAC bylo implementováno v souladu se zadáním práce. Část textového výpisu z konzole, při běhu programu KBCT_RVKAC, lze nalézt v příloze B. Byla provedena měření času běhu kryptografického algoritmu `Show`, který je počítán na čipové kartě. Variabilními parametry byly celkový počet atributů a počet odhalených atributů. Naměřené hodnoty jsou vyneseny do grafů viz obrázek 7.8 a porovnány s implementací KVAC. Pro relevantní porovnání výsledků měření bylo nutné do vydaných atributů počítat také revokační handler m_{r}, který ovšem nemůžeme odhalit. Z tohoto důvodu je maximální počet odhalených atributů vždy menší o jedna. Časy pro schéma KVAC jsou vyznačeny šedou barvou, zbylé barvy červená a modrá jsou vyhrazeny pro implementaci RKVAC. Vyšší hodnoty pro danou implementaci značí měření včetně přenosu dat na čipovou kartu a zpět.

Dále byly provedeny funkční testy. V nich bylo otestováno, zdali uživatel nemůže podvádět a tvrdit, že je držitelem falešných atributů. Také bylo otestováno, zda uživatel nemůže podvrhnout revokační handler a stát se tak nerevokovatelný.



Obr. 7.8: Porovnání rychlostí ověřovacího protokolu RKVAC ve srovnání se schématem KVAC. Červená barva – čas algoritmu RKVAC, modrá – čas algoritmu RKVAC včetně komunikace, tmavě šedá – čas aloritmu KVAC, světle šedá – čas algoritmu KVAC včetně komunikace.

Závěr

Hlavním cílem této bakalářské práce byla funkční implementace přiděleného anonymního pověřovacího schématu na platformu čipových karet. Tato schémata jsou perspektivní pro použití v moderních autentizačních systémech. Vedoucím práce byla přidělena schémata využívající anonymního atributového pověření KVAC a jeho rozšíření o revokaci do schématu RKVAC. Pro dosažení cíle je v kapitole 3 provedena analýza současných platforem čipových karet a kryptografických knihoven z pohledu kryptografické podpory a výkonnosti. Na základě této analýzy byla pro implementaci vybrána platforma MultOS z důvodu široké podpory kryptografických operací na EC a rychlosti provádění těchto operací. V prvních kapitolách je uveden teoretický základ a popsány základní stavební bloky přidělených schémat.

V praktické části je nestíněna architektura jednotlivých schémat, blíže specifikovány a popsány použité protokoly a kryptografické algoritmy. Popis samotné implementace staví v první řadě na zadaných a zvolených parametrech systému. Použité parametry jsou řádně specifikovány. Pro každé ze zadaných schémat byly vytvořeny dvě aplikace. Jedna obsluhuje entitu uživatele a byla implementována na programovatelnou čipovou kartu Multos ML4. Druhá aplikace inicializuje vzájemnou komunikaci a zajišťuje funkce pro entity vydavatele a ověřovatele. Tato aplikace byla implementována na platformu Raspbian GNU/Linux. Aplikace KVAC schématu podporují vydání, správu a ověření až deseti uživatelských atributů. Schéma RKVAC se liší od KVAC ve schopnosti revokovat uživatele. V rámci RKVAC implementace byla podpora uživatelských atributů snížena na devět a desátý atribut je vyhrazen pro revokační handler, který nelze odhalit. Vytvořené aplikace byly popsány a byly přiblíženy jejich funkce a metody. Jelikož při běhu systému mezi sebou komunikují dvě aplikace navzájem, byla také zdokumentována struktura a časová posloupnost jednotlivých příkazů. Část ověřovacího protokolu, konkrétně algoritmus `show`, byl také podroben měření časové náročnosti. Z výsledků měření je patrné, že na výslednou efektivitu/rychlost protokolu má vliv počet atributů v pověření a počet odhalených atributů. Vztah je takový, že každým přidáním atributu (který se neodhaluje) do pověření snižujeme rychlost výpočtů o cca 100 ms. Naopak odhalením atributu se o tuto hodnotu čas zkrátí. Protokol je nejoptimálnější při počtu pěti vydaných atributů a dosahuje času 797 ms pro KVAC. Celková doba ověření, při odhalení pouze jednoho atributu, je 993 ms včetně výpočtů na Raspberry Pi, což je ještě pro uživatele přijatelná hodnota. Protokol RKVAC umožňuje revokaci, výpočetní nároky jsou však mnohonásobně vyšší, cca 2 s nárůst oproti KVAC.

