VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Jan Hlinka

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

KRYPTOGRAFIE NA VÝKONOVĚ OMEZENÝCH ZAŘÍZENÍCH
CRYPTOGRAPHY ON CONSTRAINED DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Jan Hlinka

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. Jan Hajný, Ph.D.

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Jan Hlinka ***ID:*** 203703

Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Kryptografie na výkonově omezených zařízeních

POKYNY PRO VYPRACOVÁNÍ:

Student na vybrané platformě výkonově omezených zařízení (Raspberry Pi) implementuje atributové autentizační schéma založené na wBB podpisu (CDH16, CDDH19). Toto schéma bude kompatibilní s aplikací pro Multos čipovou kartu. Student provede měření výkonosti implementovaných protokolů pro vydávání, prokazování a ověřování atributů.

DOPORUČENÁ LITERATURA:

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

[2] Debian - dokumentace [online]. [cit. 2019-09-06]. Dostupné z: https://www.debian.org/doc/#manuals

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

Vedoucí práce: doc. Ing. Jan Hajný, Ph.D.

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

ABSTRAKT

Tato práce popisuje implementaci strany ověřovatele, revokační autority a vydavatele pro atributovou autentizaci pomocí čipové karty a terminálu v podobě výkonově omezeného zařízení. Vybrané schéma, RKVAC, slouží k realizaci atributové autentizace, tedy autentizace se záměrem ochránit citlivá osobní data uživatele. Schéma vyhovuje principům Privacy by Design a Privacy by Default, které jsou součástí nařízení, jako je například GDPR (General Data Protection Regulation) a eIDAS (Electronic Identification and Trust Services).

KLÍČOVÁ SLOVA

atributová autentizace, elektronický podpis, eliptické křivky, Raspberry Pi 3 model B+, čipové karty, weak Boneh-Boyen, RKVAC

ABSTRACT

This thesis describes the implementation of a verifier, revocation authority and issuer side of an attribute authentication algorithm, using device with limited performance capabilities. Chosen scheme, RKVAC, functions as an attribute authentication protocol. Which are a protocols designed to protect private user data. RKVAC Scheme complies with Privacy by Design and Privacy by Default principles, which are a part of regulations such as GDPR (General Data Protection Regulation) and eIDAS (Electronic Identification and Trust Services).

KEYWORDS

Attribute authentication, digital sign, elliptic curves, Raspberry Pi 3 model B+, smart cards, weak Boneh-Boyen, RKVAC

HLINKA, Jan. *Kryptografie na výkonově omezených zařízeních*. Brno, 2020, 40 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Jan Hajný, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Kryptografie na výkonově omezených zařízeních“ 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 semestrální práce panu doc. Ing. Janu Hajnému, Ph.D za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále děkuji panu Ing. Petru Dzurendovi, Ph.D za odbornou pomoc, konzultace a podnětné návrhy při vypracování praktické části mé bakalářské práce.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Anonymní autentizace slouží k ochraně osobních dat uživatele při jejich využití k autentizaci. Systémy anonymní autentizace vyhovují principům Privacy by Design a Privacy by Default, které jsou kořenovou součástí nařízení, jako je například *General Data Protection Regulation* (GPDR) a *Electronic Identification and Trust Services* (eIDAS). Jedním z používaných typů anonymní autentizace je atributová autentizace, jejíž implementací na výkonově omezeném zařízení se zabývá tato práce.

První část práce popisuje princip atributové autentizace, její protokoly, entity a základní vlastnosti. Dále se práce podrobněji zabývá popisem podpisového schématu wBB, algebraického *Message Authentication Code* (MAC) s využitím wBB podpisu a jejich využití ve schématu atributové autentizace *Keyed-Verification Anonymous Credentials* (KVAC).

Praktická část se zabývá implementací strany vydavatele a ověřovatele pro atributovou autentizaci v jazyce C. Prvkem této části je také stručný popis platformy Raspberry Pi, na níž je aplikace implementována, operačního systému Raspbian, vývojového prostředí NetBeans a nástrojů využitých při implementaci algoritmů wBB a MAC_{wBB} ve schématu atributové autentizace KVAC. Následně jsou popsány důležité části zdrojového kódu vytvořené aplikace. Jako první je vysvětlen princip komunikace mezi zařízením Raspberry Pi a čipovou kartou. Dále je popsána implementace strany vydavatele, revokační autority a ověřovatele ve schématu atributové autentizace KVAC.

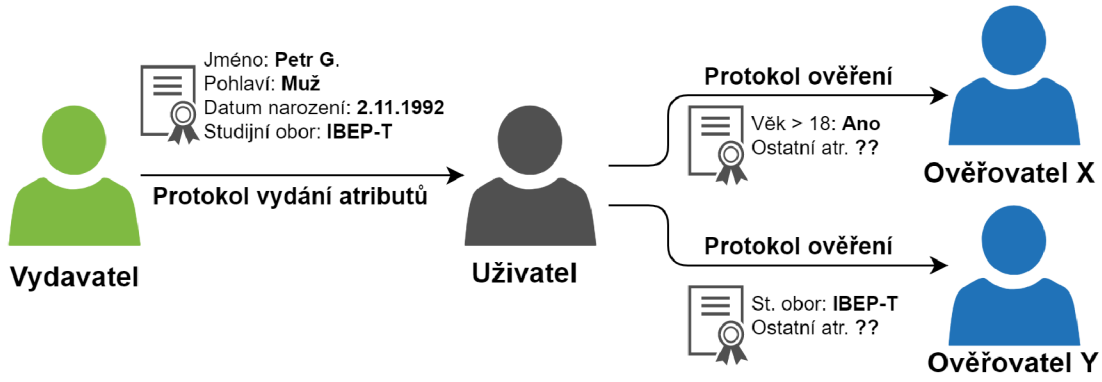
1 Atributová autentizace

Základní myšlenkou anonymní autentizace [1] je ochrana osobních informací uživatele a jeho digitální identity v přístupových systémech. Systémy anonymní autentizace respektují principy Privacy by Design a Privacy by Default. Tyto principy jsou kořenovou součástí nařízení, jako je například GDPR a eIDAS. Systémy atributové autentizace stále využívají některé osobní informace uživatele, ale pouze vybrané informace nezbytné k provozu systému. K ochraně soukromí uživatelů v systémech anonymní autentizace obvykle slouží tyto prostředky:

- **Anonymizace** [1] – Anonymizace se zabývá ochranou dat, pomocí kterých by mohlo dojít k identifikaci uživatele. Jedním z prostředků ochrany citlivých dat za pomocí anonymizace v praxi je šifrování osobních údajů. Při využití této metody je třeba data šifrovat jak v centrální databázi, tak při samotném přenosu. Důsledkem použití tohoto řešení je potřeba uložit potřebné kryptografické klíče na všech zařízeních, tudíž jsou zvýšeny nároky na zabezpečení a je vyžadována dodatečná správa klíčů.
- **Pseudonymizace** [1] – Představuje záměnu jednoznačného identifikátoru za pevně určený pseudonym. Zvolený pseudonym je stále jednoznačným identifikátorem, ale není již přímo spojitelný s identitou uživatele. Ověřovatelé tedy nejsou schopni konkrétně identifikovat uživatele. Ověřovatelé jsou pouze schopni poznat, zda se jedná o právoplatného uživatele systému. Systémy využívající pseudonymizaci vyžadují dodatečné databázové zařízení pro mapování pseudonymů s reálnými identifikátory uživatelů, tj. rodnými čísly, bydlišti a podobnými daty. Při využití této praktiky vzniká otázka vztahující se k umístění a dodatečnému zabezpečení zařízení pro mapování citlivých informací uživatelů. Ideální řešení této situace vyžaduje využití administrátorů určených pro správu pouze tohoto zařízení.
- **Atributová autentizace** [1] – Autentizace, kde se uživatel prokazuje pomocí vybraných atributů namísto osobního identifikátoru. Atributem může být například věk uživatele. Atributy jsou uloženy v certifikované formě na uživatelském zařízení. Nevýhodou je výpočetní náročnost a potřeba využít pokročilé mechanismy asymetrické kryptografie. Jedná se o nejnovější přístup vyhovující principům Privacy by Design a Privacy by Default. Tato práce je zaměřena na tuto metodu anonymní autentizace.

1.1 Úvod do atributové autentizace

Ve schématech atributové autentizace [1] není ověřována identita konkrétního uživatele jako v obvyklých autentizačních systémech. Namísto identity uživatele je ověřováno vlastnictví zvoleného atributu (vlastnosti). Vzorem ověřovaných atributů je například věk uživatele, studijní obor, řidičské oprávnění atd. Na základě této vlastnosti je možno tvrdit, že atributová autentizace přirozeně splňuje předpoklady k ochraně osobních údajů uživatele. Zajišťuje nespojitelnost jednotlivých ověření. Ověřování vlastnictví atributu nevyžaduje dodatečné osobní údaje k autentizaci. Osobní údaje mohou být jednoduše nahrazeny atributem. Například datum narození je zaměnitelné za atribut "Starší 18 let" a podobně. Uživatel, který úspěšně prokáže vlastnictví požadovaného atributu, obdrží přístup, a stále si zachová anonymitu v rámci systému. V případě porušení stanovených pravidel systému může být uživatel, který pravidla porušil, ze systému revokován. Revokace je ochranný mechanismus, který vybraného uživatele ze systému vyloučí nebo odhalí jeho identitu. Obrázek 1.1 znázorňuje příklad atributové autentizace. Vydavatel předá uživateli certifikované atributy na zvolené médium. Přidělené atributy dále uživatel prezentuje při ověření. Ověřovatel kontroluje pouze atributy relevantní pro objekt, ke kterému se uživatel snaží získat přístup. Ostatní atributy v rámci ochrany soukromí nekontroluje.



Obr. 1.1: Schéma základní atributové autentizace

Příklady využití systému atributové autentizace:

- Prokázání věku – Uživatel prokáže vlastnictví atributu dokazujícího plnoletost namísto odhalení data narození.
- Přístup do prostoru – Uživatel prokáže vlastnictví atributu dokazujícího způsobilost pro vstup do daného prostoru.
- Nárok na slevu – Uživatel prokáže vlastnictví atributu nároku pro slevu, jakým například může být: student, dítě a senior. Ověření tohoto atributu je provedeno bez odhalení data narození uživatele.

1.2 Entity a vlastnosti atributové autentizace

Základní topologie systémů atributové autentizace zahrnuje tyto entity[1]:

- **Uživatel (U)** (User) – Entita prokazující vlastnictví svých atributů (vlastností). K prokázaní slouží vhodné technologie (SW, HW, čipová karta, token).
- **Ověřovatel (V)** (Verifier) – Entita ověřující zvolené atributy uživatele žádajícího o přístup ke konkrétní službě nebo prostoru. K ověření je použito vhodné programové a hardwarové vybavení.
- **Veřejná autorita (A)** (Public authority) – Entita spravující uživatele a atributy. Některé systémy rozdělují veřejné autority na dvě samostatné entity:
  - **Vydavatel (I)** (Issuer) – Důvěryhodná autorita, vydává platné a relevantní atributy uživatelům.
  - **Revokační autorita (RA)** (Revocation authority) – Důvěryhodná třetí strana schopná revokovat a zneplatnit uživatelské atributy. V některých systémech je rovněž schopná odkrýt identitu uživatele, který porušil pravidla systému.