Literatura

[1] Nařízení Evropského parlamentu a Rady (EU) 2016/679 ze dne 27. dubna 2016 o ochraně fyzických osob v souvislosti se zpracováním osobních údajů a o volném pohybu těchto údajů a o zrušení směrnice 95/46/ES (obecné nařízení o ochraně osobních údajů). In: *Úřední věstník Evropské unie*. Květen 2016, L119, s. 1-88. Dostupné také z URL:<`http://data.europa.eu/eli/reg/2016/679/oj`>

[2] Nařízení Evropského parlamentu a Rady (EU) č. 910/2014 ze dne 23. července 2014 o elektronické identifikaci a službách vytvářejících důvěru pro elektronické transakce na vnitřním trhu a o zrušení směrnice 1999/93/ES In: *Úřední věstník Evropské unie*. Srpen 2014, L257, s. 73-114. Dostupné také z URL:<`http://data.europa.eu/eli/reg/2014/910/oj`>

[3] BARKER, Elaine. *Recommendation for Key Management–Part 1: General* [online]. Revize 5. National Institute of Standards and Technology Special Publication, 2020 [cit. 2020-05-24]. DOI: 10.6028/NIST.SP.800-57pt1r5. Dostupné z URL:<`https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf`>

[4] HANKERSON, Darrel R., Scott A. VANSTONE a A. J. MENEZES. *Guide to elliptic curve cryptography*. New York: Springer, 2003. ISBN 0-387-95273-X.

[5] OCHODKOVÁ, Eliška. *Matematické základy kryptografických algoritmů* [online]. Ostrava, Plzeň: Vysoká škola báňská, Západočeská univerzita, 2011 [cit. 2020-05-24]. Dostupné z URL:<`http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/mat_zaklady_kryptografickych_algoritmu.pdf`>

[6] BONEH, Dan a Xavier BOYEN. Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. *Journal of Cryptology* [online]. 2008, **21**(2), 149-177 [cit. 2020-05-24]. DOI: 10.1007/s00145-007-9005-7. ISSN 0933-2790. Dostupné z URL:<`http://link.springer.com/10.1007/s00145-007-9005-7`>

[7] HAJNÝ J. Autentizace pomocí Zero-Knowledge protokolů. *Crypto-World* [online]. 2008, roč. 10, č. 9, s. 7-13 [cit. 2020-05-24]. ISSN 1801-2140. Dostupné z URL:<`http://crypto-world.info/casop10/crypto09_08.pdf`>

[8] DAMGARD, Ivan. *On Σ-protocols* [online]. V.2. CPT, 2010 [cit. 2020-05-26]. Dostupné z URL:<`https://www.cs.au.dk/~ivan/Sigma.pdf`>

[9] SCHNORR, C. P. Efficient signature generation by smart cards. *Journal of Cryptology*[online]. 1991, **4**(3), 161-174 [cit. 2020-05-26]. DOI:

10.1007/BF00196725. ISSN 0933-2790. Dostupné z URL:<http://link.springer.com/10.1007/BF00196725>

[10] MENEZES, Alfred J., Paul C.van OORSCHOT a Scott A. VANSTONE. *Handbook of applied cryptography*. Vyd. 1. Boca Raton: CRC Press, 1997, 780 s. ISBN 08-493-8523-7.

[11] BURDA, Karel. *Zabezpečovací systémy* [online]. 2011. Brno: Vysoké učení technické v Brně, 2011 [cit. 2020-02-21]. Dostupné z URL:<https://moodle.vutbr.cz/enrol/index.php?id=212562>

[12] RANKL, Wolfgang; EFFING, Wolfgang. *Smart card handbook*. 4th ed. Přeložil Kenneth COX. Chichester: John Wiley, 2010. ISBN 978-0470743676.

[13] Java Card Downloads. *Oracle*[online]. Redwood Shores: Oracle, 2019 [cit. 2020-02-21]. Dostupné z URL:<https://www.oracle.com/java/technologies/javacard-downloads.html>

[14] *MULTOS Developer-s Reference Manual* [online]. London: MAOSCO Limited, 2019 [cit. 2020-02-21]. MAO-DOC-TEC-006. Dostupné z URL:<https://www.multos.com/uploads/MDRM.pdf>

[15] IDPrime Smart Cards. *Gemalto World leader in Digital Security*[online]. Meudon Cedex: Gemalto NV, 2019 [cit. 2020-02-21]. Dostupné z URL:<https://safenet.gemalto.com/multi-factor-authentication/idprime-md-pki-smart-cards/>

[16] DZURENDA, Petr. *Kryptografická ochrana digitální identity*. Brno, 2019. Disertační práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací.

[17] DZURENDA, Petr, Sara RICCI, Jan HAJNY a Lukas MALINA. Performance Analysis and Comparison of Different Elliptic Curves on Smart Cards. In: *15th Annual Conference on Privacy, Security and Trust (PST)* [online]. Calgary, Canada: IEEE Computer Society, 2017, s. 1-10 [cit. 2020-04-08]. DOI: 10.1109/PST.2017.00050. ISBN 978-1-5386-2487-6. Dostupné také z URL:<https://ieeexplore.ieee.org/document/8476956/>

[18] LYNN, Ben. *The pairing-based cryptography (pbc) library*. [software]. [přístup únor 2020]. Dostupné z URL:<https://crypto.stanford.edu/pbc/>

[19] MIRACL UK Ltd. *Multiprecision Integer and Rational Arithmetic Cryptographic Library*. [software]. [přístup únor 2020]. Dostupné z URL:<https://github.com/miracl/MIRACL>

[20] KANAOKA, Akira. *Tepla elliptic curve and pairing library.* [software]. [přístup únor 2020]. Dostupné z URL:<https://github.com/TEPLA/tepla-library>

[21] ARANHA, D. F. a další. *RELIC is an Efficient LIbrary for Cryptography.* [software]. [přístup únor 2020]. Dostupné z URL:<https://github.com/relic-toolkit/relic>

[22] SHIGEO, Mitsunari. *Mcl library.* [software]. [přístup únor 2020]. Dostupné z URL:<https://github.com/herumi/mcl>

[23] ORACLE CORPORATION. *Java™ Smart Card I/O API.* [software]. [přístup únor 2020]. Dostupné z URL:<https://docs.oracle.com/javase/7/docs/jre/api/security/smartcardio/>

[24] ROUSSEAU, Ludovic. *PCSC lite project.* [software]. [přístup 2020]. Dostupné z URL:<https://pcsclite.apdu.fr>

[25] CAMENISCH, Jan, Manu DRIJVERS, Petr DZURENDA and Jan HAJNY. Fast Keyed-Verification Anonymous Credentials on Standard Smart Cards. In: *ICT Systems Security and Privacy Protection* [online]. Springer Nature Switzerland, 2019, s. 1-13 [cit. 2020-05-09]. ISBN: 978-3-030-22312-0. Dostupné z URL:<http://link.springer.com/10.1007/978-3-030-22312-0_20>

[26] CAMENISCH, Jan, Manu DRIJVERS and Jan HAJNY. Scalable Revocation Scheme for Anonymous Credentials Based on n-times Unlinkable Proofs. In: *WPES '16 Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society* [online]. NY, USA: ACM New York, 2016, s. 123-133 [cit. 2020-05-09]. ISBN: 978-1-4503-4569-9. Dostupné z URL:<https://doi.org/10.1145/2994620.2994625>

Seznam symbolů, veličin a zkratek

ABC	Attribute-Based Credentials
AES	Advanced Encryption Standard
AID	Application Identifier
ALU	Application Load Unit
APDU	Application Protocol Data Unit
API	Application Programming Interface
CLA	Class
DF	Dokazovací Faktor
EC	Eliptic Curve
ECC	Elliptic Curve Cryptography
ECDH	Elliptic-curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EEPROM	Electrically Erasable Programmable Read-Only Memory
eIDAS	electronic Identification, Authentication and Trust Services
GDPR	General Data Protection Regulation
ICC	Integrated Circuit Card
ID	Identifier
IEC	standards for International Electrotechnical Commission
INS	Instruction
ISO	the International Organization for Standardization
KVAC	Keyed-Verification Anonymous Credentials
L_c	Length command
L_e	Length expected
MAC	Message Authentication Code
MCL	portable and fast pairing-based Cryptography Library
MIRACL	Multiprecision Integer and Rational Arithmetic Cryptographic Library
NIST	National Institute of Standards and Technology
OSI	Open Systems Interconnection
PBC	Pairing Based Cryptography
PC	Personal Computer
PK	Proof of Knowledge
RA	Revokační Autorita
RAM	Random Access Memory
RELIC	Efficient LIbrary for Cryptography
RKVAC	Revocable Keyed-Verification Anonymous Credentials
ROM	Read-Only Memory

RSA	Rivest, Shamir, and Adelman
SC	Smart Card
SHA	Secure Hash Algorithm
SIM	Subscriber Identity Module
SPK	Signature Proof of Knowledge
SW	Status Word
TEPLA	University of Tsukuba Elliptic Curve and Pairing Library
USB	Universal Serial Bus
wBB	weak Boneh-Boyen
ZK	Zero-Knowledge