Základními čtyřmi protokoly/fázemi, ze kterých se obecně systémy atributové autorizace skládají, jsou:

- **Setup** – Fáze zahrnuje nastavení pravidel, generování kryptografických parametrů, klíčů a jejich případnou distribuci.
- **Vydávací protokol/Vydání atributu** – Nový uživatel je registrován u vydavatele nebo veřejné autority. Po úspěšné registraci jsou uživateli přiděleny kryptograficky zabezpečené atributy a další parametry.
- **Ověřovací protokol/Prokázání atributu** – Uživatel prokazuje vlastnictví svých atributů ověřovateli bez zveřejnění své identity. Po úspěšném ověření je uživateli poskytnut přístup do systému či k vybrané službě.
- **Revokační protokol/Odstranění atributu** – Poruší-li uživatel v systému stanovená pravidla, pak ověřovatel předá data z protokolu prokázání atributu revokační autoritě s možností revokace. Po posouzení incidentu dojde k vyloučení uživatele ze systému nebo také k odhalení jeho identity.

Bezpečnostní vlastnosti a vlastnosti zvyšující ochranu soukromí schémat atributové autentizace:

- **Anonymita** (Anonymity) – Uživatel prokazuje vlastnictví atributů bez odhalení vlastní identity.
- **Nespojitelnost** (Unlinkability) – Jednotlivé relace protokolu prokázaní vlastnictví atributu jsou vzájemně nespojitelné. Ověřovatelé nemohou určit, zda dané relace patří jednomu, nebo více uživatelům.
- **Nesledovatelnost** (Untraceability) – Vydané certifikované atributy jsou randomizovány, a vydavatel tak nemůže spojit konkrétní přístupy a identifikovat uživatele systému.
- **Nepřenositelnost** (Non-transferability) – Uživatel má unikátní privátní klíč, který je uložen na bezpečném úložišti.
- **Selektivní odhalení atributů** (Selective disclosure of attributes)– Uživatel si může zvolit, jaké atributy během protokolu prokázání odhalí.
- **Revokace** (Revocation) – Revokační autorita může odvolávat ze systému nebo i odhalit identitu uživatele.

Anonymizace a pseudonymizace nedisponují všemi výše uvedenými vlastnostmi.

1.3 Kryptografické metody v atributové autentizaci

Tato část se zabývá popisem základních kryptografických metod [1] a stavebních bloků pro schémata atributové autentizace.

1.3.1 Kryptografické závazky

V rámci komunikace se entita zavazuje k dané tajné hodnotě (číslu), která během komunikace s protější stranou není odhalena. Zavázaná hodnota tedy zůstává skrytá. Závazek [2] zabraňuje zavazující se entitě změnit zavázanou hodnotu v pozdější fázi protokolu. Kryptografický závazek je značen jako $c = commit(r, w)$, kde r je náhodné číslo a w je tajná hodnota. Závazková funkce *commit* může být například modulární mocnění, pro které platí $c = g^w \bmod p$, kde g je generátor prvočíselné grupy, p je velké prvočíslo a w je tajemství, ke kterému se uživatel zavazuje. Zpětný výpočet w za využití c je velice složitý. Jedná se o problém diskrétního logaritmu. Závazky poskytují následující vlastnosti:

- **Skrytí** – Pro ověřovatele je těžké/nemožné získat ze závazku c tajnou hodnotu w.
- **Svázání** – Pro uživatele je těžké/nemožné nalézt stejný závazek ke dvěma různým hodnotám.

1.3.2 Protokoly s nulovou znalostí

Protokoly nulové znalosti – *Zero Knowledge* (ZK) [2] zajišťují bezpečnost osobních informací. ZK protokoly řeší, zda uživatel zná, nebo nezná tajnou informaci. Uživatel tedy prokazuje znalost tajné informace, kterou například může být heslo. Během komunikace s ověřovatelem ovšem tuto informaci neodhalí. Protokoly s nulovou znalostí musí splňovat tyto tři vlastnosti:

- **Úplnost** – Uživatel, který zná tajnou informaci, je vždy schopen vytvořit správnou dopověď na výzvu ověřovatele, a tedy být úspěšně ověřen.
- **Spolehlivost** – Uživatel, který nezná tajnou informaci, je vždy ověřovatelem zamítnut. Není schopen přesvědčit ověřovatele o znalosti tajné informace.
- **Nulová znalost** – Ověřovatel zjistí pouze, zda uživatel zná tajnou informaci. Žádné osobní informace nejsou při komunikaci odhaleny.

ZK protokoly je možné rozdělit na interaktivní a neinteraktivní. Interaktivní protokoly dodržují tento sled zpráv: závazek od uživatele, výzva od ověřovatele a odpověď od uživatele. Neinteraktivní protokoly komunikaci mezi entitami nezahrnují. Uživatel zašle důkaz pouze v jedné zprávě.

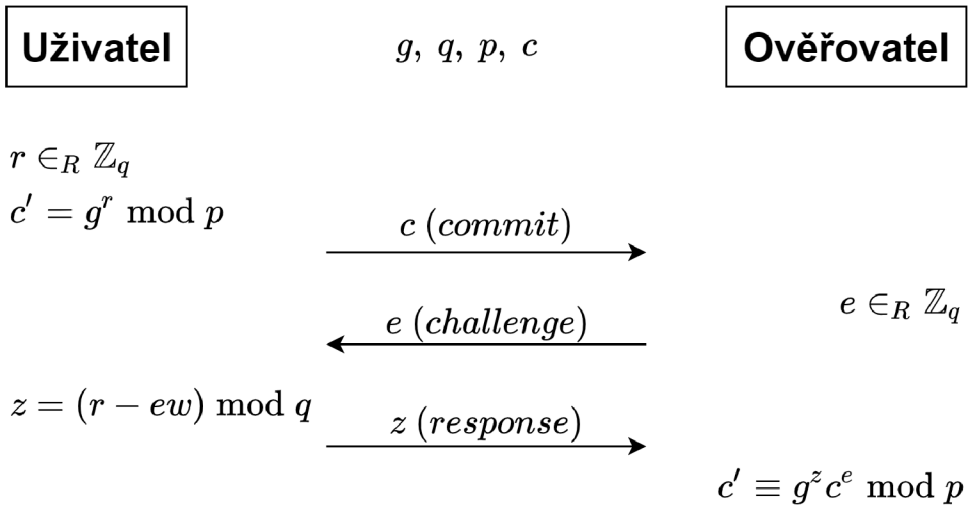
1.3.3 Sigma protokoly

Sigma protokol (Σ -protokol) [1], stejně jako ZK protokoly, slouží k prokázání znalosti tajné informace bez jejího zveřejnění. Σ -protokoly jsou ovšem rychlejší a efektivnější než ZK protokoly. Jejich vlastnosti je tedy činí vhodnějšími pro reálnou implementaci. Základní vlastnosti Σ -protokolů jsou totožné s vlastnostmi ZK protokolů s výjimkou příbytku třícestnosti (3-way). Třícestnost definuje způsob komunikace mezi uživatelem a ověřovatelem. Komunikace proběhne v rámci tří zpráv. První zprávou je zpráva commit (závazek) od uživatele ověřovateli, kdy se uživatel zavazuje k jedné určité tajné hodnotě. Druhá zpráva je výzva ověřovatele pro uživatele (challenge). Poslední zpráva je odpovědí (response) uživatele ověřovateli. Díky posledním dvěma zprávám je ověřovatel schopen poznat, zda uživatel zná tajnou hodnotu.

1.3.4 Důkazy znalosti diskrétního logaritmu

PKDL [1] slouží k prokázání znalosti tajné hodnoty w, která je součástí závazku. Vše ovšem probíhá bez zveřejnění dané tajné hodnoty. Následujícím příkladem je Σ -protokol založen na Schnorrově protokolu. V příkladu na obrázku 1.2 jsou sdíleny veřejné parametry g, p, q a c mezi stranami, kde g je generátorem grupy, p je prvočíslem, q je řádem grupy a c je závazkem. Závazek c je stanoven za pomocí vztahu: $c = g^w \bmod p$. Závazek je určen a sdílen mezi stranami před začátkem

ověření znalosti w. Prvním krokem je závazek uživatele c' k náhodné hodnotě r. V druhém kroku uživatel obdrží výzvu od ověřovatele v podobě náhodné hodnoty e, na kterou odesílá v poslední zprávě odpověď z. Na základě poslední kontroly ověřovatel zjistí, zda uživatel zná tajnou hodnotu w.



Obr. 1.2: Příklad PKDL

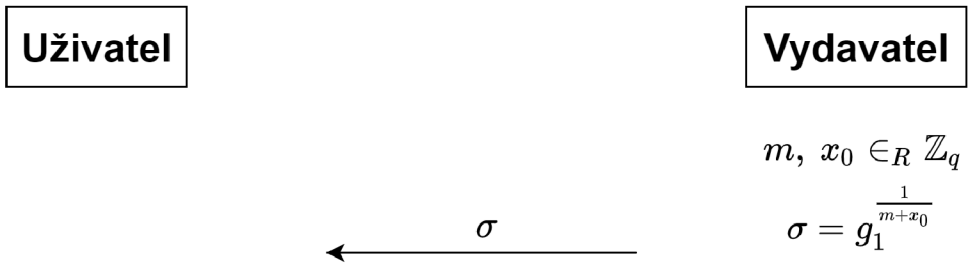
1.3.5 Podpisové schéma weak Boneh-Boyen

Podpisové schéma wBB [1] zajišťuje nespojitelnost podpisu uživatele s podpisem původním za pomocí randomizace. Podpis zůstává po randomizaci stále platný. Toto je zásadní pro ochranu soukromí uživatele. Jednou ze získaných vlastností randomizací podpisu je nesledovatelnost. Nesledovatelnost znemožňuje ověřovateli spojit nový podpis uživatele s podpisem původním. Druhou získanou vlastností je dříve zmíněná nespojitelnost. Schéma pracuje s eliptickou křivkou. Schéma je možno rozdělit na následující sekce:

- **Setup** – Algoritmus na základě stanovených podmínek vygeneruje veřejné parametry – cyklické grupy $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$, zvolí náhodný soukromý klíč $sk : r \in_{\mathrm{R}} \mathbb{Z}_{\mathrm{q}}$ a určí veřejný klíč $pk = g_2^{sk}$. Tyto stanovené parametry jsou dále předány všem stranám algoritmu (vydavatel, ověřovatel a uživatel).
- **Sign** – Algoritmus vypočte podpis $\sigma = \frac{1}{g_1^{x+m}}$ za pomocí zprávy $m \in_{\mathrm{R}} \mathbb{Z}_{\mathrm{q}}$ a soukromého klíče sk.
- **Verify** – Platnost podpisu je ověřena algoritmem za pomocí zprávy m a veřejného klíče pk. Podpis je platný, když $e(\sigma, pk) \cdot e(\sigma_{\mathrm{m}}, g_2) = e(g_1, g_1)$ (operace $e()$ značí bilineární párování).

- **Prove** – Slouží k prokázání autenticity vytvořených hodnot. Za pomocí náhodné hodnoty $r \in_{\mathrm{R}} \mathbb{Z}_{\mathrm{q}}$ je spočten nový podpis $\hat{\sigma} = \sigma^r$ a pomocná hodnota $\bar{\sigma} = \hat{\sigma}^{-m} g_1^r$. Pomocí Σ -protokolu se prokáže, zda bylo postupováno dle algoritmu.

Využití **wBB** podpisu umožňuje prokázat znalost podpisu bez jeho zveřejnění, což značně přispívá k ochraně soukromí.