Seznam příloh

A Obsah přiloženého CD

- / ..... kořenový adresář přiloženého CD
  - kody_wBB ..... zdrojové kódy pro základní schémata
    - MULTOS
      - eloyalty_MACwBB_final.c
      - eloyalty_wBB_final.c
    - Java
      - MAC_wBB_final.java
      - wBB_final.java
  - kody_KVAC ..... zdrojové kódy pro schéma KVAC
    - MULTOS
      - eloyalty.alu
      - eloyalty.c
      - multosecc.h
      - scAppSetup.h
      - README.md
    - Raspberry_Pi
      - main.cpp
      - pcscRem.cpp
      - pcscRem.h
      - README.md
  - kody_RKVAC ..... zdrojové kódy pro schéma RKVAC
    - MULTOS
      - eloyalty.alu
      - eloyalty.c
      - multosecc.h
      - scAppSetup.h
      - README.md
    - Raspberry_Pi
      - main.cpp
      - pcscRem.cpp
      - pcscRem.h
      - README.md
  - xmmorav24_KBCT.pdf ..... text bakalářské práce

B Výpis z konzole při běhu programu

```
PC/SC
 (i) SCardEstablishContext: Command successful. (0x0)
Available readers:
0 OMNIKEY AG CardMan 3121 00 00
SCardConnect: Command successful. (0x0)
 State: 0x34
 Protocol: T1
 ATR (length 19 bytes): 3B 6F 00 00 80 31 E0 6B 84 31 02 07 0F 55 55 55 55 55 55

Select app:
  >>> 00 A4 04 00 04 F0 00 00 01 00
  <<< 90 00

Setup SC app:
  >>> 80 00 09 00 E0 10 0A 68 00 91 84 83 20 B0 28 B9 61 60 B9 9E A3 D8 4B 43 A1 52
      45 10 85 80 31 6C A0 0A 34 A6 06 21 D2 F9 C4 18 CC 85 D7 1F 54 29 6D 8E A8 4D
      9E FF 41 07 A2 EF 4F 08 F0 1C B8 35 1D 9F 4C 85 AF 18 68 AE AC 44 0F BF DD 3E
      90 71 97 77 85 3D DF BF 9F B2 D2 93 9D 4D FB F7 D7 E9 DB 15 04 0B 60 0D B9 D2
      83 45 3E 65 59 18 9D 8D 95 F7 50 4F 5B FF 01 C4 A7 49 05 3F 71 2D 36 4B 25 F7
      43 0B FD 11 A9 C5 33 6C 06 F3 10 7A F3 56 D4 81 B1 2F 95 6D E6 A6 CA 99 76 FE
      12 FC F6 46 49 59 FF 6B AA 1A 3B 78 09 E0 3D 66 A5 A9 2C E8 9E 49 16 3E 1D 7B
      13 48 F4 12 06 3C 4E F2 60 CC 6C F6 AA 73 64 0B 70 71 57 DE 4A 95 FF 1A 51 4F
      4B FB 48 60 05 49 E9 2B 4C 2E 05 F5 60 BE 3E E5 DF AD 2F 2D 3F 00
  <<< 91 AF
  >>> 80 00 09 07 40 1F 33 52 04 E8 6B 52 C4 28 33 7C 31 4F AF 49 1E 39 E7 11 E2 2D
      C6 2E AD 19 6E 08 CD 44 6C D7 A0 13 B6 D0 B8 6A EF B1 F6 11 80 83 BB C0 55 D2
      53 17 75 7C A2 53 5F 0E A5 85 6F 01 01 46 04 14 23 00
  <<< 90 00
(sk_I, params_I)<--MAC.Setup(1^k)
  <--
 MAC.Setup OK
(params_RA, sk_RA, pk_RA, RL)<--SetupRA(1^k, ver_max)
  >>> 80 30 00 00 15
  <<< 02 0F 84 31 FF FF FF FF FF FF 1A 2B 3C 4D 5E 6F 00 40 00 FF 01 90 00
  >>> 80 31 00 02 A1 05 98 E9 D5 AF 28 DA A8 1E 9F 4C EA 88 B5 AB 75 5F D4 A5 86 30
      EE 16 77 67 C3 C6 45 37 A1 E8 A0 04 16 D8 F8 96 D8 3F EB 00 3D 7C 01 33 2C A6
      C3 45 B2 4B AB 07 FC BE C9 FB 3F E3 02 1A 46 CD AC CE 0F 3D 2E 55 FB 4F 47 A0
      CA F4 DA 07 87 52 B5 A5 05 F2 5C C9 C1 84 00 C4 EE E5 41 48 A6 D9 68 E1 06 04
      9B 38 33 29 29 5B C4 12 FC 5A 90 B0 2C B4 83 33 9C 24 9A 08 2C 67 05 AD B9 79
      CB 18 C1 80 11 08 F5 2E 09 B0 74 DA EC A9 EE 50 3E 75 9C F4 2D AC 5D F5 BF 74
      E9 06 70 19 CE 20 B0 74 EC A7 00

                                        ⋮



Elapsed Time = 2970.38[ms]
Proof of knowledge ACCEPT!

RUN SUCCESSFUL (total time: 8s)
```