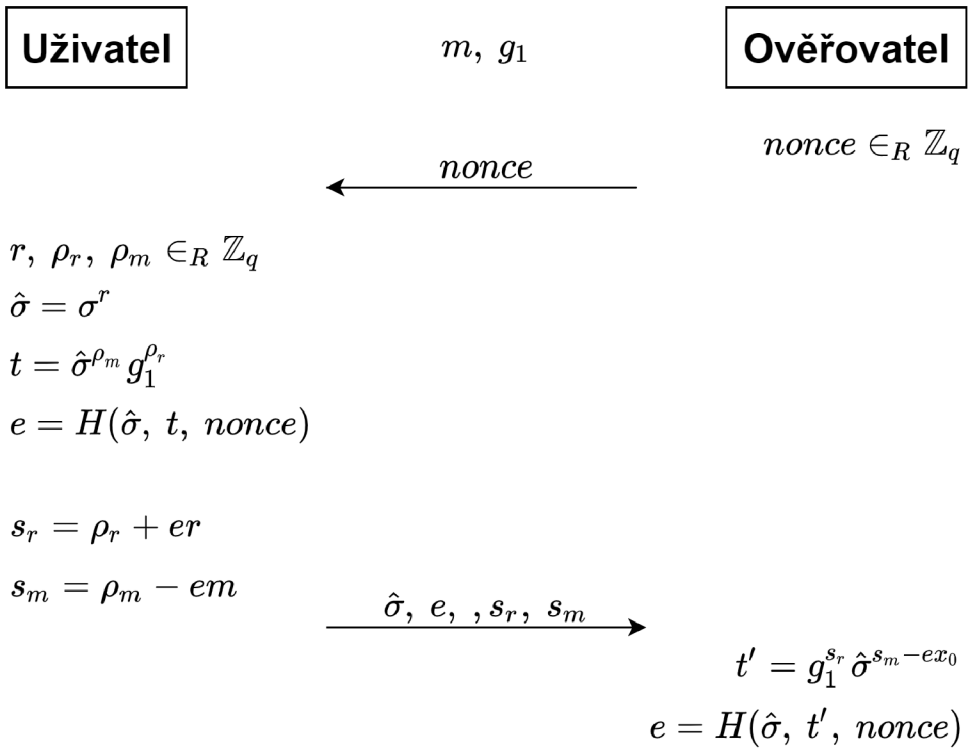
Obr. 1.3: Vydání wBB

Obrázky 1.3 a 1.4 znázorňují optimalizované schéma pro ověření znalosti podpisu wBB. Schémata se liší oproti předchozímu popisu algoritmu wBB ve fázi ověření (verify). Za účelem usnadnění implementace algoritmu pro ověření znalosti podpisu wBB byla nahrazena operace bilineárního párování.

1.3.6 Algebraický MAC s využitím wBB podpisu

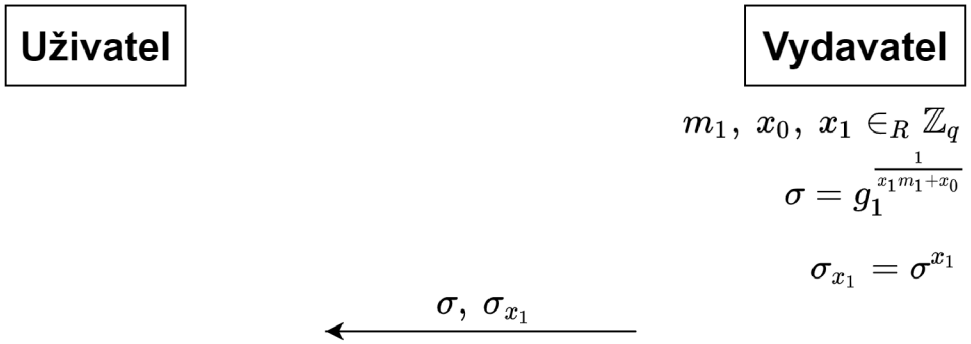
Algebraický MAC [1] je symetrický algoritmus využívající wBB podpis definovaný v části 1.3.5. Tento algoritmus zajišťuje integritu a autentičnost bloku zpráv. Podobně jako wBB podpis je algoritmus MAC_{wBB} rozdělen na několik částí. Konkrétně se jedná o následující algoritmy:

- **Setup** – Algoritmus na základě určených podmínek vygeneruje veřejné parametry – cyklické grupy $(q, G_1, G_2, G_{\mathrm{T}}, g_1, G_2, e)$, zvolí náhodné $(x_0, x_1, \ldots, x_{\mathrm{n}}) \in_R \mathbb{Z}_{\mathrm{q}}$ jako klíč a určí $(x_0, x_1, \ldots, x_{\mathrm{n}}) = (g_2^{x_0}, \ldots, g_2^{x_{\mathrm{n}}})$. Tyto stanovené parametry jsou dále předány všem stranám algoritmu (vydavatel, ověřovatel a uživatel).
- **Sign** – Algoritmus vypočte podpis $\sigma = g_1^{\frac{1}{x_0 + m_1 x_1 + \ldots + m_{\mathrm{n}} x_{\mathrm{n}}}}$ za pomocí zpráv $(m_1, \ldots, m_{\mathrm{n}}) \in_{\mathrm{R}} \mathbb{Z}_{\mathrm{q}}$ a soukromého klíče sk a pomocné hodnoty $(\sigma_{\mathrm{x_0}}, \ldots, \sigma_{\mathrm{x_n}}) = (\sigma^{x_0}, \ldots, \sigma^{x_{\mathrm{n}}})$. Podpis a pomocné hodnoty jsou předány uživateli a ověřovateli při vydání podpisu.

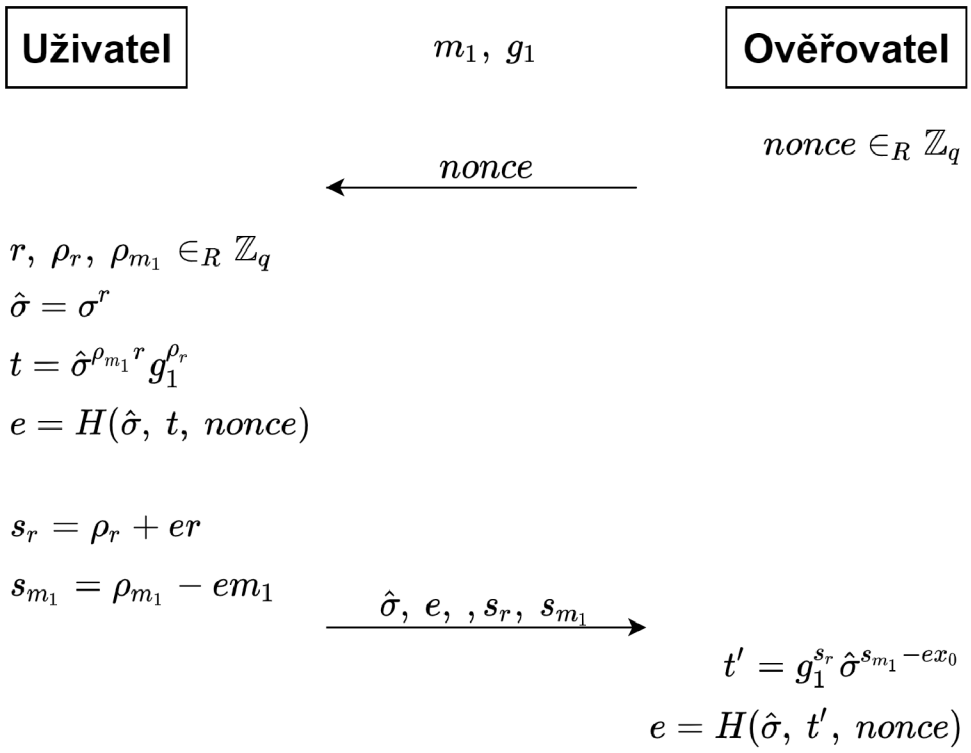


Obr. 1.4: Ověření znalosti wBB podpisu

- **Verify** – Platnost podpisu je ověřena algoritmem za pomocí podpisu σ, zpráv $(m_1, \ldots, m_\mathrm{n})$ a klíče sk. Podpis je platný, když platí $g = \sigma^{x_0 + m_1 x_1 + \ldots + m_\mathrm{n} x_\mathrm{n}}$.
- **Prove** – Je stanoven randomizační element $r \in_\mathrm{R} \mathbb{Z}_\mathrm{q}$ a pomocí něj je vytvořen podpis $\hat{\sigma} = \sigma^r$ a pomocné hodnoty $(\hat{\sigma}_{\mathrm{x}_0}, \ldots, \hat{\sigma}_{\mathrm{x}_\mathrm{n}}) = (\hat{\sigma}^r_{\mathrm{x}_0}, \ldots, \hat{\sigma}^r_{\mathrm{x}_\mathrm{n}})$. Pomocí Σ -protokolu $(\hat{\sigma}_{\mathrm{x}_0} = g^r \hat{\sigma}^{-m_1}_{\mathrm{x}_1}, \ldots, \hat{\sigma}^{-m_\mathrm{n}}_{\mathrm{x}_\mathrm{n}})$ se prokáže, zda bylo postupováno dle algoritmu.



Obr. 1.5: Vydání MAC_wBB



Obr. 1.6: Ověření znalosti kódu MAC_{wBB}

Obrázky 1.5 a 1.6 znázorňují schéma algoritmu, který byl implementován v praktické části semestrální práce. Jedná se o algoritmus pro ověření znalosti podpisu. Toto schéma je připraveno pro ověření více atributů. V rámci této implementace byl však použit pouze jeden.

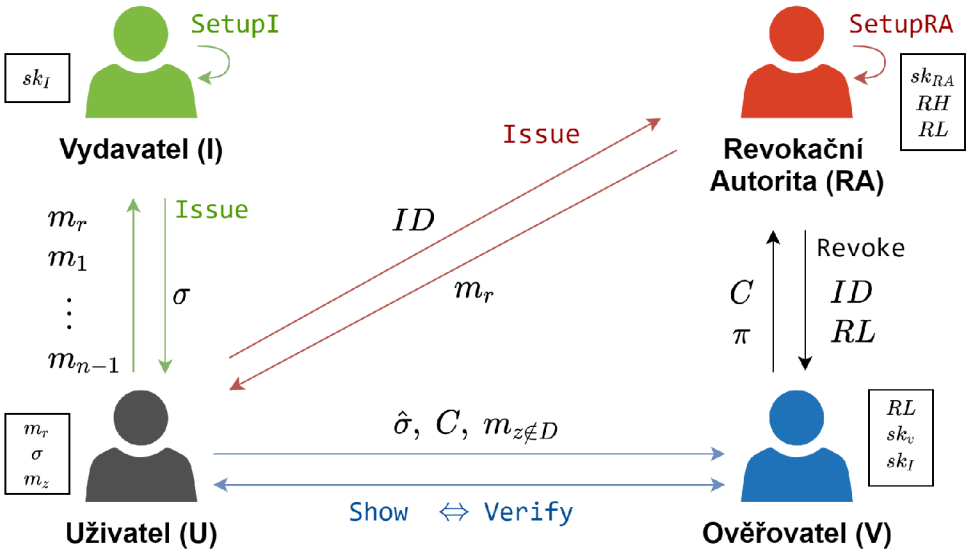
1.4 Implementované schéma RKVAC

Schéma atributové autentizace [1], implementované v této bakalářské práci, využívá oba dříve definované algoritmy (MAC a wBB). Jedná se o schéma atributové autentizace s revokací. Systém byl navržen s cílem vyřešit nedostatky stávajících systémů (U-Prove, Idemix). Revokační mechanizmy tohoto systému umožňují přímou revokaci, odstranění ze systému, de-anonymizaci uživatele. Tyto mechanizmy je možno po menších úpravách použít v rámci jiných schémat atributové autentizace. Toto schéma poskytuje uživateli veškeré vlastnosti pro ochranu soukromí, které jsou definovány v části 1.2. Systém je navržen tak, aby všechny operace strany uživatele bylo možné provést na čipových kartách, tudíž veškeré výpočetně náročnější operace jsou prováděny na straně ověřovatele.

1.4.1 Struktura implementovaného schématu

Struktura tohoto schématu je shodná s obecnou strukturou schématu atributové autentizace. Entity v tomto schématu jsou tedy totožné s entitami blíže definovanými v sekci 1.2. Zmíněné entity dále vykonávají následující algoritmy:

- **SetupI/SetupRA** - Generování veřejných parametrů a klíčů. Tyto algoritmy provádí strana vydavatele (I) a revokační autority (RA).
- **Issue** - Vydání osobních atributů uživateli. Protokol provádí vydavatel (I) a revokační autorita (RA).
- **Show** $\leftrightarrow$ **Verify** - Uživatel (U) prokazuje vlastnictví osobních atributů, aniž by dané atributy odhalil ověřovateli (V).
- **Revoke** - Revokace a zneplatňení uživatele (U). Protokol je prováděn revokační autoritou (RA).



Obr. 1.7: RKVAC schéma

1.4.2 Popis použitých kryptografických protokolů

Tato sekce podrobně popisuje použité kryptografické protokoly v implementovaném schématu.

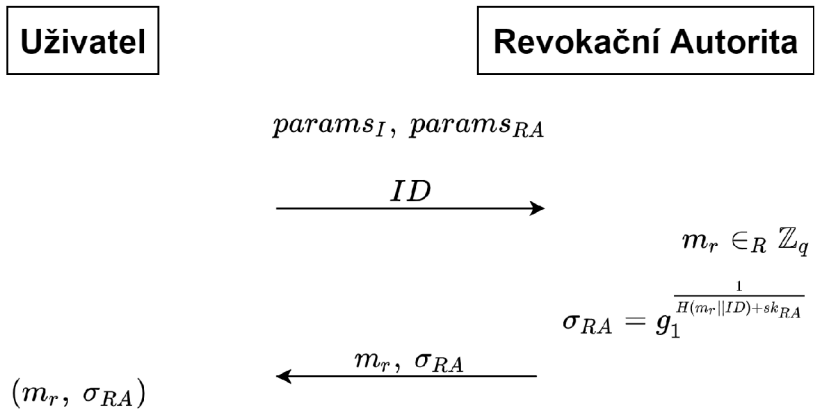
- **SetupI** - Vstupem je bezpečnostní parametr κ. V rámci této práce je tímto parametrem předem stanovená eliptická křivka BN-256. Výstupem je klíč vydavatele $sk_I = (x_0, \ldots, x_n) \in_R \mathbb{Z}_q$ a parametry $params_I = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$, které jsou vstupem všech následujících algoritmů. Tento protokol je prováděn vydavatelem (I).

- **SetupRA** - Vstupním parametrem κ je předem určená eliptická křivka BN-256. V rámci tohoto protokolu je určen maximální počet ověřovacích relací za jednu časovou epochu $ver_{max} = k^j$. Výstupem protokolu jsou parametry $params_{RA} = (k, j, (\alpha_1, \ldots, \alpha_j), (h_1, \ldots, h_j), \{(e_1, \sigma_{e_1}), \ldots, (e_k, \sigma_{e_k})\})$, které jsou vstupem všech následujících protokolů. Jednotlivé parametry $params_{RA}$ jsou získány následujícím způsobem:
  - Náhodné hodnoty $(\alpha_1, \ldots, \alpha_j) \in_R \mathbb{Z}_q$, pomocí kterých jsou vypočteny body $(h_1, \ldots, h_j)$, kde $h_j = g_1^{\alpha_j}$. Tyto body jsou dále použity při ověřování.
  - Pár klíčů (sk_{RA}, pk_{RA}), kde klíč sk_{RA} je náhodnou hodnotou ($sk_{RA} \in_R \mathbb{Z}_q$), pomocí které je dále stanoven klíč $pk_{RA} = g_2^{sk_{RA}}$.
  - Náhodné hodnoty $(e_1, \ldots, e_j) \in_R \mathbb{Z}_q$, které jsou podepsány a vzniknou podpisy $(\sigma_{e_1}, \ldots, \sigma_{e_k})$, kde $\sigma_{e_k} = \frac{1}{e_k + sk_{RA}}$.
- **Issue** - Vydání parametrů je provedeno revokační autoritou (RA), která vydá uživateli revokační handlery (m_r, σ_{RA}) a vydavatelem(I), který uživateli předá podpisy $(\sigma, \sigma_{x_1}, \ldots, \sigma_{x_{n-1}}, \sigma_{x_r})$ na jednotlivé atributy $(m_1, \ldots, m_{n-1}, m_r)$. Dané hodnoty jsou získány následujícím způsobem:
  - Hodnoty vydané revokační autoritou: m_r je náhodnou hodnotou, pomocí které je vypočten podpis $\sigma_{RA} = g_1^{\frac{1}{H(m_r||ID)+sk_{RA}}}$.
  - Podpisy vydané vydavatelem: Uživatel (U) předá vydavateli (I) své atributy $(m_1, \ldots, m_{n-1}, m_r)$. Na základě těchto atributů je vydavatelem následně vytvořen podpis $\sigma = g_1^{\frac{1}{x_0 + m_1 x_1 + \ldots + m_{n-1} x_{n-1} + m_r x_r}}$. Posledním krokem je vytvoření podpisů pro každý z klíčů $\sigma_{x_{(1,\ldots,n-1,r)}} = \sigma^{x_{(1,\ldots,n-1,r)}}$.

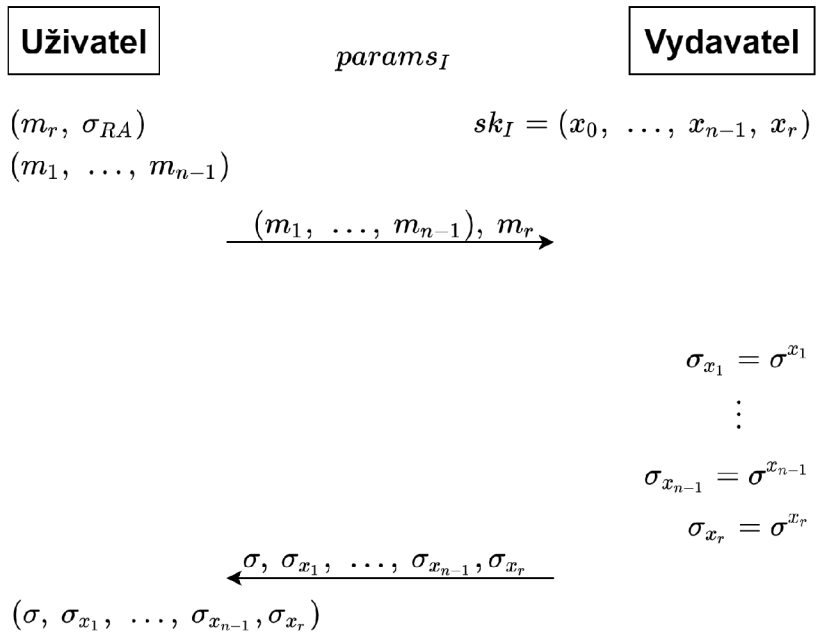
  Všechny výstupní parametry Issue protokolu jsou trvale uloženy na přenosném zařízení uživatele.
- **Show** - Uživatel obdrží od ověřovatele výzvu *nonce* a údaj o aktuálním čase *epoch*. Za pomocí unikátní hodnoty i spočtené z veřejných parametrů RA (náhodné hodnoty e_k a hodnot α_j), hashe hodnoty $H(epoch)$ a revokačního handleru m_r dále vytvoří pseudonym $C = g_1^{\frac{1}{i - m_r + H(epoch)}}$. Následně uživatel za pomocí osobních atributů, včetně revokačního handleru $(m_1, \ldots, m_{n-1}, m_r)$, jednotlivých podpisů σ a hodnoty *epoch* sestaví důkaz $\pi = (e, s_{m_{z \notin D}}, s_v, s_{m_r}, s_i, s_{e_1}, \ldots, s_{e_k})$. Vytvořený důkaz je následně odeslán spolu s odhalenými atributy $m_{z \in D}$ (kde D značí množinu odhalených atributů) a body $(C, \hat{\sigma}, \hat{\sigma_{e_1}}, \ldots, \hat{\sigma_{e_k}}, \bar{\sigma_{e_1}}, \ldots, \bar{\sigma_{e_k}})$ k ověření znalosti atributů.
- **Verify** - Ověřovatel odešle uživateli *nonce* a údaj o aktuální časové epoše *epoch*. Uživatel sestaví důkaz a ten odešle ověřovateli. Ověřovatel ověří výsledný důkaz π a nahlédne, zda pseudonym C není na revokační listině. Výstupem ověření je 0/1, tedy zda byl důkaz přijat či zamítnut.

- **Revoke** - Vstupem je seznam revokačních handlerů, revokační listina, soukromý klíč revokační autority a transkript ověřovací transakce. Výstupem je revokační listina rozšířená o všechna C, které je daný uživatel schopen vypočítat. Revokace probíhá mezi ověřovatelem a revokační autoritou.

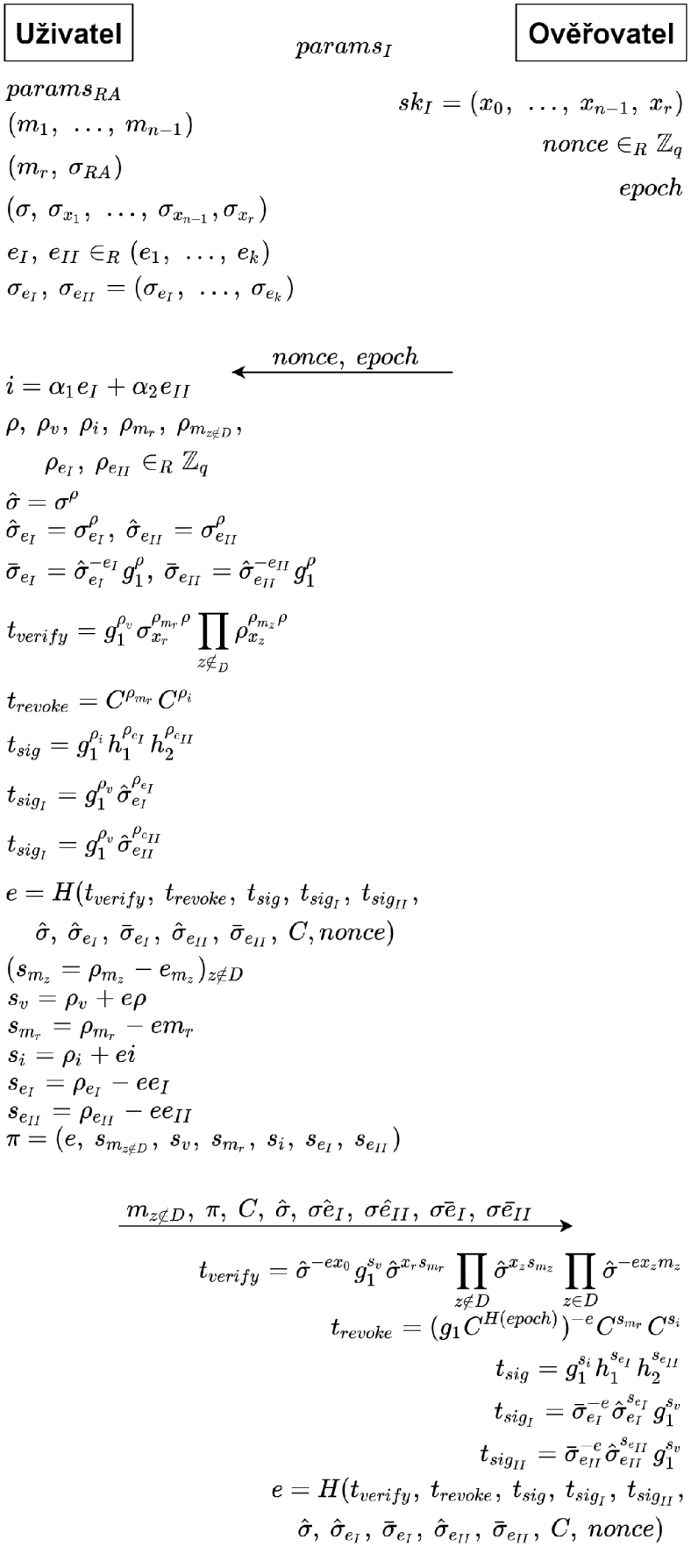
V rámci této práce je toto schéma implementováno pouze pro ověření, nikoliv revokaci, ačkoliv je **SetupRA** a **IssueRA** součástí výsledného kódu. Dále nejsou implementovány operace s výpočty bilineárního párování.



Obr. 1.8: RKVAC - vydání revokačního handleru



Obr. 1.9: RKVAC - vydání atributů



Obr. 1.10: Ověření v RKVAC schématu

2 Praktická implementace

Prvním krokem k vypracování tohoto zadání byla volba vhodné platformy pro implementaci algoritmu atributové autentizace. Jelikož cílem práce bylo realizovat stranu terminálu atributové autentizace na výkonově omezeném zařízení, jednou z nejvhodnějších platforem byla platforma Raspberry Pi. V tomto případě konkrétně Raspberry Pi 3 Model B+. Nejprve byl protokol implementován pouze na zařízení Raspberry Pi, bez použití komunikace s čipovou kartou. Bylo tak učiněno pro usnadnění testování implementace na kartě a ověření, zda jsou výpočty na straně terminálu správné.

2.1 Raspberry Pi

Jedná se o modulární počítač malých rozměrů (velikost kreditní karty) [3] vytvořený s úmyslem poskytnout vhodnou platformu pro rozvoj programovacích schopností, digitální tvorby a podobných dovedností, které umožňují běžné PC. Srdcem této platformy je poměrně malý výkonný čtyřjádrový procesor architektury ARM. Typy procesoru se liší s jednotlivými modely Raspberry Pi. Konkrétní zařízení zvolené pro tuto práci je Raspberry Pi 3 model B+ [4]. Jedná se o finální verzi Raspberry Pi 3. Následující verzí je Raspberry Pi 4. Tento konkrétní model (model B+) má oproti modelu B výkonnější procesor (Broadcom BCM2837, 1,4 GHz) umožňující použití 64-bitové verze operačního systému. Ovšem v tomto případě je použit 32-bitový operační systém standardní pro platformu Raspberry Pi. Zařízení umožňuje připojení do sítě za pomocí Ethernet adaptéru nebo bezdrátového připojení (od verze Pi 3). V tomto případě umožňuje připojení Gigabit Ethernet (1000Base), na rozdíl od standardního modelu B, který nabízí pouze Fast Ethernet (100Base). Zařízení má vlastní GPU (grafický procesor) umožňující vykreslování obrazu ve FullHD rozlišení (1920x1080 pixelů) a 30/60 snímků za vteřinu. Kapacita operační paměti (RAM) je 1 GB a není možné ji změnit bez většího zásahu do zařízení. K připojení monitoru slouží HDMI port. K zařízení je rovněž možno připojit modul s displejem (běžným i dotykovým). Mezi další užitečné připojitelné moduly se řadí například různé typy senzorů (teplota, tlak apod.), GPS modul a číslicové klávesy. Pro ovládání zařízení jsou poskytnuty USB (Universal Serial Bus) porty k připojení vybraných periférií. Zařízení má celkem 4 USB porty. K napájení slouží 5,1 V zdroj s micro USB konektorem. Stejně jako u ostatních podobných zařízení je náhlé odpojení od zdroje škodlivé a může způsobit problémy. Kompatibilními operačními systémy jsou vybrané distribuce systému Linux pro procesory typu ARM. Operační systém je uložen na SD kartě. Konkrétní doporučenou distribucí operačního systému Linux je Raspbian.

2.2 Operační systém Raspbian

Raspbian [5] je 32-bitovým operačním systémem. Jedná se o volně dostupný operační systém pro Raspberry Pi. Systém je doporučen výrobcem a zároveň je nejvíce rozšířeným operačním systémem pro zařízení Raspberry Pi. Raspbian je mírně upraveným portem Linux distribuce Debian. Změny spočívají ve změně způsobu ukládání hodnot aplikací do registrů, tedy přechod ze "soft float ABI"na "hardware floating point", který je podporován procesorem Raspberry PI. Původní "přímý"port využívající "soft float"vyžadoval přesun dat do registru pro "hard float"během chodu programu, a tedy způsoboval pokles výkonu zařízení. Raspbian obsahuje téměř všechny aplikační balíčky jako ARM verze systému Debian. Raspbian poskytuje uživateli k ovládání jak grafické rozhraní, tak příkazový řádek, jak je u operačních systémů Linux zvykem. Systém je možno nainstalovat manuálně nebo za pomocí instalačního nástroje Noobs, který je volně dostupný na stránkách výrobce Raspberry Pi. Součástí procesu instalace za pomocí nástroje Noobs je příprava SD karty pro uložení operačního systému. Prvním krokem je tedy volba karty a následné formátování. Tyto kroky musí být provedeny na jiném zařízení než na zařízení Raspberry Pi, na které má být daný operační systém nainstalován. Po dokončení příprav systému na SD kartě proběhne instalace na samotném zařízení Raspberry Pi. Po poměrně krátké instalaci je zařízení okamžitě připraveno k použití. Systém po instalaci obsahuje množství předinstalovaných aplikací. Mezi dané aplikace se řadí například webový prohlížeč Mozilla Firefox a různé aplikace pro rozvoj programovacích schopností.

2.3 Vývojové prostředí NetBeans

Pro vytvoření aplikace představující strany vydavatele a ověřovatele atributové autentizace bylo použito vývojové prostředí NetBeans [10]. NetBeans umožňuje uživateli vyvíjet software v mnoha programovacích jazycích. Primárním programovacím jazykem vývojového prostředí NetBeans je Java. Vývojové prostředí NetBeans je rovněž vhodné pro vývoj webových aplikací s využitím HTML5, JavaScript a CSS. Další možné alternativy jsou PHP a C/C++. Vybraným jazykem pro tuto implementaci je jazyk C. Hlavní funkcí tohoto vývojového prostředí, která byla klíčová pro volbu NetBeans, je "Remote deployment". Remote deployment umožňuje uživateli kompilovat a spouštět aplikace na vzdáleném zařízení. Tato funkce v kombinaci s využitím SSH (Secure Shell) dovoluje uživateli pracovat na aplikaci a správě Raspberry Pi vzdáleně, a tudíž snižuje požadavky na dodatečný hardware, jako je například monitor, klávesnice apod. K vytváření spustitelných souborů při vývoji v jazyce C na operačních systémech Linux slouží soubor Makefile, který je generován prostře-

dím NetBeans. Parametry pro vytváření spustitelných souborů a jejich spouštění je nutno upravit, jelikož některé knihovny použité pro tuto práci vyžadují dodatečný argument při volání funkcí a knihoven.

2.4 Použité nástroje systému Linux

V rámci implementace wBB podpisu a MAC_{wBB} algoritmu byly použity následující nástroje:

- **PCSC-lite** [6]– Balíček služeb pro operační systém Linux zaměřený na komunikaci s čipovými kartami a jejich čtečkami. Součástí balíčku nástrojů jsou knihovny pro programování v různých programovacích jazycích. Použité PCSC knihovny pro tuto implementaci (`PCSC/wintypes.h a PCSC/winscard.h`) jsou součástí balíčku `PCSC-lite` pro operační systémy Linux. V rámci této aplikace byly tyto knihovny použity pro komunikaci s čipovou kartou (více v části 2.5).
- **OpenSSL** [11]– Je balíček nástrojů pro operační systémy Linux. Jedná se o jeden z nejpoužívanějších balíčků zaměřujících se na kryptografii. Součástí tohoto balíčku jsou knihovny vývoje aplikací. V rámci této implementace jsou použity knihovny (`openssl/sha.h, openssl/ec.h a openssl/obj_mac.h`) pro práci s eliptickými křivkami a SHA-1 hashem. Významné prostředky poskytnuté knihovnami OpenSSL:
  - Prostředky knihovny BN [8]– Umožňují uživateli práci s celými čísly libovolné délky. Jádrem je datový typ `BIGNUM`. Většina funkcí knihovny pracuje s tímto datovým typem.
  - Prostředky knihovny EC [9]– Slouží k operacím na eliptických křivkách. Struktura `EC_POINT` představuje bod eliptické křivky, jejíž součástí jsou souřadnice daného bodu. Struktura `EC_GROUP` definuje parametry zvolené křivky.
- Standardní C knihovny – Mezi použité standardní knihovny jazyka C se řadí stdio.h, stdlib.h, time.h, unistd.h a string.h.

2.5 Komunikace s čipovými kartami

Komunikaci mezi zařízením Raspberry Pi a čipovou kartou umožňují funkce balíčku nástrojů PCSC-lite. V případě této práce byly použity knihovny `PCSC/wintypes.h` a `PCSC/winscard.h`. Tyto knihovny poskytují množství datových typů a funkcí pro komunikaci s čipovými kartami a čtečkami. Jelikož komunikace není zaměřením této konkrétní práce, byl využit vzorový kód [7] dostupný na stránkách vydavatele této knihovny. Vzorový kód představuje různé způsoby navázaní spojení a komunikace

s čipovou kartou. Tento vzorový kód byl upraven pro potřeby této práce. Prvním krokem je navázaní spojení se čtečkou čipových karet a následně také s čipovou kartou. Po úspěšném navázání spojení je zahájena transakce umožňující komunikaci s čipovou kartou bez nutnosti opakovaného navazování spojení s čipovou kartou. Transakce je zahájena následující funkcí:

```
rv = SCardBeginTransaction(hCard);                                    1
```

Proměnná *rv* (return value) představuje návratovou hodnotu funkce pro debugging. Proměnná *hCard* nese informaci o zvolené kartě pro komunikaci. Po zahájení transakce je možno zahájit komunikaci. Komunikace s čipovými kartami probíhá formou *Application Protocol Data Unit* (APDU) zpráv. APDU zprávy jsou `BYTE` řetězce ve stanoveném formátu. První zaslanou APDU zprávou po zahájení komunikace je vždy *Application ID* (AID) select. Pomocí AID select APDU zprávy je zvolena konkrétní aplikace čipové karty, která bude využita. Konkrétní odesílaná AID select zpráva vypadá následovně:

```
BYTE pbAIDSelect[] = {0x00, 0xA4, 0x04, 0x00, 0x04,                   1
                      0xF0, 0x00, 0x00, 0x01};                        2
```

První čtyři bajty `(0x00, 0xA4, 0x04, 0x00)` jsou pevně určeny pro AID select. Pátý bajt určuje délku vybraného AID v bajtech. Zbylé bajty popisují zvolené AID. Další APDU zprávy již volají konkrétní instrukce čipové karty. Příkladem takové APDU zprávy je zpráva pro zasílání podpisu sigma na čipovou kartu:

```
BYTE pbSendData[255] = {0x80, 0x10, 0x00, 0x00, 0x61, 0x04};          1
```

První bajt této zprávy určuje instrukční třídu zprávy. Druhý bajt obsahuje instrukční kód, který volá konkrétní funkci naprogramovanou na čipové kartě. Následující dva bajty jsou parametry. V rámci této aplikace tyto bajty nemají žádný význam, ale musí být součástí APDU zprávy. Pátý bajt určuje délku odesílaných dat (v tomto případě 97 bajtů). Následující bajt (`0x04`) stanovuje, že odesílaný bod křivky má pouze X a Y souřadnice. Ačkoliv je celá APDU zpráva 10 bajtů dlouhá, v ukázce je deklarováno 255 bajtů (maximální velikost APDU zpráv). Jelikož se nejedná o jediná data, která v tomto programu terminál zasílá, tak je pro zvýšení přehlednosti a efektivity vytvořena jediná proměnná *pbSendData* pro odesílání dat. Data, konkrétní instrukce a velikost dat jsou do proměnné vkládány za pomocí funkce `memcpy()`. V tomto případě konkrétně:

```
memcpy(pbReceiveData+1, &instruction , 1);
memcpy(pbSendRequest+4, &size , 1);
EC_POINT_get_affine_coordinates_GFp(curve, sigma, x, y, NULL)
  BN_bn2bin(x, x_bin);
  BN_bn2bin(y, y_bin);
  memcpy(pbSendData+5, fill , 1);
  memcpy(pbSendData+6, x_bin , VALUE_SIZE);
  memcpy(pbSendData+6+VALUE_SIZE, y_bin , VALUE_SIZE);
```

Proměnná *instruction* nese kód použité instrukce k odeslání `EC_POINT` bodu *sigma_ra* a revokačního handleru *mr*. Souřadnice bodu eliptické křivky jsou získány za pomocí funkce `EC_POINT_get_affine_coordinates_GFp()`. Souřadnice jsou uloženy do proměnných x a y. Tyto hodnoty datového typu `BIGNUM` je dále třeba převést na řetězce použitím funkce `BN_bn2bin()` (výstupem jsou řetězce *x_bin* a *y_bin*). Získané řetězce jsou následně přidány k APDU zprávě *sigma_ra* pomocí funkce `memcpy()`. Stejným způsobem jsou vytvořeny APDU zprávy pro odeslání ostatních dat. Zasílání APDU zpráv je provedeno pomocí následujících funkcí:

```
dwSendLength = 102;
dwRecvLength = sizeof(pbRecvBuffer);
rv = SCardTransmit(hCard, pioSendPci, pbSendData,
  dwSendLength, &pioRecvPci, pbRecvBuffer, &dwRecvLength);
```

První řádek stanovuje délku odesílaných dat. Druhý příkaz určuje velikost bufferu pro přijímaná data. V tomto případě se jedná pouze o potvrzení od karty o vykonání instrukce. Následně funkce `SCardTransmit()` data odešle.

Žádost o poskytnutí dat čipovou kartou je pevně stanovena APDU zpráva, která má tuto formu:

```
BYTE pbSendRequest[5] = {0x80, 0xC0, 0x00, 0x00};
```

První bajt opět určuje instrukční třídu zprávy. Druhý bajt definuje žádost o vyzvednutí dat ve vybraném formátu. Následující dva bajty jsou opět bajty nesoucí dodatečné parametry, které nejsou v této aplikaci využity. Pátý a poslední bajt není deklarován, jelikož určuje délku očekávaných dat a ta je pro každý příjem dat stanovena zvlášť.

2.6 Implementace SetupI a SetupRA

Protokoly `SetupI` a `SetupRA` jsou součástí jednoho kódu nazvaného `RKVAC_GEN`. Jelikož v tomto případě terminál vykonává funkci vydavatele, revokační autority

a ověřovatele zároveň, tak není nutno vybraný kód rozdělovat na jednotlivé aplikace. Tato část kódu je spuštěna jen jednou a vygeneruje vstupní parametry všech ostatních protokolů. Parametry jsou manuálně zkopírovány a vloženy na zařízení uživatele (kartu) a terminál. Samotná implementace těchto částí by mohla být optimalizována a automatizována. Úprava této části kódu na jednotlivé samostatné aplikace by nebyla nijak složitá.

Výstupem protokolu `SetupI` jsou parametry křivky $params_I = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$, parametr e je vynechán, jelikož se v implementaci nepracuje s bilineárním párováním. Výstupem protokolu SetupRA jsou parametry $params_{RA} = (k = 10, j = 2, (\alpha_1, \alpha_2), (h_1, h_2), \{(e_1, \sigma_{e_1}), (e_{10}, \sigma_{e_{10}})\})$. Protokol je nastaven pro 100 ověření za jednu epochu, tedy $k^j = 100$, kde $k = 10$ a $j = 2$.

Prvním krokem je stanovení vstupního parametru pro oba protokoly, `SetupI` a `SetupRA`. Vstupním parametrem je eliptická křivka BN-256 z knihovny `herumi/mcl`. K práci s eliptickými křivkami s knihovnou `OpenSSL/ec.h` je třeba vytvořit objekt `ctx` pomocí funkce `BN_CTX_new()`. Daný objekt je dále využíván u všech ostatních operací s body eliptické křivky. Po vytvoření objektu je již možno vytvořit generátor (*generator1*) a vložit parametry křivky do proměnných. Druhý generátor (*generator2*) je získán umocněním prvního generátoru. Zde je generátor umocněn na druhou (*exp* = `0x02`).

Výpis 2.1: Nastavení křivky OpenSSl

```
//Set up the BN_CTX                                                    1
if (NULL == (ctx = BN_CTX_new()));                                     2
                                                                       3
//Create the curve instance                                            4
curve = EC_GROUP_new_curve\_GFp(p, a, b, ctx);                         5
                                                                       6
//Create the generator point                                           7
generator1 = EC_POINT_new(curve);                                      8
EC_POINT_set_affine_coordinates(curve, generator1, x, y, ctx);         9
                                                                       10
//Set the generator and the order for the curve                        11
EC_GROUP_set_generator(curve, generator1, order, NULL);                12
                                                                       13
//Creating second generator point                                      14
generator2 = EC_POINT_new(curve);                                      15
EC_POINT_mul(curve, generator2, NULL, generator1, exp, ctx);           16
```

Funkce `EC_GROUP_new_curve_GFp()` vytvoří instanci křivky, ke které přiřazujeme veškeré parametry, jako je generátor a řád. Generátor a řád je přiřazen za pomocí

funkce `EC_GROUP_set_generator()`. Funkce `EC_POINT_mul()` představuje násobení (umocnění) bodu na křivce. Vykonaná operace tedy představuje $g_2 = g_1^2$.

Jakmile je toto základní nastavení dokončeno, protokol zahájí výpočty jednotlivých parametrů. Prvním krokem je vygenerování náhodných hodnot *alpha1*, *alpha2* a *sk*, pomocí kterých jsou vypočítány body *h1* a *h2*. Body jsou získány následovně:

Výpis 2.2: Část implementace SetupRA

```
h = EC_POINT_new(curve);
for(j=0; j < J; j++)
{
  for (i = 0; i < VALUE_SIZE; i++)
      alpha_bin[i] = rand();
  alpha = BN_bin2bn(alpha_bin, 32, NULL);

  //h = g1^alpha
  EC_POINT_mul(curve, h, NULL, generator1, alpha, ctx);
}
```

Konstanta J představuje hodnotu j v protokolu. V této implementaci se jedná o hodnotu 2. Tato hodnota určuje počet vygenerovaných hodnot *alpha* a vytvořených bodů h.

Posledním krokem je vygenerování náhodných hodnot e_k a vytvoření podpisů $\sigma_{e_k} = g_1^{\frac{1}{e_k+sk_{RA}}}$ na ně. K výpočtu těchto podpisů je použit soukromý klíč revokační autority sk_{RA}, který byl vygenerován společně s proměnnými *alpha1* a *alpha2* v předchozí ukázce zdrojového kódu. Operaci $\frac{1}{e_k+sk_{RA}}$ představuje funkce `BN_mod_inverse()`, která je konkrétně v kódu použita následovně:

Výpis 2.3: Tvorba bodu σ

```
//e+sk
BN_mod_add(exp, e, sk, order, ctx);

//1(e+sk)
BN_mod_inverse(exp, exp, order, ctx);

//Calculating sigma_e
sigma_e = EC_POINT_new(curve);
EC_POINT_mul(curve, sigma_e, NULL, generator1, exp, ctx);
```

Tyto operace jsou vykonány k-krát, a tedy vygenerují k náhodných hodnot e_k a podpisů σ_{e_k}

2.7 Protokoly vydání

Oba protokoly vydání, `IssueRA` a `IssueI`, jsou součástí hlavní části kódu, která má rovněž na starosti ověření znalosti atributů uživatele. Důvodem je, že terminál (zařízení Raspberry Pi) představuje revokační autoritu, vydavatele a ověřovatele zároveň. Rozdělit kód pro jednotlivé strany by nebyl závažný problém.

2.7.1 Protokol vydání revokačních parametrů

Vstupem tohoto protokolu jsou parametry křivky $params_I = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$, vytvořeny protokolem `SetupI`, a parametry $params_{RA} = (k = 10, j = 2, (\alpha_1, \alpha_2), (h_1, h_2), \{(e_1, \sigma_{e_1}), (e_{10}, \sigma_{e_{10}})\})$, které jsou vytvořeny protokolem `SetupRA`. Výstupem je revokační handler *mr* a podpis *sigma_ra*. Revokační handler *mr* je náhodná hodnota.

K vytvoření podpisu *sigma_ra* potřebuje revokační autorita *ID* uživatele. Tato hodnota je od karty získána zasláním APDU zprávu s instrukcí `0x05`. Karta navrátí terminálu hodnotu *ID* uživatele a revokační autorita může zahájit tvorbu podpisu. Součástí vytvoření podpisu $\sigma_{RA} = g_1^{\frac{1}{H(mr||ID)+sk_{RA}}}$ je tvorba hashe hodnot *mr* a *ID*. Zvolený hash pro tyto hodnoty je `SHA-256`. Hash je proveden za pomocí knihovny `openssl/sha.h`. Tvorba hashe vypadá následovně:

Výpis 2.4: Tvorba hashe SHA-256

```
//Creating hash of ID and mr                                    1
SHA256_CTX sha256_ctx;                                          2
SHA256_Init(&sha256_ctx);                                       3
SHA256_Update(&sha256_ctx, mr_bin, VALUE_SIZE);                 4
SHA256_Update(&sha256_ctx, id_bin, VALUE_SIZE);                 5
SHA256_Final(hash_ra, &sha256_ctx);                             6
```

Funkce pro vytvoření hashe, podobně jako při práci s křivkami, vyžadují deklaraci vlastního objektu *sha256_ctx*. Jakmile je objekt vytvořen, funkcí `SHA256_Init()` se inicializuje tvorba hashe. Pomocí volání funkce `SHA256_Update()` jsou vkládána data, která budou hashována. Následným zavoláním funkce `SHA256_Final()` je na základě vložených dat vytvořen hash.

Po získání hashe je vytvořen podpis, který je následně odeslán společně s revokačním handlerem *mr* uživateli. APDU zpráva pro odeslání zmíněných dat je označena instrukcí `0x10`. Odesláním dat uživateli je vydání revokačních handlerů dokončeno. Vydání revokačních handlerů průměrně trvá 281 ms (průměr za 100 měření).

2.7.2 Protokol vydání atributů

Tento protokol je vykonán vydavatelem. Vstupními parametry jsou parametry křivky $params_I = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$. Výstupem jsou σ podpisy $(\sigma, \sigma_{x_1}, \sigma_{x_2}, \sigma_{x_r})$ na jednotlivé atributy uživatele (*m1*, *m2*), včetně revokačního handleru *mr*.

V prvním kroku vydavatel přijme atributy *m1*, *m2* a *mr* od uživatele. Tyto atributy si terminál vyžádá od karty uživatele za pomocí APDU zprávy s instrukcí `0x20`. Jakmile vydavatel získá všechny atributy, může zahájit vydání všech podpisů $(sigma,\ sigma_x1, sigma_x2, sigma_xr)$. Nejkomplikovanější částí tohoto protokolu je tvorba podpisu $\sigma = g_1^{\frac{1}{x_0+m_1x_1+m_2x_2+m_rx_r}}$. V rámci těchto výpočtů nejsou použity žádné dříve nezmíněné funkce.

Protože by výsledné 4 body přesáhly maximální velikost přenášených dat APDU zprávou, data jsou odeslána za pomocí dvou zpráv. Zpráva s instrukcí `0x25` slouží k přenosu bodů *sigma* a *sigma_xr*. Druhá zpráva s instrukcí `0x26` nese body *sigma_x1* a *sigma_x2*. Po odeslání těchto bodů je tento protokol kompletní. Průměrná doba vykonání této části kódu je 630 ms (průměr za 100 měření).

2.8 Protokol ověření

Implementace protokolu pro ověření uživatelovy znalosti atributů je nejdelší částí kódu implementace. V rámci této konkrétní implementace se kód nachází ve stejném programu, jako předchozí dva popsané protokoly (`IssueRA`, `IssueI`). Vstupními parametry jsou $params_I = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$ a klíč $sk_I = (x_0, x_1, x_2, x_r)$. Výstupem tohoto protokolu je 0/1, tedy zda je důkaz znalosti přijat či zamítnut.

Prvním krokem je vygenerování výzvy *nonce* a údaje o aktuálním čase *epoch*. Tyto hodnoty jsou následovně zaslány kartě uživatele k dalšímu zpracování. Generování hodnoty *nonce* a vytvoření hodnoty *epoch* vypadá následovně:

Výpis 2.5: Vytvoření hodnoty *epoch*

```
//generating random nonce                                          1
for (i = 0; i < VALUE_SIZE; i++)                                   2
  nonce_bin[i] = rand();                                           3
                                                                   4
//creating epoch                                                   5
struct tm *t = localtime(&now);                                    6
strftime(epoch_bin, sizeof(epoch_bin)-1, "%d/%m/%Y", t);           7
```

Do proměnné *t* je uložen časový údaj z momentu zavolání funkce `localtime(&now)`. Tento údaj je dále převeden na řetězec ve zvoleném formátu funkcí `strftime()`. Je stanoveno, že maximální počet pokusů je 100 na jeden den, použitý formát je

"Den/Měsíc/Rok". K odeslání zmíněných hodnot je použita APDU zpráva s instrukcí `0x30`.

Po odeslání výzvy jsou zavolány instrukce karty, které vytvoří důkaz a odešlou jej terminálu. Celkový počet hodnot k ověření je příliš velký na jednu APDU zprávu, a tudíž je rozdělen do několika zpráv. První zpráva s instrukcí `0x50` zahájí přenos hodnot $\hat{\sigma}$, $\hat{\sigma}_{e_I}$a $\hat{\sigma}_{e_{II}}$. Zprávou s instrukcí `0x51` jsou vyžádány hodnoty C, $\bar{\sigma}_{e_I}$ a $\bar{\sigma}_{e_{II}}$. A poslední zprávou s instrukcí `0x52` jsou přijaty hodnoty důkazu $\pi = (s_{m_1}, s_v, s_{m_r}, s_i, s_{e_I}, s_{e_{II}})$ a zvolený odhalený atribut m_2. Příklad zpracování přijatých dat:

Výpis 2.6: Zpracování přijatých dat od karty

```
/* PROCESSING RECEIVED DATA */                                    1
  for (i = 0; i < VALUE_SIZE; i++)                                2
    x_bin[i] = pbRecvBuffer[i+1];                                 3
                                                                  4
  for (i = 0; i < VALUE_SIZE; i++)                                5
    y_bin[i] = pbRecvBuffer[i+1+VALUE_SIZE];                      6
                                                                  7
  if (NULL == (x = BN_bin2bn(x_bin, VALUE_SIZE, NULL)));          8
  if (NULL == (y = BN_bin2bn(y_bin, VALUE_SIZE, NULL)));          9
                                                                  10
  sigma_roof = EC_POINT_new(curve);                               11
  EC_POINT_set_affine_coordinates(curve, sigma_roof,              12
                                        x, y, ctx);               13
```

Ukázka je zbavena výpisu přijímaných dat a ukazuje pouze jejich zpracování a uložení. Veškerá přijatá data od karty jsou uložena do bufferu *pbRecvBuffer*. Jejich pořadí je pevně dáno a stanoveno na základě domluvy s kolegou Janem Brodou, který implementoval protokol na kartě uživatele. Klíčem ke správnému zpracování dat je důsledná orientace v bufferu při ukládání dat. Hodnota `'+1'` zajišťuje, že při ukládání je přeskočen bajt s hodnotou `0x04`, který je vždy umístěn před X souřadnicí bodu křivky. Karta tento bajt používá při práci s body eliptické křivky. Konstanta `VALUE_SIZE` má velikost 32 bajtů, tedy velikost jedné souřadnice bodu křivky `BN-256`. Po uložení hodnoty do `BYTE` řetězce je hodnota převedena na proměnnou datového typu `BIGNUM`, se kterou pracují funkce knihoven `OpenSSL`. Poslední dva řádky jsou inicializace bodu křivky *sigma_roof* a následné přiřazení uložených souřadnic x a y tomuto bodu.

V rámci zavolání instrukce `0x50` karta vyhodnotí, zda nebyl překročen limit ověření za jeden den. Jestli je limit překročen, karta navrátí hodnotu `0x64 0x04`. Terminál tedy po zavolání této instrukce vždy kontroluje první dva bajty navrácené hodnoty. Kontrola vypadá následovně:

Výpis 2.7: Kontrola počtu pokusů

```
if(memcmp(pbRecvBuffer, limit_exc, 2)==0){
    printf("\n\nNumber␣of␣tries␣exceeded!");
    goto end;
  }
```

Funkce `memcmp()` porovná řetězce, a pokud jsou shodné, navrátí hodnotu `0`. V tomto případě kód nahlásí překročení maximálního limitu ověření a přejde na konec programu.

Po zpracování všech potřebných parametrů je zahájen výpočet ověřovacích bodů pro ověření uživatelovy znalosti atributů. Zmíněné body jsou: t_{verify}, t_{revoke}, t_{sig}, t_{sig_I} a $t_{sig_{II}}$. Výpočty jednotlivých bodů jsou totožné s výpočty na obrázku 1.10.

Výpis 2.8: Ukázka výpočtu ověřovacího bodu

```
//calculating t_verify_v
t_ver_v = EC_POINT_new(curve);
BN_mod_mul(exp, x0, e, order, ctx);
EC_POINT_mul(curve, t_ver_v, NULL, sigma_roof, exp, ctx);
EC_POINT_invert(curve, t_ver_v, ctx);
EC_POINT_mul(curve, ec_temp, NULL, generator1, sv, ctx);
EC_POINT_add(curve, t_ver_v, t_ver_v, ec_temp, ctx);
BN_mod_mul(exp, xr, smr, order, ctx);
EC_POINT_mul(curve, ec_temp, NULL, sigma_roof, exp, ctx);
EC_POINT_add(curve, t_ver_v, t_ver_v, ec_temp, ctx);
BN_mod_mul(exp, x1, sm1, order, ctx);
EC_POINT_mul(curve, ec_temp, NULL, sigma_roof, exp, ctx);
EC_POINT_add(curve, t_ver_v, t_ver_v, ec_temp, ctx);
BN_mod_mul(exp, x2, e, order, ctx);
BN_mod_mul(exp, exp, m2, order, ctx);
EC_POINT_mul(curve, ec_temp, NULL, sigma_roof, exp, ctx);
EC_POINT_invert(curve, ec_temp, ctx);
EC_POINT_add(curve, t_ver_v, t_ver_v, ec_temp, ctx);
```

Při výpočtech byly použity pomocné proměnné *exp* (BIGNUM) a *ec_temp* (`EC_POINT`). Funkce `EC_POINT_invert()` vypočítá inverzi vybraného bodu na křivky.

Posledním krokem ověření je výpočet ověřovacího hashe a následné porovnání s hashem, který byl zaslán kartou. Zvolený typ hashe je `SHA-256`. Formát a pořadí vstupních dat hashe musí být shodný s formátem a pořadím na kartě. Při převodu jednotlivých souřadnic bodů na řetězce za pomocí funkce `BN_bn2bin()` mohou nastat problémy, protože délka hodnoty datového typu `BIGNUM` není fixní. Při převodu dat z proměnné typu `BIGNUM` na řetězec dané fixní délky může nastat chyba. Například při ukládání 30 bajtů dat z proměnné typu `BIGNUM` do řetězce o délce 32 bajtů,

dojde k uložení dat do prvních 30 bajtů, namísto posledních 30 bajtů. Jestliže cílový řetězec již obsahoval nějaká data, poslední 2 bajty zůstanou nepozměněné, a řetězec tedy bude obsahovat data nevhodná pro tvorbu hashe. Pro vyřešení tohoto problému byla vytvořena následující funkce:

Výpis 2.9: Funkce checkZeroes

```
void checkZeroes(const BIGNUM *a, unsigned char
        *buff, size_t VALUE_SIZE){
  unsigned char temp[VALUE_SIZE];
  int x;
  x = VALUE_SIZE - BN_num_bytes(a);
  if(x > 0)
  {
    for(int i=0; i < x; i++)
      buff[i] = 0x00;
    BN_bn2bin(a, temp);
    memcpy(buff + x, temp, sizeof(temp));
  }
  else{
    BN_bn2bin(a, buff);
  }
}
```

Vstupními parametry funkce `checkZeroes()` jsou hodnota *a* datového typu `BIGNUM`, která má být převedena na řetězec, řetězec *buff*, do kterého bude výsledek uložen, a velikost `VALUE_SIZE`, která stanovuje očekávanou velikost výsledného řetězce. Funkce nejprve zkontroluje velikost hodnoty *a*. Jestliže je rozdíl velikostí x mezi `VALUE_SIZE` a *a* větší než `'0'`, funkce vloží na prvních x bajtů hodnotu `0x00` a následně za ně vloží hodnotu *a* převedenou za pomocí funkce `BN_bn2bin()`. V případě, kdy je rozdíl x roven `'0'`, funkce převede hodnotu *a* pomocí funkce `BN_bn2bin()`. Převod pomocí této funkce je prováděn při přípravě dat pro vytvoření hashe. Je potřeba zajistit, že data vstupují do hashovací funkce ve stejném formátu jako na kartě.

Po vytvoření hashe ověřovatele následuje jeho porovnání s hashem e uživatele. Porovnání je provedeno následujícím způsobem:

Výpis 2.10: Porovnání výsledných hashů

```
if(memcmp(hash_verify, x_bin, SHA256_DIGEST_LENGTH) == 0)
    success = 1;
if(success == 1){
    printf("\n\nVerification␣successful!␣(%dms)",
               t_ver - t_issuer);
    goto end;
}
else
    printf("Verification␣failed!(%dms)",
               t_ver - t_issuer);
```

Jestliže jsou hashe shodné, proměnná *success* je nastavena do hodnoty `'1'` a program následně přejde na konec. V případě, že se hashe neshodují, hodnota *success* zůstane `'0'`, program ohlásí neshodu hashů a za pomocí APDU zpráv s instrukcemi `0x15` a `0x16` vyžádá ověřovací body $t_{verify}, t_{revoke}, t_{sig}, t_{sig_I}$ a $t_{sig_{II}}$ karty pro debugování. Průběh celého protokolu ověření je průměrně 3700 ms (3425 ms karta), což není příliš vhodné pro reálné využití. Průměr je spočten na základě 100 měření. Úspěšnost ověření je 100

Jak již bylo dříve zmíněno, protokol byl nejprve implementován pouze na zařízení Raspberry Pi. Implementace byla vytvořena ve spolupráci s kolegou Janem Brodou. Cílem bylo seznámit se s jednotlivými operacemi protokolu a vytvořit benchmark, podle kterého byla dále vytvořena implementace operací na kartě uživatele. Na základě této implementace byl vytvořen druhý program, který již zahrnoval komunikaci s kartou. Průběh implementace celého protokolu bez komunikace s kartou je průměrně 350 ms, což je značně rychlejší než při komunikaci s kartou.

```
ISSUING REVOCATION HANDLER

Received ID:    99 de b1 25 5b 73 35 fd 80 9a 4a 8f 90 52 41 e2 e1 91 c2 a9 67 a6 0c ed 29 bd d1 95 9c 62 7e 51
Sending sigma RA.

Revocation authority part successful! (274ms)

ISSUING ATTRIBUTES

Printing received atributes:
m1:     ce 48 f1 5e da d3 3c b6 5b 96 8f 83 6e 12 f4 27 b9 b5 18 cd d3 c1 44 63 b1 8c 3c 2c df 62 cd 5a
m2:     5c a9 0c 2b dc 9e 49 77 c4 15 cf e8 92 6b fb 22 3a d0 62 f3 bd 12 01 28 fe f6 d1 86 c8 77 11 31
mr:     35 f2 74 fb f7 a4 f9 9d dc a5 c3 7f c1 c2 47 bb ae bb e2 61 68 b6 15 c8 90 cc a0 0f 3c 61 5d 10
Sending sigma points.

Issuer part successful! (717ms)

VERIFICATION

Printing received data:
Sigma_roof_x:           18 a7 e1 02 25 39 a6 85 20 5d ba 9c dd 78 cc 9d 22 c0 f9 ec 20 38 b6 a1 b3 54 ae c0 9d fe cb c4
Sigma_roof_y:           14 9f 57 2a fb ed 81 1b 15 ff 5a cc ff 8e 36 df 2e 4e b3 74 d8 fa 10 7c c8 b4 04 bc 90 52 ae 68
Sigma_roof_ei_x:        24 8c 37 da 94 ce 01 d1 f9 0a b7 8c 26 56 d6 38 07 b8 83 d4 d6 da a2 87 e8 5d 34 e5 60 27 06 25
Sigma_roof_ei_y:        19 20 6b 66 75 27 32 31 e1 d1 57 b6 29 cc be d6 7c 74 89 5b ee d2 ff b7 ea 8f 86 1d 2d 08 ba 09
Sigma_roof_eii_x:       19 37 32 8b 5b 18 b1 49 32 01 83 ed b9 c5 4b 98 42 58 2a 3d d3 1a 70 20 71 bf 9e 8e 5e ba 40 b4
Sigma_roof_eii_y:       19 fd 60 64 9a 4b bd 9a 08 be 21 9d d7 c3 4b ce 86 7e b5 61 2b 98 42 51 11 39 cd 8a cf 88 09 bd
C_x:                    0d fb e7 19 9c 86 77 e8 22 3d f1 64 4b f1 a8 37 df 99 7a 51 6c 57 36 b9 9f 6b 18 36 ff cd f2 76
C_y:                    05 13 92 c3 c6 cf dc de 4c 56 13 aa 0f be 41 9b 79 4b cc c8 e2 4b 40 7b fb de 83 b9 73 22 23 d8
Sigma_neg_ei_x:         11 6f c3 b1 cf b7 76 90 4f 1e 62 ff 96 ff f2 21 d5 f1 85 e6 59 7a a7 70 f9 18 30 d6 77 1b b0 2e
Sigma_neg_ei_y:         05 bf 72 53 51 38 2b 13 75 1b ee ed 7e e7 54 52 99 e5 16 35 0e e8 ce fd a2 70 89 d8 d7 c1 50 31
Sigma_neg_eii_x:        23 3b e9 e8 29 f2 29 f1 6a ac 3f 2e 13 7b 1f 7c ae b6 4f ab b9 58 0c 23 5b 08 f0 2a a2 1c d8 a0
Sigma_neg_eii_y:        09 54 da 00 38 d8 9c 99 06 8c 07 84 dd b1 0d 26 89 fb aa a3 77 7e ce 9c d1 c6 b5 1d fb 94 02 4e
e:                      ef ae e8 1f b9 37 49 75 32 ec 4a 6e 8c c9 63 db d2 c0 e4 e8 14 bc 1e a6 cd 45 5c 70 bc 99 07 2a
sm1:                    01 e0 19 8f 1f d2 1a 8e 3a 9a 81 11 77 88 7a 46 d6 8f 9f 61 dd 80 cc ad f8 3c 32 c6 0f b8 24 99
sv:                     11 ff 35 bf 21 55 ed b0 ab 0f 54 b0 ab 59 70 2e e6 b1 8a 20 23 af 87 61 0d 1c 57 e8 d9 91 9a 0a
smr:                    16 2e ba 5c 90 43 59 ee 75 6b 29 01 39 62 1c 38 ed be 66 f4 59 58 f0 00 43 fa 48 21 37 2b 95 6c
si:                     0c 83 be ed 04 75 39 66 33 98 e8 3f 36 c7 b5 e8 30 f5 0b bd 5b 65 23 72 0f 19 b3 6d a7 cc d8 10
sei:                    0d d7 82 6d 86 d2 af c6 33 41 e8 c1 4f 21 29 4a 6a 58 98 a1 1b f9 50 7b 2a 5c e4 af 61 cb 0e 07
seii:                   0e 08 4e 5e 14 24 6d f7 c2 14 e2 bb 61 1a 62 56 6f 44 34 58 6f 6f c7 fb a0 ba 04 f7 3f ac 74 b5
m2:                     5c a9 0c 2b dc 9e 49 77 c4 15 cf e8 92 6b fb 22 3a d0 62 f3 bd 12 01 28 fe f6 d1 86 c8 77 11 31

Creating verifying hash.

hash_verifier:  ef ae e8 1f b9 37 49 75 32 ec 4a 6e 8c c9 63 db d2 c0 e4 e8 14 bc 1e a6 cd 45 5c 70 bc 99 07 2a
hash_user:      ef ae e8 1f b9 37 49 75 32 ec 4a 6e 8c c9 63 db d2 c0 e4 e8 14 bc 1e a6 cd 45 5c 70 bc 99 07 2a

Verification successful! (3733ms)
RUN SUCCESSFUL (total time: 5s)
```

Obr. 2.1: Výstup aplikace

Závěr

Cílem této práce bylo vytvořit aplikaci představující stranu vydavatele, ověřovatele a revokační autority pro atributovou autentizaci použitím schématu RKVAC. Aplikace byla po kryptografické stránce úspěšně vytvořena pro ověření dvou atributů. Přidání ověření dalšího atributu je dosažitelné s nenáročnými úpravami aplikací. V aplikaci implementace setup protokolů stačí navýšit konstantu attr, a tím se automaticky navýší počet generovaných parametrů. Úprava aplikace pro ověření je poněkud komplikovanější a vyžadovala by rozšíření kódu, jak na kartě, tak na terminálu. Na straně terminálu by úpravy zahrnovaly kopírování a rozšiřování již existujících částí kódu.

Strana uživatele (čipová karta) byla implementována kolegou Janem Brodou. Vzájemná spolupráce a komunikace byla stěžejní pro tvorbu této práce. V rámci realizace práce jsme se potýkali s mnohými problémy při výpočtech hlavních kryptografických hodnot. Řešení těchto problémů vyžadovalo časově náročné synchronizované přepočítávání těchto zmíněných hodnot. Jedním z problémů bylo přetékání hodnot při výpočtech na straně karty. Pro vyvarování se tomuto problému bylo nutné pečlivě sledovat stav problémových hodnot a v případě nutnosti upravit jejich velikost. V první části práce byly popsány teoretické principy atributové autentizace. Součástí tohoto popisu byl popis kryptografických metod spjatých s atributovou autentizací, jejich vlastnostmi a entitami. Dále první část obsahovala podrobnější pohled na podpisové schéma weak BonehBoyen a algebraický MAC využívající wBB podpis. Tyto algoritmy jsou součástí implementovaného schématu RKVAC v praktické části této práce.

Praktická část charakterizuje využité prvky při praktické implementaci schématu RKVAC a odůvodnění, proč byly dané prvky použity. Jako první je představeno zařízení Raspberry Pi, na kterém je tato práce realizována. Součástí této charakterizace jsou vlastnosti a parametry zmíněného zařízení. Následně je popsán operační systém Raspbian, který je používán na zařízení Raspberry Pi, vývojové prostředí NetBeans s funkcí Remote Deployment a nástroje operačního sytému Linux, jejichž knihovny umožnily práci s eliptickými křivkami a komunikaci s čipovými kartami.

Další částí je popis implementace strany vydavatele, revokační autority a ověřovatele ve schématu atributové autentizace RKVAC. Prvním krokem bylo stanovení hodnot využívaných tímto algoritmem a vydání podpisů σ na stanovené hodnoty. Součástí tohoto kroku a následujícího kroku je komunikace mezi aplikací a čipovou kartou. V rámci implementace byla zprovozněna obousměrná komunikace mezi zařízením a kartou, pomocí níž byla předávána jednotlivá data algoritmu. Posledním a nejdůležitějším krokem byla implementace strany ověřovatele.

Aplikace je vykonána v průměru za 4,6 s, což je pro reálné použití nevhodné.

Ověření je časově nejnáročnější operací této implementace. Pro zrychlení běhu celé aplikace je třeba optimalizace výpočtů na straně karty a zvolení vhodnější knihovny pro kryptografické operace na straně terminálu. Další možnou budoucí úpravou vytvořené aplikace je její přepsání do jiného programovacího jazyka pro zefektivnění, zrychlení a zjednodušení programu. Vhodnými programovacími jazyky pro tuto implementaci jsou Java a C++ díky množství dostupných knihoven jak pro komunikaci s čipovými kartami, tak pro využívané kryptografické operace.

Literatura

[1] HAJNÝ, JAN A KOL.: *Technická zpráva k projektu TAČR TL02000398. 2019* [cit. 8. 6. 2020].

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

[3] *Raspberry Pi FAQ* [online]. 2020. [cit. 8. 6. 2020]. Dostupné z URL: <https://www.raspberrypi.org/documentation/faqs/>.

[4] *Raspberry Pi 3 Model B+* [online]. 2020. [cit. 8. 6. 2020]. Dostupné z URL: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>.

[5] *Raspbian FAQ* [online]. 2020. [cit. 8. 6. 2020]. Dostupné z URL: <https://www.raspbian.org/RaspbianFAQ>.

[6] *PCSC-lite: API* [online]. 2020. [cit. 8. 6. 2020]. Dostupné z URL: <https://pcsclite.apdu.fr/api/group__API.html#details>.

[7] *PCSC sample in C* [online]. 2020. [cit. 8. 6. 2020]. Dostupné z URL: <https://ludovicrousseau.blogspot.com/2010/04/pcsc-sample-in-c.html>.

[8] *OpenSSL Bn library* [online]. 2020. [cit. 8. 6. 2020]. Dostupné z URL: <https://www.openssl.org/docs/man1.0.2/man3/bn.html>.

[9] *OpenSSL EC library* [online]. 2020. [cit. 8. 6. 2020]. Dostupné z URL: <https://www.openssl.org/docs/man1.0.2/man3/ec.html>.

[10] *NetBeans IDE features* [online]. 2020. [cit. 8. 6. 2020]. Dostupné z URL: <https://netbeans.org/features/>.

[11] *OpenSSL Documentation* [online]. 2020. [cit. 8. 6. 2020]. Dostupné z URL: <https://www.openssl.org/docs/>.

Seznam symbolů, veličin a zkratek

GPDR	General Data Protection Regulation
eIDAS	Electronic Identification and Trust Services
MAC	Message Authentication Code
wBB	weak Boneh-Boyen
KVAC	Keyed-Verification Anonymous Credentials
PKDL	Proofs of Knowledge of Discrete Logarithm - důkaz znalosti diskrétního logaritmu
ZK	Zero Knowledge - protokoly s nulovou znalostí
Σ **-protokol**	Sigma protokol
APDU	Application Protocol Data Unit
AID	Application ID